



Department of Computer Science

BSc (Hons) Computer Science (Network Computing)

Academic Year 2021 - 2022

**Analysing network traffic to detect header injections
through machine learning algorithms**

Craig Saville

1924742

A report submitted in partial fulfilment of the requirements for the degree of
Bachelor of Science

Brunel University London
Department of Computer Science
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

With the majority of information being moved onto the internet through cloud storage or simply connected networks, mitigating vulnerabilities has never been so important. A study conducted into TCP/IP stacks over an 18 month period called Project Memoria found nearly 100 vulnerabilities in TCP/IP stacks and showed “An analysis of a quarter of a million devices affected by the Project Memoria vulnerabilities showed that the highest number of vulnerable systems are housed by the government and healthcare sectors, followed by manufacturing, retail, and financial.” (Kovacs, 2021). The main goal of covert channel communication is primarily data smuggling and provides the means to leak proprietary and intellectual property or to execute code that should otherwise be prevented by company security policies. Botnets are just one type of malicious code that uses covert channels to control the already implanted malicious code, examples of these from the past are programs such as Tribal flood network and Loki.

In this report we will explore packet structures and attempt to create a system to identify potentially malicious packets within network traffic.

I have begun by creating a cyber range to mimic a typical network with a server VM, user VMs doing typical things like browsing and an attacker VM performing attacks and recorded network traffic on a separate VM, by designing a program to hopefully detect these attacks we can mitigate some of the dangers of modern computing where information is largely kept accessible anywhere.

This project will then use a stacked ensemble made from a selection of machine learning models to detect the traffic and evaluate the best results of the machine learning algorithms.

Acknowledgements

Professor Panos Louvieris for supervising my final year and providing guidance when needed.

Jonathan Wong, for being a great rubber duck when I needed to talk things out even if most of the time, he had no idea what I was on about.

I certify that the work presented in the dissertation is my own unless referenced.

Signature Craig Saville

Date 01/04/2022

Total Words: 8564

Table of Contents

Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Tables.....	iv
List of Figures.....	v
1 Introduction	7
1.1 Aims and Objectives.....	7
1.2 Project Approach.....	8
1.3 Dissertation Outline	9
2 Background research and information.....	10
2.1 Network protocols.....	10
2.2 Machine learning libraries.....	11
2.3 Machine learning algorithms.....	13
2.4 Unsupervised packet-based anomaly detection in virtual networks.....	17
2.5 Other papers.....	18
3 Methodology	18
4 Design and Data collection.....	20
4.1 Requirements.....	20
4.2 The Network.....	20
4.3 Data capture	21
5 Machine learning Implementation	24
5.1 Training the models	24
5.2 User interface.....	31
6 Testing and Evaluation	32
6.1 Testing methodology	32
6.2 Tests.....	34
7 Conclusions.....	38
7.1 Future Work.....	38
References	40

List of Tables

Table 1 ML Library comparison	12
-------------------------------------	----

List of Figures

Figure 1 TCP Header layout	10
Figure 2 IPV4 header (Guru99, 2022)	11
Figure 3 H2O cluster details.....	13
Figure 4 Random forest (Sruethi, 2021).....	14
Figure 5 Neutral Network (IBM Cloud Education, 2020).....	15
Figure 6 Gradient boosting flowchart (Zhang, 2021).....	16
Figure 7 Stacking architecture (Opengeni.us.org, 2021)	17
Figure 8 - Feature set used in background paper (Spiekermann and Keller, 2021)	18
Figure 9 – Basic network layout.....	21
Figure 10 Wireshark example.....	21
Figure 11 - Covert_TCP command line example.....	22
Figure 12 Initial RF training.....	24
Figure 13 confusion matrix example.....	25
Figure 14 Random forest confusion matrix.....	25
Figure 15 Random forest model details.....	26
Figure 16 Neural network.....	26
Figure 17 Neural network confusion matrix.....	27
Figure 18 random forest variable importance	27
Figure 19 GBM defining	28
Figure 20 initial GBM confusion matrix	28
Figure 21 configured GBM	29
Figure 22 GBM model details.....	29
Figure 23 Stacked ensemble	30
Figure 24 Stacked Ensemble details	30
Figure 25 Basic user interface.....	31
Figure 26 User interface after packet analysis.....	31
Figure 27 Dataset 1 results.....	34
Figure 28 random forest dataset1 CM.....	34
Figure 29 Dataset 2 results.....	35
Figure 30 GBM dataset 2 CM.....	35
Figure 31 Dataset3 results.....	36
Figure 32 Random forest dataset3 CM	36
Figure 33 Metric comparison	37

Figure 34 other algorithms.....	43
---------------------------------	----

1 Introduction

1.1 Aims and Objectives

We are firmly in the age of cloud computing where the majority of a person's data is accessible from anywhere in the world.

Malicious actors can utilize covert channels to evade detection from typical NIDS and send or extract information from networks due to the way packets are structured.

Within this project I aim to create a machine learning model which will analyse network traffic in order to detect covert channel attacks, specifically packets with data injected into their headers. To achieve my goal, I will need to do the following:

1. Research background data to find other papers on this subject and to help identify best algorithms and features to include in my project.
2. Design and implement a network in which to gather data for my models.
3. Create a number of datasets containing recorded network traffic with both normal browsing data and attacks to simulate standard network traffic.
4. Train various models and evaluate the ability of the model to correctly identify packets.
5. Gather best performing models to form base of stacked ensemble for final product and compare the ensemble to base models.

1.2 Project Approach

This project was developed by an iterative process in order to essentially create a vertical slice of the project, each iteration essentially ended with a model that could potentially be used as the final product, however, would not perform as well as my actual final product which is a combination of the models produced each iteration. To begin I researched the field in order to find the appropriate tools and knowledge in order to effectively produce my data as data is one of the most important steps in any project like this. Once I had the main training dataset, I began an iterative process of training a model, testing it and fine tuning where possible in order to achieve solid results, I would then repeat the steps in order to have a number of well performing models in which to combine into a stacked ensemble model in order to achieve the best results I could. At the start of some iterations in the process I would return to the data collection stage and create a new dataset, however I used the same training data from the initial iteration to train every model, the new datasets created would be saved to be used in the last stage to confirm that my results are reproduceable on separate datasets that my models hadn't learned from.

1.3 Dissertation Outline

This dissertation is laid out with the following structure:

Chapter 2 contains an overview of the background research on technologies and previous papers used within my project.

Chapter 3 explores my methodology used when approaching my project and the steps taken in order to develop my project.

Chapter 4 details my initial project data collection exploring the tools used, how they work and the thoughts around how to create a fair dataset.

Chapter 5 explores the creation of the various models and training.

Chapter 6 details the testing and evaluation of the base models and ensemble model.

Chapter 7 presents my conclusions.

2 Background research and information

2.1 Network protocols

Although TCP/IP is the way the internet works in reality, the OSI model is a conceptual model that is typically referred to when talking or learning about network communications so I will refer to OSI layers.

Header fields at various OSI layers can be used for covert channel data transfer, these can be fields like:

- IP Identification field
- TCP Acknowledgement
- TCP Sequence number
- TCP option fields

TCP is one of the main protocols used on the internet. It is a reliable protocol and provides flow control and ensures that the data reaches the proper destination in the order that it was sent including reordering lost or delivered out of order packets, although this has a tradeoff in the speed of the protocol which is why it tends to be used for data that needs to be reliable. The TCP protocol is on layer 4 of the OSI model.

TCP header structure

The header of a TCP segment contains data requires for connection and data transmission. This header data precedes the payload and is typically 20 bytes in size with an extra optional field that can contain 40 bytes of data.

Transmission Control Protocol (TCP) Header 20-60 bytes

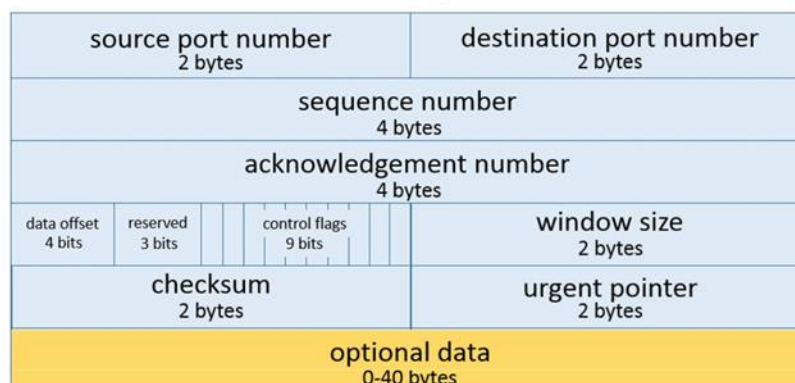


Figure 1 TCP Header layout

Raw data can be injected into these headers in order to transfer data possibly bypassing network security like network intrusion detection systems, this data could be in the form of malicious code for execution or secure files being exfiltrated from the network. The specific field I will largely use is the sequence number field. Sequence number fields are used to keep track of how much data has been transferred and received. This header is also the one that contains the source and destination port numbers, Covert_TCP allows any port to be specified so I will change the ports used frequently in order for them to be used like ephemeral ports and my models will not use ports as an important variable.

IPV4 header

The IPV4 header is another header in which attackers can inject data, IP headers are on layer 3 of the OSI model and are typically between 20 and 60 bytes in length. IP headers contain information needed for routing and delivery. There are several fields that could be used for injecting in the IPV4 header although in my data I will only use the Identification field. The Identification field in the IPV4 header is a 16bit field used to aid in assembling fragments of an IPv4 datagram by having the sender assign an identifying value. This will also be the header Covert_TCP will use to spoof the source IP addresses in order for my data to have a range of source IPs for the attacks.

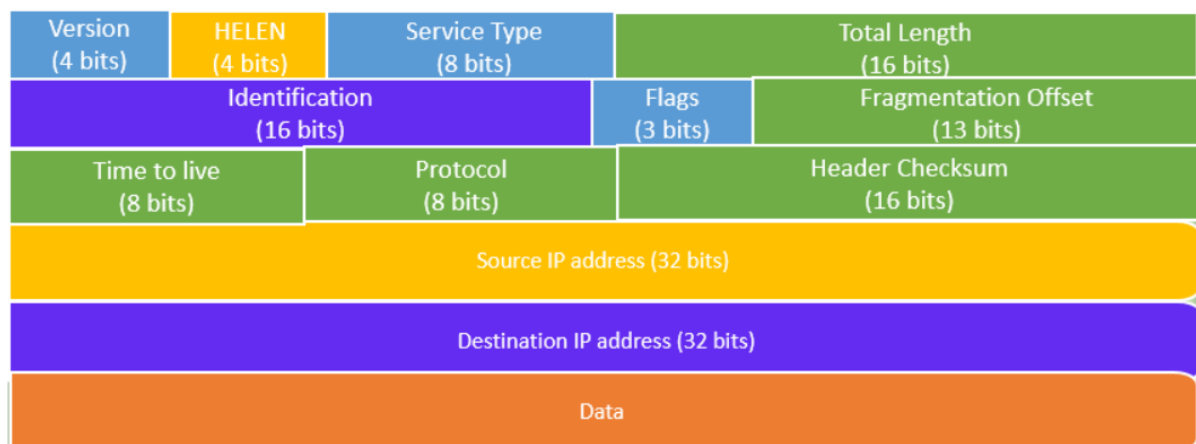


Figure 2 IPV4 header (Guru99, 2022)

2.2 Machine learning libraries

Machine learning has become quite popular in recent times leading to many vast libraries designed for machine learning in multiple languages. My initial research began with finding

some preferred library or libraries to use. There were multiple candidates each with their own pros and cons for use some of which are detailed in the table below:

Library	Pros	Cons
Scikit-learn	Well established library with multiple algorithms built in. Popular and heavily documented	Uses python which I have little to no experience in. Not good with data manipulation, other libraries are better like Pandas and NumPy.
TensorFlow	Extensive documentation. Flexible architecture allows running on CPU, GPU and TPU. Models built on TPUs perform computations faster than CPU and GPUs.	Limited GPU support, only supports NVIDIA GPUs. Benchmarks show slower than competitors.
Rpart	Rpart good for building classification and regression trees. Built in R, which I have experience in.	Limited built-in algorithms.
H2O.ai	Uses JVM, can be implemented with both R and Python Many built-in algorithms Runs in memory cluster Ease of use.	Limited feature engineering, zero feature creation.

Table 1 ML Library comparison

Having explored the documentation for the various libraries, H2O.ai was selected as it had a good selection of algorithms to implement and being about to use either Python or R. Due to having no real experience in Python for this project, I chose to opt for the R version of H2O.ai.

As stated, H2O runs a cluster in memory which allows for customisation for things like memory allocation and number of cores the cluster has.

```
java version "15.0.1" 2020-10-20
Java(TM) SE Runtime Environment (build 15.0.1+9-18)
Java HotSpot(TM) 64-Bit Server VM (build 15.0.1+9-18, mixed mode, sharing)

Starting H2O JVM and connecting: Connection successful!

R is connected to the H2O cluster:
  H2O cluster uptime:      1 seconds 419 milliseconds
  H2O cluster timezone:    Europe/London
  H2O data parsing timezone: UTC
  H2O cluster version:     3.36.0.2
  H2O cluster version age:  2 months and 2 days
  H2O cluster name:        H2O_started_from_R_Craig_ega437
  H2O cluster total nodes: 1
  H2O cluster total memory: 3.98 GB
  H2O cluster total cores: 12
  H2O cluster allowed cores: 4
  H2O cluster healthy:     TRUE
  H2O Connection ip:       localhost
  H2O Connection port:     54321
  H2O Connection proxy:    NA
  H2O Internal Security:   FALSE
  H2O API Extensions:      Amazon S3, Algos, Infogram, AutoML, Core V3, TargetEncoder, Core V4
  R Version:               R version 4.1.2 (2021-11-01)

> |
```

Figure 3 H2O cluster details

The H2O library also has an open source user interface called H2O flow which allows for a blend of command line and graphical user interface, however while I explored this option, it was not used as I made my own user interface using shiny.

2.3 Machine learning algorithms

Within machine learning there are many differing models that can be applied to this project. Within this section I will explore some of the algorithms I plan on implementing to test for my final product.

Random forest

Random forest is a supervised machine learning algorithm that is popular in classification and regression problems. It uses a multitude of decision trees based on different samples when training. Having many decision trees (forests) trained independently help correct for overfitting of singular decision trees and will generally outperform a singular decision. RF constructs a strong learner by using the large number of decision trees to predict outcomes based on a voting system known as aggregation, which in turn produces better precision. According to the Random Forest developer Leo Breiman “Random forests does not overfit. You can run as many trees as you want.” (Breiman and Cutler, n.d.), Although this is debated as some people do find overfitting when using random forest. A well calibrated forest can have a very high level of

precision with little data compared to other machine learning algorithms, this makes random forests a very popular algorithm.

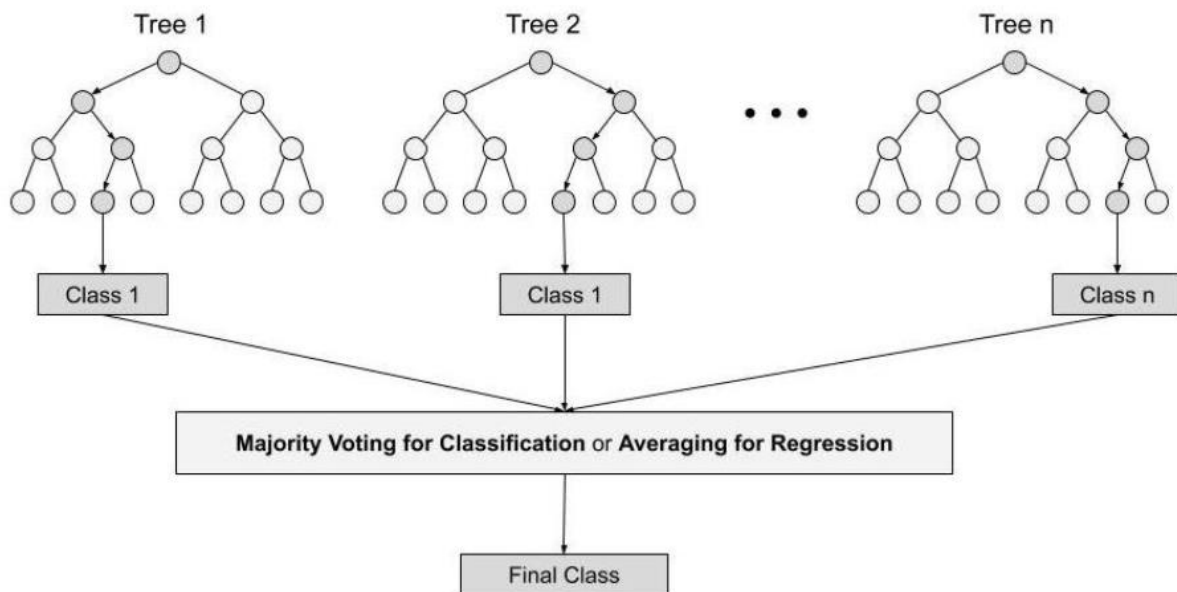


Figure 4 Random forest (Sruethi, 2021)

Neural network

Neural networks also called artificial neural networks or simulated neural networks are inspired by the human brain. Neural networks are comprised of node layers containing an input layer, a number of hidden layers and then an output layer. Each node layer is made up of a set of neurons. Each neuron can be thought of as its own linear regression model which is composed of input data, weights, an activation function and an output which is fed into the next layer as the input. As the neural networks on H2O contain many hidden layers they are technically deep neural networks (DNNs), also known as deep learning.

During training a neural networks weights and thresholds are initially set to random values, as training data is fed through the weights and thresholds are continually adjusted until the output of the neural network is similar to that of the training data.

The H2O library uses feedforward artificial neural networks which are trained with stochastic gradient descent using back-propagation.

Backpropagation is a widely used algorithm for training feed forward neural networks, it computes the gradient of the loss function with respect to the weights of the network. Due to the efficiency of the algorithm, it is possible to use gradient methods for training multilayer networks by updating weights to minimise loss, in H2Os case it uses stochastic gradient descent.

Stochastic gradient descent is an optimisation algorithm commonly used in machine learning to optimize the loss function, at each step the algorithm will calculate the gradient for one observation picked at random rather than the whole dataset.

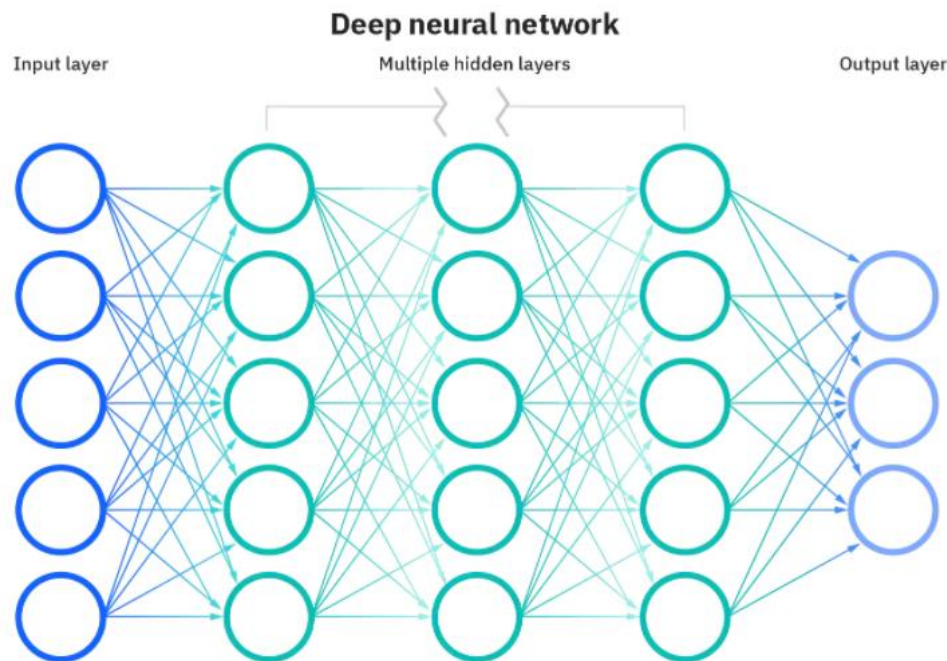


Figure 5 Neural Network (IBM Cloud Education, 2020)

Gradient boosting machines

Gradient boosting trains many models in a gradual and sequential manner which it then uses in an ensemble method to create a strong model from the various weaker models. Each additional weak model added to the overall GBM reduces the mean squared error of the overall model.

GBMs have a choice of distributions for the loss function depending on the response column format. If the distribution is not specified the model will attempt to guess which distribution is best suited however this may not always lead to the best results as it can be incorrect. Bernoulli distribution was specified for my models, Bernoulli is best used when the response column is a 2-class categorical. Some other available distributions are :

Multinomial - the response column must be categorical

Gaussian - the response column must be numeric

Ordinal – the response column must be categorical and with at least 3 levels.

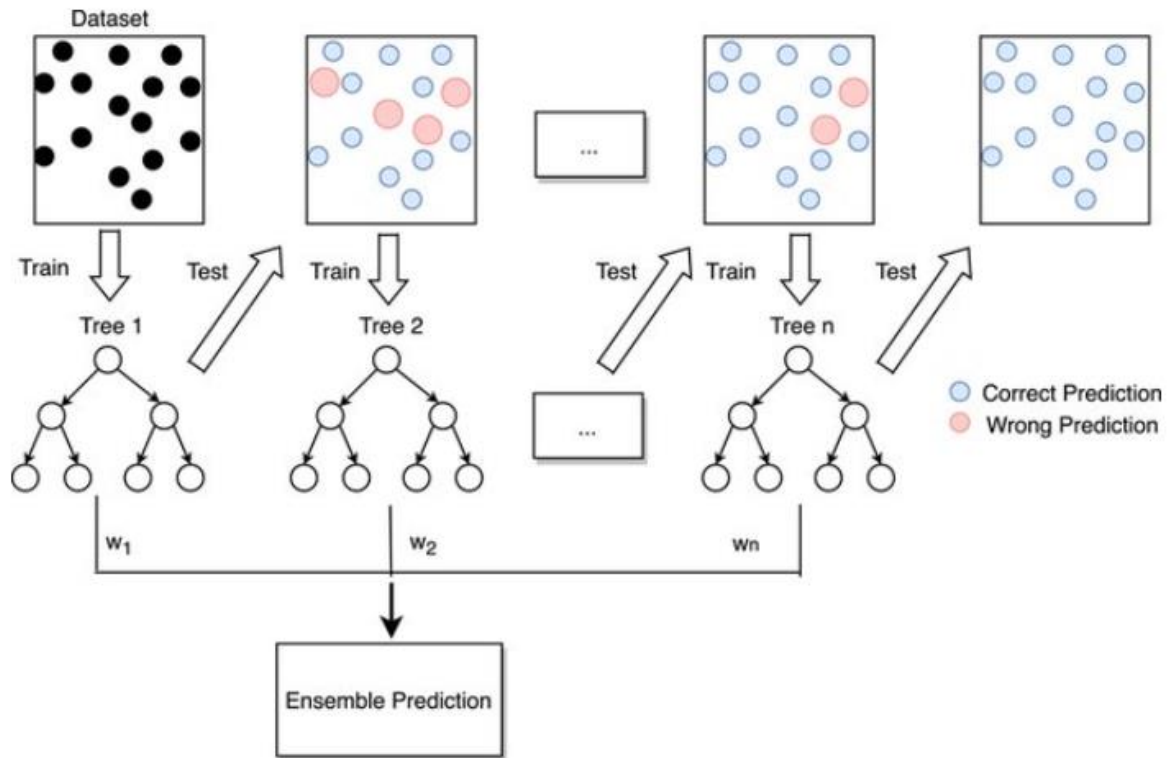


Figure 6 Gradient boosting flowchart (Zhang, 2021)

GBM hyperparameter tuning is key in GBM modelling as they are prone to overfitting. This can be helped by tuning the number of iterations which is called early stopping. Early stopping monitors the models performance on a separate test data set and attempts to select the inflection point where performance on test dataset begins to decrease whilst performance on the training dataset continues to improve as overfitting occurs.

Stacked ensembles

Many of the modern machine learning algorithms are ensemble learners including random forest and gradient boosting machines in which they use a collection of weaker learners to form a single strong learner. Stacked ensembles are different in the way they work as rather than bagging (random forests) or boosting (GBM) they use stacking. Stacking also sometimes known as super learning or super regression, is a class of algorithms that uses a second-level “meta learner” to find the optimal combination of the base learners. Stacking requires the use of cross-validation to form “level-one” data which is the data that the metalearning algorithm is trained on. Due to the way the metalearner is trained the is required to use the same number of folds in the cross validation stage of training the base models.

The Architecture of Stacking:

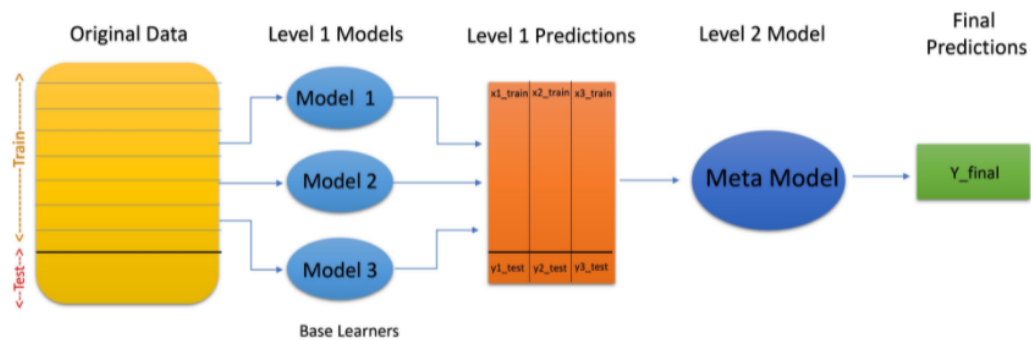


Figure 7 Stacking architecture (Opendenius.org, 2021)

Stacking can have N number of base models also called level 1 models and will utilize the capabilities of both well performing and weakly performing models to make predictions with better performance than any of the base models in the ensemble. Within H2O the metalearner model can be specified from a selection of algorithms including GLM, GBM, DRF, Deep Learning, NaiveBayes or XGBoost. The metalearner algorithm is fully customisable with the ability to specify the hyperparameter values for the algorithm in order to achieve better results.

2.4 Unsupervised packet-based anomaly detection in virtual networks

The ability to detect packets accurately can be used to ensure privacy and security of a network and its data. Particularly with the evolution of cloud environments, digital services and resources have become standard in our everyday life. This paper focused entirely on virtual networks and detecting general anomalies within the virtual network protocols on an ever-changing virtual network comprising of virtual machines, virtual networks and virtual storage, however, I found some key information which I can use to help focus my models on.

Feature set

Feature selection within machine learning helps to narrow down the parameters and aspects of the network traffic and can eliminate complexity by removing redundant or irrelevant data in the dataset.

Within this paper I found some of the features the authors found best to use for detecting anomalies which will give me some insight in which features I should begin to test with when training my models. I will aim to minimise the features used to reduce noise in my model by excluding features that do not provide a significant uplift in my testing accuracy.

Deployed feature set.		
Feature	Layer	Description
frame.len	–	Length of the entire frame
eth.src	2	MAC address of the source
eth.dst	2	MAC address of the destination
eth.type	2	Protocol ID of the upper layer
ip.version	3	Version (4 or 6) of the IP packet
ip.len	3	Length of the IP packet
ip.flags	3	Flags of the IP protocol header
ip.fragment	3	Fragmentation of the packet
ip.dst	3	IP address of the destination
ip.src	3	IP address of the source
ip.proto	3	Protocol ID of the upper layer
ip.tos	3	Type of service
udp.srcport	4	Destination port of the UDP datagram
udp.dstport	4	Source port of the UDP datagram
tcp.srcport	4	Destination port of the TCP datagram
tcp.dstport	4	Source port of the TCP datagram
tcp.flags	4	Flags of the TCP datagram
tcp.urgent_pointer	4	Urgent flag of the TCP datagram

Figure 8 - Feature set used in background paper (Spiekermann and Keller, 2021)

2.5 Other papers

Analysis of TCP/IP header attacks and how to prevent, this paper by Hirushan Sajindra gave me great insight into the TCP protocols and the different types of attacks that are available to malicious actors. Whilst much of the paper features many different types of TCP header attacks like SYN flooding, RST attacks and more, it gave me great insight into the TCP protocol and how the three way handshake and general communication between client and servers worked.

3 Methodology

During this project I aimed to use a loose implementation of CRISP-DM and Agile in order to keep my project on track. I aim to keep to the general phases of CRISP-DM without the heavy documentation that it requires, using this CRISP-DM Agile approach allows me to go through the process several times and adjust when needed in case of setbacks like if I find a dataset isn't best suited or too big or small to efficiently train

The Cross Industry Standard Process for Data Mining (CRISP-DM) was published in 1999 and has become the most common methodology for data mining, analytics and data science projects. CRISP-DM consists of six phases:

1. Business understanding – What does the business need? What is its end goal?

2. Data understanding – What data do we have or need?
3. Data preparation – Is our data clean? Do we need to process it?
4. Modelling – Which modelling techniques should we apply?
5. Evaluation – Which model best meets the objectives?
6. Deployment – How will our product be accessed by users?

The first phase, Business understanding is essentially asking for the objectives of the project which was outlined earlier in this report (see section 1.1).

The next phase - Data Understanding – is about learning what data we have and what it represents. In this project I have collected my data myself using no outside data, this allows me to know exactly what the data consists of, like what formats it comes in and required me to think considerably about how to make the data as unbiased as possible.

Initially using the feature set mentioned earlier from the paper Unsupervised packet-based anomaly detection in virtual networks, I realised that if I included the IP Address source as a main feature that I would end up with a very bias model only targeting the addressed I had used or flagging any address that isn't used often which would provide very inaccurate results in a real-world scenario but perfect results within my tests. In seeking a way to provide as unbiased a dataset as possible I would have to change all key source info like the IP address of the attacker and port numbers between each set of attacks whilst also not making source addresses a key feature in my feature set.

Phase 3 -Data Preparation – is the pre-processing part of the project, this phase is critical to the accuracy of the ending model and is one of the longest steps, columns that have a single value are useless for modelling and should be ignored or removed, “When a predictor contains a single value, we call this a zero-variance predictor because there truly is no variation displayed by the predictor.” (Kuhn and Johnson, 2019).

Phase 4 – Modelling – This phase is the part where I will actually create my models, it consists of testing a range of machine learning algorithms in order evaluate the best to combine into my stacked ensemble for the end product.

Phase 5 – Evaluation – within this phase I will be looking into the base models of my ensemble and my finished ensemble to explore which algorithm performs best and the performance of the overall ensemble compared to its base models.

In real world scenarios there would be a phase 6 – Deployment – however as this is a final year project, I am not deploying an end product to an enterprise.

4 Design and Data collection

4.1 Requirements

The final product should have a user friendly design able to take in data in some form and output the malicious packets whilst not flagging the standard traffic. Due to the flags being malicious or benign (standard traffic), it is essentially a binary classification task therefore classification algorithms would be best suited for the task.

4.2 The Network

One of the most vital tasks is that of the initial data collection. I will need to gather data in order to train and evaluate my models, due to the fact that I plan on gathering my data personally rather than relying on any public data I will need to consider the method in which I set up and perform the task. In order to gain reliable data, I will need multiple machines in which to record traffic from and at least one to perform the attacks to be detected. To create this network, I will use a number of VMs (Virtual Machines) in order to create multiple instances without significant cost. I have chosen Oracles VM VirtualBox to create the VMs on as I have had previous experience with this software and found it to be reliable and easy to use whilst also being free. The operating systems that the VMs run didn't matter in general so I chose Kali Linux as it comes with a vast majority of the software that I was going to use pre-installed as default, so it is best suited for the task.

Producing reliable data that isn't bias is key in my implementation as models are only as good as the dataset they learn from. In my initial planning I considered some of the ways my data could create a bias, to help prevent bias from having a single machine performing the attacks I will be required to change the IP and ports that I use during the attacks so that my models don't just flag everything from a single IP address. I will also need to have standard browsing traffic in order to simulate a normal working network in order to create some noise on the network by making the victim machines browse the internet.

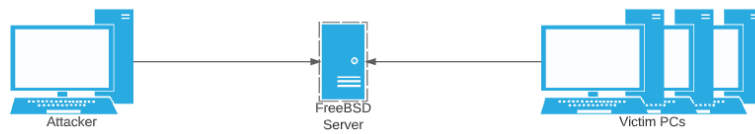


Figure 9 – Basic network layout

4.3 Data capture

Wireshark

I will use Wireshark to capture the network traffic. Wireshark is one of the worlds most widely used network analysers and will allow me to capture the packets being sent through the network. I considered using other data capture software like Tcpdump which all work very similarly, however, Wireshark having a graphical front end rather than a command line interface was a personal preference along with having experience in using Wireshark previously.

No.	Time	Source	Destination	Protocol
31	5.168068504	firefox.settings.services.mozilla.com	10.0.2.5	TCP
32	5.168074527	10.0.2.5	firefox.setti...	TCP
33	5.172034478	firefox.settings.services.mozilla.com	10.0.2.5	TLSv1.3
34	5.172039671	10.0.2.5	firefox.setti...	TCP
35	5.173270597	10.0.2.5	firefox.setti...	TLSv1.3
36	5.173418801	10.0.2.5	firefox.setti...	TLSv1.3
37	5.173517230	firefox.settings.services.mozilla.com	10.0.2.5	TCP
38	5.190207454	firefox.settings.services.mozilla.com	10.0.2.5	TLSv1.3
39	5.190217519	10.0.2.5	firefox.setti...	TCP
40	5.196626677	10.0.2.5	firefox.setti...	TLSv1.3
41	5.214407485	firefox.settings.services.mozilla.com	10.0.2.5	TLSv1.3
42	5.214427385	10.0.2.5	firefox.setti...	TCP
43	6.004741398	56.54.3.5	10.0.2.5	TCP
44	6.004758116	10.0.2.5	56.54.3.5	TCP

▶ Frame 43: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
 ▶ Ethernet II, Src: PcsCompu_50:4c:14 (08:00:27:50:4c:14), Dst: PcsCompu_d2:56:77 (08:00:27:d2:56:77)
 ▶ Internet Protocol Version 4, Src: 56.54.3.5 (56.54.3.5), Dst: 10.0.2.5 (10.0.2.5)
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 40
 Identification: 0x7300 (29440)
 ▶ Flags: 0x00
 Fragment Offset: 0
 Time to Live: 64
 Protocol: TCP (6)
 Header Checksum: 0xc090 [validation disabled]
 [Header checksum status: Unverified]

0000	08 00 27 d2 56 77 08 00	27 50 4c 14 08 00 45 00	..'.Vw.. 'PL...E.
0010	00 28 73 00 00 00 40 06	c0 90 38 36 03 05 0a 00	..(S...@..86....
0020	02 05 1b 39 22 60 55 0b	00 00 00 00 00 00 50 02	..9"U.....P.
0030	02 00 d3 fe 00 00 00 00	00 00 00 00 00 00 00 00

Identification (ip.id), 2 bytes
 Packets: 5477 · Displayed: 5477 (100.0%) · Profile: Default

Figure 10 Wireshark example

Wireshark allows the user to see a wide range of information about the network traffic in real time. Wireshark's main downside is the format of the data it outputs. Wireshark saves the recorded streams of data into PCAP files which are harder to work with, so in order to get data that I can easily work with I will need to get the data into CSV format which can be done in several ways, I opted to use a PCAP to CSV file converter due to the size of the files. Wireshark has an option to export the captured traffic directly into CSV format, however when this is done it does not export the header information, instead it only exports the table that is shown on the user interface and the internal packet structure and information is lost, thus I had to find another way to get my dataset into a format I could work with, exporting to JSON file types was the only exporting method on Wireshark that kept the header information intact however I still needed a way to convert JSON files into CSV. My initial attempt to convert my exported JSON dataset to CSV using an R library called JSONlite failed due to the size of the file, the performance of the program ground to a halt, thus I was required to use an online converter in order to get my datasets into the CSV format that I could work with.

Covert_Tcp

Covert_Tcp is a network tool that allows data to be injected into a number of fields of the headers of packets. This tool has a number of great features that will allow me to vary my attack data to ensure minimal bias.

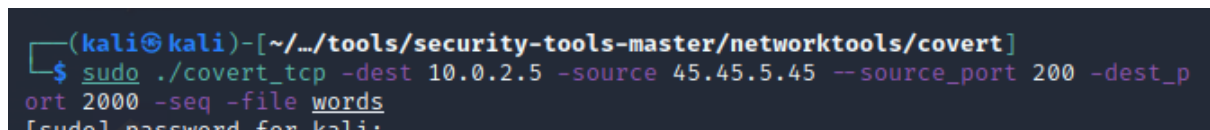
A terminal window with a dark background and light-colored text. The prompt is `(kali@kali)-[~/tools/security-tools-master/networktools/covert]`. The command entered is `$ sudo ./covert_tcp -dest 10.0.2.5 -source 45.45.5.45 --source_port 200 -dest_port 2000 -seq -file words`. The output shows `[sudo] password for kali:` followed by a blank line.

Figure 11 - Covert_TCP command line example

One way in which to minimise bias in my data is by spoofing IP addresses which is a technique in which a program allows for a machine on a network to send data that seems to come from a number of different machines with different addresses, this is done by modifying the source address field in the IPV4 packet header. Covert_tcp allows for this with a simple `-source` option in your command which will edit the source address to any IP address you specify thus making traffic appear to come from various differing sources.

In order to further reduce bias, I will perform attacks using several different header fields to inject data into in the hopes of a wider detection range so attackers can't just select a different header field to bypass my model. In section 2.1 the different fields of network protocol headers I would be using were explored, Covert_TCP allows injecting into multiple different fields by

varying the command in my data set I have varied the fields including the TCP sequence number and the IPV4 Identification field for my data injection.

5 Machine learning Implementation

5.1 Training the models

When training the models, I would use a confusion matrix to initially understand how well the models were performing once trained with the best base algorithms being selected for my ensemble.

N-fold is a cross-validation method h2o uses to validate a model internally, when specifying `nfolds` within h2o the algorithm will create `nfolds+1` models, within my project I used `nfolds=5` throughout so there are 5 cross-validation models and then a main model returned at the end. The first 5 models get built on just 80% of the training data with 20% held back, then for the final model 100% of the training data is used. My final model was planned to be a stacked ensemble

When training stacked ensembles, you are required to use the same number of `nfolds` through all of the base models and keep the cross-validation predictions by setting `keep_cross_validation_predictions = TRUE`. Stacked ensembles within h2o require all base models to be trained on the same training set, however, it allows for each base model to use different feature sets across the models.

Random Forest

In section 2.3 I explained how random forest works. The h2o library allows random forests to have a user specified number of trees, my model will have 50 trees within it. Larger forests require higher computational costs and while can be good, more trees does not always equal more accuracy.

```
58
59 # Train & Cross-validate a RF
60 my_rf <- h2o.randomForest(x = x,
61                           y = y,
62                           training_frame = train,
63                           ntrees = 50,
64                           nfolds = nfolds,
65                           keep_cross_validation_predictions = TRUE,
66                           seed = 1)
67
68
```

Figure 12 Initial RF training

Once my model was trained I initially used a confusion matrix to quickly summarise if my model was actually performing well. A confusion matrix, also known as an error matrix, indicates how successful a classification model is by summarising the prediction results as follows:

TP: true positive, correct prediction of a positive class.

TN: true negative, correct prediction of a negative class.

FP: False positive, incorrect prediction claiming positive when negative.

FN: False negative, incorrect prediction claiming negative when positive.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Figure 13 confusion matrix example

The above example is laid out the same way H2O outputs its confusion matrixes with the actual label on the vertical and the predicted labels on the horizontal. This was purely a primary way to detect good accuracy models, accuracy isn't the best evaluation technique to judge models as a whole but can be used initially to help select good candidates. Unbalanced classification problems like this one where one class is in a great amount, in this case benign traffic, and the other class, malicious traffic, has very little data comparatively with only 2.1% of the packets being malicious in the test set.

```
> h2o.confusionMatrix(my_rf, packets_h2o)
Confusion Matrix (vertical: actual; across: predicted)
      0    1  Error  Rate
0     5094   0 0.000000  =0/5094
1         0 112 0.000000  =0/112
Totals 5094 112 0.000000  =0/5206
> |
```

Figure 14 Random forest confusion matrix

The above matrix shows that my random forest model correctly predicted 100% of the test data, whilst this suggests amazing results and a perfect model it could also be caused by overfitting and cause issues on other datasets. In the main evaluation stage, we shall explore

this further by testing on secondary datasets I created and explore any mistakes made by my model.

```
> my_rf
Model Details:
=====

H2OBinomialModel: drf
Model ID:   DRF_model_R_1648560776282_261
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth
1             50             50             6907             3             7             4.82000
  min_leaves max_leaves mean_leaves
1           4          13          6.54000

H2OBinomialMetrics: drf
** Reported on training data. **
** Metrics reported on Out-Of-Bag training samples **

MSE: 3.039045e-05
RMSE: 0.005512753
LogLoss: 0.0002446348
Mean Per-Class Error: 0
AUC: 1
AUCPR: 1
Gini: 1
R^2: 0.9984823

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
      0  1  Error  Rate
0    3594  0 0.000000  =0/3594
1      0  75 0.000000  =0/75
Totals 3594 75 0.000000  =0/3669
```

Figure 15 Random forest model details

Neural network

Training a neural network in H2O is fairly straight forward, as stated earlier the nfolds validation is required to be the same in all models or they can't be integrated into a stacked ensemble at a later stage.

```
69 #NN in h2o
70 my_nn <- h2o.deeplearning(x = x,
71                           y = y,
72                           training_frame = train,
73                           nfolds = 5,
74                           keep_cross_validation_predictions = TRUE,
75                           seed=1
76 )
--
```

Figure 16 Neural network.

There are quite a few hidden options set by the H2O library that were left as default due to them being adequate for my data. As described earlier in section 2.3 neural networks learn through supervised learning building knowledge from data sets where the answer is provided in advance and then fine tuning themselves to find the correct answers increasing their accuracy of their predictions. This approach means that neural networks go over the same dataset multiple times, this is called an epoch. The epoch number is a critical hyperparameter for the

algorithm, my model has an epoch number of 10, so it will go through my entire dataset 10 times updating the weights each iteration. Another feature that was left default but is important to the model is the hidden layers, in H2O this defaults to `hidden= 200, 200` where the first number specifies the number of neurons per layer and the second number specifies the number of layers.

```
> h2o.confusionMatrix(my_nn, packets_h2o)
Confusion Matrix (vertical: actual; across: predicted)
      0      1      Error      Rate
0      5093      1 0.000196 =1/5094
1         1     111 0.008929 =1/112
Totals 5094    112 0.000384 =2/5206
> |
```

Figure 17 Neural network confusion matrix

Once my model is trained, I check the confusion matrix like with my other models, the above matrix shows that there were two packets predicted incorrectly from the testing dataset. This is not as accurate as the random forest but also could be mean the neural network is not overfitted as the random forest may be.

Variable importance

Variable importance isn't a metric but is helpful to understand how each model is looking at the dataset and which variables it prioritises in order to make a prediction. These variables are chosen during training so will not change between datasets. Each model gives different weighting to different variables which helps the final ensemble as there is less chance of a bias being built from each model using the same variables.

Random forest model gives the most weighting to the `time_epoch` variable with its weighting making up 39.9% of the decision.

```
> h2o.varimp(my_rf)
Variable Importances:
      variable relative_importance scaled_importance percentage
1      X_source.layers.frame.frame.time_epoch      1191.698486      1.000000      0.399805
2 X_source.layers.frame.frame.time_delta_displayed      480.039307      0.402819      0.161049
3      X_source.layers.frame.frame.time_delta      431.224091      0.361857      0.144672
4      X_source.layers.frame.frame.time_relative      230.259949      0.193220      0.077250
5      X_source.layers.eth.eth.src_tree.eth.lg      169.799591      0.142485      0.056966
6      X_source.layers.frame.frame.number      117.932785      0.098962      0.039565
7      X_source.layers.eth.eth.src_tree.eth.src.oui      85.565887      0.071802      0.028707
8      X_source.layers.eth.eth.src_tree.eth.addr.oui      82.649765      0.069355      0.027728
9      X_source.layers.eth.eth.src_tree.eth.src.lg      55.469795      0.046547      0.018610
10     X_source.layers.eth.eth.dst_tree.eth.dst.lg      30.742102      0.025797      0.010314
11     X_source.layers.eth.eth.dst_tree.eth.lg      29.631523      0.024865      0.009941
12     X_source.layers.eth.eth.dst_tree.eth.dst.oui      25.816940      0.021664      0.008661
13     X_source.layers.eth.eth.dst_tree.eth.addr.oui      22.196083      0.018626      0.007447
14           X_source.layers.eth.eth.type      13.044382      0.010946      0.004376
15     X_source.layers.frame.frame.cap_len      10.867319      0.009119      0.003646
16           X_source.layers.frame.frame.len       3.764052      0.003159      0.001263
```

Figure 18 random forest variable importance

Each of the models have different variables they deem the most valuable. I won't go into each of the models variable importance as it is not a key metric however it is important to note that

they all use different variables as their most important in order for the models to predict classes using different information.

GBM

Gradient Boosting Machines in H2O are defined through the `h2o.gbm()` method. H2O's GBM sequentially builds regression trees on all the features of the dataset in a fully distributed way with each tree being built in parallel, this allows for quick models to be made as many of the parameters are predefined.

```
46 # Train & Cross-validate a GBM
47 my_gbm <- h2o.gbm(x = x,
48                   y = y,
49                   training_frame = train,
50                   distribution = "bernoulli",
51                   ntrees = 20,
52                   max_depth = 3,
53                   min_rows = 2,
54                   learn_rate = 0.2,
55                   nfolds = nfolds,
56                   keep_cross_validation_predictions = TRUE,
57                   seed = 1)
58
```

Figure 19 GBM defining

When training my GBM model I initially had very poor results when looking at the initial confusion matrix, I believe this was due to poor configuration of my initial model as I opted to go for the default options for many of the configurations, this led me to retrain the model with more specified parameters to attempt to get better accuracy as shown in the snippet above. The changes from default that I ended up needing to make for better results were an increased learning rate which defaults at 0.1 and was increased to 0.2, the max depth was reduced to `max_depth = 3` from a default of 5, having an increased depth leads to a more complex model and can lead to overfitting. GBM in H2O have a number of distributions that can be specified, due to my data having 2 class categories, malicious or benign GBM uses bernoulli distribution.

```
> h2o.confusionMatrix(modelgbm, packets_h2o)
Confusion Matrix (vertical: actual; across: predicted)
      0    1   Error   Rate
0     5091  3 0.000589  =3/5094
1       25 87 0.223214  =25/112
Totals 5116 90 0.005378  =28/5206
> |
```

Figure 20 initial GBM confusion matrix

The confusion matrix above was the model trained with majority of the parameters left as default which yielded many false negatives when compared to the final GBM model. As discussed earlier GBM does have a tendency to overfit however having decreased the max depth should be enough to help prevent overfitting, despite the increased accuracy.

```
> h2o.confusionMatrix(my_gbm, packets_h2o)
Confusion Matrix (vertical: actual; across: predicted)
      0    1   Error   Rate
0     5093   1 0.000196  =1/5094
1         0 112 0.000000  =0/112
Totals 5093 113 0.000192  =1/5206
```

Figure 21 configured GBM

Once I had a better understanding of the GBM parameters and managed to get a more accurate GBM model on my testing dataset as seen above, my model details ended up as seen in the figure below.

```
> my_gbm
Model Details:
=====

H2OBinomialModel: gbm
Model ID:   GBM_model_R_1648665477546_976
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth
1              20                20                2687           3           3      3.00000
  min_leaves max_leaves mean_leaves
1           4           8       6.05000

H2OBinomialMetrics: gbm
** Reported on training data. **

MSE:  4.14841e-05
RMSE: 0.006440815
LogLoss: 0.000801931
Mean Per-Class Error: 0
AUC: 1
AUCPR: 1
Gini: 1
R^2: 0.9979282

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
      0    1   Error   Rate
0     3594   0 0.000000  =0/3594
1         0 75 0.000000  =0/75
Totals 3594 75 0.000000  =0/3669
```

Figure 22 GBM model details

Stacked ensemble

My stacked ensemble is a combination of multiple models explored earlier.

```
106
107 #rf, nn , gbm
108 # Train a stacked ensemble using the GBM and RF above
109 ensemble2 <- h2o.stackedEnsemble(x = x,
110                                y = y,
111                                training_frame = train,
112                                base_models = list(my_gbm, my_rf, my_nn))
113
```

Figure 23 Stacked ensemble

In the figure below you can see the algorithms making up the stacked ensemble. The confusion matrix listed is that of the initial training set rather than the test set as it is listing the models details. The meta learner used is a GBM model.

```
> ensemble2
Model Details:
=====

H2OBinomialModel: stackedensemble
Model ID: StackedEnsemble_model_R_1648665477546_1215
Number of Base Models: 3

Base Models (count by algorithm type):

deeplearning      drf      gbm
               1      1      1

Metalearner:

Metalearner algorithm: glm
Metalearner hyperparameters:

H2OBinomialMetrics: stackedensemble
** Reported on training data. **

MSE: 2.622926e-07
RMSE: 0.0005121451
LogLoss: 0.0002119541
Mean Per-Class Error: 0
AUC: 1
AUCPR: 1
Gini: 1

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
      0  1  Error  Rate
0    3594  0 0.000000  =0/3594
1      0  75 0.000000  =0/75
Totals 3594 75 0.000000  =0/3669
```

Figure 24 Stacked Ensemble details

5.2 User interface

In order for my final product to be user friendly a graphical user interface needs to be included. The R library shiny was selected for its ease of use for making UI components.

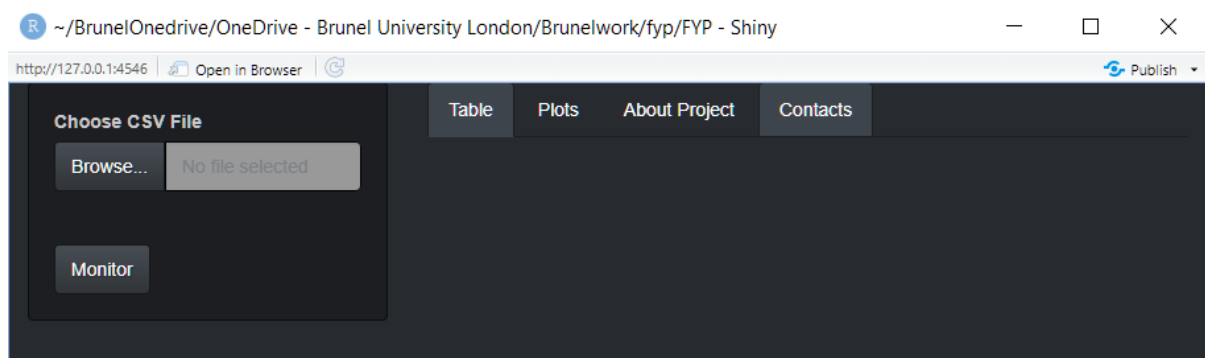


Figure 25 Basic user interface

I made the interface minimalistic so that it was intuitive on its use. All the user needs to do is to choose the dataset to be analyzed and the output the model predicts to be malicious will be displayed on a table for the user to inspect with details of the time of the packet source address and destination address so users can see potentially compromised machines on the network.

X_index	X_type	X_source.layers.frame.frame.time	X_source.layers.ip.ip.addr	X_source.layers.ip.ip.dst
packets-2022-02-16	doc	Feb 16, 2022 16:13:54.571912621 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:13:55.577046102 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:13:56.579826957 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:13:57.586461421 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:13:58.604425369 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:13:59.604804485 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:14:00.604979297 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:14:01.615371250 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:14:02.615535236 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:14:03.617647996 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:14:04.643578833 EST	192.0.0.1	10.0.2.5
packets-2022-02-16	doc	Feb 16, 2022 16:14:05.660934249 EST	192.0.0.1	10.0.2.5

Figure 26 User interface after packet analysis

6 Testing and Evaluation

6.1 Testing methodology

Due to collecting my own data set for initial training I have the ability to create entirely new datasets to perform my evaluation on. At several stages during this project, I created datasets that my models were not trained or tested on, I did these separately and at different times on different days to ensure a various mix of data as time is a variable within the datasets that may have been measured. Throughout this section I will compare both the base models and the ensemble model across several datasets to evaluate their ability to predict varied datasets and see if there is a noticeable uplift from the base model to the ensemble.

As this is a classification task there are several metrics that are classed as important and should be explored, however, some metrics are inappropriate for my evaluation due to the imbalanced data. For data that has very few of one class, in this case very few malicious packets compared to benign packets, then a model that always picks benign will have a very high accuracy in that metric. For these use cases, it is best to select a metric that does not consider true negatives or that considers the relative size of the true negatives. Along with some of the standard metrics, I will also explore metrics that are specifically good for imbalanced data like Matthews correlation coefficient (MCC).

Accuracy, Precision and Recall

Accuracy, precision, and recall are typically three of the standard metrics used to evaluate models. This is just a quick summary of these metrics as they are not truly suitable for my evaluation however metrics that I do use, like F1, uses precision and recall within it, and further down I will explore why I do not use these metrics using Accuracy as an example, so a brief understanding is required.

In binary classification Accuracy is the number of correct predictions made as a ratio of all predictions made. The Accuracy equation is:

$$\text{Accuracy} = \left(\frac{\text{Number of correctly predicted}}{\text{Number of observations}} \right)$$

Precision is the number of true positives divided by all positive samples both true and false, Precision equation:

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

Recall is the ratio of correctly predicted positive observations compared to the total of that class

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

F1 score

F1 score is a method to solve the class imbalance problems that some other metrics like the above have. F1 takes into account not only the number of errors that models make, but also the type of errors that are made. F1 score is based off both the precision and recall scores discussed previously. The F1 score is defined as the “harmonic mean of precision and recall” (Korstanje, 2021)

Matthews correlation coefficient

Despite being a crucial issue in machine learning there is no widespread consensus on a single chosen measure yet for overall evaluation. Accuracy, precision, recall are amongst the most popular however these statistical measures can show overoptimistic results especially on imbalanced datasets.

The MCC is a more reliable statistical method that will only produce a high score if the models predictions obtained good results in all of the categories proportionally, both to the size of the positive elements and the size of the negative elements within the dataset.

AUCPR

Area under the precision-recall curve is a metric used to evaluate how well a binary classification model is able to distinguish between precision recall pairs. The precision recall curve does not care about true negatives so this metric can be suitable for an imbalanced dataset in which there is significantly less true positives than true negatives.

The datasets

The datasets are very similar with variations largely being in the number of packets, I made varied datasets in order to ensure that my models weren't influenced by some unknown happening at the time of the initial training dataset. By creating several datasets over a number

of weeks it ensures my results are reproduceable and that my models can predict attacks using this method of header injection, all datasets consist of a mix of packets with data injected into either IPV4 ID field or data injected into TCP sequence fields.

Dataset 1- moreatt - contains 5443 total packets

Dataset 2 – presdemo - contains 5124 total packets

Dataset 3 – test – contains 876 total packets.

6.2 Tests

Dataset 1

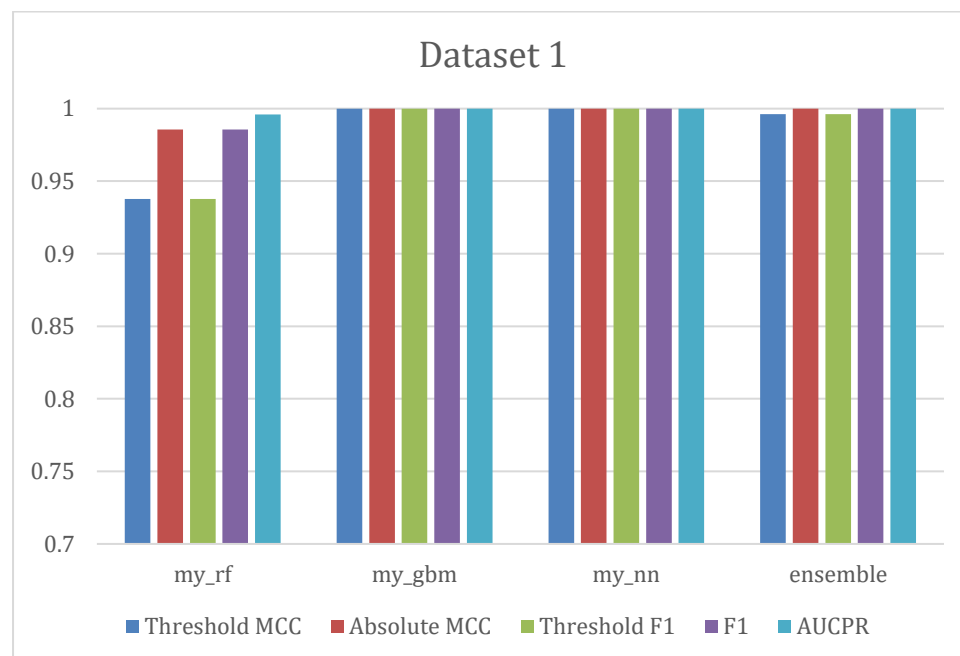


Figure 27 Dataset 1 results

When testing with my dataset 1, two of my base algorithms performed perfectly, however, my random forest was the weak algorithm in this test data as it flagged two false positives as you can see in the confusion matrix for the random forest below.

```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
      0   1   Error   Rate
0     5371  2 0.000372  =2/5373
1         0 69 0.000000  =0/69
Totals 5371 71 0.000368  =2/5442
```

Figure 28 random forest dataset1 CM

Despite being the weaker of the algorithms it is still a strong model and the ensemble model still scored excellently due to the other algorithms performing exceptionally well.

Dataset 2

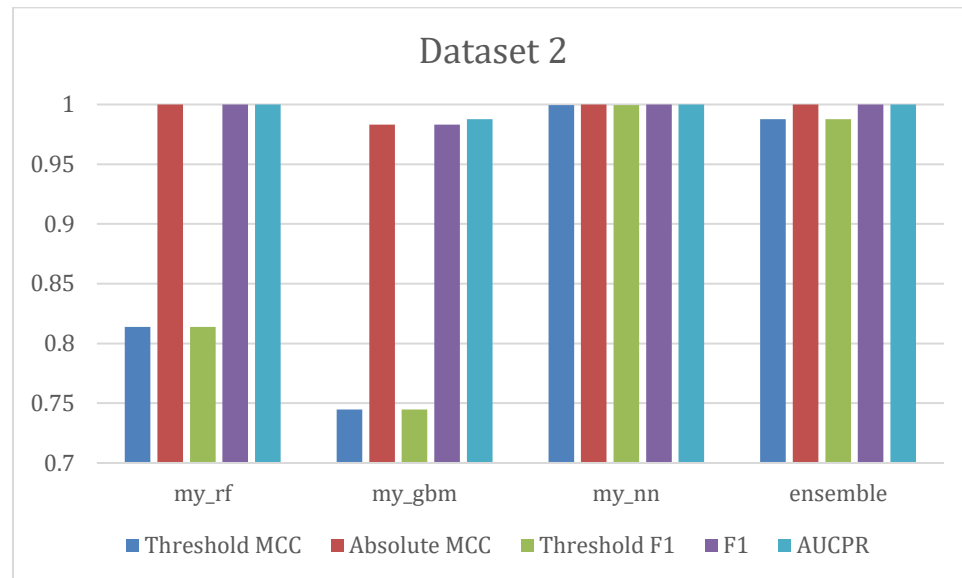


Figure 29 Dataset 2 results

Testing dataset 2 found a reversal of best performing algorithms with random forest actually coming second best and the GBM model scoring worst of the models. Note that due to this dataset having a smaller number of malicious packets overall the random forest from dataset 1 and the GBM in dataset 2 scored similarly in the MCC and F1 scores, roughly 0.985 despite the random forest flagging two false positives and the GBM only flagging one as seen below:

```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
      0   1   Error   Rate
0     5093  1 0.000196 =1/5094
1         0  29 0.000000 =0/29
Totals 5093 30 0.000195 =1/5123
```

Figure 30 GBM dataset 2 CM

Dataset 3

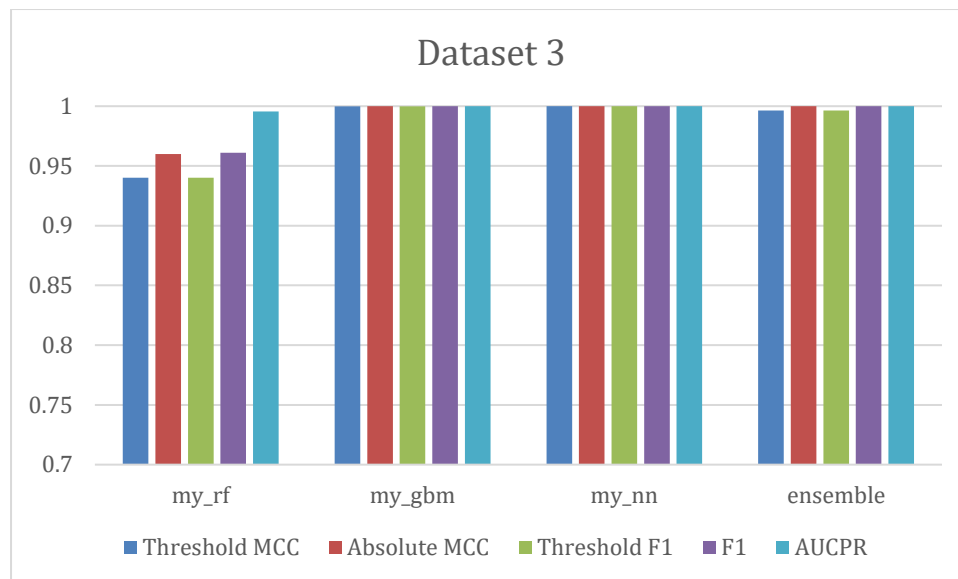


Figure 31 Dataset3 results

Dataset 3 saw a return of the random forest being the weaker of the algorithms with it actually flagging up 3 false positives and all the other algorithms performing excellently, dataset 3 is by far the smallest of the datasets, however it also has one of the higher concentrations of malicious packets with 37 of the 875 packets being malicious.

```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
      0  1  Error  Rate
0      835  3 0.003580  =3/838
1         0 37 0.000000  =0/37
Totals 835 40 0.003429  =3/875
```

Figure 32 Random forest dataset3 CM

Within dataset 3 we find that random forest flagged 3 false positives. Given this higher number of malicious packets despite the smaller size lets us demonstrate how important the selection of evaluation metrics is to the overall evaluation stage. If we compare an evaluation metric that is not suitable for our dataset, for instance the random forests accuracy between dataset 2 and dataset 3

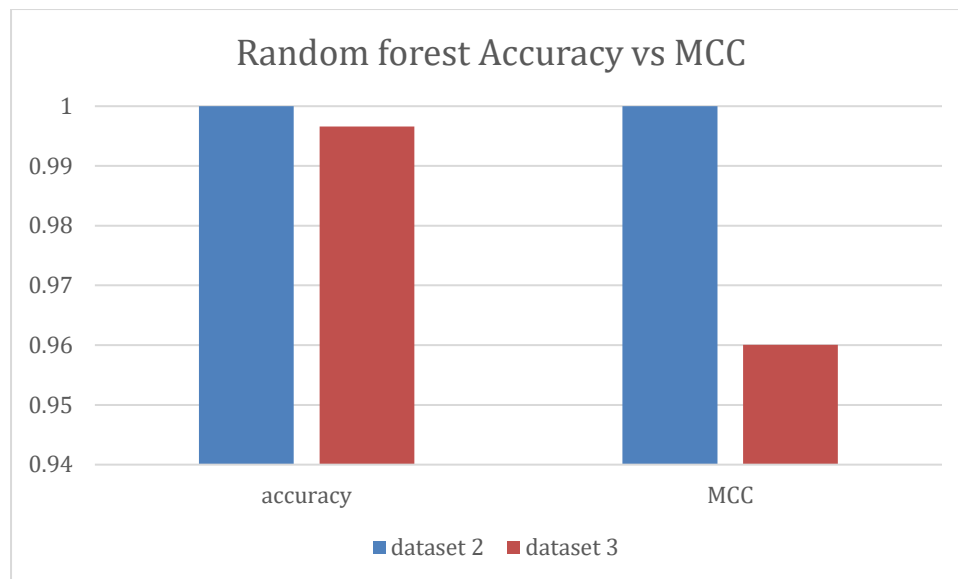


Figure 33 Metric comparison

On the graph, with the maximum being a 1 and minimum being 0, each 0.01 on the graph is one percent. There is a 0.35% change in the accuracy of the random forest when compared between the two datasets despite the algorithm flagging none incorrectly in dataset 2 and flagging 3 packets incorrectly in dataset 3. When compared to the MCC metric you can see a drop of 4% in the scoring of the random forest between datasets 2 and 3. This emphasises the importance of understanding your data and selecting the correct metrics to apply rather than going with standard metrics, and why I did not use the standard accuracy, precision and recall that is common for machine learning evaluation.

7 Conclusions

Having evaluated the models in on various data in Section 6, I found that my ensemble model worked well to counteract the weakness of any one particular model, like my random forest on dataset 2 for example.

One model that worked well throughout the entirety was the neural network, which I have been considering if it is due to having worked with neural networks in the past so having more experience in tuning them or if a neural network was just the best single model for the task at hand. My final product works well to detect the covert attacks within my datasets, however due to having such a limited dataset I am only confident in its ability to detect attacks made by the Covert_TCP tool, due to time and partially knowledge constraints I could not create more datasets with other tools nor could I risk using outside due to how I applied for the ethics approval which limits my ability to test on a wider range of data.

Section 2 detailed my background research into the network set up, how to perform the attacks and my learning about other works in the field, meeting my objective 1.

Within section 4 I detailed how I built the network and created the data used throughout this project, thus meeting my 2 and 3 objectives.

Section 5 and 6 gives insight into my training and testing of models used to detect the covert channel attacks, there are also a few code snips from failed models within appendix C that were not implemented into my final product due to failings in their performance. This completes my objective 4 and 5

In conclusion, this model has been proven to successfully detect these covert channel attacks accurately within my datasets.

7.1 Future Work

Time was a considerable constraint for the project, by the time I had evaluated the datasets I had and wanted to expand I didn't have time to source more data and possibly need to change my Breo submission.

There is also the issue of the complicated process I had to change my dataset into a format to be used, due to files being large by the time they are finished recording any built in library in R or Java struggled with the size of the variable, therefore a pipe could potentially be defined and used in place of saving recorded packets in large files, real time machine learning is a fascinating concept that I looked into for this project however once again time constraints heavily limited the ability to implement a pipe and convert my data into a useful format. I found this project to

be very useful in learning about multiple fields, it had networking, AI and cybersecurity and I believe I will continue developing it in the future as I learn more.

References

Kovacs, E., 2021. *Nearly 100 TCP/IP Stack Vulnerabilities Found During 18-Month Research Project* | *SecurityWeek.Com*. [online] Securityweek.com. Available at:

<<https://www.securityweek.com/nearly-100-tcpip-stack-vulnerabilities-found-during-18-month-research-project>> [Accessed 4 February 2022].

Kuhn, M. and Johnson, K., 2019. Feature engineering and selection.

Sruethi, E., 2021. [image] Available at:

<<https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>> [Accessed 13 March 2022].

IBM Cloud Education, 2020. [image] Available at: <<https://www.ibm.com/uk-en/cloud/learn/neural-networks>> [Accessed 8 March 2022].

Zhang, T., 2021. Gradient boosting machine flowchart. [image] Available at:

<https://www.researchgate.net/publication/351542039_Improving_Convection_Trigger_Functions_in_Deep_Convective_Parameterization_Schemes_Using_Machine_Learning> [Accessed 30 March 2022].

Guru99, 2022. [image] Available at: <<https://www.guru99.com/ip-header.html>> [Accessed 21 March 2022].

Opengenius.org, 2021. [image] Available at: <<https://iq.opengenus.org/stacking-in-machine-learning/>>.

Breiman, L. and Cutler, A., n.d. *Random forests - classification description*. [online]

Stat.berkeley.edu. Available at:

<https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm> [Accessed 22 March 2022].

Korstanje, J., 2021. *The F1 score*. [online] Medium. Available at:

<<https://towardsdatascience.com/the-f1-score-bec2bbc38aa6>> [Accessed 29 March 2022].

Appendix A Personal Reflection

A.1 Reflection on Project

While my project on paper was a success, I find myself wondering how well it would work on outside data. I believe my model works well, as seen in the testing section, however, if I could redo my project I would have looked into using multiple tools earlier in the lifecycle as by the time I had tested all of my datasets it was too late in the lifecycle to discover and learn about another tool or to risk needing to change my Breo application in order to receive authorization for using outside data and having to source that data. Even if I had found data easily, I would have had to learn about it like the ratio of one class to another and types of attacks used in order to evaluate it properly. I personally believe there wouldn't be enough time to do both and would need to stick to either making the data myself and knowing its general composition or using outside data.

As time was so heavily limited, I found myself needing to reduce scope of what I would have like to have implemented vs what I would need for a minimum viable product, like the data pipe with real time machine learning. Real time machine learning is a fascinating subject I found during my background research and would have liked to have implemented it here, perhaps if I had used outside data, I would have had time to proceed with the data pipe and real-time machine learning.

A.2 Personal Reflection

Final year projects are nearly always a learn as you do style of work that you look back on and wish you had done some things differently.

Overall, I have learned a great deal about most of my final year modules as this project integrated all of them well. Building the network and performing the attacks gave me great insight into the network computing aspect of computer science, building the models and investigating their performance gave me amazing technical and practical knowledge on machine learning, the different libraries and languages used and vast amounts of algorithms and metrics used for different areas like classification or regression, and learning about header injection attacks and some of the tools used for them gave me theoretical knowledge on the various types of attacks that can be performed, more than just injection, there is SYN flooding, TCP RST attacks and TCP session hijacking just to name a few, all of this contributed to my Cybersecurity area of knowledge. In the end it was an amazing project to work on and I have learned a significant amount through my time producing it.

Appendix B Ethics Documentation

B.1 Ethics Confirmation



College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London
Kingston Lane
Uxbridge
UB8 3PH
United Kingdom
www.brunel.ac.uk

8 December 2021

LETTER OF CONFIRMATION

Applicant: Mr Craig Saville
Project Title: FYP: Detecting covert channel attacks
Reference: 34088-NER-Dec/2021- 35570-1

Dear Mr Craig Saville

The Research Ethics Committee has considered the above application recently submitted by you.

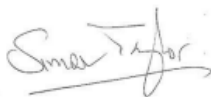
The Chair, acting under delegated authority has confirmed that, according to the information provided in your application, your project does not require ethical review.

Please note that:

- You are not permitted to conduct research involving human participants, their tissue and/or their data. If you wish to conduct such research, you must contact the Research Ethics Committee to seek approval prior to engaging with any participants or working with data for which you do not have approval.
- The Research Ethics Committee reserves the right to sample and review documentation relevant to the study.
- If during the course of the study, you would like to carry out research activities that concern a human participant, their tissue and/or their data, you must inform the Committee by submitting an appropriate Research Ethics Application. Research activity includes the recruitment of participants, undertaking consent procedures and collection of data. Breach of this requirement constitutes research misconduct and is a disciplinary offence.

Good luck with your research!

Kind regards,



Professor Simon Taylor

Chair of the College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London

Appendix C Other Appendices

More relevant material

The algorithms explored in the main body of the dissertation are the ones that made it through to the final product, I spend some time building and testing other algorithms including Kmeans, Support vector machines and naïve bayes, however these were unsuccessful in their tuning or

just poor performance comparatively so were not included, I only have a few lines left from their time within my code. These went through a number of attempts to get them working well with various parameters but in the end had to be abandoned for the sake of the project moving forward.

```
77
78 my_km <- h2o.kmeans(training_frame = train,
79                     validation_frame = test,
80                     k = 2,
81                     seed = 1,
82
83
84
85 )
86
87 my_svm <- h2o.psvm(x=x,
88                  y=y,
89                  training_frame = train,
90
91                  )
92
93 my_nb <- h2o.naiveBayes(x=x,
94                       y=y,
95                       training_frame = train,
96                       nfolds = nfolds,
97                       seed = 1
98
99 )
100
101
```

Figure 34 other algorithms

In my software I have included the data.r that has the training of the models and evaluation of the models and is the main part of the software that should be looked at for insight into how the models were trained and work, my final app is the shiny app consisting of server.r, ui.r and app.r, however due to the final product code being a lean codebase just loading a saved model I have included all code produced throughout this project along with the datasets used.