Contents

1.0 Aim 2
1.1 results summary 2
2.0 Method of implementation 2
2.1 Importing the data 2
2.1.1 Data Categorization 3
2.2 Splitting the data 3
2.3 Data encoding/tokenization 4
3.0 Network selection 4
3.1 Architecture 5
3.2 Components 5
3.3 Solver 5
3.4 Reflected Linear Units (ReLU) 6
4.0 Gathering Metrics 7
5.0 Results 7
Confusion matrix 7
5.1 Roc Graph 8
5.2 Precision and recall 9
6.0 Conclusion 9

Figure 1 data inported 3
Figure 2 all attack_cat 3
Figure 3 selecting data 4
Figure 4 encoded data 4
Figure 5 Neural Network architecture 5
Figure 6 inefficient) Sebastian Ruder. (2016). 6
Figure 7 (efficient) Sebastian Ruder. (2016) 6
Figure 8 Relu calcution 6
Figure 9 k-fold accuracy .... **Error! Bookmark not defined.**
Figure 10 Confusion Matrix if classifier model 8

Table 1 Results from model trained 7

1.0 Aim

The aim of the project is to categorize and learn from network traffic data with machine learning. This report will cover how the data is imported and the given model used to learn and categorize the packet capture data. This will also cover the use of scoring and analysing the classifier performance, this will use a mixture of accuracy data, confusion matrix data and a ROC chain.

1.1 results summary

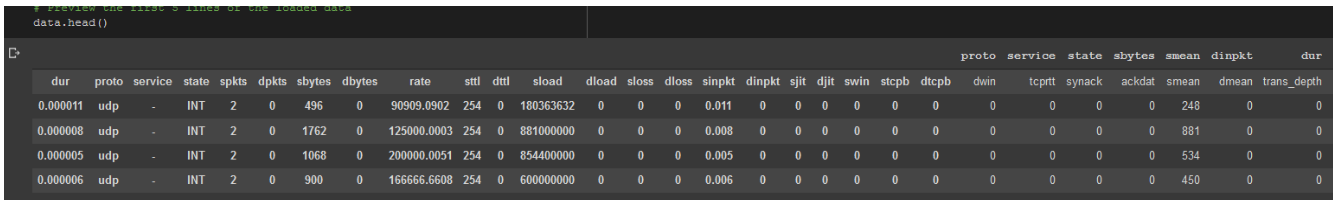
The results gathered to show that a neural network is an efficient way in which data can be categorised but it is slow which if applied to an IDS system this could be an issue to the system as it would take time to process the data. A neural network was selected because of the method of which it processes the data and the fact optimizers can be added in which make the network accurate, the accuracy of the given network were 69% which proves that there can be room for improvement but overall it is above the 50% threshold showing it can make a good prediction from the dataset.

2.0 Method of implementation

Section 2.0 will be split into subsections detailing how the data was imported and tokenized/encoded and the creation of the neural network used. This will include the ways which the data was encoded and split into training sets X Y and Xtest Ytest and the meds used to create the network and how the given classifier network works.

2.1 Importing the data

The first stage of the project was to load the file which can be done with pandas, this is to see and show the data and find any issues within the dataset which can be mitigated. The file was loaded in and encoded with UTF-8-sig which allows telling the library how to read in the information, without the encoding option this can create issues with the testing as the model doesn't know how to read the given data.



```
# Preview the first 5 lines of the loaded data
data.head()
```

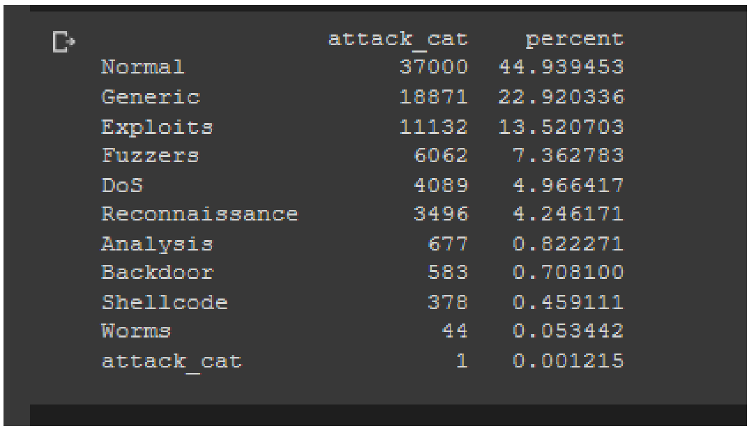
																						proto	service	state	sbytes	smean	dinpkt	dur
dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss	dloss	sinpkt	dinpkt	sjit	djit	swin	stcpb	dtcpb	dwin	tcprtt	synack	ackdat	smean	dmean	trans_depth
0.000011	udp	-	INT	2	0	496	0	90909.0902	254	0	180363632	0	0	0	0.011	0	0	0	0	0	0	0	0	0	0	248	0	0
0.000008	udp	-	INT	2	0	1762	0	125000.0003	254	0	881000000	0	0	0	0.008	0	0	0	0	0	0	0	0	0	0	881	0	0
0.000005	udp	-	INT	2	0	1068	0	200000.0051	254	0	854400000	0	0	0	0.005	0	0	0	0	0	0	0	0	0	0	534	0	0
0.000006	udp	-	INT	2	0	900	0	166666.6608	254	0	600000000	0	0	0	0.006	0	0	0	0	0	0	0	0	0	0	450	0	0

Figure 1 data inported

2.1.1 Data Categorization

To view and get an understanding of the data set, it was decided to split the data up into sub sections and display the average attack vector. This was implemented by listing the attack_cat and dividing by the amount and creating an average then displaying the output as shown in figure X

The justification behind this was to grasp the scope of what the model will be learning and training form.



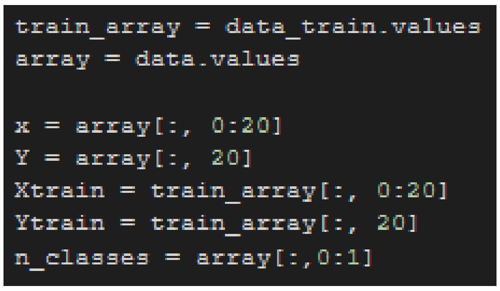
```
                    attack_cat    percent
Normal                   37000  44.939453
Generic                  18871  22.920336
Exploits                 11132  13.520703
Fuzzers                   6062   7.362783
DoS                       4089   4.966417
Reconnaissance            3496   4.246171
Analysis                   677   0.822271
Backdoor                   583   0.708100
Shellcode                  378   0.459111
Worms                       44   0.053442
attack_cat                   1   0.001215
```

Figure 2 all attack_cat

2.2 Splitting the data

The use of data splitting is to have both the dataset variables defined, this is to allow a set to be trained form (x_train) and tested form X_test. Both files are different, the theory is if it can train for a file which is somewhat like the training file it can learn from the test file better.

The data was both split into test and train split.



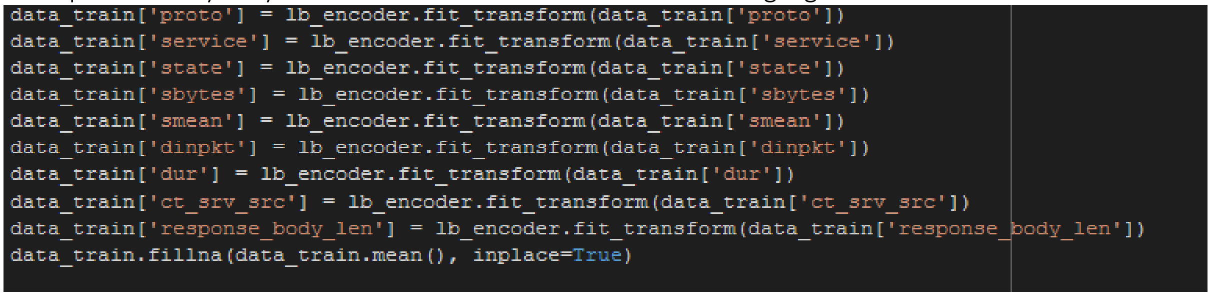
```
train_array = data_train.values
array = data.values

x = array[:, 0:20]
Y = array[:, 20]
Xtrain = train_array[:, 0:20]
Ytrain = train_array[:, 20]
n_classes = array[:,0:1]
```

Figure 3 selecting data

2.3 Data encoding/tokenization

For the model to understand the data this has to be encoded/tokenized, which is having the data converted into a numeric value for the model to learn from, there are different types of encoding and different ways in which the data can be converted into values. For the model currently used sklearn encoder was used which is a basic encoder that converts the values into a single-digit encoded format, all the categories of the dataset where selected, as the more data used the more accurate the network would be as the larger the data span the more it can process and learn the different types of attack categories. Once the data was encoded, the data was then pushed into arrays and labelled Xtest,Xtrain
And split correctly they were then loaded into a machine-learning algorithm.



```
data_train['proto'] = lb_encoder.fit_transform(data_train['proto'])
data_train['service'] = lb_encoder.fit_transform(data_train['service'])
data_train['state'] = lb_encoder.fit_transform(data_train['state'])
data_train['sbytes'] = lb_encoder.fit_transform(data_train['sbytes'])
data_train['smean'] = lb_encoder.fit_transform(data_train['smean'])
data_train['dinpkt'] = lb_encoder.fit_transform(data_train['dinpkt'])
data_train['dur'] = lb_encoder.fit_transform(data_train['dur'])
data_train['ct_srv_src'] = lb_encoder.fit_transform(data_train['ct_srv_src'])
data_train['response_body_len'] = lb_encoder.fit_transform(data_train['response_body_len'])
data_train.fillna(data_train.mean(), inplace=True)
```

Figure 4 encoded data

3.0 Network selection

Once the data was processed the model had to be created, in this section, it will cover how the model works, the use of activation and optimization and the ways in which the results were collected.

A neural network was selected as it is a good method to classify the data used, it has different components which make it accurate as a method to classify and learn from the data, with the use of optimizers and the activation this will prove to allow the model to be accurate when classifying the data.

3.1 Architecture

Input layer: this is a node which provides the information to the network, this is the first layer of the neural network and provides all the information to it. Much like humans, we take in all information this is the first part of that.

Hidden layer: the hidden layer is a layer which has no access to the outside data, this is where the data is processed, an example would be reading a maths book than in your head working out the answer. This would be roughly how this works for the data being processed.

Output layer: The output layer is the layer which outputs the findings, this will be the accuracy. after doing the maths equation in your head, this would be written out the answer on paper.

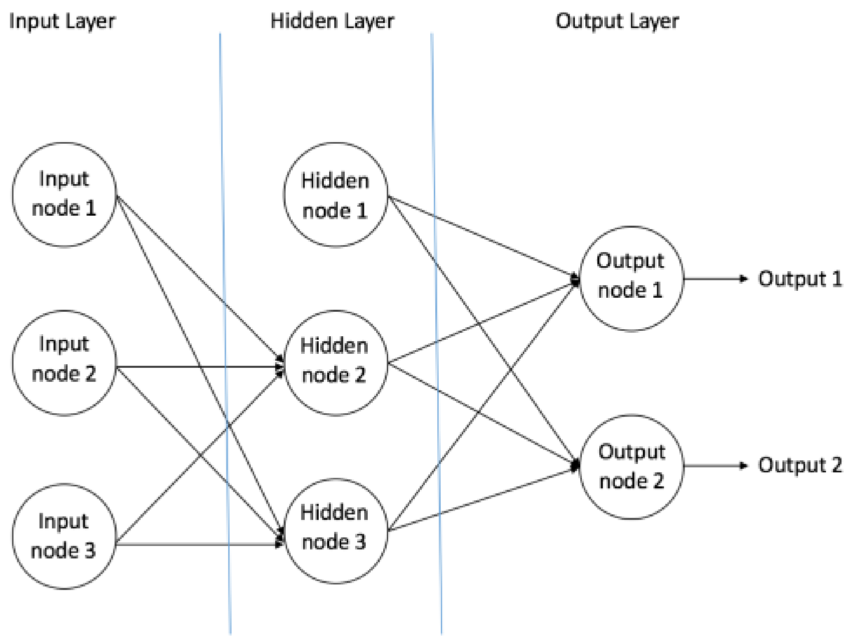


Figure 5 Neural Network architecture

3.2 Components

Within the model there are two different functions which help the model learn and make an accurate prediction of the date when processing it, these are referred to as a solver and an activation function

3.3 Solver

Adaptive Moment Estimation (Adam) is an algorithm specifically made for deep learning. Adam uses *momentum[]* which learns to form its past steps and gradient to work out what direction to go to. Adam incorporates momentum by increasing the use of fractions and the previous gradients to the most recent gradient thus becoming more accurate as it learns.

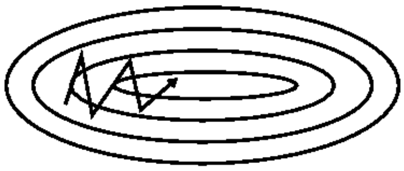


Figure 6 inefficient) Sebastian Ruder. (2016).

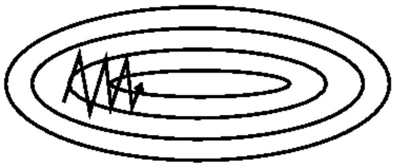


Figure 7 (efficient) Sebastian Ruder. (2016)

3.4 Reflected Linear Units (ReLU)

Relu was used as the main activation function for the model to predict form. Relu was selected as it is a quick way to make calculations and helps improve accuracy.

Relu works by having the data on a gradient, so an example would be: Is there data >0? If the data is greater than zero it is equal to the input size and it has learned if it is a negative it is zero and the gradient has not changed. There are some issues with using Relu which was taken into consideration such as dead_relu problem which is where the values are never updated when learning which means it is somewhat stagnant and cannot progress with the learning.

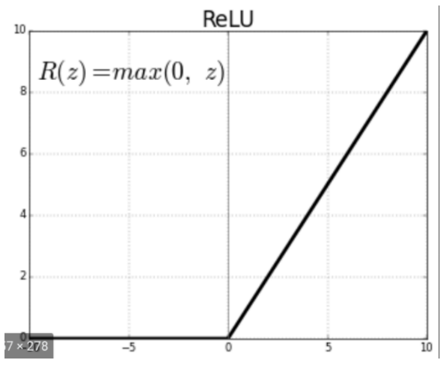


Figure 8 Relu calcution

4.0 Gathering Metrics

There are different ways in which metrics can be measured for machine learning models this section of the report will list and cover the different methods that were used for this model.

Accuracy: accuracy is a method sued to well calculate the overall accuracy of the model, it was done in this project to see at how accurate the neural network was calculated. This calculation is done by having the total number of correct predictions divided by the total number of predictions.

K-fold: k-fold is a way to validate data my shuffling the test set by K amount then calculating an acreage-based form the Penticton

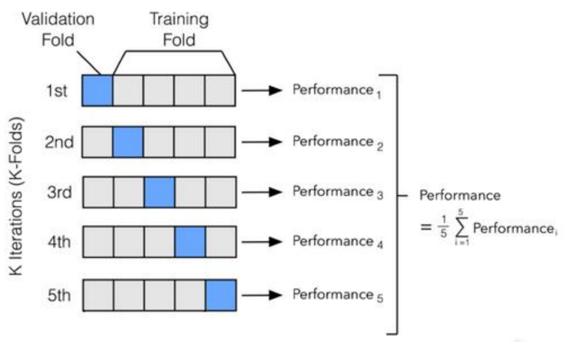


Figure 9 Ashfaque, Johar & Iqbal, Amer. (2019). Introduction to Support Vector Machines and Kernel Methods.

5.0 Results

Table 1 Results from model trained

Accuracy	K-fold(3)
0.763591153289	0.79857898 0.78031628 0.69639994

Confusion matrix

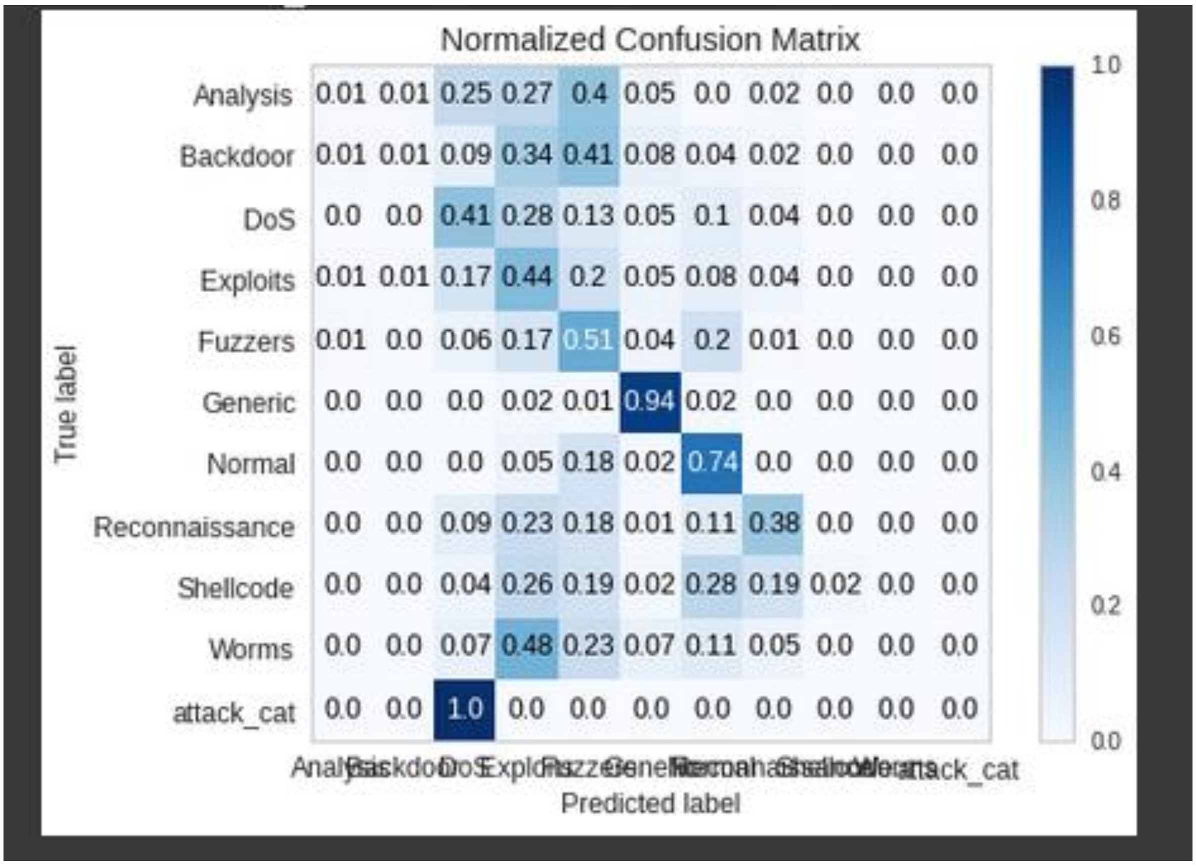
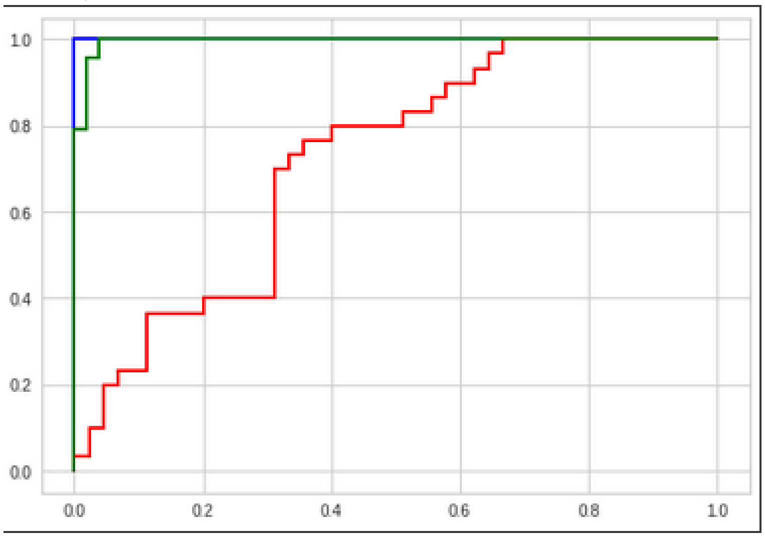


Figure 10 Confusion Matrix if classifier model

5.1 Roc Graph

For the ROC graph shows the data had to be split into three sections then the ROC chain had to be implemented, as shown in figure X shows that the model learned correctly and the first three values form the data set where correctly identified.

- Green Normal
- Blue Genric
- Red Exploits



As the figure shows, normal traffic was the easiest to classify, the graph shows that the exploits where header to classify when learning. The exploits had a higher false positive rate meaning that this part of the data was harder to classify overall and make predictions from, this may be because the exploit data was all different exploits therefore different types of attack traffic and different protocols where used during the network dataset creation.

5.2 Precision and recall

This graph is used to show the relationship of the precision and the recall for how accurate the classifier is. This another visual representation to show the total number of relevant results.

Precision is the percentage of correct results and recall is the overall correct results, this is graphed and used to define the over all effectiveness of the classification and how well the model is at training.

An attempt was made to implement as a graph and output the data but, there where library issues with sklearn and YellowBrick.

6.0 Conclusion

Overall the system proved to be accurate and functioned as intended to detect attacks and packets used to attack the network, the confusion matrix does show that it is accurate at detecting normal traffic which is a positive as things other than that will be an outlier flagged as something somewhat malicious. The model was selected because of the solver functions which did take some time for the mode to learn and make a prediction and train, but it was accurate at detection which is overall the main purpose. Another way to classify the data could have been used such as naive Bayes or an SVM. as they are quicker and somewhat learner the same. The neural network has two main functions of relu and softmax which were why this was used as it is an added layer to make it accurate. The other graphs do show what data was getting categorised and more data would need to be collected on what exploits where used as this can vary and may generate false positive results if this model was used in a intrusion detection system.

References

(2017). *sklearn.metrics.confusion_matrix.* Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Last accessed 20/04/2020.

Sebastian Ruder. (2017). *An overview of gradient descent optimization algorithms.* Available: https://ruder.io/optimizing-gradient-descent/. Last accessed 05/03/2020.

na. (2017). *sklearn.preprocessing.LabelEncoder.* Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. Last accessed 20/04/2020.

na. (2019). *1.17. Neural network models (supervised).* Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html. Last accessed 20/04/2020.