Project overview for portfolio upload

Project Title:

Machine Learning: Classifier to categorise network packet data according to attack category.

Project Description:

Building a classifier in Python to categorise network packet data according to their attack category (Normal, Fuzzers, Analysis, Backdoor, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms). Using standard metrics to measure and discuss the performance of the classifier. Produced visualisation graphs of the data based on the obtained results

Project Tooling:

- Python
- Python Libraries
    - Sickitlearn
    - Numpy
    - pandas

Project processes and techniques:

- Machine Learning
- Reading data in python
- Building a classifier in Python
- Building visualisation graphs (ROC, recall, confusion matrix)
- Using random forest sorting algorithm

Grade: A

Appendices have been removed to avoid plagiarism from future students, however I am happy to provide additional information or discuss the work further if desired.

Table of Contents

1. Introduction

After the recent hacktivist attack the technical staff at Scottish Glen have been recording incoming network traffic to ensure that another attack is not to happen again. However, upon inspection of the recorded network traffic there is some concerning information being obtained. Therefore, Scottish Glen have requested that the IT Manager develop a classifier to categorise network packet data to identify for suspicious traffic. To develop a system that is beneficial to Scottish Glen it is important the IT manager outlines the process in the following report.

2. Machine Learning Algorithms

Machine learning algorithms can be categorised into 4 types: ***Supervised, Unsupervised, Reinforcement*** and ***Recommender Systems***. Each of these categories will have its own algorithms that can be utilised by a user to solve specific problems. For this report, the model being constructed is a classifier, therefore the algorithm that should be chosen is from the supervised category, figure 1-1 shows some of the possible algorithms to choose from. Random Decision Forest (random forest) was the model chosen for several benefits.

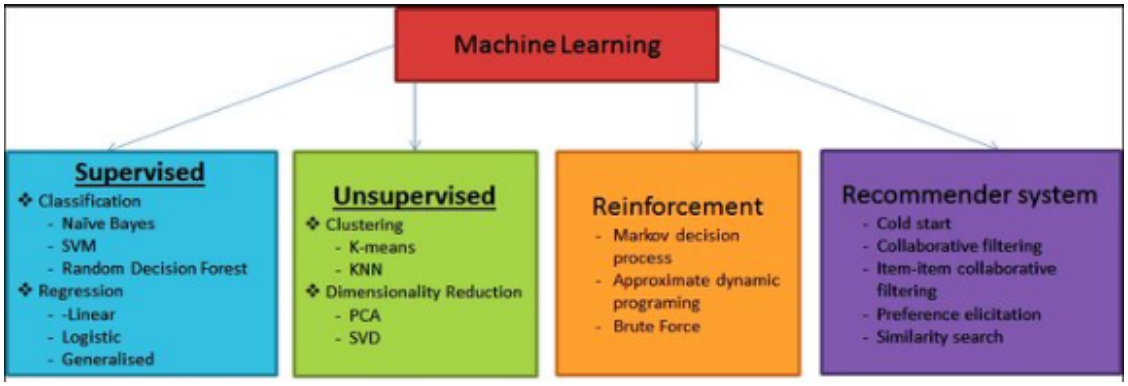


Figure 2-1- Graphical Representation of the spread of machine learning algorithms
Source : (Jindal, Gupta and Bhushan, 2019)

The random forest algorithm uses a combination of separate decision trees within its model, this use of multiple decision trees is indicative of its name – multiple trees join to create a forest. The algorithm works through each node of the decision tree working on a subset of features to calculate a final, single output that is the prediction of all the combined trees. Through combining different decision trees under one model, it is assumed this will improve overall prediction accuracy and robustness (Wainberg *et al,* 2016) .This process of taking outputs from multiple different decision trees is known as ensemble learning and is visualised in figure 1-2. Ensemble learning is an advantage over other supervised learning algorithms such as Support Vector Machines, Naïve Byes or Decision Trees as those that use ensemble learning are better suited to deal with imbalanced data (Hemanth *et al.,* 2019). Imbalanced data can often skew results and provide an unreliable model (Wainberg *et al,* 2016) Ensemble methods can combine multiple models on different subsets of the data, providing well rounded results on these imbalanced data set. In reference to the created model which

accompanies the report, it was important to have a model which could account for how the model will pick up on varying attack types, not adhering to any trend in the data.

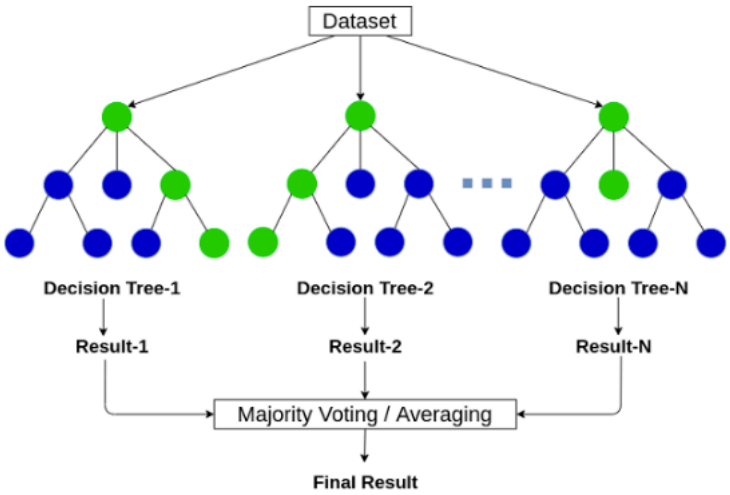


Figure 2-2 Graphical display of how a random forest traverses data to buid a model
Source - https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

However, with the nature of a random forest model being a collection of multiple decision trees, this will often lead to large quantities of separate trees and thus nodes. This will generate large amounts of complexity within the model (Nabila Farnaaz and M.A. Jabbar, 2016). This complexity is where the benefits of something such as stability comes in as it can provide a reliable result, traversing many paths and effectively capturing the nuances of the data. However also will lead to 'computational complexity', this is where models require more computational power, leading to longer training times and inefficient modelling for cases where the user requires a quick result. Models such as Support Vector Machines (SVM) may be more suitable for situations where the efficiency of a model matters more over their accuracy, such as real-time systems due to SVMs being able to find the optimal decision boundary in these spaces (Ahmad *et al.,* 2018).

When considering this trade-off between speed and precision it becomes clear why Random Forest is the preferred classifier for analysing network packet data and classifying them into distinct groups. Implementing a model such as random forest will allow for the best level of precision when compared to other algorithms such SVM or naïve bytes (Jiong Zhang, Zulkernine and Haque, 2008). It is important for this model to be as accurate as possible for Scottish Glen as they do not want to be constantly altered for false positives or false negatives. A model which does not predict these things correctly will enable the company that is using the model to allocate resources to dead ends, which is not feasible for a smaller company such as Scottish Glen.

3. Design of Classifier

When designing a classier, it is important that certain areas of the model are created correctly as they will have a direct impact on the accuracy and reliability of the predictions made by the model (Alshamy, *et al,* 2021). For the correct steps to be conducted, a data pipeline must be implemented to provide the creator of the model with concrete steps to produce the optimal model. Figure 3-1 is a variation of this data pipeline, and it covers some of the most crucial steps that are outlined from the following report.

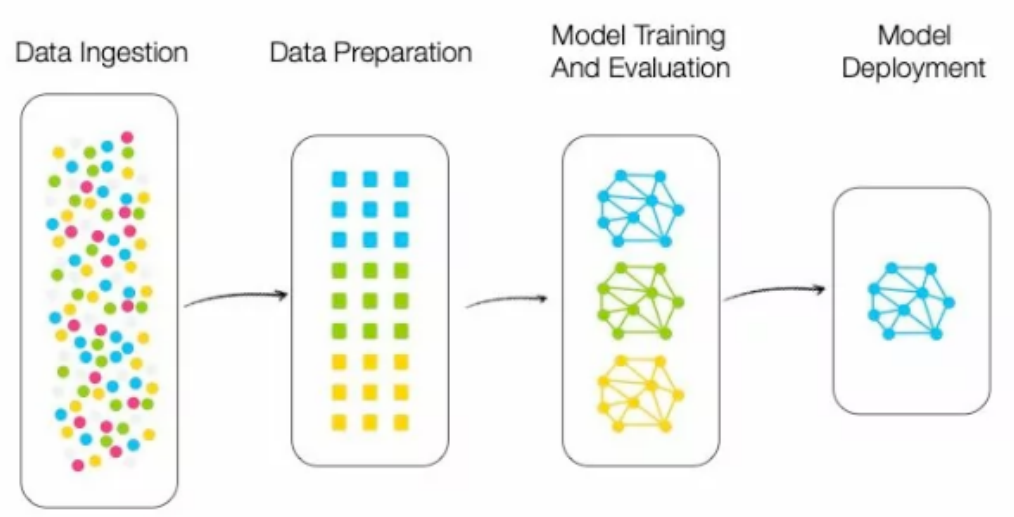


Figure 3-1 – Data Pipline Example
Source - https://www.forepaas.com/en/blog/data-pipelines-ai-pipelines-2/

3.1 Data Ingestion & Pre-processing

Data ingestion and data pre-processing are considered some of the most important parts of the data pipeline. This is because correct pre-processing and data ingestion will ensure that the raw data which will be used by the model is suitable for its use case. In doing so, it will aim to provide the model with the best chance of optimal training and finally the output.

The work carried out in pre-processing included numerous actions to increase data quality and provide the best compatibility for the for the Random Forest classier. The first step taken was to address the missing values denoted by '-' in the rows. This was done through creating a new designated category of 'Unknown' with an attempt to prevent skewed results.

Alongside this, label encoding was used. Label encoding was used to transform the categorical variables such as ('proto', 'service' and 'state') into numerical data so that the model knows how to handle the information in these columns of the data. This was also done for the target variable. This had to be done for the target variable when implementing the visualisation graphs. This was because the ROC graph was not able to take In categorical data and therefore a process involving binarize was used. Therefore, when considering the visualisation graphs in section 3.4 it is important to consider the labels with their allocated numerical values from table 3-1.

Table 3-1

0	Analysis
1	Backdoor
2	DoS
3	Exploits
4	Fuzzers
5	Generic
6	Normal
7	Reconnaissance
8	Shellcode
9	Worms

Finally, data was already split by Scottish Glen into training and testing, this allowed each dataset to be used straight away, simply ingesting the data as CSV files into the structure data frame format.

3.2 Modelling

The created model implements a random forest classifier with some parameters, which include ***n_jobs = -1***, ***random_state=3*** and ***n_estimators=102***. The first parameter (n_jobs) is related to the performance of the random forest. As previously stated in Section 2, often Random Forrest models can be demanding on the system and provide a large running time. Therefore, the value of -1 will allow all CPU cores of the processor to be utilised, speeding up the training process.

Next the ***random_state=3*** parameter sets the random seed. The random seed is used to determine whether the numbers generated during the model's execution is reproducible. It is important that when testing the behaviour and efficiency of the model, the results can be reproduced for comparison, setting this to a number can allow for this.

Finally, ***n_estimators=102*** sets the number of decision trees to be used in this random forest. Increasing the number of trees can lead t a more robust model, although as discussed in section 2 this can also increase computational complexity which is a trade-off that is often balanced by practitioners and will be by Scottish Glen.

Whilst the initial Random Forrest Classifier will produce a model of a certain precision (recorded in section 3.3), further hyperparameter tuning is vital for Scottish Glen to achieve the best possible model. This tuning can be done through techniques such as GridSearchCV.

GridSearchCV is a sickitlearn function which would allow the classifier to tuned through external combinations of hyperparameters. GridsearchCV works through searching a predefined grid of hyperparameters, evaluating model performance using cross validation (Anurag Verma, 2023). These parameters could include ***max_depth*** to determine the maximum depth of the trees or ***bootstrap*** to determine whether bootstrap samples are used when building trees.

Other techniques can be used to improve machine performance. One such method is SMOTE or Synthetic Minority Over-Sampling Technique. This technique is used to counteract class imbalance, where one class is will significantly outnumber the others. This technique uses 'synthetic samples' for minority classes, helping to oversample the minority classes and boost the model's precision (Alshamy, *et al,* 2021).

3.3 Standard Metrics

To evaluate the classification model used in this report, several standard metrics were used: precision, recall, F1-Score and support Understanding these metrics are crucial to understanding the effectiveness of the model.

Score: 0.9075173519028635
Classification Report:

	precision	recall	f1-score	support
Analysis	0.01	0.01	0.01	677
Backdoor	0.01	0.08	0.02	583
DoS	0.56	0.12	0.20	4089
Exploits	0.63	0.77	0.69	11131
Fuzzers	0.30	0.59	0.40	6061
Generic	1.00	0.97	0.98	18871
Normal	0.96	0.76	0.85	36998
Reconnaissance	0.93	0.80	0.86	3495
Shellcode	0.35	0.71	0.47	378
Worms	0.64	0.16	0.25	44
accuracy			0.76	82327
macro avg	0.54	0.50	0.47	82327
weighted avg	0.84	0.76	0.78	82327

Accuracy Score: 0.7571877998712452

Figure 3-3 – Classification report with labels

Precision is a vital metric in classifier evaluation. Precision measures the accuracy of positive predictions. In the context of this classifier, precision indicates the percentage of instances correctly identified within each specified attack category. For example, a precision score of the 'exploits' label has a score of 0.63, showing that 63% of Instances predicted as exploits were correct – this highlights the need for improvement of precision in the classifier.

Recall, also known as sensitivity, and it is a metric that measures the ability of the classifier to correctly identify all relative instances within the dataset, disregarding false negatives. Unlike precision, recall prioritises capturing all instances of the positive class.

F1-Score is a measure of the classifier's performance through balancing precision and recall. The f1-score can give a quantifiable answer to how balanced the model's precision and recall is. With a good f1-score it indicates that the classifier was able to achieve high precision and high recall at the same time This can be beneficial for analysing classifiers as it can identify which classifiers are most effective at decision making.

Finally, support indicates the number of actual occurrences of each class in the dataset. The support metric will provide an insight to the distribution of the defined classes and allows the user to understand the reliability of the evaluation results of each class.

3.4 Communication of Results

When looking at the results of the implemented classifier, multiple different visualisation tools can be implemented to provide different information on the accuracy of the classification with a confusion matrix, the performance of a classifier with an ROC graph and the precision of the model, with the associated recall drop off in the precision curve.

Confusion Matrix

A confusion matrix is a table that can describe the performance of the implemented classification model. The matrix outlined in figure 3-4 provides a breakdown of the correct and incorrect classification for each of the specified classes. The darker green cells indicate higher values where the model mad most of its predictions.

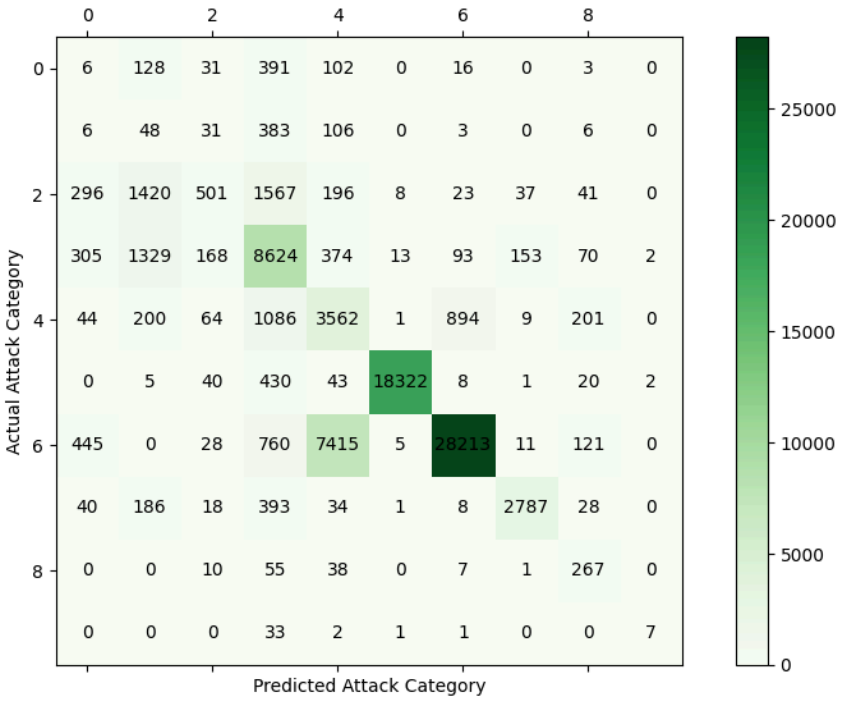


Figure 3-4 – Confusion Matrix

Receiver Operating Characteristics (ROC)

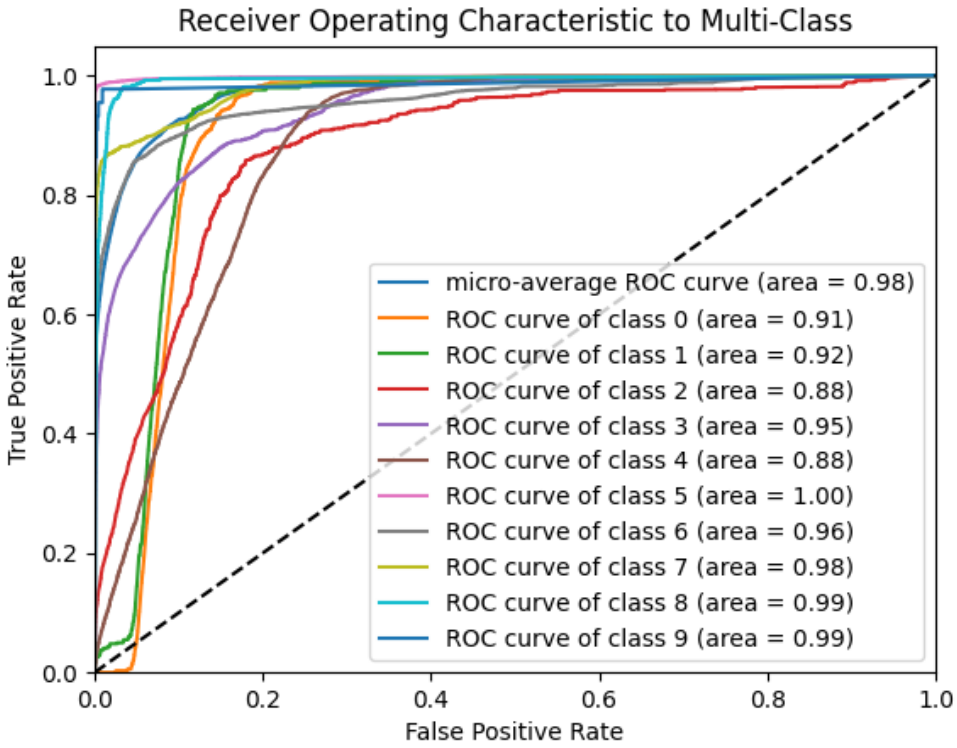


Figure 3-5 ROC graph

Each colour of line represents a different class. As a result of the program created, the class labels are not included as they were changed to numerical data to allow for the curves to be implemented. The classes are defined in section 3.1 above. The ROC is a graphical representation of the performance of a classifier. As this is a multi-class problem the ROC curve was extended to handle multiple classes. The approach used here was the one vs all approach. This is where a separate model is trained for each class predicted against other classes. This means that there is now as many curves as there was classes. The performance from figure 3-5 indicates that some classes perform very well, showing near perfect classification, whilst others still need tweaked to perform better in the testing.

Precision Curve

A precision recall curve is a useful visualisation tool for recording the performance of the classifier upon execution. The curve displayed in figure 3-6 plots the precision (y-axis) against the recall (x-axis) for different thresholds, showing the previously discussed trade off allowing for analysis of how the classifier performs this trade off, such as between making precision predictions and capturing relevant instances. By using the information in the precision curve visualisation it can allow for further insight to be learnt about the designed model.

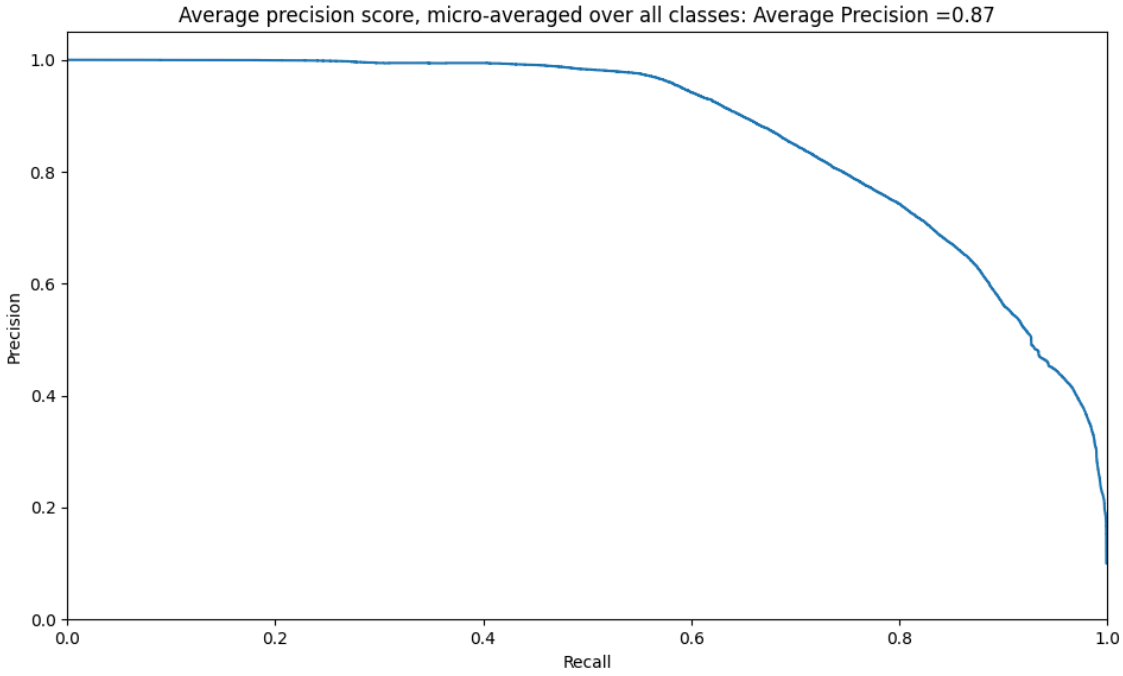


Figure 3-6 Precision recall graph

4. Conclusion

The model outlined in the report, alongside what has been created already provides a solid foundational ground for a network traffic classifier that attempt to categorise malicious network traffic into specific categories. The visualisation and results in the classification report show a model which has a respectable score. However, there needs to be further hyperparameter tuning made by Scottish Glen to have the model be suitable for its use case. This can be done through the suggested methods outlined in the modelling section of the report. Overall, there is usable model for Scottish Glen, alongside enough information provided in this report for future work to be carried out.

References

Ahmad, I., Basheri, M., Iqbal, M.J. and Rahim, A. (2018) 'Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection', *IEEE Access,* 6, pp. 33789-33795. doi: 10.1109/ACCESS.2018.2841987 https://ieeexplore.ieee.org/document/8369054

Alshamy, R., Ghurab, M., Othman, S. and Alshami, F. 'Intrusion Detection Model for Imbalanced Dataset Using SMOTE and Random Forest Algorithm', *Advances in Cyber Security,* , pp. 361-378. doi: 10.1007/978-981-16-8059-5_22 http://link.springer.com/10.1007/978-981-16-8059-5_22

Chaudhary, A., Kolhe, S. and Kamal, R. (2016) 'An improved random forest classifier for multi-class classification', *Information processing in agriculture,* 3(4), pp. 215-222. doi: 10.1016/j.inpa.2016.08.002 https://dx.doi.org/10.1016/j.inpa.2016.08.002

Hemanth, J., Fernando, X., Lafata, P. and Baig, Z. (2019) 'A Review on Random Forest: An Ensemble Classifier', *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018,* 26, pp. 758-763. doi: 10.1007/978-3-030-03146-6_86 http://ebookcentral.proquest.com/lib/SITE_ID/reader.action?docID=5625034&ppg=804

Jindal, M., Gupta, J. and Bhushan, B. (2019) 'Machine learning methods for IoT and their Future Applications', *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS),* , pp. 430-434. doi: 10.1109/ICCCIS48478.2019.8974551 https://ieeexplore.ieee.org/document/8974551

Jiong Zhang, Zulkernine, M. and Haque, A. (2008) 'Random-Forests-Based Network Intrusion Detection Systems', *IEEE transactions on systems, man and cybernetics. Part C, Applications and reviews,* 38(5), pp. 649-659. doi: 10.1109/TSMCC.2008.923876 https://ieeexplore.ieee.org/document/4603103

Michael Wainberg, Babak Alipanahi and Brendan J. Frey (2016) 'Are Random Forests Truly the Best Classifiers?', *Journal of Machine Learning Research,* https://jmlr.org/papers/volume17/15-374/15-374.pdf

Nabila Farnaaz and M.A. Jabbar (2016) 'Random Forest Modeling for Network Intrusion Detection System', *Procedia Computer Science,* 89, pp. 213-217. doi: 10.1016/j.procs.2016.06.047

Prinzie, A. and Van den Poel, D. (2008) 'Random Forests for multiclass classification: Random MultiNomial Logit', *Expert Systems with Applications,* 34(3), pp. 1721-1732. doi: 10.1016/j.eswa.2007.01.029 https://dx.doi.org/10.1016/j.eswa.2007.01.029

Appendices

Appendix A – Code Screenshots

```
#CMP510 - Engineering Resilient Systems coursework
# Classifer to categorise network packets according to attack category
# Random Forest chosen & visualisation using : Confusion Matrix, Precision Recall Curve, ROC

#importing required libraries - some might not be used due to unused code in program
from sklearn.calibration import label_binarize
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, roc_curve, auc, precision_recall_curve, confusion_matrix, average_precision_score
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

#reading the testing and training data
training_data = pd.read_csv("CMP510_training_dataset1.csv")
testing_data = pd.read_csv("CMP510_testing_dataset1.csv")


#edit the head value to display more or less data from the dataset
print ("Loading training Data:")
print (training_data.head())
print ("")
print ("Loading testing Data:")
print (testing_data.head())


#data preprocessing
'''…
#assigning missing values in the service colun with 'unknown'
training_data.loc[training_data['service'] == '-', 'service'] = 'Unknown'
testing_data.loc[testing_data['service'] == '-', 'service'] = 'Unknown'

#transforming categorical features to numerical values
label_encoder = LabelEncoder()
categorical_cols = ['proto', 'service', 'state']
for col in categorical_cols:
    training_data[col] = label_encoder.fit_transform(training_data[col])
    testing_data[col] = label_encoder.fit_transform(testing_data[col])

# #transforming target variable to numerical values
target_label_encoder = LabelEncoder()
training_data['attack_cat'] = target_label_encoder.fit_transform(training_data['attack_cat'])
testing_data['attack_cat'] = target_label_encoder.fit_transform(testing_data['attack_cat'])

# Split the data
X_training_data = training_data.drop('attack_cat', axis=1)
y_training_data = training_data['attack_cat']

X_testing_data = testing_data.drop('attack_cat', axis=1)
y_testing_data = testing_data['attack_cat']

#trying to increase the accuracy of the RF using gridsearch CV
# param_grid = {
#     'n_estimators': [100, 200, 300, 400, 500],
#     'max_depth': [None, 10, 20, 30, 40, 50],
#     'min_samples_split': [2, 5, 10],
#     'min_samples_leaf': [1, 2, 4],

# }
# clf = RandomForestClassifier()
# grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)


# initilising and training random forest classifier
clf = RandomForestClassifier(n_jobs=-1, random_state=3, n_estimators=102)
trained_model = clf.fit(X_training_data, y_training_data)
print ("")
# Printing the accuracy of the model on the training data
print ("Score: ", trained_model.score(X_training_data, y_training_data))
```

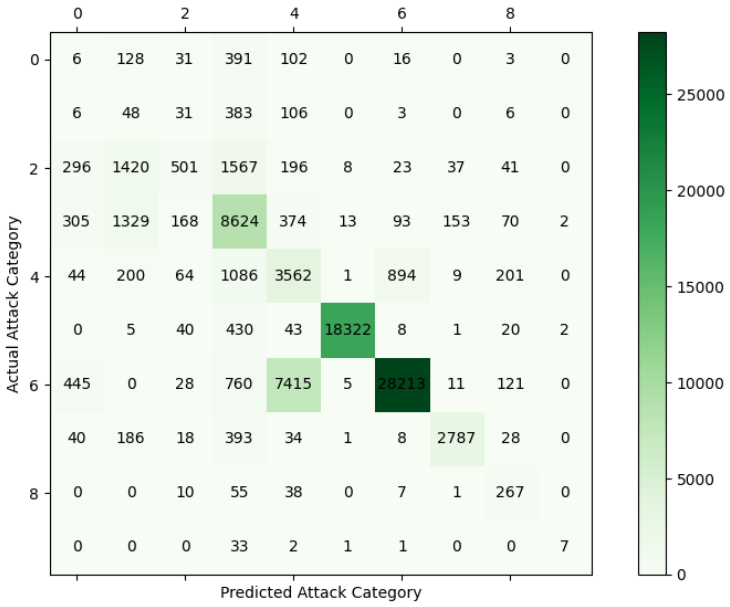
```
# Predict on the testing data
y_predicting_data = clf.predict(X_testing_data)

# Evaluate the model
# Then ccuracy of the model on the testing data
print('Classification Report:')
print(classification_report(y_testing_data, y_predicting_data))
print('Accuracy Score:', accuracy_score(y_testing_data, y_predicting_data))




# Compute confusion matrix
compute_matrix = confusion_matrix(y_testing_data, y_predicting_data)

#creating a visualisation for the confusion matrix to remove the basic text one
figure_plot, axes_plot = plt.subplots()
object_on_axes = axes_plot.matshow(compute_matrix, cmap=plt.cm.Greens)
figure_plot.colorbar(object_on_axes)
plt.xlabel('Predicted Attack Category')
plt.ylabel('Actual Attack Category')

#annotating the plot
for (i, j), z in np.ndenumerate(compute_matrix):
        axes_plot.text(j, i, str(z), ha='center', va='center')

plt.show()

### ROC IMPLIMENTATION ####

""" REFERENCES -> PLEASE READ
The following code below to impliment the ROC
has been pulled together from multiple sources online, some of it i found quite difficult to
implement therefore i have just moved over the structure and adjusted for this project.
Therefore  I wanted to ensure i correctly referenced them, further references will be given in the report,
however I also wanted to ensure i referenced here. This ends around line 163 for the ROC implimentation
https://stats.stackexchange.com/questions/2151/how-to-plot-roc-curves-in-multiclass-classification
https://stackoverflow.com/questions/50941223/plotting-roc-curve-with-multiple-classes
https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
https://scikit-learn.org/stable/modules/multiclass.html#multiclass
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html#sklearn.preprocessing.LabelBinarizer
https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-binarization
https://towardsdatascience.com/multiclass-classification-evaluation-with-roc-curves-and-roc-auc-294fd4617e3a

"""

### IMPORTANT -> For this to work it is important to undo the commenting for the label encoder on line 48 as this
# will enable the labels to be taken as numerical values
#binarising the values for ROC and precision recall curve implementation & visualisation
y_testing_data_binarized = label_binarize(y_testing_data, classes=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
n_classes = y_testing_data_binarized.shape[1]

#initistialising the classifier and predicting probabilities
classifier = OneVsRestClassifier(clf)
y_score = classifier.fit(X_training_data, y_training_data).predict_proba(X_testing_data)

#initialising dictionaries to store false positive, true postive, ROC and AUC scores
false_positive_rates = dict()
true_positive_rates = dict()
roc_auc = dict()


for i in range(n_classes):
    false_positive_rates[i], true_positive_rates[i], _ = roc_curve(y_testing_data_binarized[:, i], y_score[:, i])
    roc_auc[i] = auc(false_positive_rates[i], true_positive_rates[i])


# Compute micro-average true positive rate
fpr_micro, tpr_micro, _ = roc_curve(y_testing_data_binarized.ravel(), y_score.ravel())
roc_auc_micro = auc(fpr_micro, tpr_micro)
```

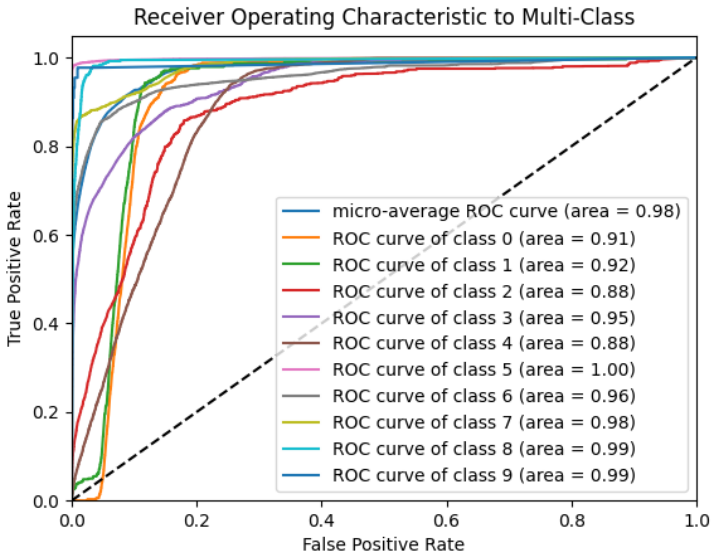
```
# plotting ROC for each class
plt.figure()
plt.plot(fpr_micro, tpr_micro,
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc_micro))
for i in range(n_classes):
    plt.plot(false_positive_rates[i], true_positive_rates[i], label='ROC curve of class {0} (area = {1:0.2f})'
                                   ''.format(i, roc_auc[i]))

#formating the labels, legend and look of the ROC
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic to Multi-Class')
plt.legend(loc="lower right")
plt.show()



### PRECISION RECALL CURVE IMPLEMENTATION
""" REFERENCES -> Please read
 Much like above, difficult implementaiton albeit was easier after understanding how to implement ROC
 Despite this though, code heavily referenced from the following links:
 https://stackoverflow.com/questions/50941223/plotting-roc-curve-with-multiple-classes
 https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#:~:text=The%20precision%2Drecall%20curve%20shows,a%20low%20false%20negative%20rate.
 https://stackoverflow.com/questions/56090541/how-to-plot-precision-and-recall-of-multiclass-classifier
 https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
 """

#initilisaing dictionares to store preciison, recall and average precision scores
precision = dict()
recall = dict()
average_precision = dict()

#computing precision recall and average precision scores
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_testing_data_binarized[:, i], y_score[:, i])
    average_precision[i] = average_precision_score(y_testing_data_binarized[:, i], y_score[:, i])

precision["micro"], recall["micro"], _ = precision_recall_curve(y_testing_data_binarized.ravel(), y_score.ravel())
average_precision["micro"] = average_precision_score(y_testing_data_binarized, y_score, average="micro")

#plotting and formatting precision recall curve simiarly to above
plt.figure()
plt.step(recall['micro'], precision['micro'], where='post')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Average precision score, micro-averaged over all classes: Average Precision ={0:0.2f}'.format(average_precision["micro"]))
plt.show()
```

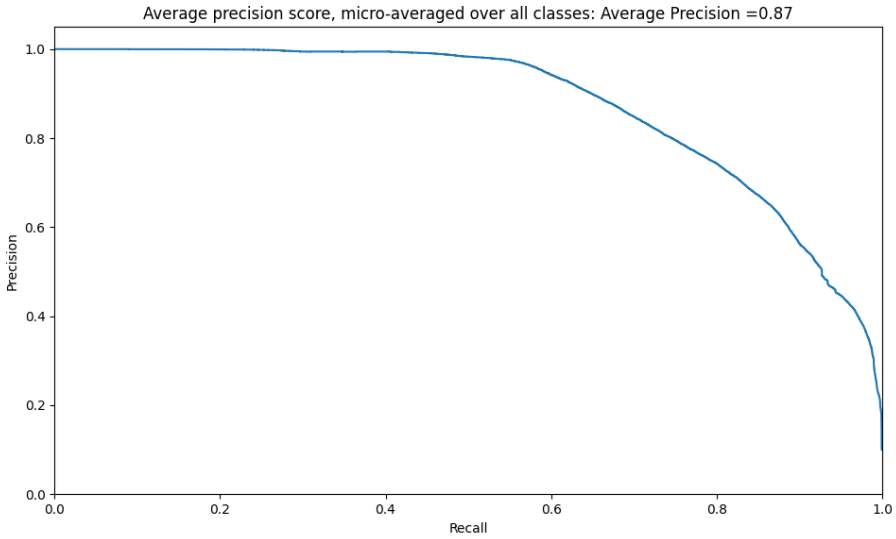
Appendix B – Program Outputs

```
Score:  0.9075173519028635
Classification Report:
                precision    recall  f1-score   support

      Analysis       0.01      0.01      0.01       677
      Backdoor       0.01      0.08      0.02       583
           DoS       0.56      0.12      0.20      4089
      Exploits       0.63      0.77      0.69     11131
       Fuzzers       0.30      0.59      0.40      6061
       Generic       1.00      0.97      0.98     18871
        Normal       0.96      0.76      0.85     36998
Reconnaissance       0.93      0.80      0.86      3495
     Shellcode       0.35      0.71      0.47       378
         Worms       0.64      0.16      0.25        44

      accuracy                           0.76     82327
     macro avg       0.54      0.50      0.47     82327
  weighted avg       0.84      0.76      0.78     82327

Accuracy Score: 0.7571877998712452
```

```
Score:  0.9075173519028635
Classification Report:
              precision    recall  f1-score   support

           0       0.01      0.01      0.01       677
           1       0.01      0.08      0.02       583
           2       0.56      0.12      0.20      4089
           3       0.63      0.77      0.69     11131
           4       0.30      0.59      0.40      6061
           5       1.00      0.97      0.98     18871
           6       0.96      0.76      0.85     36998
           7       0.93      0.80      0.86      3495
           8       0.35      0.71      0.47       378
           9       0.64      0.16      0.25        44

    accuracy                           0.76     82327
   macro avg       0.54      0.50      0.47     82327
weighted avg       0.84      0.76      0.78     82327
```







Appendix C – References for coding

Inserting miscellaneous references here as they do not fit in the references section but were used to guide implementation to developing the model.

https://github.com/harshilpatel1799/IoT-Network-Intrusion-Detection-and-Classification-using-Explainable-XAI-Machine-Learning/blob/main/UNSW-NB%2015%20Dataset%20Modified%20(Explainable%20AI)%20ML%20Methods%20Comparison%20%20-%20Attack%20or%20Normal%20Response.py

https://github.com/kolxy/cyber-security-analysis

https://github.com/harshilpatel1799/IoT-Network-Intrusion-Detection-and-Classification-using-Explainable-XAI-Machine-Learning/blob/main/UNSW-NB%2015%20Dataset%20Modified%20(Explainable%20AI)%20ML%20Methods%20Comparison%20%20-%20Attack%20or%20Normal%20Response.ipynb

https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e#:~:text=Prediction%20of%20missing%20values%3A&text=Using%20the%20other%20features%20which,the%20feature%20having%20missing%20value.

https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/#:~:text=Precision%2DRecall%20Curves%20in%20Python&text=The%20precision%20and%20recall%20can,precision%2C%20recall%20and%20threshold%20values.

https://www.geeksforgeeks.org/how-to-convert-categorical-variable-to-numeric-in-pandas/

https://stats.stackexchange.com/questions/2151/how-to-plot-roc-curves-in-multiclass-classification

https://stackoverflow.com/questions/50941223/plotting-roc-curve-with-multiple-classes

https://stackoverflow.com/questions/44244435/how-to-binarize-randomforest-to-plot-a-roc-in-python

https://stackoverflow.com/questions/60636444/what-is-the-difference-between-x-test-x-train-y-test-y-train-in-sklearn

https://stackoverflow.com/questions/70963332/plot-roc-curve-with-y-label-and-y-pred-both-with-binary-values

https://stackoverflow.com/questions/56090541/how-to-plot-precision-and-recall-of-multiclass-classifier

https://www.scikit-yb.org/en/latest/api/classifier/rocauc.html#multi-class-rocauc-curves

https://towardsdatascience.com/multiclass-classification-evaluation-with-roc-curves-and-roc-auc-294fd4617e3a

https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248

https://scikit-learn.org/stable/modules/multiclass.html#multiclass

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.binarize.html

https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-binarization

https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#:~:text=The%20precision%2Drecall%20curve%20shows,a%20low%20false%20negative%20rate.

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html#sklearn.preprocessing.LabelBinarizer

https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html