A Project Report

On

IoT Device Security

BY

Khush Bhuta

2022A7PS1333H

Under the supervision of

Dr. Barsha Mitra

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

CS F376: DESIGN PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(August to December - 2024)

ACKNOWLEDGEMENTS

I am extremely grateful to Dr. Barsha Mitra for entrusting me with the opportunity to pursue this project on IoT Device Security. I am grateful for her insightful feedback and encouragement that has significantly increased my understanding on this topic. I would like to thank my teammates – Chaitanya and Shobhan for guiding me, and resolving any queries that I had during the course of this project. This project offered me an incredible opportunity to apply my skills to use and encounter a real-world problem.

Lastly, I would like to thank the CS-IS Department of BITS Hyderabad for providing me with the infrastructure and resources to be able to pursue this project.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**IoT Device Security**" submitted by Mr. Khush Bhuta (ID No. 2022A7PS1333H) in partial fulfillment of the requirements of the course CS F376, Design Project Course, embodies the work done by him under my supervision and guidance.

Date: 6th December 2024 **(Dr. Barsha Mitra)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

With the surge in the application of lightweight IoT Devices like sensors and smart devices, it has become crucial to protect them from cybercriminals due to their vulnerabilities and nature. These devices become prime targets during cyber attacks to get access to the network these devices are relying on. Due to this rise in the use of IoT devices, it has become essential to protect them by recognizing any kind of anomalies that enter the network. As a result the security concerns have heightened rapidly. To serve this purpose, a lot of research has been done in the field of Behavioral Fingerprinting and Anomaly Detection. Most techniques rely on network-level features to fingerprint these devices using the packet payload and packet headers captured in the form of PCAP files. Training machine learning classification models on this helps us develop individual device fingerprints. However, network level features do not constitute the only source of data that can be used to develop fingerprints. Recent studies have been working on developing device fingerprints using hardware features like Time-Domain and Frequency-Domain analysis and combining the model with sole – Network level features. This model yields much better metrics (Accuracy, Precision, Recall and F1-Score) compared to previous models that use only one of the type of features. Our aim is to implement this multi-modal setup on small IoT Devices and recognize them with high accuracy and train Machine Learning and Deep Learning models. Besides a multimodal setup, we aim at testing our models on unsampled open source IoT Datasets to check their validity.

CONTENTS

1. Literature Review

IoT Device Security was a completely new domain for me, and hence to gain knowledge on every topic that surrounds it, I performed recursive literature review. This helped me gain insights about the latest developments in this field and the research that has been going on in the previous years. I came across several papers and went through the interesting papers that had been cited in the same. This helped me gain knowledge on how papers are formally published and cited. The following are the research papers I went through along with some key insights on each one.

- **IoTDevID: A Behavior-Based Device Identification Method for the IoT -** This paper emphasized on a Behavior-based device Identification method using Non-IP and Low-energy protocols. The study used two open-source IoT Datasets (UNSW IoT, Aalto Dataset). The protocols used by them were Bluetooth, Zigbee and ZWave. Packet-level feature exploration involved various protocol types, payload characteristics and behavioral characteristics. They focused on a Feature Importance Voting to evaluate the importance of various features to gain an optimized feature vector. The approach also used a Genetic Algorithm selection (GA) – to select the most discriminative features from the pool, optimizing them specifically for a Decision Tree classifier. This study gained significantly better results than previous studies like – IOTSense and IOTSentinel.

- **A systematic framework for categorizing IoT Device Fingerprinting Mechanisms –** This paper talks about the various properties of achieving effective fingerprints for devices, and various fingerprinting terminology. The three essential properties for achieving effective fingerprints are – Unique identity, integrity and Reproducibility (features used in the fingerprinting process must be stable in the presence of environmental changes). It talks about the difference between Passive and Active Fingerprinting, Static vs Dynamic feature fingerprinting, Adaptive vs Fixed fingerprinting, Rule-based vs Machine Learning based fingerprinting and Network-packet vs Flow-based fingerprinting.

- **Optimal Feature Set Selection for IoT Device Fingerprinting and Edge Infrastucture using Machine Learning –** This study used the UNSW open-source IoT dataset to perform an 81% reduction in feature size, to optimize the feature vector. Edge devices are resource constrained and the Machine Learning classification models are computationally intensive. The research paper highlights a concise set of steps with which we can efficiently reduce the feature vector size while maintaining the same level of accuracy and precision in classification. The results highlighted the One vs Rest strategy for identifying new IoT devices into the network and reduced the feature vector size from 111 to 21.

- **Multi-level IoT Device Identification –** This study aims to improve the scalability and accuracy of IoT Device Fingeprinting by overcoming limitations like incomplete class detection, and the need for frequent model updates. It emphasizes the use of payload entropy as a feature for device identification since it helps to classify devices more effectively. The framework comprises of two steps – manufacturer identification and device identification.

- **Fingerprinting Analog IoT Sensors for secret-free Authentication –** This paper emphasized on the use of Public Physically Unclonable Features (PPUF's) and other secret-free methods to authenticate devices. The study employed the challenge-response method to identify anomalies in the network under the assumption that the temperature variation in the environment of the sensors does not exceed a delta of 0.5 degrees. The receiver is bootstrapped with the matched responses. The remote-party sends a challenge to which the local-party must produce a valid response. No two devices must have an identical set of responses to all the available challenges. The challenge for IoT devices is in the form of an oscillating voltage of a fixed pattern/ frequency. The response includes specific parameters if the output waveforms.

- **Identification of Devices based on Hardware and Software features –** This is the most recent and sole study that focuses on developing device fingerprints for smart devices using Hardware and Software fingerprint features. It demonstrates the use of an integrated model of Hardware and Software model to generate very high classification accuracy. Hardware level features provide a stable device signature, but are susceptible to various signal transmission environments. Software features are more intricate and less directly linked to the device. The multi-level authentication method leverages the Time and Frequency domains as hardware features and performs – Multimodal feature extraction, feature fusion and Hardware attribute information identification. Software features are extracted by network traffic analysis using behavioral fingerprinting methods. This involves a series of three steps – behavioral information modeling, sequence information modeling and Software attributes information identification.

  The hardware feature extraction process used the GRU (Gated Recurrent Unit Model) for time-domain analysis and the CNN Model for Frequency Domain analysis. These features are combined by the process of multimodal fusion. For the software feature extraction, Entropy is used as an important feature and byte locations having very low entropy are considered stable and selected as key-blocks. The key-blocks are concentrated at the front and back of the feature set.

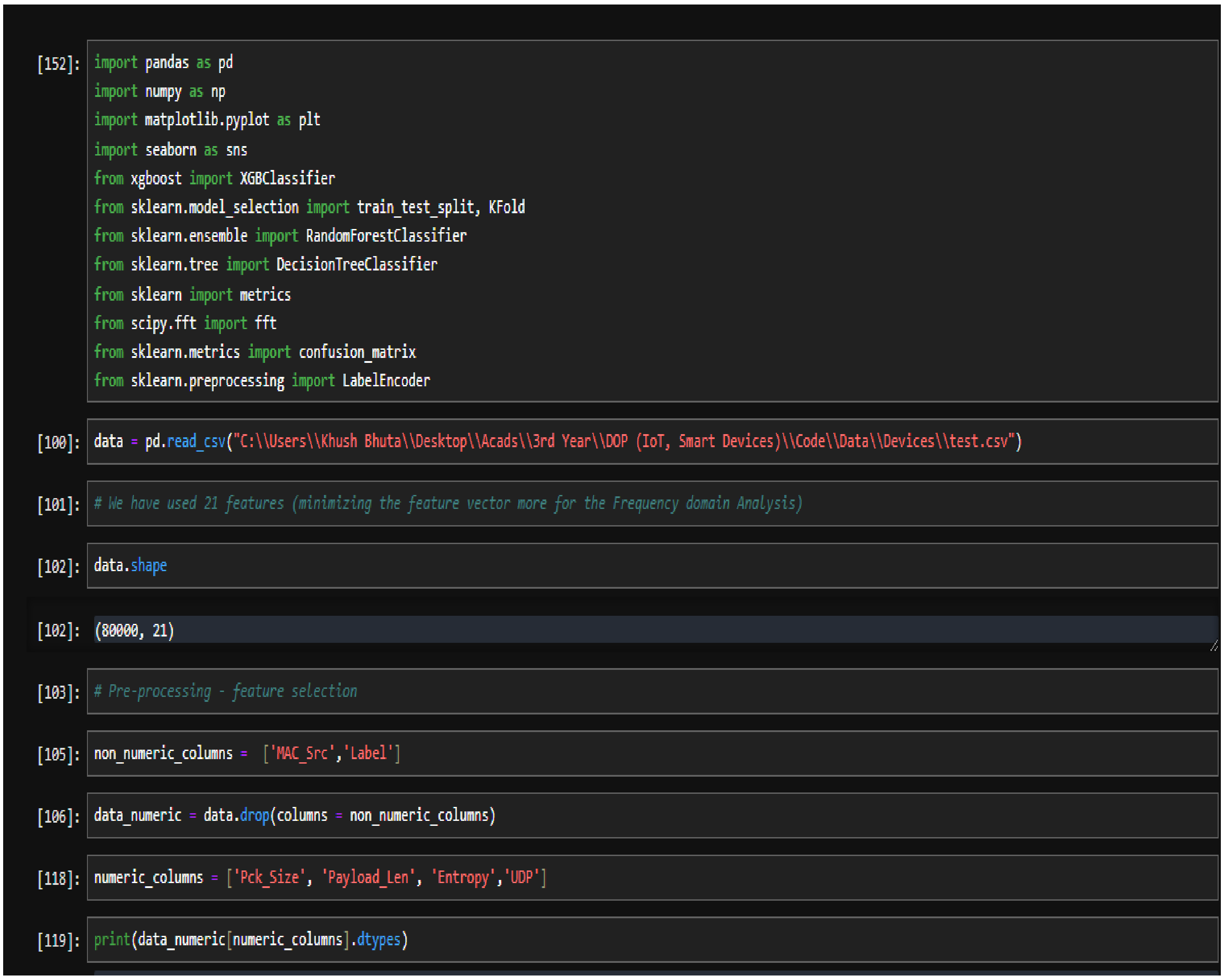
Some other papers that I have performed literature review on are:

1. **Locality-sensitive network traffic fingerprinting for Device Level Identification.**
2. **Sliding Window based Temporally Aware Network Traffic Analyzer for IoT Device Fingerprinting.**
3. **Context-Aware Behavioral Fingerprinting of IoT Devices via Network Traffic Analysis.**
4. **DEFT: A Distributed IoT Fingerprinting Technique.**

2. FFT on Hardware Features (Pre-Midsem)

The data used for this model was obtained from the test.csv file which contained 10,000 random packets captured from each sensor in a random order to accurately train the model. The packets were randomly picked from a very huge dataset of 6,00,000 packets for each sensor.

Using the features and network-traffic data extracted in the form of PCAP files by our seniors using WireShark, I was able to perform Frequency – Domain Analysis on selected network-level features by applying Fast-fourier transforms. Using the information provided in some hardware-level feature papers that were reviewed by my teammates, we narrowed down to 4 features to perform the FFT – Packet-size, payload-length, Entropy, and one of the network protocols – UDP/ TCP. Since the data contained the MAC_addresses and the Labels for each of the sensor, I deleted them from the feature set, to avoid overfitting.

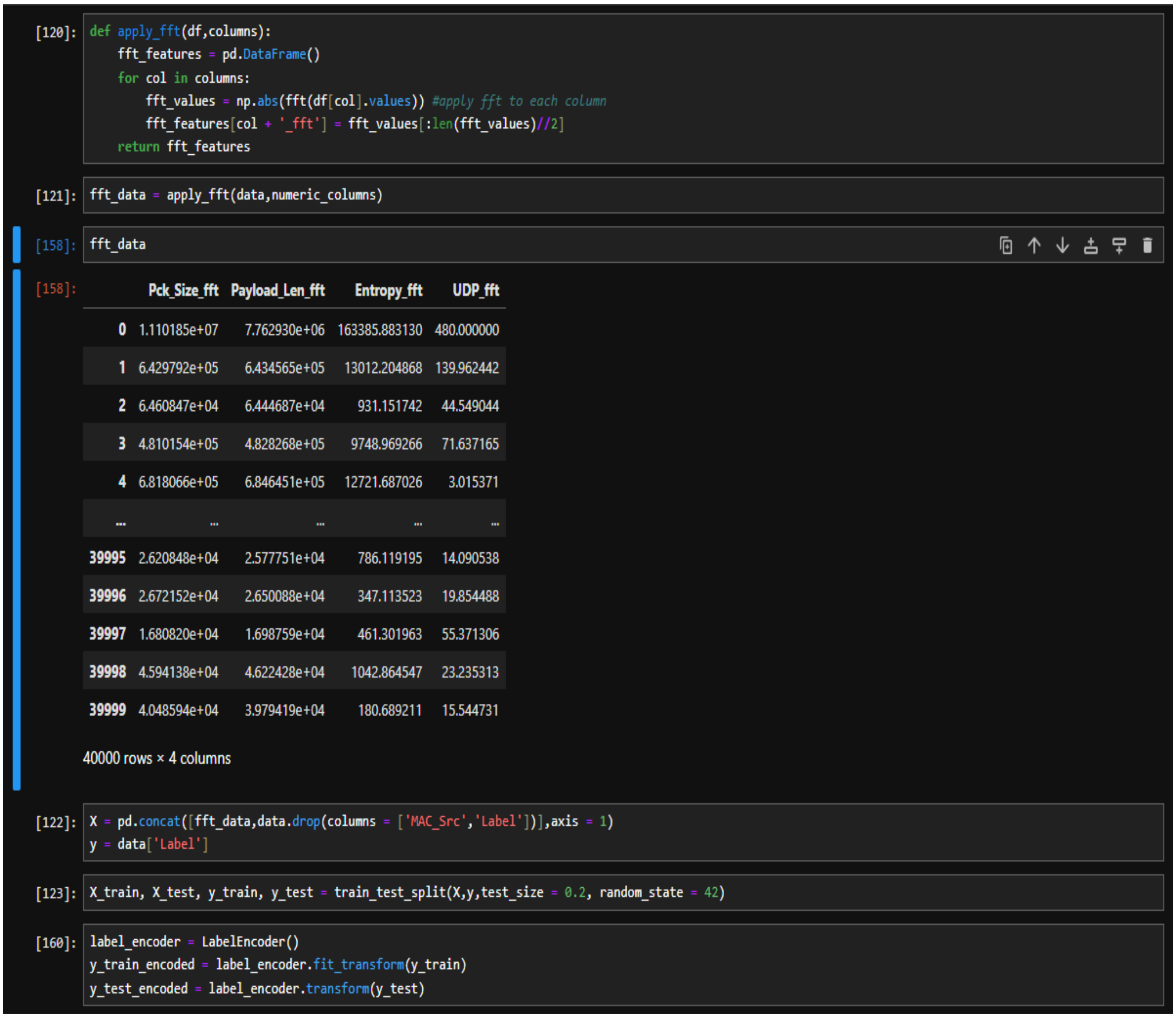


```
[152]: import pandas as pd
       import numpy as np
       import matplotlib.pyplot as plt
       import seaborn as sns
       from xgboost import XGBClassifier
       from sklearn.model_selection import train_test_split, KFold
       from sklearn.ensemble import RandomForestClassifier
       from sklearn.tree import DecisionTreeClassifier
       from sklearn import metrics
       from scipy.fft import fft
       from sklearn.metrics import confusion_matrix
       from sklearn.preprocessing import LabelEncoder

[100]: data = pd.read_csv("C:\\Users\\Khush Bhuta\\Desktop\\Acads\\3rd Year\\DOP (IoT, Smart Devices)\\Code\\Data\\Devices\\test.csv")

[101]: # We have used 21 features (minimizing the feature vector more for the Frequency domain Analysis)

[102]: data.shape

[102]: (80000, 21)

[103]: # Pre-processing - feature selection

[105]: non_numeric_columns =  ['MAC_Src','Label']

[106]: data_numeric = data.drop(columns = non_numeric_columns)

[118]: numeric_columns = ['Pck_Size', 'Payload_Len', 'Entropy','UDP']

[119]: print(data_numeric[numeric_columns].dtypes)
```

Using the scipy package, I imported the fft (fast-fourier transformation) library. Using this, I performed the operation on each of the 4 selected features. I created a function named apply_fft which created a new dataframe called fft_data, and I proceeded the analysis on that dataframe.

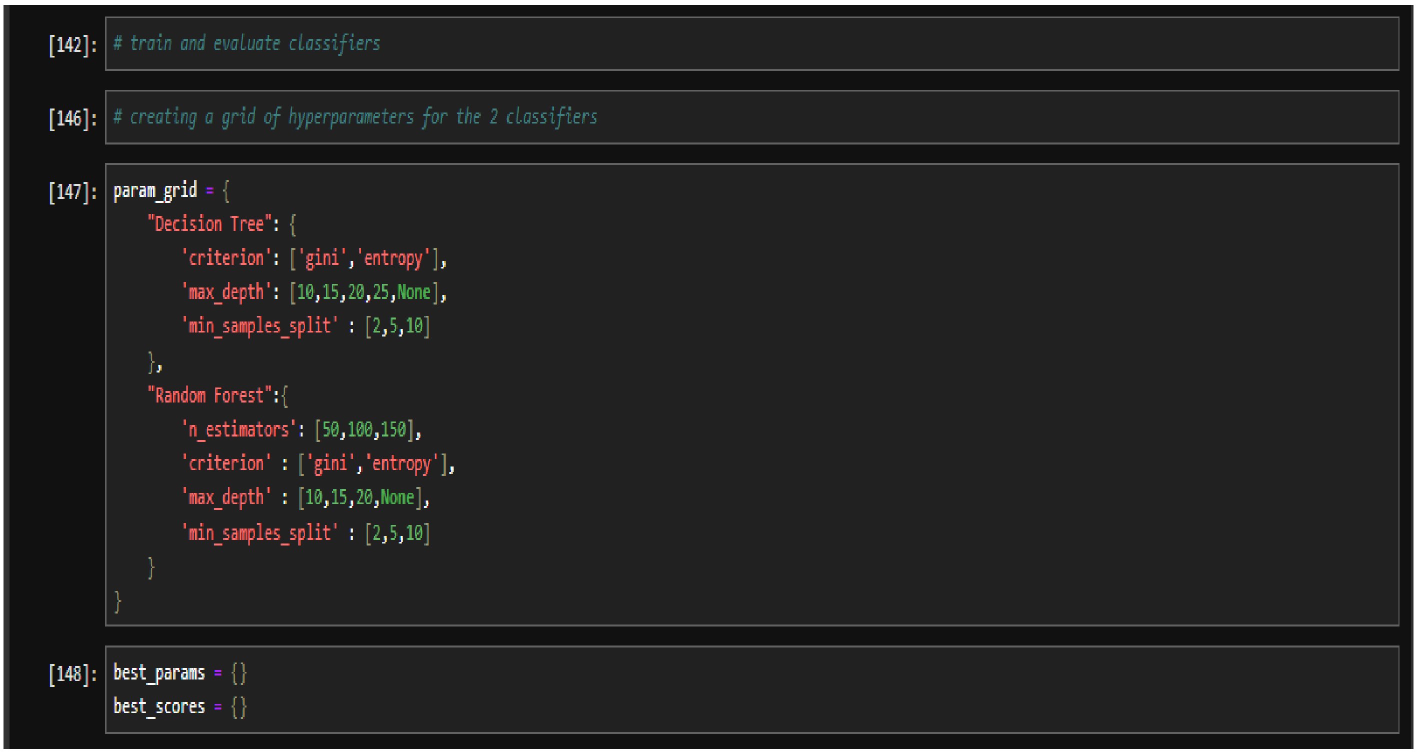
The next step was to create a train-test split to train the Machine Learning model. I used a train-test split of 20% (testing) and 80% (training) data. This is considered optimal to avoid overfitting/ underfitting and this was the same split used by our seniors for training their classifiers.

Lastly, I have defined a lable-encoder to ensure that I can classify the sensors correctly according to their MAC addresses on deriving the results for the same.



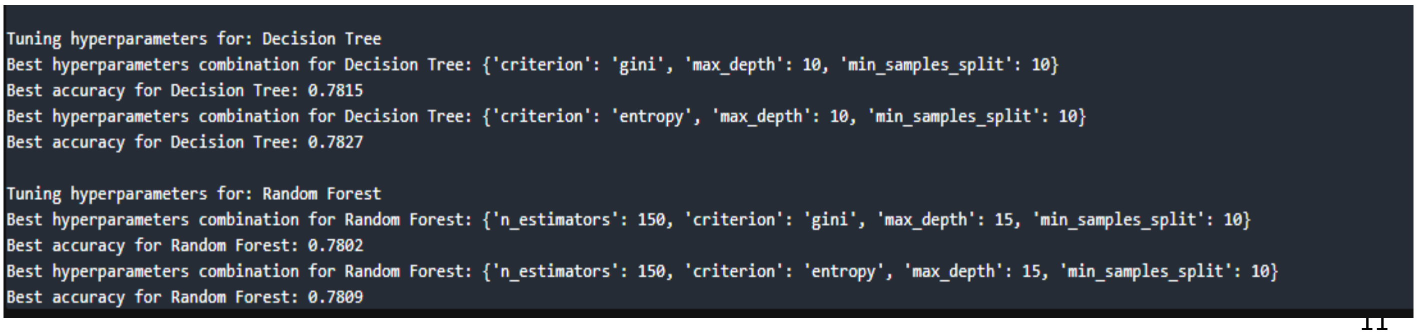
```
[120]: def apply_fft(df,columns):
           fft_features = pd.DataFrame()
           for col in columns:
               fft_values = np.abs(fft(df[col].values)) #apply fft to each column
               fft_features[col + '_fft'] = fft_values[:len(fft_values)//2]
           return fft_features

[121]: fft_data = apply_fft(data,numeric_columns)

[158]: fft_data

[158]:
        Pck_Size_fft  Payload_Len_fft    Entropy_fft     UDP_fft
    0   1.110185e+07     7.762930e+06  163385.883130  480.000000
    1   6.429792e+05     6.434565e+05   13012.204868  139.962442
    2   6.460847e+04     6.444687e+04     931.151742   44.549044
    3   4.810154e+05     4.828268e+05    9748.969266   71.637165
    4   6.818066e+05     6.846451e+05   12721.687026    3.015371
  ...            ...              ...            ...         ...
39995   2.620848e+04     2.577751e+04     786.119195   14.090538
39996   2.672152e+04     2.650088e+04     347.113523   19.854488
39997   1.680820e+04     1.698759e+04     461.301963   55.371306
39998   4.594138e+04     4.622428e+04    1042.864547   23.235313
39999   4.048594e+04     3.979419e+04     180.689211   15.544731

40000 rows × 4 columns

[122]: X = pd.concat([fft_data,data.drop(columns = ['MAC_Src','Label'])],axis = 1)
       y = data['Label']

[123]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 42)

[160]: label_encoder = LabelEncoder()
       y_train_encoded = label_encoder.fit_transform(y_train)
       y_test_encoded = label_encoder.transform(y_test)
```

The next step in the creation of the model was to optimize the hyperparameters. Based on the previous experiments by our seniors and the literature review done by me, I observed that the Random Forest and Decision Tree classifiers provided the best classification accuracy. Since the a Convolutional Neural Network model would require a lot of epochs and computation, I was not able to perform it on my system. I aim to perform it post-midsems. For optimizing the hyperparameters, I created a parameter grid for the 2 classifiers that I used. I also created 2 empty dictionaries to store the parameters obtaining the best results.

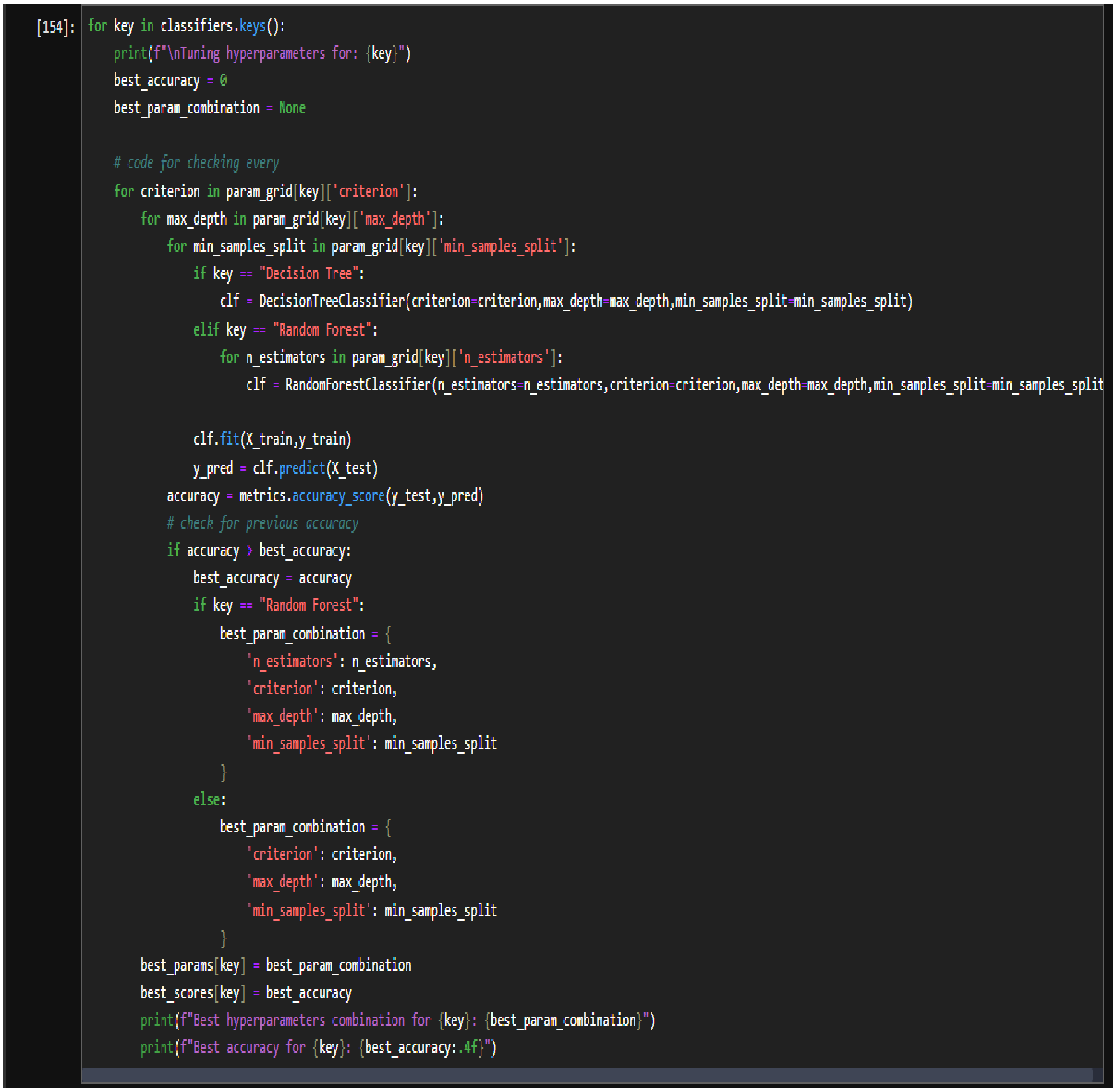


```
[142]: # train and evaluate classifiers

[146]: # creating a grid of hyperparameters for the 2 classifiers

[147]: param_grid = {
           "Decision Tree": {
               'criterion': ['gini','entropy'],
               'max_depth': [10,15,20,25,None],
               'min_samples_split' : [2,5,10]
           },
           "Random Forest":{
               'n_estimators': [50,100,150],
               'criterion' : ['gini','entropy'],
               'max_depth' : [10,15,20,None],
               'min_samples_split' : [2,5,10]
           }
       }

[148]: best_params = {}
       best_scores = {}
```

For tuning the hyperparameters, I ran loops on each of the arguments of the classifiers and fit them to the model. I proceeded to yield the accuracy of classification and stored the hyperparameters with maximum accuracy in the empty dictionaries created earlier. These hyperparameters yielded accuracy of upto 78.27% in the Decision Tree classifier and 78.09% in the Random Forest classifier.



```
Tuning hyperparameters for: Decision Tree
Best hyperparameters combination for Decision Tree: {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 10}
Best accuracy for Decision Tree: 0.7815
Best hyperparameters combination for Decision Tree: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 10}
Best accuracy for Decision Tree: 0.7827

Tuning hyperparameters for: Random Forest
Best hyperparameters combination for Random Forest: {'n_estimators': 150, 'criterion': 'gini', 'max_depth': 15, 'min_samples_split': 10}
Best accuracy for Random Forest: 0.7802
Best hyperparameters combination for Random Forest: {'n_estimators': 150, 'criterion': 'entropy', 'max_depth': 15, 'min_samples_split': 10}
Best accuracy for Random Forest: 0.7809
```

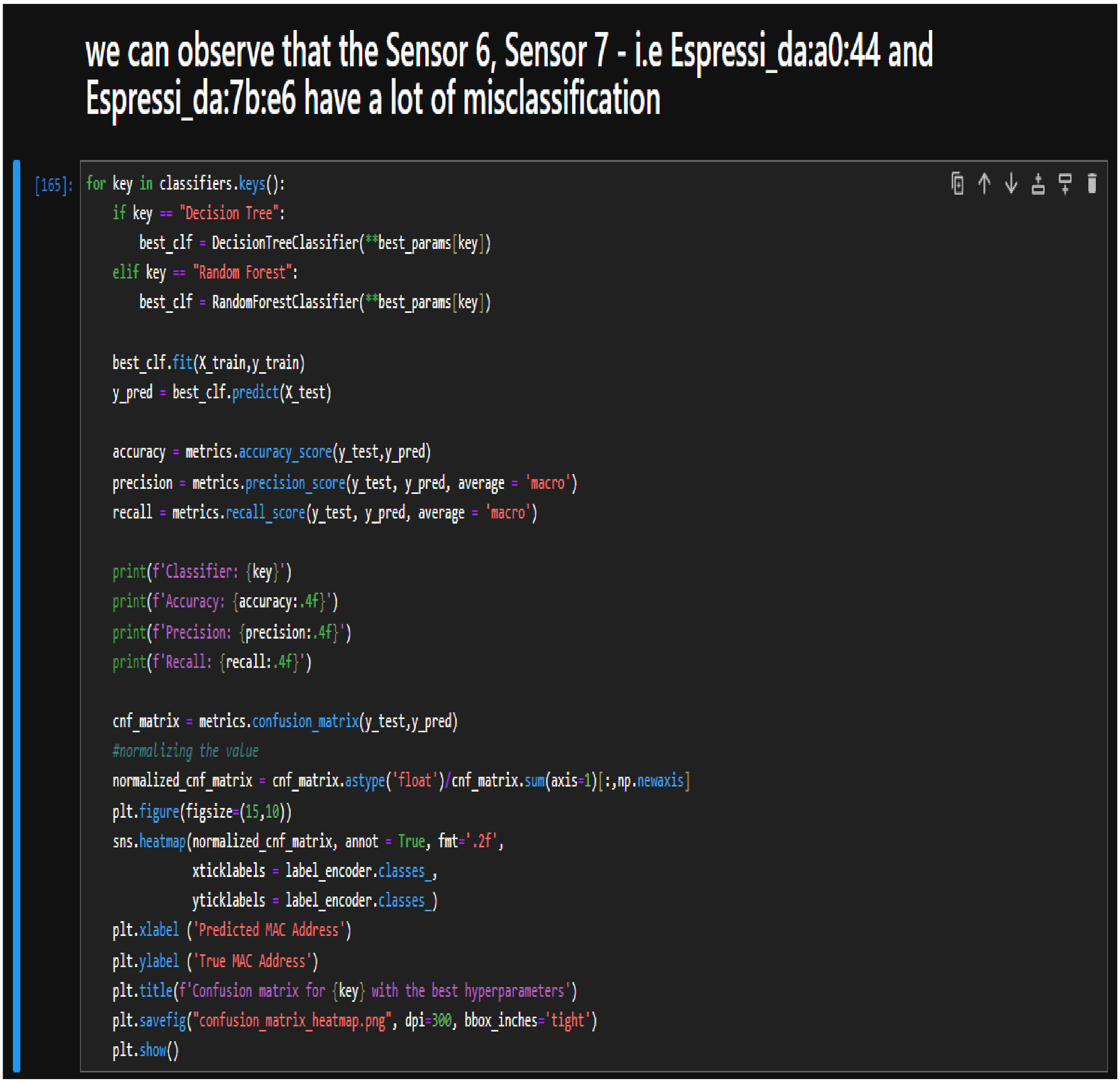


```
[154]: for key in classifiers.keys():
           print(f"\nTuning hyperparameters for: {key}")
           best_accuracy = 0
           best_param_combination = None

           # code for checking every
           for criterion in param_grid[key]['criterion']:
               for max_depth in param_grid[key]['max_depth']:
                   for min_samples_split in param_grid[key]['min_samples_split']:
                       if key == "Decision Tree":
                           clf = DecisionTreeClassifier(criterion=criterion,max_depth=max_depth,min_samples_split=min_samples_split)
                       elif key == "Random Forest":
                           for n_estimators in param_grid[key]['n_estimators']:
                               clf = RandomForestClassifier(n_estimators=n_estimators,criterion=criterion,max_depth=max_depth,min_samples_split=min_samples_split

                       clf.fit(X_train,y_train)
                       y_pred = clf.predict(X_test)
                   accuracy = metrics.accuracy_score(y_test,y_pred)
                   # check for previous accuracy
                   if accuracy > best_accuracy:
                       best_accuracy = accuracy
                       if key == "Random Forest":
                           best_param_combination = {
                               'n_estimators': n_estimators,
                               'criterion': criterion,
                               'max_depth': max_depth,
                               'min_samples_split': min_samples_split
                           }
                       else:
                           best_param_combination = {
                               'criterion': criterion,
                               'max_depth': max_depth,
                               'min_samples_split': min_samples_split
                           }
               best_params[key] = best_param_combination
               best_scores[key] = best_accuracy
               print(f"Best hyperparameters combination for {key}: {best_param_combination}")
               print(f"Best accuracy for {key}: {best_accuracy:.4f}")
```

The above is the code for optimizing the hyperparameters for the best accuracy.

In the next step, I defined the 2 classifiers with the optimized parameters and got results for 4 important metrics to evaluate the performance – Accuracy, Precision, Recall and F1-Score. The formula for each is given as follows:

- Accuracy = (TP + TN) / (TP + TN + FP + FN)
- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1 – score = (Precision*Recall) / (Precision + Recall)

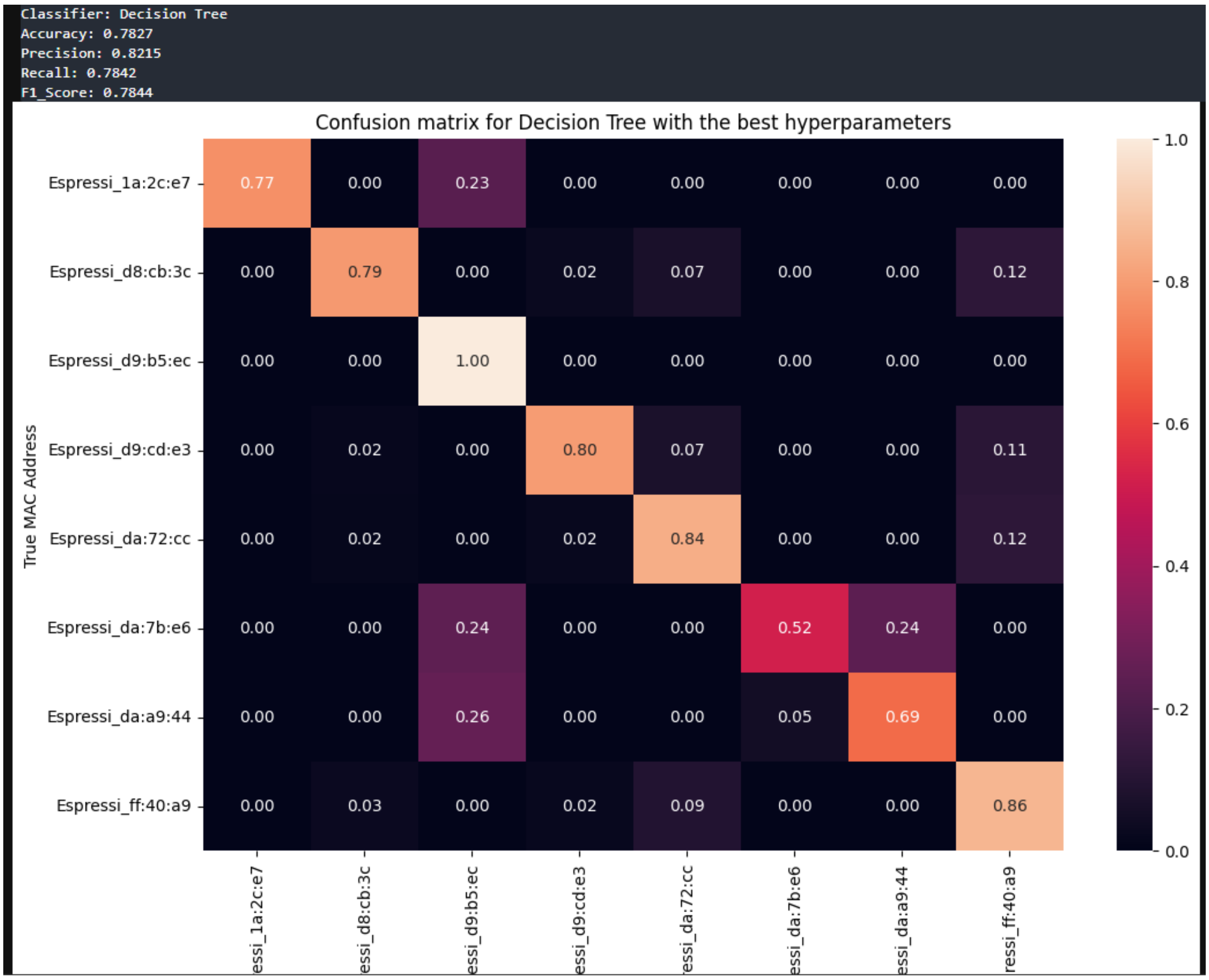


we can observe that the Sensor 6, Sensor 7 - i.e Espressi_da:a0:44 and Espressi_da:7b:e6 have a lot of misclassification

[165]:
```python
for key in classifiers.keys():
    if key == "Decision Tree":
        best_clf = DecisionTreeClassifier(**best_params[key])
    elif key == "Random Forest":
        best_clf = RandomForestClassifier(**best_params[key])

    best_clf.fit(X_train,y_train)
    y_pred = best_clf.predict(X_test)

    accuracy = metrics.accuracy_score(y_test,y_pred)
    precision = metrics.precision_score(y_test, y_pred, average = 'macro')
    recall = metrics.recall_score(y_test, y_pred, average = 'macro')

    print(f'Classifier: {key}')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')

    cnf_matrix = metrics.confusion_matrix(y_test,y_pred)
    #normalizing the value
    normalized_cnf_matrix = cnf_matrix.astype('float')/cnf_matrix.sum(axis=1)[:,np.newaxis]
    plt.figure(figsize=(15,10))
    sns.heatmap(normalized_cnf_matrix, annot = True, fmt='.2f',
                xticklabels = label_encoder.classes_,
                yticklabels = label_encoder.classes_)
    plt.xlabel ('Predicted MAC Address')
    plt.ylabel ('True MAC Address')
    plt.title(f'Confusion matrix for {key} with the best hyperparameters')
    plt.savefig("confusion_matrix_heatmap.png", dpi=300, bbox_inches='tight')
    plt.show()
```

The above code snippet gives us the confusion matrix and heatmap for each of the classifiers. To recognize the device, we have used the MAC Addresses of each sensor (NodeMCU here). Since the data contains packets from 8 separate NodeMCUs we receive an 8x8 output heatmap. The classification results are depicted in the next images.

Decision Tree Classifier Heatmap:

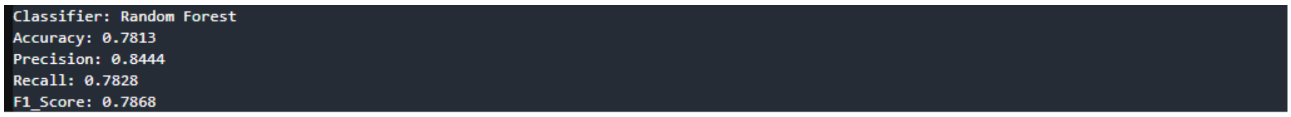


```
Classifier: Decision Tree
Accuracy: 0.7827
Precision: 0.8215
Recall: 0.7842
F1_Score: 0.7844
```

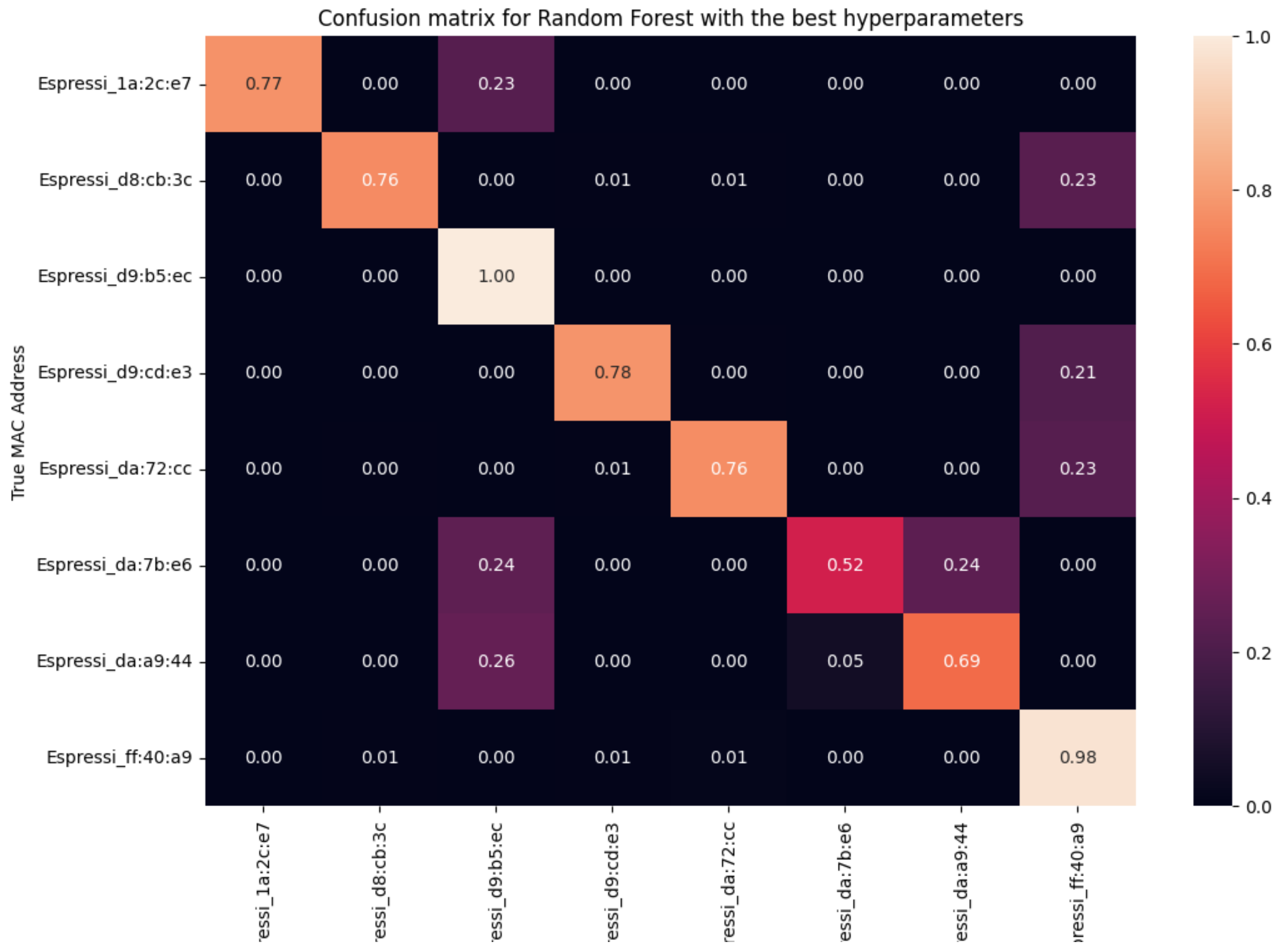
Confusion matrix for Decision Tree with the best hyperparameters

True MAC Address	essi_1a:2c:e7	essi_d8:cb:3c	essi_d9:b5:ec	essi_d9:cd:e3	essi_da:72:cc	essi_da:7b:e6	essi_da:a9:44	ressi_ff:40:a9
Espressi_1a:2c:e7	0.77	0.00	0.23	0.00	0.00	0.00	0.00	0.00
Espressi_d8:cb:3c	0.00	0.79	0.00	0.02	0.07	0.00	0.00	0.12
Espressi_d9:b5:ec	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
Espressi_d9:cd:e3	0.00	0.02	0.00	0.80	0.07	0.00	0.00	0.11
Espressi_da:72:cc	0.00	0.02	0.00	0.02	0.84	0.00	0.00	0.12
Espressi_da:7b:e6	0.00	0.00	0.24	0.00	0.00	0.52	0.24	0.00
Espressi_da:a9:44	0.00	0.00	0.26	0.00	0.00	0.05	0.69	0.00
Espressi_ff:40:a9	0.00	0.03	0.00	0.02	0.09	0.00	0.00	0.86

Random Forest Classifier Heatmap:



```
Classifier: Random Forest
Accuracy: 0.7813
Precision: 0.8444
Recall: 0.7828
F1_Score: 0.7868
```



	essi_1a:2c:e7	essi_d8:cb:3c	essi_d9:b5:ec	essi_d9:cd:e3	essi_da:72:cc	essi_da:7b:e6	essi_da:a9:44	ressi_ff:40:a9
Espressi_1a:2c:e7	0.77	0.00	0.23	0.00	0.00	0.00	0.00	0.00
Espressi_d8:cb:3c	0.00	0.76	0.00	0.01	0.01	0.00	0.00	0.23
Espressi_d9:b5:ec	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
Espressi_d9:cd:e3	0.00	0.00	0.00	0.78	0.00	0.00	0.00	0.21
Espressi_da:72:cc	0.00	0.00	0.00	0.01	0.76	0.00	0.00	0.23
Espressi_da:7b:e6	0.00	0.00	0.24	0.00	0.00	0.52	0.24	0.00
Espressi_da:a9:44	0.00	0.00	0.26	0.00	0.00	0.05	0.69	0.00
Espressi_ff:40:a9	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.98

We observed that for both the classifiers, the accuracy was close to 78.1% and the precision was close to 84%. However there was strong misclassification between 3 devices – D2,D5,D6. This indicates that the sensor data received by all the 3 sensors exhibits very similar physical features. The problem we faced was in mapping these sensors with their respective MAC addresses. The sensor D6 having MAC Address –

Espressi_da:7b:e6 is the NodeMCU containing the Smoke sensor and Sound sensor. We are yet to find the MAC addresses for the rest of the NodeMCUs and classify them accordingly.

Progress Made Post-Midsem:

1. Improved the performance of classifiers trained before midsem significantly by using the 'stacking' technique to increase the feature vector size.
2. Performed FFT (Fast Fourier Transformation) on numeric PCAP data to capture frequency-domain patterns. This decomposition into sinusoidal components can help in capturing periodic patterns and help Machine Learning models to improve their performance. It helps in capturing cyclic behavior, highlight deviations (anomalous spikes) and focus only on dominant frequency components.
3. Trained Machine Learning classifiers to accurately classify IoT Devices on Multiple open-source Datasets.
4. Binary and Multiclass Classification on UNSW-NB15 Anomaly Detection Dataset.
5. Performed Multiclass Classification on the UNSW Traffic Traces Dataset.
6. Researched and learnt about Neural Networks and trained a CNN Model for classification but obtained sub-optimal results.

Datasets Used:

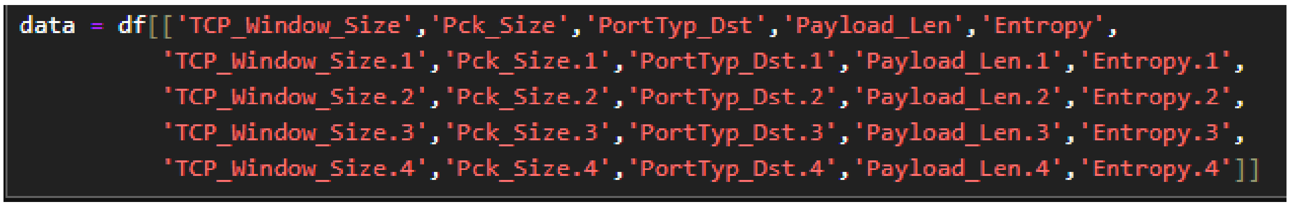
- UNSW NB-15 Network Intrustion Dataset (Link)
- UNSW IoT Analytics (Traffic Traces) Dataset (Link)

3. *Improving previous Model on test.csv*

The test.csv dataset consisted of 80,000 packets from 8 separate devices (10k packets from each NodeMCU).

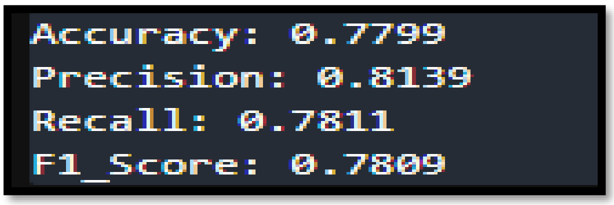
It has been samples and was used to provide a benchmark for the results of the other open-source datasets. However, this dataset lacks temporal features (due to random sampling), and hence it is difficult to perform time-domain analysis on it.

To improve the performance however, I used the stacking approach to stack the fft_transformed features onto each other for the same device, and thus reducing the number of rows and increasing the number of columns. 5 consecutive entires were stacked (since they were sorted according to their Labels initially) creating a new dataset test_5.csv with 16,000 rows. This dataset showed a great surge in the performance metrics. The image shown below contains the feature vector set used for training the model.

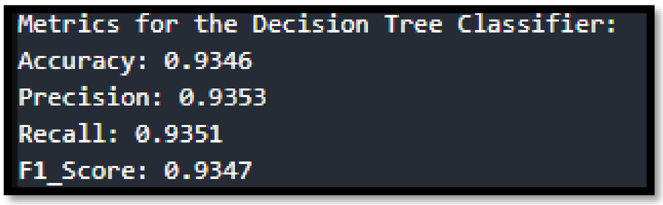


```
data = df[['TCP_Window_Size','Pck_Size','PortTyp_Dst','Payload_Len','Entropy',
          'TCP_Window_Size.1','Pck_Size.1','PortTyp_Dst.1','Payload_Len.1','Entropy.1',
          'TCP_Window_Size.2','Pck_Size.2','PortTyp_Dst.2','Payload_Len.2','Entropy.2',
          'TCP_Window_Size.3','Pck_Size.3','PortTyp_Dst.3','Payload_Len.3','Entropy.3',
          'TCP_Window_Size.4','Pck_Size.4','PortTyp_Dst.4','Payload_Len.4','Entropy.4']]
```

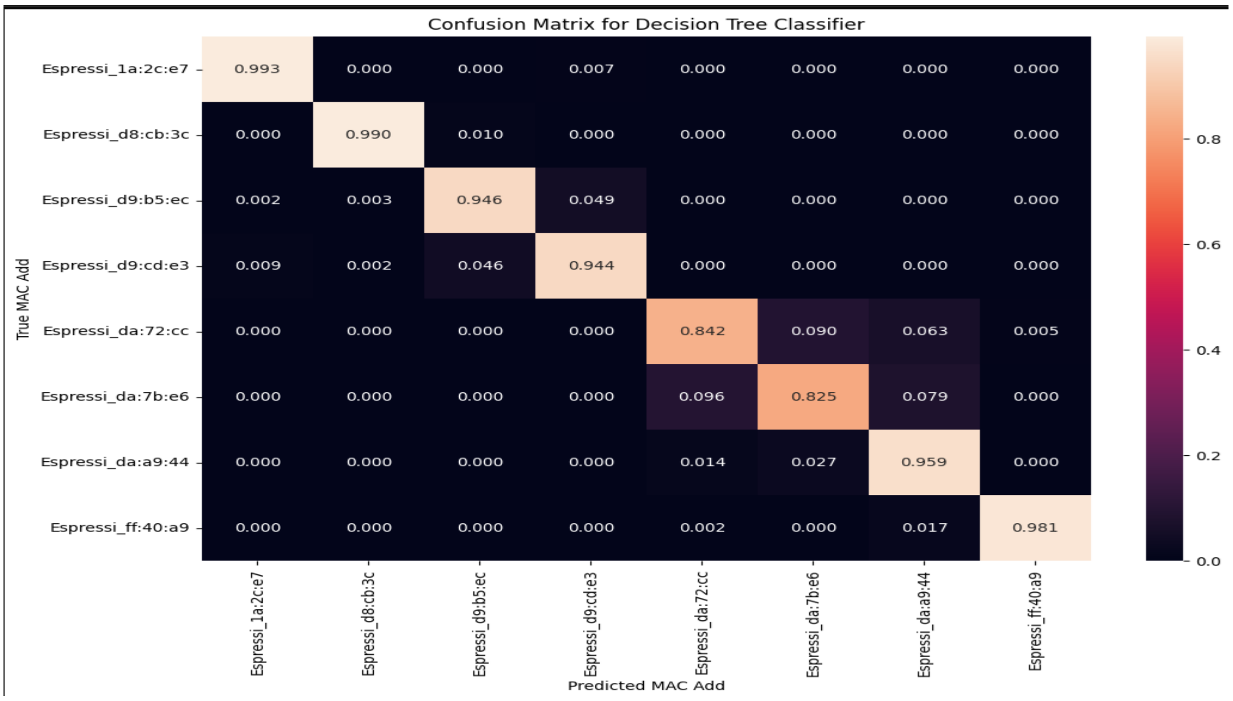
Decision Tree Classifier: (Left image shows the initial metrics)



```
Accuracy: 0.7799
Precision: 0.8139
Recall: 0.7811
F1_Score: 0.7809
```



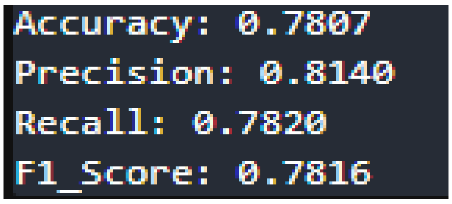
```
Metrics for the Decision Tree Classifier:
Accuracy: 0.9346
Precision: 0.9353
Recall: 0.9351
F1_Score: 0.9347
```



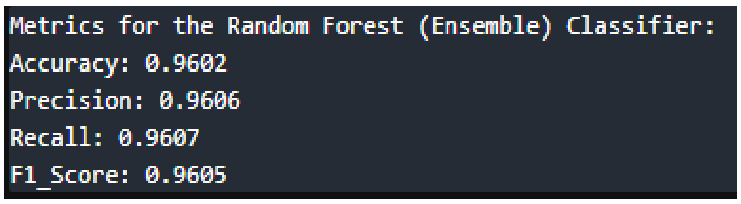
Confusion Matrix for Decision Tree Classifier

True MAC Add \ Predicted MAC Add	Espressi_1a:2c:e7	Espressi_d8:cb:3c	Espressi_d9:b5:ec	Espressi_d9:cd:e3	Espressi_da:72:cc	Espressi_da:7b:e6	Espressi_da:a9:44	Espressi_ff:40:a9
Espressi_1a:2c:e7	0.993	0.000	0.000	0.007	0.000	0.000	0.000	0.000
Espressi_d8:cb:3c	0.000	0.990	0.010	0.000	0.000	0.000	0.000	0.000
Espressi_d9:b5:ec	0.002	0.003	0.946	0.049	0.000	0.000	0.000	0.000
Espressi_d9:cd:e3	0.009	0.002	0.046	0.944	0.000	0.000	0.000	0.000
Espressi_da:72:cc	0.000	0.000	0.000	0.000	0.842	0.090	0.063	0.005
Espressi_da:7b:e6	0.000	0.000	0.000	0.000	0.096	0.825	0.079	0.000
Espressi_da:a9:44	0.000	0.000	0.000	0.000	0.014	0.027	0.959	0.000
Espressi_ff:40:a9	0.000	0.000	0.000	0.000	0.002	0.000	0.017	0.981

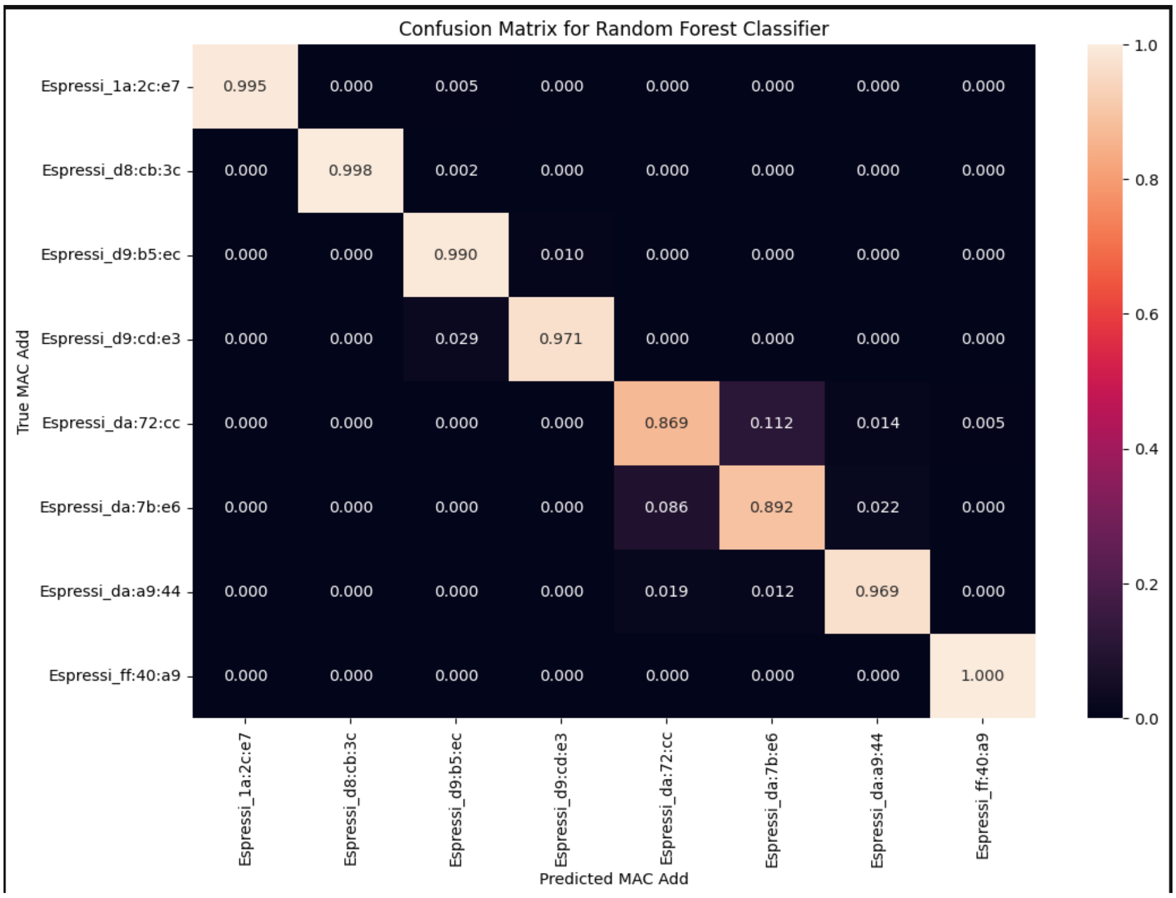
Random Forest Classifier: (Left image shows the initial metrics)



```
Accuracy: 0.7807
Precision: 0.8140
Recall: 0.7820
F1_Score: 0.7816
```



```
Metrics for the Random Forest (Ensemble) Classifier:
Accuracy: 0.9602
Precision: 0.9606
Recall: 0.9607
F1_Score: 0.9605
```



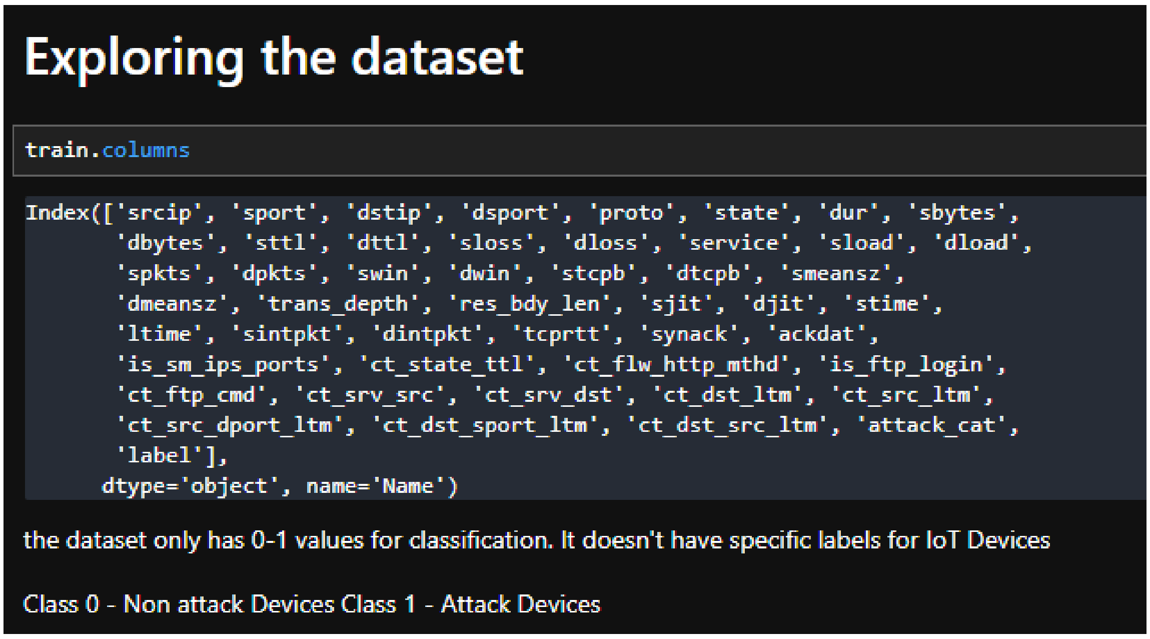
	Espressi_1a:2c:e7	Espressi_d8:cb:3c	Espressi_d9:b5:ec	Espressi_d9:cd:e3	Espressi_da:72:cc	Espressi_da:7b:e6	Espressi_da:a9:44	Espressi_ff:40:a9
Espressi_1a:2c:e7	0.995	0.000	0.005	0.000	0.000	0.000	0.000	0.000
Espressi_d8:cb:3c	0.000	0.998	0.002	0.000	0.000	0.000	0.000	0.000
Espressi_d9:b5:ec	0.000	0.000	0.990	0.010	0.000	0.000	0.000	0.000
Espressi_d9:cd:e3	0.000	0.000	0.029	0.971	0.000	0.000	0.000	0.000
Espressi_da:72:cc	0.000	0.000	0.000	0.000	0.869	0.112	0.014	0.005
Espressi_da:7b:e6	0.000	0.000	0.000	0.000	0.086	0.892	0.022	0.000
Espressi_da:a9:44	0.000	0.000	0.000	0.000	0.019	0.012	0.969	0.000
Espressi_ff:40:a9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

The random forest classifier has a macro avg. accuracy of 96.02% a feature vector of size 25 due to stacking. The high values of Precision, Recall and F1_Score also provide decent results.

4. ML Classifiers on UNSW NB15 – Dataset

Dataset Characteristics:

- The dataset contains 49 PCAP features captured through network traffic.
- I used 11 numeric features out of the available to perform FFT on them to capture frequency-level trends. The ‘attack_cat’ contained the name of the type of the anomaly, and the ‘label’ contained binary 0-1 values for classification where 0 indicated Non-attack devices and 1 indicated attack devices.
- The 11 features are:
  'Dur' - Record total duration
  'Sbytes' - Source to destination transaction bytes
  'Dbytes' - Destination to source transaction bytes
  'Sloss' - Source packets retransmitted or dropped
  'Dloss' - Destination packets retransmitted or dropped
  'Spkts' - Source to destination packet count
  'Dpkts' - Destination to source packet count
  'Swin' - Source TCP window advertisement value
  'Dwin' - Destination TCP window advertisement value
  'Sintpkt' - Source interpacket arrival time (mSec)
  'Dintpkt' - Destination interpacket arrival time (mSec)
- The FFT_features were concatenated with the original numeric values to obtain a bigger feature vector for better classification.



Exploring the dataset

```
train.columns
```

```
Index(['srcip', 'sport', 'dstip', 'dsport', 'proto', 'state', 'dur', 'sbytes',
       'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'service', 'sload', 'dload',
       'spkts', 'dpkts', 'swin', 'dwin', 'stcpb', 'dtcpb', 'smeansz',
       'dmeansz', 'trans_depth', 'res_bdy_len', 'sjit', 'djit', 'stime',
       'ltime', 'sintpkt', 'dintpkt', 'tcprtt', 'synack', 'ackdat',
       'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_login',
       'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ltm',
       'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'attack_cat',
       'label'],
      dtype='object', name='Name')
```

the dataset only has 0-1 values for classification. It doesn't have specific labels for IoT Devices

Class 0 - Non attack Devices Class 1 - Attack Devices

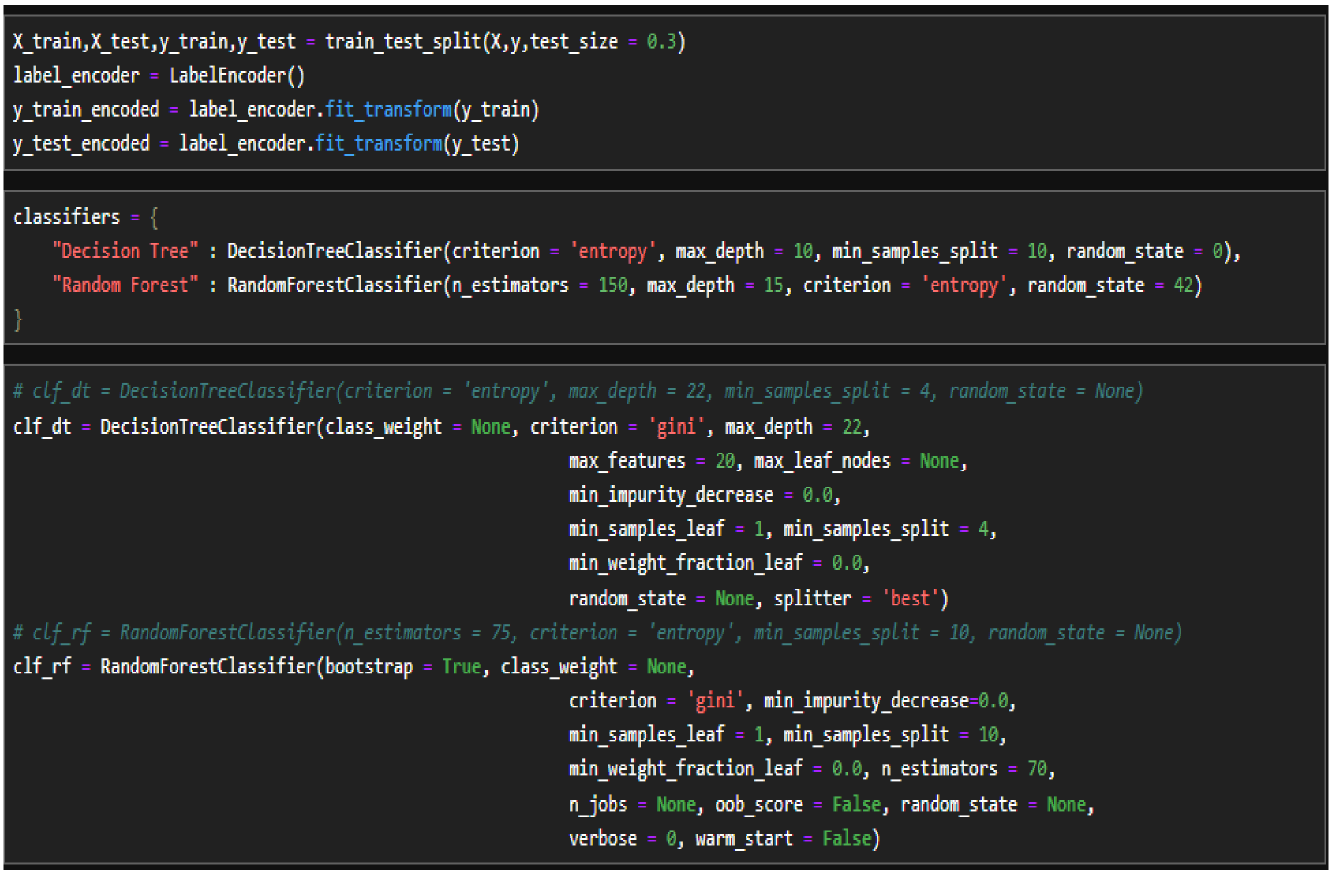
The ‘attack_cat’ feature was necessary for multi-class classification of the anomalies. The dataset was found to be highly skewed with the number of ‘0’ entries very high compared to the ‘1‘ (anomalous entries). The ‘attack_cat’ was renamed as ‘normal’ by filling the null values initially present there.

Systematically combined all the CSV files to create a sufficiently large dataset for training the model.

Decision Tree and Random Forest Classifiers were trained with the given hyperparameters and the output received was as shown.

The model performed very well for Binary Classification, but poorly for Multi-class due to the skewed dataset, which although is a realistic situation makes it difficult for classifiers to identify the attack_cat correctly.

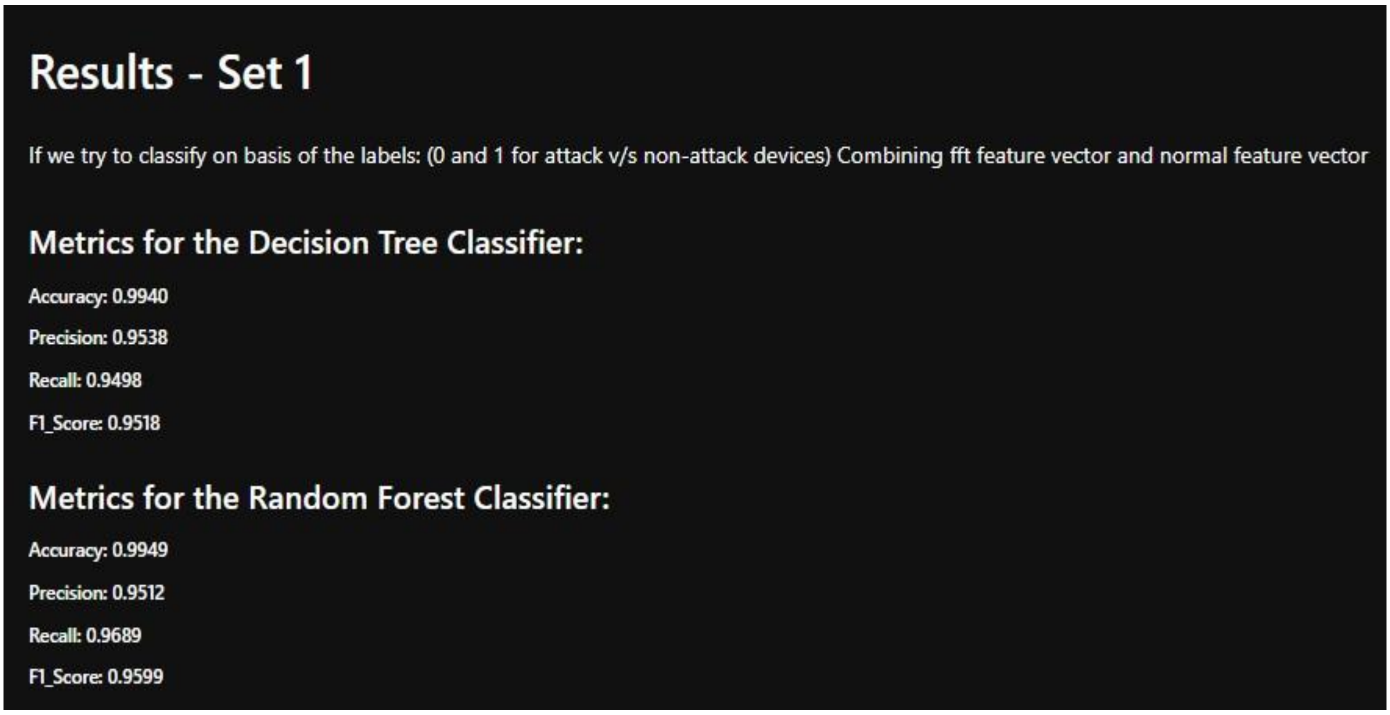
The best hyperparameters were decided using the same looping structure that was used previously for the test.csv dataset.



```python
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3)
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.fit_transform(y_test)
```

```python
classifiers = {
    "Decision Tree" : DecisionTreeClassifier(criterion = 'entropy', max_depth = 10, min_samples_split = 10, random_state = 0),
    "Random Forest" : RandomForestClassifier(n_estimators = 150, max_depth = 15, criterion = 'entropy', random_state = 42)
}
```

```python
# clf_dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = 22, min_samples_split = 4, random_state = None)
clf_dt = DecisionTreeClassifier(class_weight = None, criterion = 'gini', max_depth = 22,
                                                         max_features = 20, max_leaf_nodes = None,
                                                         min_impurity_decrease = 0.0,
                                                         min_samples_leaf = 1, min_samples_split = 4,
                                                         min_weight_fraction_leaf = 0.0,
                                                         random_state = None, splitter = 'best')
# clf_rf = RandomForestClassifier(n_estimators = 75, criterion = 'entropy', min_samples_split = 10, random_state = None)
clf_rf = RandomForestClassifier(bootstrap = True, class_weight = None,
                                                         criterion = 'gini', min_impurity_decrease=0.0,
                                                         min_samples_leaf = 1, min_samples_split = 10,
                                                         min_weight_fraction_leaf = 0.0, n_estimators = 70,
                                                         n_jobs = None, oob_score = False, random_state = None,
                                                         verbose = 0, warm_start = False)
```



Results - Set 1

If we try to classify on basis of the labels: (0 and 1 for attack v/s non-attack devices) Combining fft feature vector and normal feature vector

Metrics for the Decision Tree Classifier:

Accuracy: 0.9940

Precision: 0.9538

Recall: 0.9498

F1_Score: 0.9518

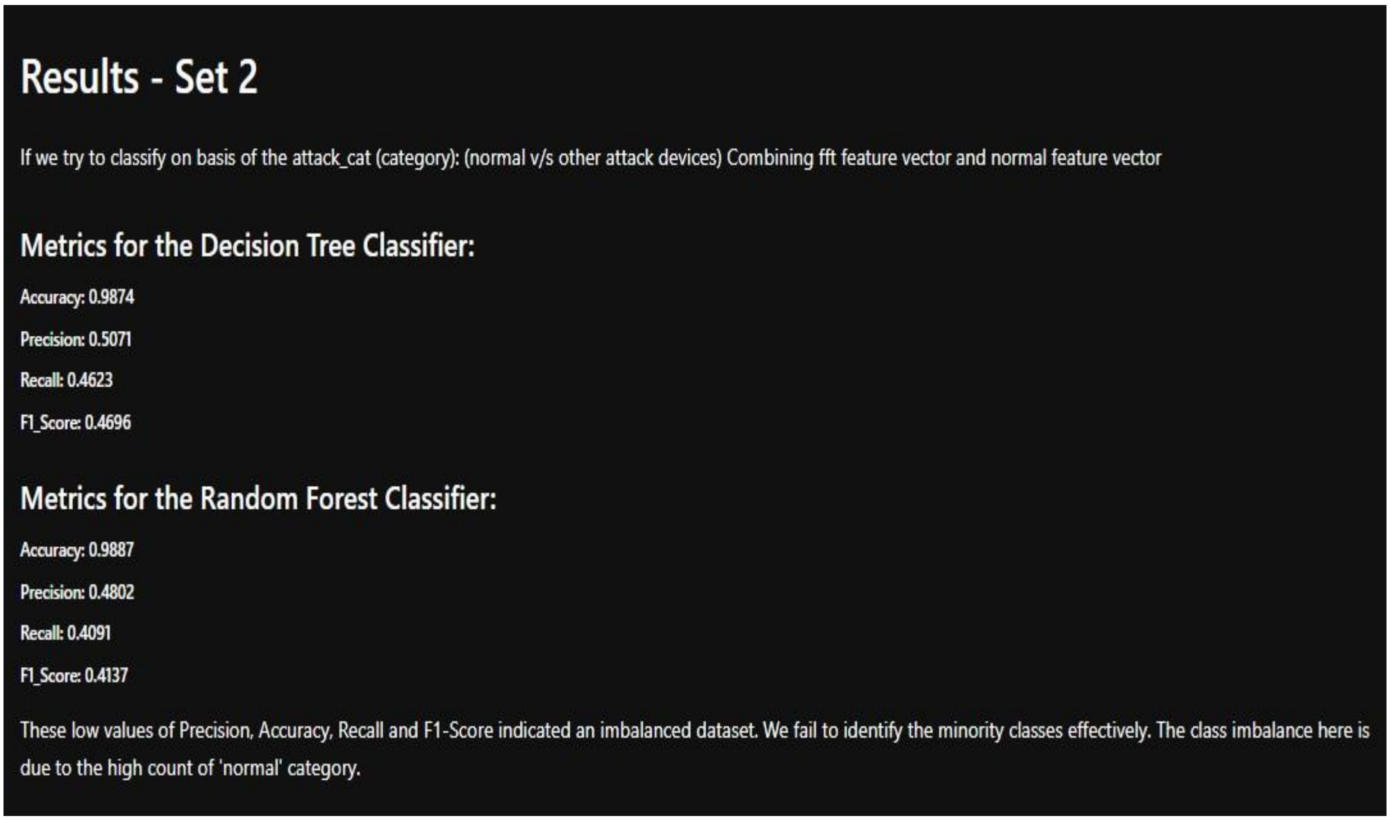
Metrics for the Random Forest Classifier:

Accuracy: 0.9949

Precision: 0.9512

Recall: 0.9689

F1_Score: 0.9599



Results - Set 2

If we try to classify on basis of the attack_cat (category): (normal v/s other attack devices) Combining fft feature vector and normal feature vector

Metrics for the Decision Tree Classifier:

Accuracy: 0.9874

Precision: 0.5071

Recall: 0.4623

F1_Score: 0.4696

Metrics for the Random Forest Classifier:

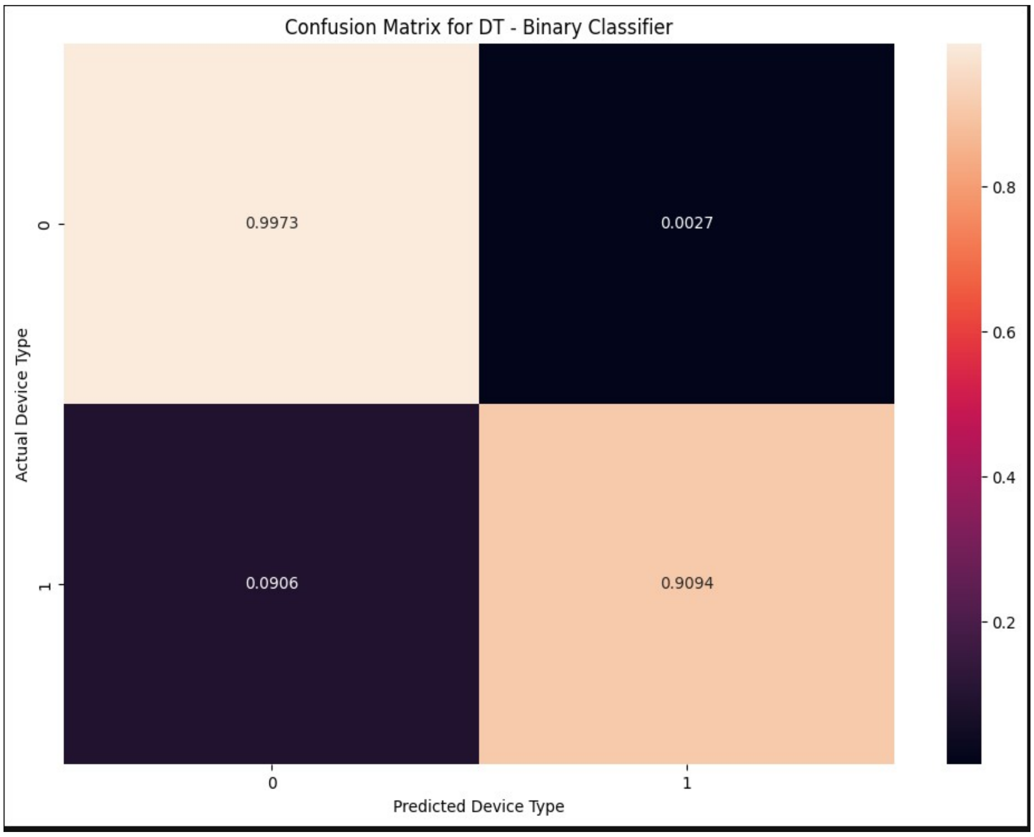
Accuracy: 0.9887

Precision: 0.4802

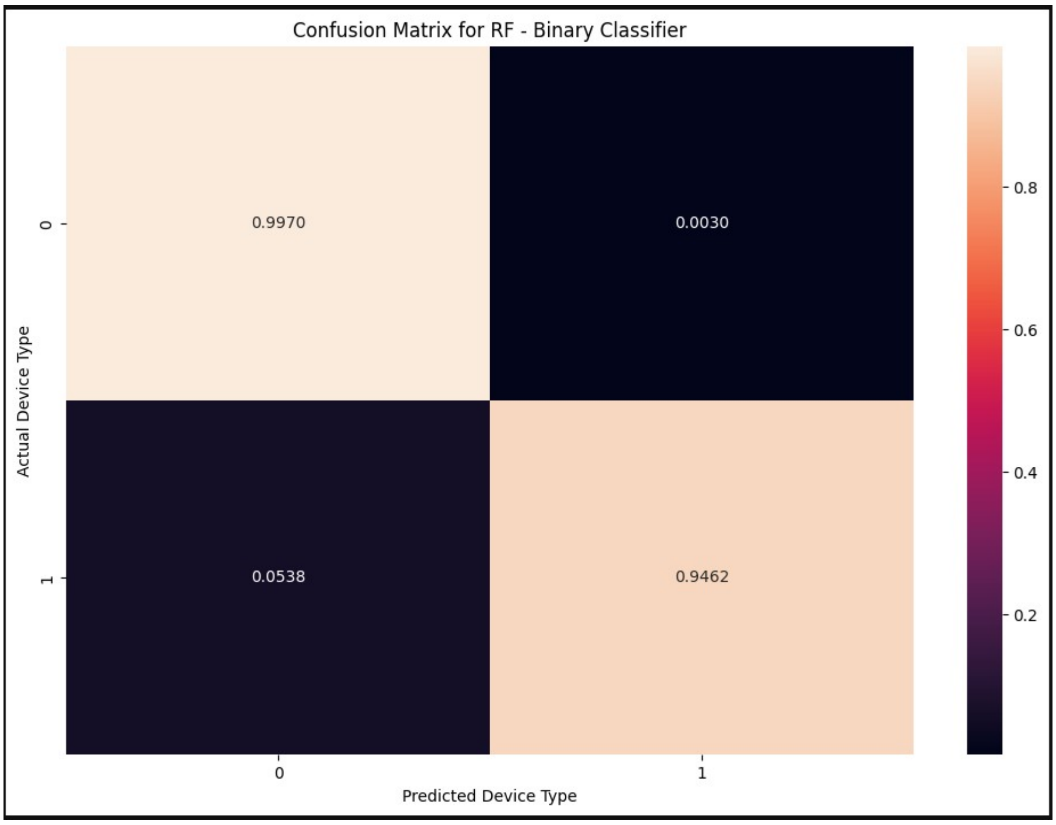
Recall: 0.4091

F1_Score: 0.4137

These low values of Precision, Accuracy, Recall and F1-Score indicated an imbalanced dataset. We fail to identify the minority classes effectively. The class imbalance here is due to the high count of 'normal' category.



Confusion Matrix for DT - Binary Classifier
0.9973
0.0027
0.0906
0.9094
Actual Device Type
Predicted Device Type
0
1
0
1
0.8
0.6
0.4
0.2

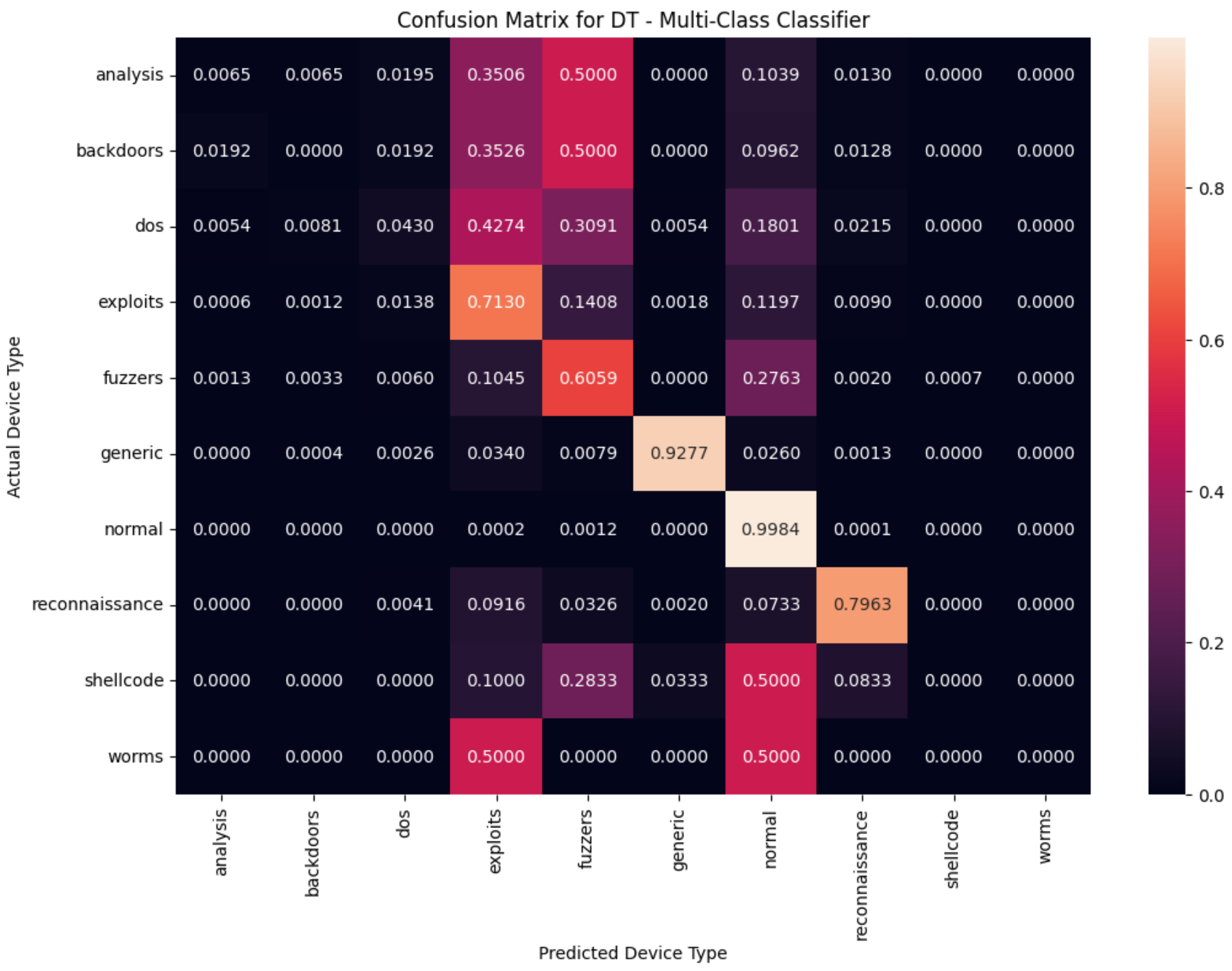


Confusion Matrix for RF - Binary Classifier
0.9970
0.0030
0.0538
0.9462
Actual Device Type
Predicted Device Type
0
1
0
1
0.8
0.6
0.4
0.2

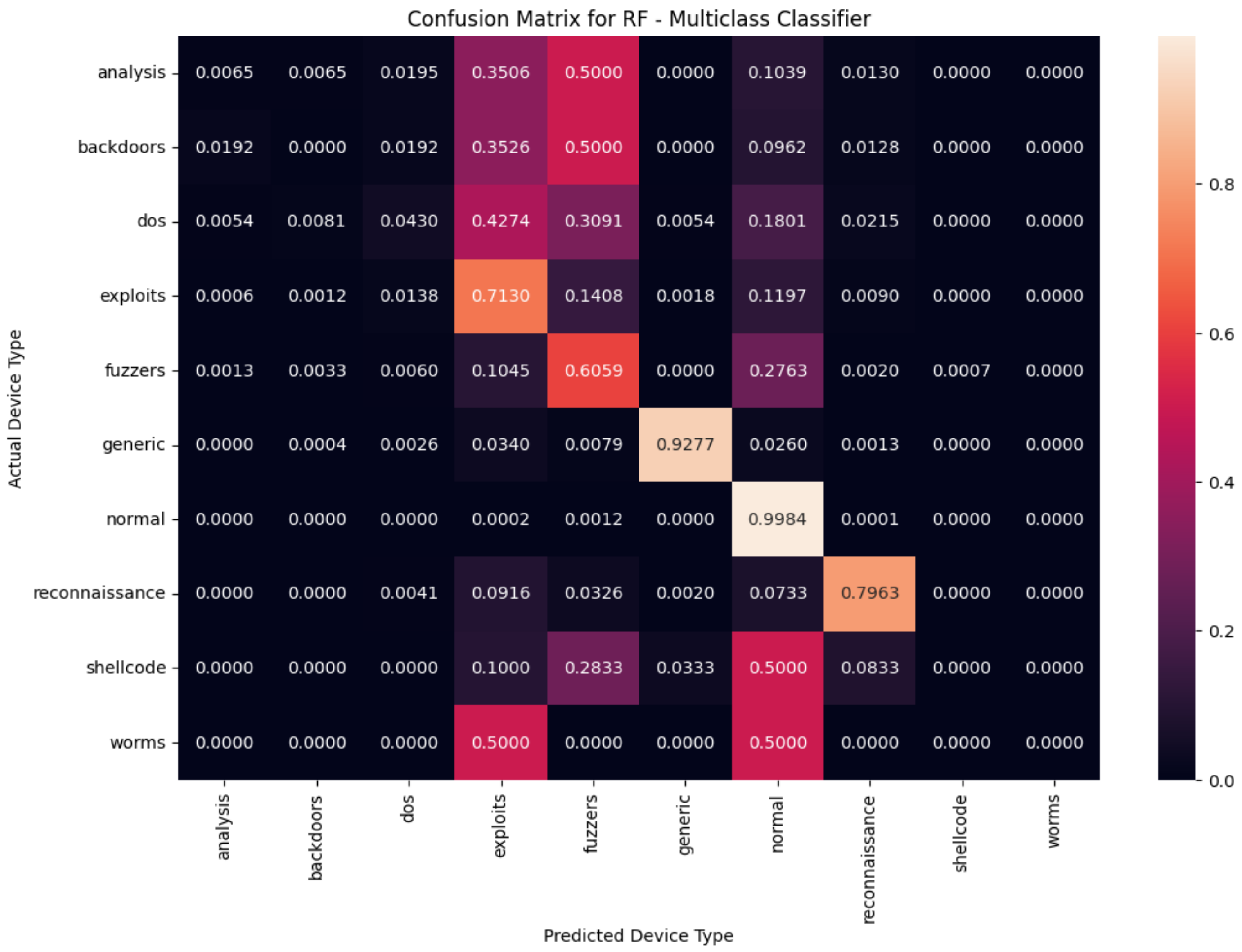
The Random Forest Classifier being an ensemble classifier performed better by classifying 94.62% of the anomalous devices correctly.

However the results obtained for the Multi-class classification were high for the accuracy metrics but very low for Recall, F1-Score and Precision due to misclassification of anomalous devices amongst themselves.

The heatmap and confusion matrix is given below for both the Decision Tree and Random Forest classifiers.



	analysis	backdoors	dos	exploits	fuzzers	generic	normal	reconnaissance	shellcode	worms
analysis	0.0065	0.0065	0.0195	0.3506	0.5000	0.0000	0.1039	0.0130	0.0000	0.0000
backdoors	0.0192	0.0000	0.0192	0.3526	0.5000	0.0000	0.0962	0.0128	0.0000	0.0000
dos	0.0054	0.0081	0.0430	0.4274	0.3091	0.0054	0.1801	0.0215	0.0000	0.0000
exploits	0.0006	0.0012	0.0138	0.7130	0.1408	0.0018	0.1197	0.0090	0.0000	0.0000
fuzzers	0.0013	0.0033	0.0060	0.1045	0.6059	0.0000	0.2763	0.0020	0.0007	0.0000
generic	0.0000	0.0004	0.0026	0.0340	0.0079	0.9277	0.0260	0.0013	0.0000	0.0000
normal	0.0000	0.0000	0.0000	0.0002	0.0012	0.0000	0.9984	0.0001	0.0000	0.0000
reconnaissance	0.0000	0.0000	0.0041	0.0916	0.0326	0.0020	0.0733	0.7963	0.0000	0.0000
shellcode	0.0000	0.0000	0.0000	0.1000	0.2833	0.0333	0.5000	0.0833	0.0000	0.0000
worms	0.0000	0.0000	0.0000	0.5000	0.0000	0.0000	0.5000	0.0000	0.0000	0.0000

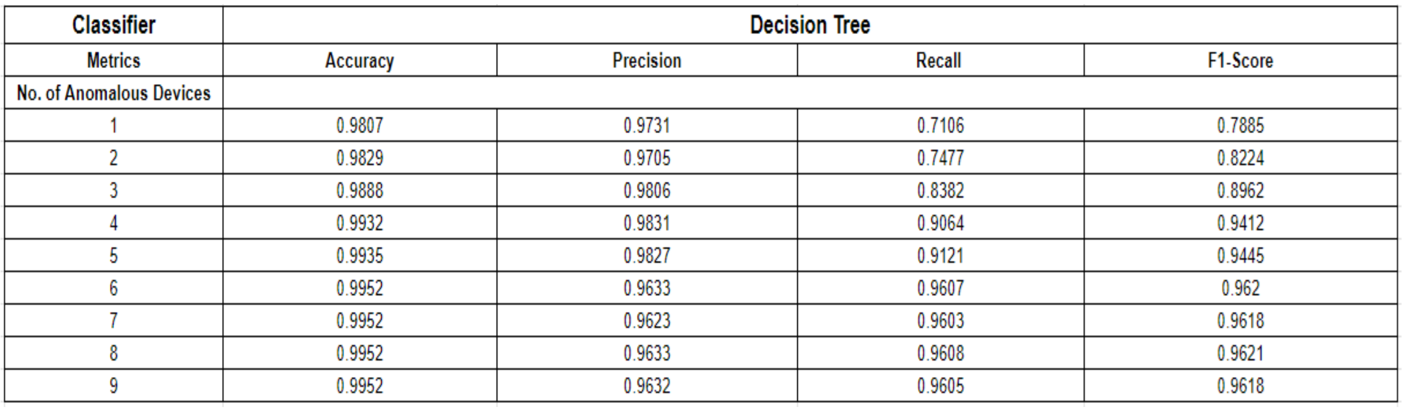


	analysis	backdoors	dos	exploits	fuzzers	generic	normal	reconnaissance	shellcode	worms
analysis	0.0065	0.0065	0.0195	0.3506	0.5000	0.0000	0.1039	0.0130	0.0000	0.0000
backdoors	0.0192	0.0000	0.0192	0.3526	0.5000	0.0000	0.0962	0.0128	0.0000	0.0000
dos	0.0054	0.0081	0.0430	0.4274	0.3091	0.0054	0.1801	0.0215	0.0000	0.0000
exploits	0.0006	0.0012	0.0138	0.7130	0.1408	0.0018	0.1197	0.0090	0.0000	0.0000
fuzzers	0.0013	0.0033	0.0060	0.1045	0.6059	0.0000	0.2763	0.0020	0.0007	0.0000
generic	0.0000	0.0004	0.0026	0.0340	0.0079	0.9277	0.0260	0.0013	0.0000	0.0000
normal	0.0000	0.0000	0.0000	0.0002	0.0012	0.0000	0.9984	0.0001	0.0000	0.0000
reconnaissance	0.0000	0.0000	0.0041	0.0916	0.0326	0.0020	0.0733	0.7963	0.0000	0.0000
shellcode	0.0000	0.0000	0.0000	0.1000	0.2833	0.0333	0.5000	0.0833	0.0000	0.0000
worms	0.0000	0.0000	0.0000	0.5000	0.0000	0.0000	0.5000	0.0000	0.0000	0.0000

The shellcode, fuzzers and worms ‘attack_cat’ categories of anomalies have strong misclassification as Normal Devices.

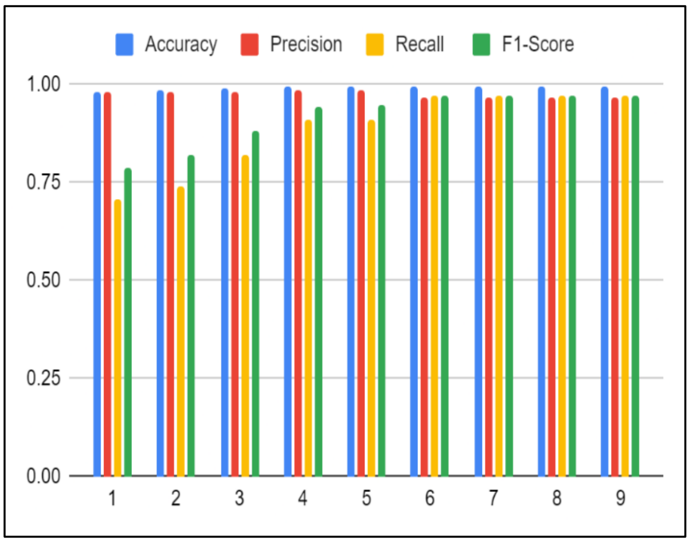
In the training dataset, we have already included all the types of anomalies that can be detected and hence our model knows what to expect. However, in a real-life situation, unpredictable anomalies can occur and our model should be strong enough to classify them as anomalies. To test this we trained a new model with different datasets, each progressively containing an increasing number of anomalies in their training datasets. The testing dataset remained the same.

This process of **Progressive out-of-Distribution (OOD) Generalization** testing was done by creating multiple datasets. Each dataset has a progressively increasing number of known anomaly cataegories included. At one extreme, the training dataset had no amonaliys categories and on the other extreme it was trained with all anomalies. For each intermediate dataset, the anomaly categories were included. The testing dataset remained consistent and contained all the known anomalies ensuring a fair evaluation of the model’s ability to generalize.

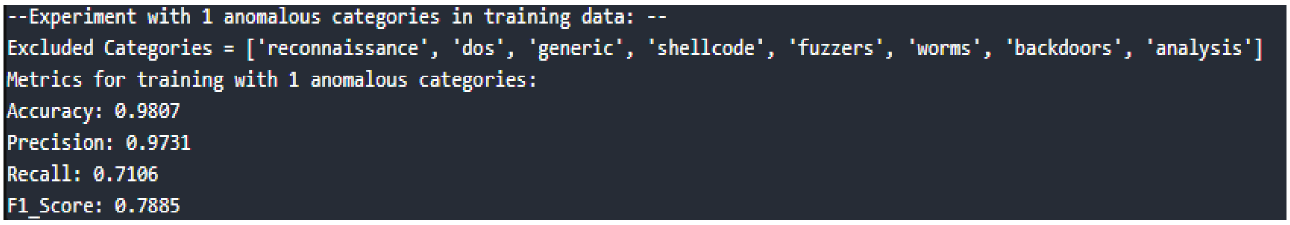


Classifier	**Decision Tree**			
Metrics	Accuracy	Precision	Recall	F1-Score
No. of Anomalous Devices				
1	0.9807	0.9731	0.7106	0.7885
2	0.9829	0.9705	0.7477	0.8224
3	0.9888	0.9806	0.8382	0.8962
4	0.9932	0.9831	0.9064	0.9412
5	0.9935	0.9827	0.9121	0.9445
6	0.9952	0.9633	0.9607	0.962
7	0.9952	0.9623	0.9603	0.9618
8	0.9952	0.9633	0.9608	0.9621
9	0.9952	0.9632	0.9605	0.9618

The graph on the right has the x-axis as the number of anomalies in the training dataset, and y-axis as measure of the 4 performance metrics. We observe that as the number of anomalies increases, all the metrics show a significant increase. The above table has summarized all the metrics as we progressively increase the number of anomalies in the training dataset.



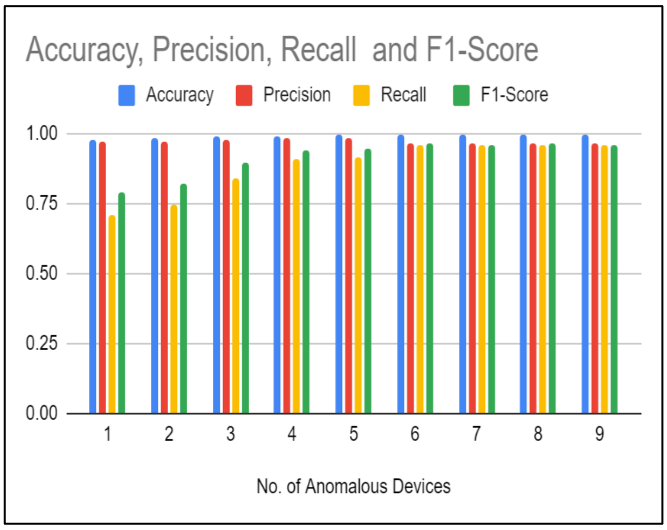
The image below shows a sample output given by the first iteration of the loop.

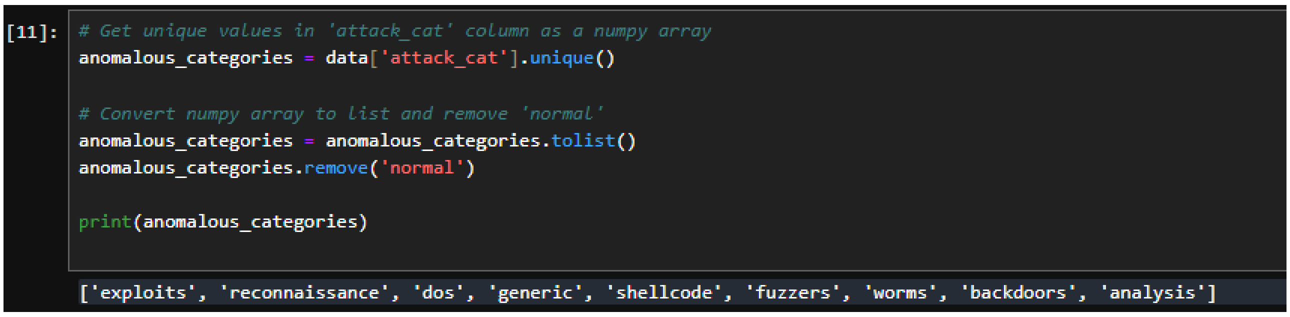


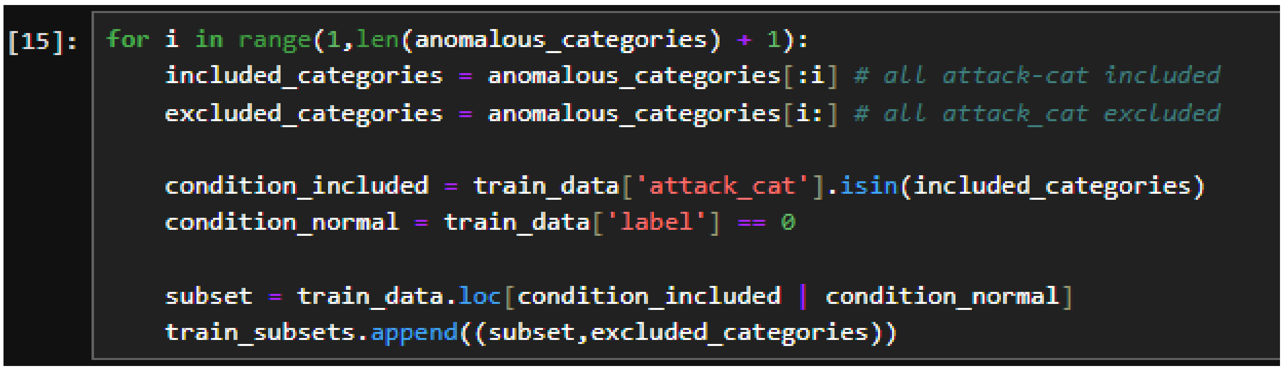
```
--Experiment with 1 anomalous categories in training data: --
Excluded Categories = ['reconnaissance', 'dos', 'generic', 'shellcode', 'fuzzers', 'worms', 'backdoors', 'analysis']
Metrics for training with 1 anomalous categories:
Accuracy: 0.9807
Precision: 0.9731
Recall: 0.7106
F1_Score: 0.7885
```

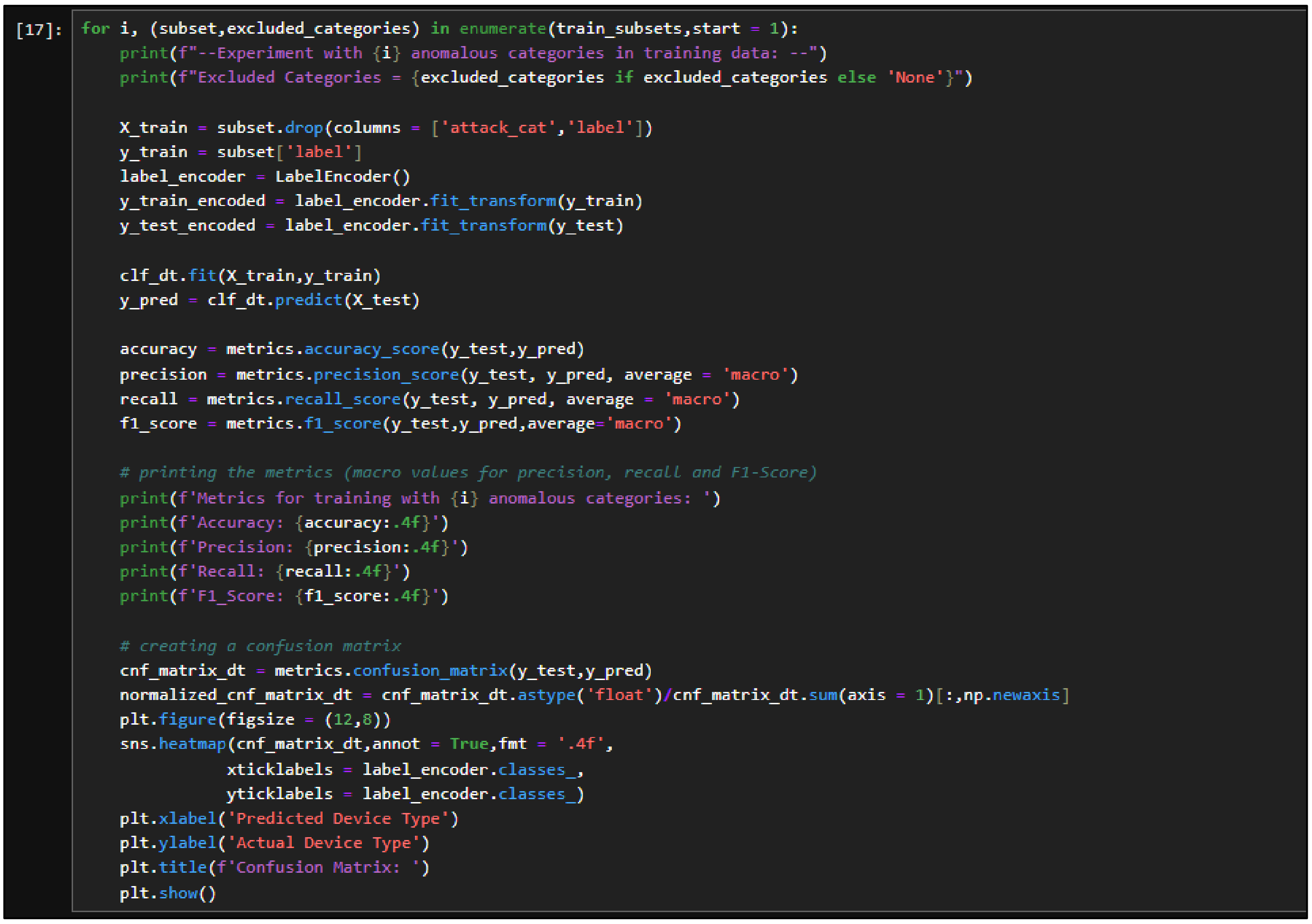
	Performance Metrics after removing anomalies from Training Dataset step-by-step (UNSW NB15 Dataset)			
Classifier	Random Forest			
No. of Anomalous Devices	Accuracy	Precision	Recall	F1-Score
1	0.9806	0.9785	0.7074	0.7863
2	0.9827	0.9807	0.7398	0.8176
3	0.9877	0.9825	0.8185	0.8826
4	0.9935	0.9872	0.9076	0.9436
5	0.9937	0.9869	0.9117	0.9459
6	0.9961	0.9684	0.9704	0.9694
7	0.9961	0.9669	0.9711	0.969
8	0.9961	0.9668	0.9711	0.9689
9	0.9961	0.968	0.971	0.9695

The Random Forest Classifier performs slightly better than the Decision Tree Classifier for the same experiment. Some of the code snippets of the main loop and the partitioning of the dataset is given below:




```
[11]: # Get unique values in 'attack_cat' column as a numpy array
      anomalous_categories = data['attack_cat'].unique()

      # Convert numpy array to list and remove 'normal'
      anomalous_categories = anomalous_categories.tolist()
      anomalous_categories.remove('normal')

      print(anomalous_categories)

      ['exploits', 'reconnaissance', 'dos', 'generic', 'shellcode', 'fuzzers', 'worms', 'backdoors', 'analysis']
```


```
[15]: for i in range(1,len(anomalous_categories) + 1):
          included_categories = anomalous_categories[:i] # all attack-cat included
          excluded_categories = anomalous_categories[i:] # all attack_cat excluded

          condition_included = train_data['attack_cat'].isin(included_categories)
          condition_normal = train_data['label'] == 0

          subset = train_data.loc[condition_included | condition_normal]
          train_subsets.append((subset,excluded_categories))
```



```
[17]: for i, (subset,excluded_categories) in enumerate(train_subsets,start = 1):
          print(f"--Experiment with {i} anomalous categories in training data: --")
          print(f"Excluded Categories = {excluded_categories if excluded_categories else 'None'}")

          X_train = subset.drop(columns = ['attack_cat','label'])
          y_train = subset['label']
          label_encoder = LabelEncoder()
          y_train_encoded = label_encoder.fit_transform(y_train)
          y_test_encoded = label_encoder.fit_transform(y_test)

          clf_dt.fit(X_train,y_train)
          y_pred = clf_dt.predict(X_test)

          accuracy = metrics.accuracy_score(y_test,y_pred)
          precision = metrics.precision_score(y_test, y_pred, average = 'macro')
          recall = metrics.recall_score(y_test, y_pred, average = 'macro')
          f1_score = metrics.f1_score(y_test,y_pred,average='macro')

          # printing the metrics (macro values for precision, recall and F1-Score)
          print(f'Metrics for training with {i} anomalous categories: ')
          print(f'Accuracy: {accuracy:.4f}')
          print(f'Precision: {precision:.4f}')
          print(f'Recall: {recall:.4f}')
          print(f'F1_Score: {f1_score:.4f}')

          # creating a confusion matrix
          cnf_matrix_dt = metrics.confusion_matrix(y_test,y_pred)
          normalized_cnf_matrix_dt = cnf_matrix_dt.astype('float')/cnf_matrix_dt.sum(axis = 1)[:,np.newaxis]
          plt.figure(figsize = (12,8))
          sns.heatmap(cnf_matrix_dt,annot = True,fmt = '.4f',
                      xticklabels = label_encoder.classes_,
                      yticklabels = label_encoder.classes_)
          plt.xlabel('Predicted Device Type')
          plt.ylabel('Actual Device Type')
          plt.title(f'Confusion Matrix: ')
          plt.show()
```

This code segment displays the performance metrics for each turn and a corresponding normalized confusion matrix for the experiment.

5. *UNSW IoT Analytics Dataset (Traffic Traces)*

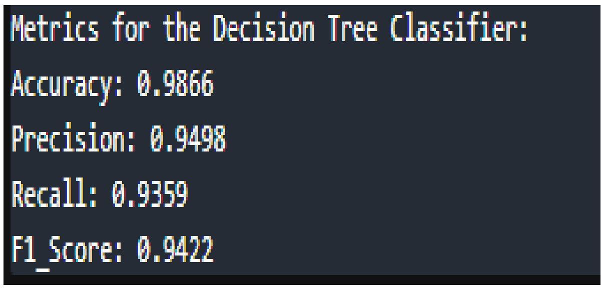
This dataset contained packet capture data from 29 IoT and Smart Devices and they were labelled using their corresponding MAC Addresses. PCAP data for 20 days was available, and I combined the data for 4 days and created a csv file. It contained over 2.5M packets and it was ensured that the data is not resampled or preprocessed.

This dataset contained very few numeric features – 'packet size' and 'port.src' and 'port.dst'. FFT was performed on these features to get better insights on the trends of the data. These fft_features were concatenated with the actual network traffic features to create a larger feature vector. Since not enough data numeric data was available, FFT could be applied on only 3 features.

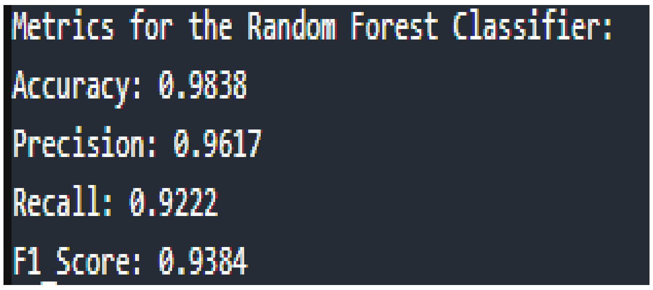
The same classifiers – Decision Tree and Random Forest, with the same tuned hyperparameters were used to perform this multiclass classification.

The next slides contain the results from these models. Unlike what was observed in all previous experiments, the Decision Tree Classifier yielded slightly better results here. The table highlights the exact macro averages of the performance metrics.

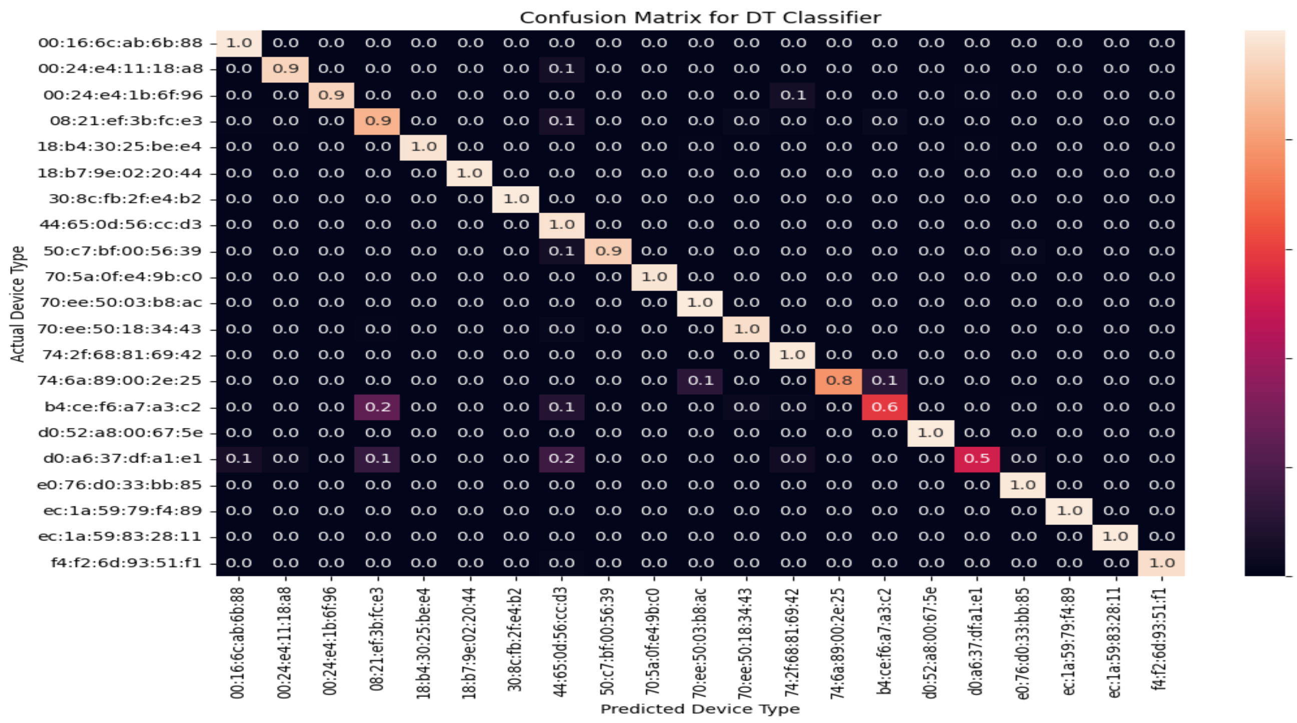
Metric	**Decision Tree**	**Random Forest**
Accuracy	0.9866	0.9838
Precision	0.9498	0.9617
Recall	0.9359	0.9222
F1 – Score	0.9422	0.9384



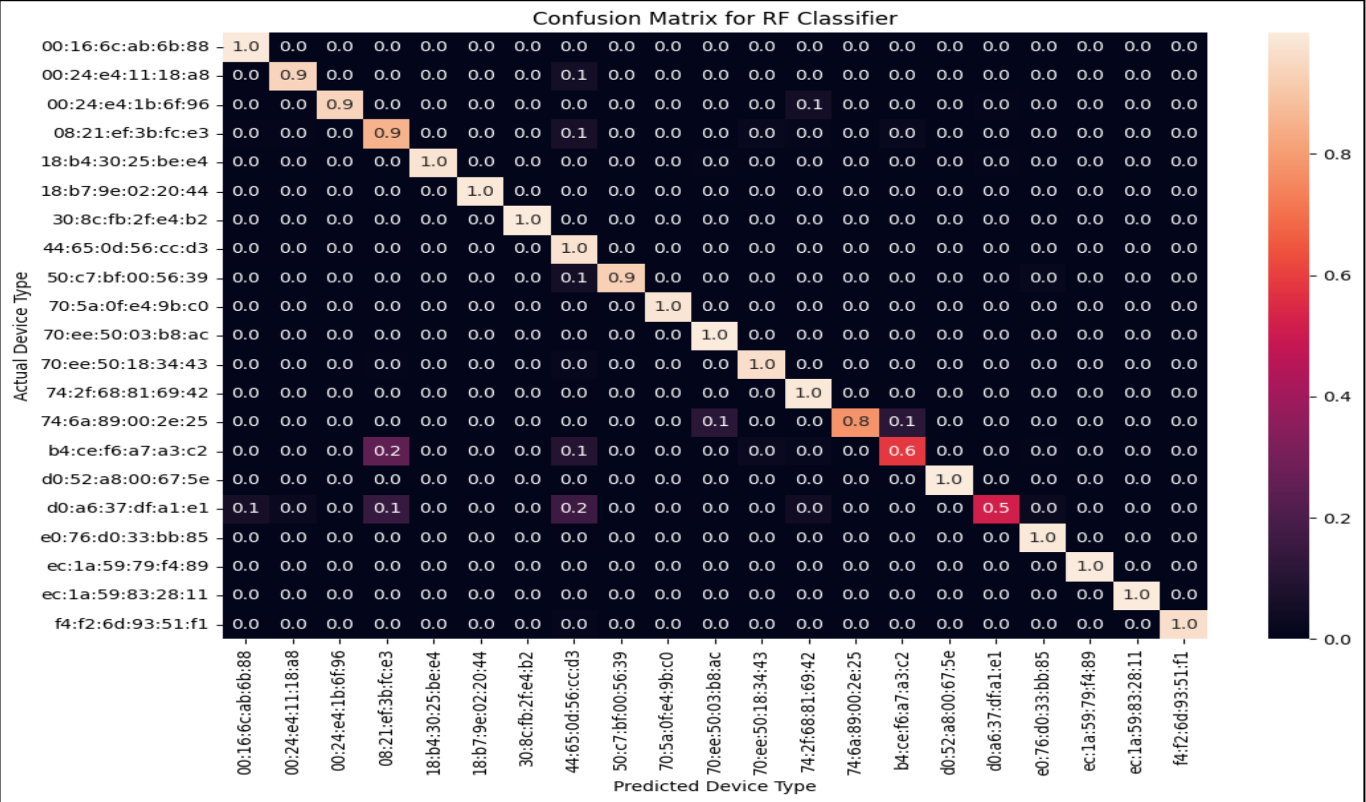
```
Metrics for the Decision Tree Classifier:
Accuracy: 0.9866
Precision: 0.9498
Recall: 0.9359
F1_Score: 0.9422
```



```
Metrics for the Random Forest Classifier:
Accuracy: 0.9838
Precision: 0.9617
Recall: 0.9222
F1_Score: 0.9384
```



	00:16:6c:ab:6b:88	00:24:e4:11:18:a8	00:24:e4:1b:6f:96	08:21:ef:3b:fc:e3	18:b4:30:25:be:e4	18:b7:9e:02:20:44	30:8c:fb:2f:e4:b2	44:65:0d:56:cc:d3	50:c7:bf:00:56:39	70:5a:0f:e4:9b:c0	70:ee:50:03:b8:ac	70:ee:50:18:34:43	74:2f:68:81:69:42	74:6a:89:00:2e:25	b4:ce:f6:a7:a3:c2	d0:52:a8:00:67:5e	d0:a6:37:df:a1:e1	e0:76:d0:33:bb:85	ec:1a:59:79:f4:89	ec:1a:59:83:28:11	f4:f2:6d:93:51:f1
00:16:6c:ab:6b:88	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
00:24:e4:11:18:a8	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
00:24:e4:1b:6f:96	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
08:21:ef:3b:fc:e3	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18:b4:30:25:be:e4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18:b7:9e:02:20:44	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30:8c:fb:2f:e4:b2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
44:65:0d:56:cc:d3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50:c7:bf:00:56:39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70:5a:0f:e4:9b:c0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70:ee:50:03:b8:ac	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70:ee:50:18:34:43	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
74:2f:68:81:69:42	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
74:6a:89:00:2e:25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.8	0.1	0.0	0.0	0.0	0.0	0.0	0.0
b4:ce:f6:a7:a3:c2	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0
d0:52:a8:00:67:5e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
d0:a6:37:df:a1:e1	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0
e0:76:d0:33:bb:85	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
ec:1a:59:79:f4:89	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
ec:1a:59:83:28:11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
f4:f2:6d:93:51:f1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0



	00:16:6c:ab:6b:88	00:24:e4:11:18:a8	00:24:e4:1b:6f:96	08:21:ef:3b:fc:e3	18:b4:30:25:be:e4	18:b7:9e:02:20:44	30:8c:fb:2f:e4:b2	44:65:0d:56:cc:d3	50:c7:bf:00:56:39	70:5a:0f:e4:9b:c0	70:ee:50:03:b8:ac	70:ee:50:18:34:43	74:2f:68:81:69:42	74:6a:89:00:2e:25	b4:ce:f6:a7:a3:c2	d0:52:a8:00:67:5e	d0:a6:37:df:a1:e1	e0:76:d0:33:bb:85	ec:1a:59:79:f4:89	ec:1a:59:83:28:11	f4:f2:6d:93:51:f1
00:16:6c:ab:6b:88	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
00:24:e4:11:18:a8	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
00:24:e4:1b:6f:96	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
08:21:ef:3b:fc:e3	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18:b4:30:25:be:e4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18:b7:9e:02:20:44	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30:8c:fb:2f:e4:b2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
44:65:0d:56:cc:d3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50:c7:bf:00:56:39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70:5a:0f:e4:9b:c0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70:ee:50:03:b8:ac	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70:ee:50:18:34:43	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
74:2f:68:81:69:42	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
74:6a:89:00:2e:25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.8	0.1	0.0	0.0	0.0	0.0	0.0	0.0
b4:ce:f6:a7:a3:c2	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0
d0:52:a8:00:67:5e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
d0:a6:37:df:a1:e1	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0
e0:76:d0:33:bb:85	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
ec:1a:59:79:f4:89	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
ec:1a:59:83:28:11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
f4:f2:6d:93:51:f1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Challenges faced:

- Inability in tracing the sensors associated with each MAC Address / NodeMCU saved from D0-D7. This caused us hindrance in making inferences from the results. To tackle this, we have planned to boot the setup, and achieve the MAC addresses for each sensor using Wireshark. We resolved this challenge by using the MAC Addresses that the NodeMCU's provided after we booted the setup again.
- Another difficulty that I faced was in the implementation of the CNN model for Frequency-domain analysis. I tried creating similar classifiers using CNN's and learnt about CNN Architectures and how they worked. However, the results yielded from the model were subpar and worse than a random classifier. Since Shobhan was already working on training a CNN model, I discarded the idea and began proceeding with the ML Classifier to test it on various datasets.
- The UNSW traffic traces dataset contained only the 'packet-size' as the numeric feature through which we could try to capture frequency-domain patterns. Hence, it is essential that we capture data such as 'inter-packet arrival time', 'source_packets', 'destination_packets', etc for frequency domain and temporal analysis.
- The accuracy of the ML model has reached a plateau and increasing it further is much more difficult, than training a CNN on it. CNN's are much powerful than these classifiers since they figure out their structures are complex and they can recognize patterns much easily.

Future Targets:

- Deploying these models on the Raspberry Pi since the setup is ready.
- Testing these models and making the classification task real-time (as soon as the packets are captured)
- Time-domain analysis can be explored to further better the model, and create a multi-modal setup involving both Hardware and Network level features for better classification.
- Exploring other domains where this type of fingerprinting strategies can be implemented and develop technology in that direction since the classification results will reach a standstill at some point.

Conclusion:

The major sector of IoT Device Fingerprinting that we are working on is relatively new and unexplored since it involves Hardware features. We explored numerous Machine Learning and Deep Learning techniques to improve the classification accuracy of the previously build models by capturing Hardware feature trends. This led to a clear jump in performance, and we wish to create a multimodal setup that combines both aspects of data transmission – Hardware and Software. The models are ready, and can easily be deployed on the Raspberry Pi to enable real-time classification. By performing multiple experiments on Open-Source datasets, we validated our model and ensured that they provide optimal output for all kinds of data that a network encounters.

References:

1. K. Kostas, M. Just and M. A. Lones, "IoTDevID: A behavior-based device identification method for the IoT", *arXiv:2102.08866*, 2021.
2. Yadav P, Feraudo A, Arief B, Shahandashti SF, Vassilakis VG (2020) Position paper: A systematic framework for categorising IoT device fingerprinting mechanisms. In: Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, pp 62–68
3. S.S. Wanode, M. Anand, B. Mitra, Optimal Feature Set Selection for IoT Device Fingerprinting on Edge Infrastructure using Machine Intelligence, in: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops, INFOCOM 2022 - Workshops, New York, NY, USA, May 2-5, 2022, 2022, pp. 1–6.
4. Jiao, R., Liu, Z., Liu, L., Ge, C., Hancke, G. Multi-level IoT device identification. In *Proc. of IEEE ICPADS '21*, 538–547 (2021). https://doi.org/10.1109/ICPADS53394.2021.00073.
5. Lorenz F., Thamsen L., Wilke A., Behnke I., Waldmüller-Littke J., Komarov I., Kao O., and Paeschke M., Fingerprinting analog IoT sensors for secret-free authentication, *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020, Honolulu, HI, USA, 1–6.
6. Jiang, Y.; Dou, Y.; Hu, A. Identification of IoT Devices Based on Hardware and Software Fingerprint Features. *Symmetry* **2024**, *16*, 846. https://doi.org/10.3390/sym16070846
7. Charyyev, Batyr, and Mehmet Hadi Gunes. "Locality-sensitive iot network traffic fingerprinting for device identification." *IEEE Internet of Things Journal* 8.3 (2020): 1272-1281.
8. Prasad, Arjun, et al. "Context-Aware Behavioral Fingerprinting of IoT Devices via Network Traffic Analysis." *SECRYPT*. 2023.
9. Thangavelu, Vijayanand, et al. "DEFT: A distributed IoT fingerprinting technique." *IEEE Internet of Things Journal* 6.1 (2018): 940-952.
10. Aftab, Muhammad, Sid Chi-Kin Chau, and Prashant Shenoy. "Efficient online classification and tracking on resource-constrained IoT devices." *ACM Transactions on Internet of Things* 1.3 (2020): 1-29.
11. Y. Shang, L. Yang, Y. Hou, W. Wang, W., Li, P., and Zhang, Y., “Fingerprinting mainstream IoT platforms using traffic analysis,” IEEE Internet of Things Journal, 9(3):2083–2086, 2022.
12. Wang, Boxiong, et al. "Efficient traffic-based IoT device identification using a feature selection approach with Lévy flight-based sine chaotic sub-swarm binary honey badger algorithm." *Applied Soft Computing* 155 (2024): 111455.
13. Zhang, Kunpeng, et al. "Enhancing IoT (Internet of Things) feature selection: A two-stage approach via an improved whale optimization algorithm." *Expert Systems with Applications* 256 (2024): 124936.