Device Sense: A Behavioral Approach to Identifying IoT Devices

Report submitted to the SASTRA Deemed to be University
in partial fulfillment of the requirements
for the award of the degree of

Bachelor of Technology

Submitted by

ATHARSH NANTHA A S
(Reg. No.: 124014004, Information and Communication Technology)
IRAIANBAN S
(Reg. No.: 124014017, Information and Communication Technology)
MUGUNDHAN S
(Reg. No.: 124014032, Information and Communication Technology)

May 2024

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Bonafide Certificate

This is to certify that the project report titled "**Device Sense: A Behavioral Approach to Identifying IoT Devices"** submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. Information & Communication Technology to the SASTRA Deemed to be University, is a bona-fide record of the work done by Mr. ATHARSH NANTHA A S(Reg. No. 124014004) , Mr. IRAIANBAN S(Reg. No. 124014017), Mr. MUGUNDHAN S(Reg. No. 124014032)during the final semester of the academic year 2023-24, in the **School of Computing**, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor : 

Name with Affiliation : Dr. S. Raj Anand, Asst. Professor-III/SOC

Date : 23-04-24

Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

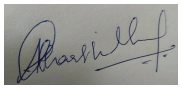
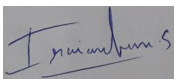
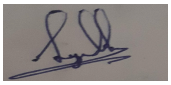
SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Declaration

We declare that the project report titled "**Device Sense: A Behavioral Approach to Identifying IoT Devices**" submitted by us is an original work done by us under the guidance of **Dr. Raj Anand S Asst. Professor-III, School of Computing, SASTRA Deemed to be University** during the final semester of the academic year 2023-24, in the **School of Computing**. The work is original and wherever We have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s) :   

Name of the candidate(s) : Atharsh Nantha A S Iraianban S Mugundhan S

Date : 23-04-24

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for their encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend my/our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Algeswaran**, Associate Dean, Students Welfare.

Our guide **Dr. Raj Anand S, Asst. Professor-III**, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us to make progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing us with an opportunity to showcase my skills through this project.

Table of Contents

Title	Page No
Bona-fide Certificate	ii
Declaration	iii
Acknowledgments	iv
List of Figures	vi
List of Tables	vii
Abbreviations	viii
Abstract	ix
Chapter 1 – Summary of the Base Paper	
1.1 Base Paper Details	1
1.2 Introduction	1
1.3 Materials	1
1.4 Methods	2
1.5 TCP Port Ranges	2
1.6 Proposed Model	2
1.6.1 Algorithm Combination	3
1.6.2 Preprocessing	5
1.7 System Architecture	6
Chapter 2 – Merits and Demerits	7
Chapter 3 – Implementation	9
Chapter 4 – Source Code	11
Chapter 5 – Snapshots	90
Chapter 6 – Efficiency and Conclusion	95
Chapter 7 – References	96

List of Figures

Figure No.	Title	Page No.
1	Feature Extraction Process	90
2	Output of Voting Classification	90
3	Comparison of isolated data and merged cross validated data result according to the features	91
4	Confusion matrix of KNN and RF combined (Aalto)	92
5	Confusion matrix of KNN and GB combined (UNSW)	93
6	Confusion matrix for Performance evaluation combined datasets	94

List of Tables

Table No.	Table name	Page No.
1	Comparison of ML algorithms (Existing model)	10
2	Comparison of Combination of ML algorithms (Proposed model)	10

Abbreviations

RF	-	Random Forest
KNN	-	K Nearest Neighbours
SVM	-	Support Vector Machine
DT	-	Decision Tree
GB	-	Gradient Booster
NB	-	Naive Bayes
PCAP	-	Packet Capture
GA	-	Genetic Algorithm
CV	-	Cross Validation

Abstract

In today's world, the Internet of Things (IoT) is really important for making things automatic, but we gotta be super careful about security, especially for spotting dodgy devices. The identification of devices is a vital mechanism to enhance the security of IoT networks by isolating potential threats. But one big problem we face in the current setup is properly gauging the security of all the gadgets hooked up to IoT devices. To fix the problems with what we already have, we present IoTDevID, a machine learning-based method aimed at discerning devices based on distinctive characteristics found in their network packets. Our project employs a meticulous feature analysis and selection process, offering a versatile and practical framework for modeling device behavior. Importantly, our methodology demonstrates superior predictive accuracy when applied to two publicly available datasets. The intrinsic feature set of our project showcases heightened predictive capabilities compared to established feature sets used in traditional device identification methods. This highlights its adaptability to previously unseen data. The main goal of our project is to stand out from the usual ways of identifying IoT devices. We want to show that we can spot devices using different methods, like ones that don't need much power and don't rely on IP addresses.

Specific Contribution

- Dataset collection and Coding.
- Algorithm selection and Dataset collection.
- Code definition and PPT.

Specific Learning

- Instead of using a single algorithm, combining different combinations of algorithms yields higher accuracy and precision.

CHAPTER 1

SUMMARY OF THE BASE PAPER

1.1 BASE PAPER DETAILS

Title **:** IoTDevID : Behavior- Based Device Identification Method for the IoT

Journal name **:** IEEE Internet of Things Journal

Publisher **:** IEEE

Year **:** July 19, 2022

Indexed in **:** Scopus

1.2 INTRODUCTION

Our research centers on the Internet of Things (IoT), encompassing objects equipped with communication systems. These devices serve vital functions in smart homes, healthcare, and manufacturing. Securing IoT devices is challenging due to resource constraints and device diversity. To tackle these hurdles, We suggest a novel approach for recognizing IoT devices. based on their network behavior rather than relying solely on specific identifiers like MAC or IP addresses. This approach has wider applicability, especially for devices using non-IP and low-energy protocols like Bluetooth, ZigBee, or ZWave. Through rigorous experimentation, we developed and validated our method, demonstrating its effectiveness across varied datasets. Our approach represents a significant advancement over previous techniques, offering improved detection capabilities and adaptability.

1.3 MATERIALS

We tested our device identification (DI) method on two public datasets, focusing on benign networks devoid of any attack data. The initial dataset, sourced from Aalto University, provides installation data for 31 devices, with some being paired, resulting in 27 classes due to shared behavior. Interestingly, some devices in this set lack their own MAC and IP addresses, posing a "transfer problem," but our method can still identify them based solely on their behavior. The second dataset, obtained from UNSW, comprises network logs from 28 IoT devices recorded over a 60-day period, Enhanced with supplementary data, we incorporate information for four devices from a separate study. We first honed our approach using the smaller Aalto dataset and then assessed its efficacy on the larger UNSW dataset to validate its generalizability.

1.4 METHODS

Previous studies commonly combined multiple packets to create fingerprints, often relying on features like MAC or IP addresses, which may not accurately represent individual devices, especially in cases of the "transfer problem" where multiple devices share the same address. To tackle this issue, we chose to utilize individual packets instead of merged ones, This approach not only conserves resources but also prevents misinterpretations arising from shared addresses. However, relying solely on individual packets may not always result in clear device identification due to potential ambiguity. To improve accuracy, we introduced an aggregated method that groups packets based on MAC addresses and machine learning labels, enabling more reliable device discrimination. By analyzing behavior before merging packets, we enhance previous approaches. Nevertheless, it's crucial to acknowledge that this technique might not be appropriate for MAC addresses impacted by the transfer problem. Thus, a blended approach for assessment is required to adequately address such instances.

1.5 TCP PORTS RANGES

- Well-Known Ports : 0 through 1023(HTTP (port 80), HTTPS (port 443), FTP (port 21), SSH (port 22), SMTP (port 25))
- Registered Ports : 1024 through 49151
- Dynamic/Private : 49152 through 65535

1.6 PROPOSED MODEL

Previous studies employed list algorithms to detect ambiguous packets in the IoT network, but their accuracy fell short. Thus, we conducted experiments with different algorithm combinations using an ensemble method. Notably, a particular combination of KNN and RF demonstrated enhanced accuracy compared to previous methods.

1.6.1 ALGORITHM COMBINATION

To make sure we classify IoT network traffic better, our suggested model combines the machine learning tricks of Random Forest (RF) and K-Nearest Neighbours (KNN). It obtains a complete feature set for in-depth examination of network traffic patterns through feature fusion, improving anomaly detection capabilities. These characteristics are the result of combining elements from the KNN and RF algorithms. By merging predictions from KNN and RF algorithms using an ensemble learning technique, the model improves overall predictive power. Parameter optimization means tweaking settings like the number of trees in RF and the value of K in KNN to get the best performance, accuracy, and efficiency.

```
ml_list = {}  # Init a dictionary
# Def parameters for RF Classifier
rf_params = {
    "bootstrap": True,
    "criterion": "gini",
    "max_depth": 18.0,
    "max_features": 8,
    "min_samples_split": 9,
    "n_estimators": 96
}
# Def parameters for KNN Classifier
knn_params = {
    "algorithm": 'brute',
    "leaf_size": 41,
    "n_neighbors": 48,
    "weights": 'distance'
}
# Create RFt Classifier
rf_classifier = RandomForestClassifier(**rf_params)
# Create KNN Classifier
```

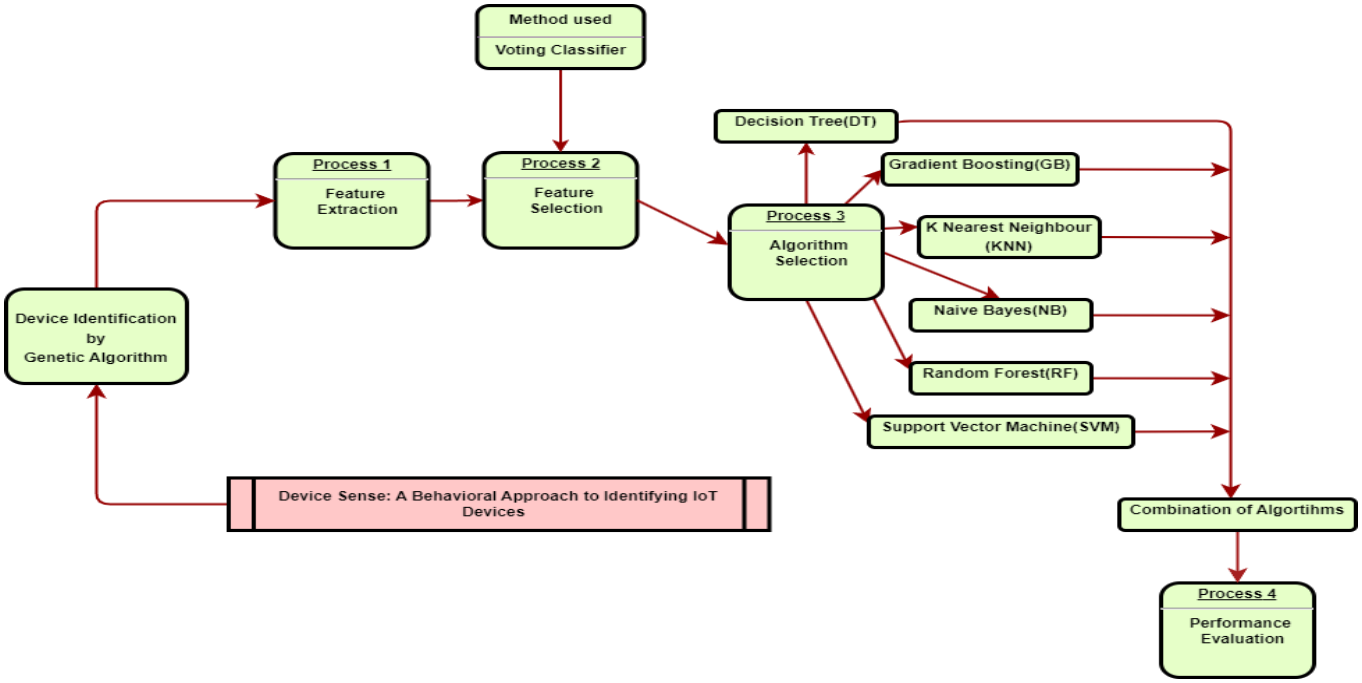
```
21: knn_classifier = KNeighborsClassifier(**knn_params)
22: # Combine classifiers using Voting Classifier
23: combined_classifier = VotingClassifier(
24:     estimators=[
25:         ("RF", rf_classifier),
26:         ("KNN", knn_classifier)
27:     ],
28:     voting="hard"
29: )
30: # Add the combined classifier to the ML model list
31: ml_list["Combined"] = combined_classifier
1: Def (rf_params)
2: Define knn_params
3: Function RandomForestClassifier(params):
4:     Initialize a Random Forest Classifier object
5:     Set classifier parameters according to the provided 'params'
6:     Return the initialized Random Forest Classifier object
7: Function KNeighborsClassifier(params):
8:     Initialize a K-Nearest Neighbors Classifier object
9:     Set classifier parameters according to the provided 'params'
10:    Return the initialized K-Nearest Neighbors Classifier object
11: # Example parameter values for (rf_params)
12: rf_params = {
13:     "bootstrap": True,
14:     "criterion": "gini",
15:     "max_depth": 18.0,
```

```
16:     "max_features": 8,
17:     "min_samples_split": 9,
18:     "n_estimators": 96
19: }
20: # Example knn_params
21: knn_params = {
22:     "algorithm": 'brute',
23:     "leaf_size": 41,
24:     "n_neighbors": 48,
25:     "weights": 'distance'
26: }
27: # Create RF Classifier object with specified parameters
28: rf_classifier = RandomForestClassifier(rf_params)
29: # Create KNN Classifier object with specified parameters
30: knn_classifier = KNeighborsClassifier(knn_params)
```

1.6.2 PREPROCESSING

The process commences with pcaps extraction, focusing on either the Aalto or UNSW dataset, followed by the exclusion of data associated with the ARP protocol. Following that, a hybrid feature selection method is utilized, merging information gain (IG) with principal component analysis (PCA). IG evaluates feature subsets based on entropy to ascertain relevance scores, while PCA diminishes dimensions while preserving crucial attribute details through orthogonal combinations. A voting classifier, incorporating various machine learning algorithms, This method finds the most important features for classification by combining predictions from separate classifiers using a strict voting system. This approach is specifically utilized for feature extraction from pcap files for classifying network traffic and detecting intrusions. The suggested model is assessed using standard metrics like F1-score, precision, recall, and accuracy. These metrics are employed to assess the effectiveness of the combined RF and KNN approach in detecting and classifying different types of network attacks.

1.7 SYSTEM ARCHITECTURE



Feature Extraction:

- Start with "Device Identification by Genetic Algorithm." Extract relevant features from device data.

Feature Selection:

- Selecting the most important feature with the help of Voting Classifier.

Algorithm Selection:

- Consider various algorithms: Decision Tree (DT), Gradient Boosting (GB), K Nearest Neighbour (KNN), Naive Bayes (NB), Random Forest (RF), Support Vector Machine (SVM) and some combinations of these mentioned algorithms.

Performance Evaluation:

- Combine the chosen datasets and evaluate their effectiveness for device identification.

CHAPTER 2

MERITS AND DEMERITS

BASE PAPER

MERITS

- Enhanced Accuracy: By putting together various machine learning methods through ensemble learning, the accuracy of device identification can be significantly improved. This ensures more reliable detection of IoT devices, even in complex and dynamic network environments.
- Robustness: Ensemble methods tend to be more robust to noise and outliers in the data compared to individual algorithms. This robustness contributes to the method's effectiveness in accurately identifying IoT devices under various conditions and scenarios.
- Adaptability: The ensemble method can adapt to different types of IoT devices and network protocols, making it suitable for diverse IoT environments. This flexibility ensures that the method remains effective even as IoT technologies evolve, and new devices are introduced.

DEMERITS

- Complexity: Ensemble methods can be more complex to implement and maintain compared to individual algorithms. This complexity might need more computational resources and expertise, potentially limiting its practicality for some applications.
- Increased Computational Overhead: Combining multiple algorithms in an ensemble approach may result in increased computational overhead, leading to longer processing times and higher resource requirements. This could be tough for deployment in resource constrained IoT environments.
- Overfitting Risk: There's a risk of overfitting when using ensemble methods, particularly if the individual algorithms are not carefully selected or if the ensemble is too complex. Overfitting can lead to reduced generalizability and accuracy of the device identification model.

PROPOSED WORK

MERITS

- Efficient Identification: This paper suggests a smarter way to recognize IoT gadgets based on how they act. It's better than the usual methods that rely on IP or MAC addresses because it can still spot devices even if they're using tricky tactics like encryption or changing their IP addresses.
- Accuracy: The method they propose is really good at figuring out which IoT devices are which, even if they're using encrypted stuff. Since encryption is common in IoT to keep things private and safe, this accuracy is super helpful.
- Generalizability: Their method works for lots of different IoT gadgets and the ways they communicate, like Zigbee or Bluetooth. This is a big deal because IoT setups can have all kinds of gadgets and ways of talking, so it's great to have a method that's flexible.

DEMERITS

- Limited Datasets: They didn't have a huge number of different examples to train their smart system, which might mean it doesn't work as well with all kinds of gadgets and situations as it could.
- Lack of Real-World Implementation: They didn't try out their method in real-life situations, so we can't be sure how well it works when it's being used, which is important to know.
- Performance Evaluation: They didn't give us a lot of info about how fast or efficient their method is, or how much stuff it needs to work properly. This info is key for deciding if it's practical to use in big IoT setups.

CHAPTER 3

IMPLEMENTATION

First, we collected datasets from Aalto and UNSW. Next, we proceeded with pre-processing based on features such as:

i. ARP (Address Resolution Protocol)
ii. Class Labels

Initially, we began by extracting PCAP files from both the UNSW and Aalto datasets. Then, within these PCAP files, we filtered out devices utilizing the Address Resolution Protocol (ARP), focusing on non-IP based devices for our study. After this filtering step, we proceeded with feature selection using a voting classifier, in which TCP_sport yields higher votes than the other features. Following feature selection, we selected a set of supervised algorithms based on their potential to improve accuracy. These algorithms were carefully assessed using various metrics including Accuracy, F1_Score, Precision, Recall, and Algorithm run time. This exhaustive evaluation process facilitated informed decision-making regarding the most fitting algorithms for our research goals.

In the further performance evaluation process, we utilized an additional metric called kappa, also known as Cohen kappa, which describes the level of agreement between two classifications. We incorporated functions such as 'mixed', where 'mixed' denotes a Boolean variable (true/false). In the context of our project, 'mixed' indicates whether the data is being mixed in some way, such as whether the training data comprises a mix of different sources or types.

Additionally, in the performance evaluation, we conducted hyperparameter optimization, wherein we adjusted several hyperparameters corresponding to respective algorithms. For instance, in the Decision Tree algorithm, we adjusted the 'max_depth' parameter, which specifies the maximum depth the node can attain. Moreover, we utilized parameters such as Gini index and entropy to split the nodes in the decision tree. The Gini index calculates the likelihood of a randomly selected element being incorrectly satisfied, while entropy measures disorder or uncertainty.

TABLE I: Comparison of ML algorithms (Existing model)

ML	Accuracy	Precision	Recall	F1-Score	Train-T	Test-T
DT	70.5%	77.4	70.6	72.7	0.128	0.004
GB	69.9%	78.9	69.3	72.5	918.3	8.312
KNN	70.5%	75.2	70.5	71.8	0.005	20.2
NB	61.7%	58.4	62.9	55.9	0.433	0.032
RF	70.8%	76.8	70.8	72.7	3.742	0.333
SVM	68.0%	69.7	63.4	64.9	101.3	64.8

- Table 1 represents the accuracy of each algorithm against Aalto and UNSW datasets, in which DT (Decision Tree) scores a higher accuracy value of 70.5% with minimum amount of time taken compared to the other algorithms. On the other hand SVM (Naive Bayes) with a low score of 61.7% . In terms of time SVM performs poorly compared to all other algorithms

TABLE II: Comparison of Combination of ML algorithms (Proposed model)

ML	Accuracy	Precision	Recall	F1-Score	Train-T	Test-T
RF & KNN	72.0%	80	73	74.5	10.62	76.9
KNN & GB	69.0%	76.0	68.0	68.49	325	38
NB & DT	68%	70	68.5	64.2	2.785	0.039
NB & RF	68%	70	69	64.45	8.641	0.865

- Table 2 displays the accuracy achieved by various combinations of algorithms enhanced through ensemble methods. Through this approach, we've attained a higher accuracy rate of 72% with Random Forest and K Nearest Neighbours, surpassing previous efforts. However, it's worth noting that the computational time required by this ensemble is considerably longer compared to prior methodologies. Conversely, the combination of Naive Bayes and Random Forest yielded the lowest accuracy among the ensembles at 68%. While we could explore alternative algorithm combinations, doing so would significantly increase computational time. Hence, we concluded our experimentation with this minimal set of combinations.

CHAPTER 4

SOURCE CODE

FEATURE EXTRACTION

```
import pandas as pd
import zipfile
import os
import shutil
def folder(f_name): #this function creates a folder named "attacks" in the program directory.
    try:
        if not os.path.exists(f_name):
            os.makedirs(f_name)
    except OSError:
        print ("The folder could not be created!")
path="captures_IoT_Sentinel.zip"
with zipfile.ZipFile(path, 'r') as zip_ref:
    zip_ref.extractall("./")
path="./captures_IoT_Sentinel/"
def find_the_way(path,file_format):
    files_add = []
    # r=root, d=directories, f = files
    for r, d, f in os.walk(path):
        for file in f:
            if file_format in file:
                files_add.append(os.path.join(r, file))
    return files_add
```

```
files_add=find_the_way(path,'.pcap')
files_add
train=[]
test=[]
validation=[]
for ii, i in enumerate(files_add):
    print(ii,i)
    if ii%5!=0:
        if ii%4==0:
            validation.append(i)
            test.append(files_add[ii+1])


        else:
            train.append(i)
len(test),len(train),len(validation)
from scapy.all import*
import math
import pandas as pd
import os
import numpy as np
def folder(f_name): #this function creates a folder.
    try:
        if not os.path.exists(f_name):
            os.makedirs(f_name)
    except OSError:
        print ("The folder could not be created!")
```

MAC_list={

UNSW IEEE TMC 2018 Data MAC and Device names

"d0:52:a8:00:67:5e":"Smart Things",

"44:65:0d:56:cc:d3":"Amazon Echo",

"70:ee:50:18:34:43":"Netatmo Welcome",

"f4:f2:6d:93:51:f1":"TP-Link Day Night Cloud camera",

"00:16:6c:ab:6b:88":"Samsung SmartCam",

"30:8c:fb:2f:e4:b2":"Dropcam",

"00:62:6e:51:27:2e":"Insteon Camera",

"e8:ab:fa:19:de:4f":"unknown maybe cam",

"00:24:e4:11:18:a8":"Withings Smart Baby Monitor",

"ec:1a:59:79:f4:89":"Belkin Wemo switch",

"50:c7:bf:00:56:39":"TP-Link Smart plug",

"74:c6:3b:29:d7:1d":"iHome",

"ec:1a:59:83:28:11":"Belkin wemo motion sensor",

"18:b4:30:25:be:e4":"NEST Protect smoke alarm",

"70:ee:50:03:b8:ac":"Netatmo weather station",

"00:24:e4:1b:6f:96":"Withings Smart scale",

"74:6a:89:00:2e:25":"Blipcare Blood Pressure meter",

"00:24:e4:20:28:c6":"Withings Aura smart sleep sensor",

"d0:73:d5:01:83:08":"Light Bulbs LiFX Smart Bulb",

"18:b7:9e:02:20:44":"Triby Speaker",

"e0:76:d0:33:bb:85":"PIX-STAR Photo-frame",

"70:5a:0f:e4:9b:c0":"HP Printer",

"08:21:ef:3b:fc:e3":"Samsung Galaxy Tab",

"30:8c:fb:b6:ea:45":"Nest Dropcam",

"40:f3:08:ff:1e:da":"Android Phone",

"74:2f:68:81:69:42":"Laptop",

"ac:bc:32:d4:6f:2f":"MacBook",

"b4:ce:f6:a7:a3:c2":"Android Phone",

"d0:a6:37:df:a1:e1":"IPhone",

"f4:5c:89:93:cc:85":"MacBook/Iphone",

"14:cc:20:51:33:ea":"TPLink Router Bridge LAN (Gateway)",

Yourthings Data MAC and Device names

'00:01:c0:18:7f:9b': 'Gateway',

'00:04:4b:55:f6:4f': 'nVidiaShield',

'00:12:16:ab:c0:22': 'ChineseWebcam',

'00:17:88:21:f7:e4': 'PhilipsHUEHub',

'00:1d:c9:23:f6:00': 'RingDoorbell',

'00:21:cc:4d:59:35': 'Wink2Hub',

'00:24:e4:2b:a5:34': 'WithingsHome',

'00:7e:56:77:35:4d': 'KoogeekLightbulb',

'08:05:81:ee:06:46': 'Roku4',

'08:86:3b:6f:7a:15': 'BelkinWeMoMotionSensor',

'08:86:3b:70:d7:39': 'BelkinWeMoSwitch',

'0c:47:c9:4e:fe:5b': 'AmazonFireTV',

'10:ce:a9:eb:5a:8a': 'BoseSoundTouch10',

'18:b4:30:31:04:b9': 'NestProtect',

'18:b4:30:40:1e:c5': 'NestGuard',

'18:b4:30:58:3d:6c': 'NestCamera',

'18:b4:30:8c:03:e4': 'NestCamIQ',

'20:df:b9:20:87:39': 'GoogleHomeMini',

'30:52:cb:a3:4f:5f': 'RokuTV',

'3c:f7:a4:f2:15:87': 'iPhone',

'40:9f:38:92:40:13': 'Roomba',

'44:73:d6:01:3d:fd': 'LogitechLogiCircle',

'48:d6:d5:98:53:84': 'GoogleHome',

'50:c7:bf:92:a6:4a': 'TP-LinkSmartWiFiLEDBulb',

'54:4a:16:f9:54:18': 'InsteonHub',

'5c:aa:fd:6c:e0:d4': 'Sonos',

'64:52:99:97:f8:40': 'ChamberlainmyQGarageOpener',

'74:c2:46:1b:8e:e2': 'AmazonEchoGen1',

'7c:64:56:60:71:74': 'SamsungSmartTV',

'7c:70:bc:5d:09:d1': 'Canary',

'94:10:3e:5c:2e:31': 'BelkinWeMoCrockpot',

'94:10:3e:cc:67:95': 'BelkinWeMoLink',

'94:4a:0c:08:7e:72': 'MiCasaVerdeVeraLite',

'a4:f1:e8:8d:b0:9e': 'AndroidTablet',

'ac:3f:a4:70:4a:d6': 'PiperNV',

'b0:4e:26:20:15:8a': 'TP-LinkWiFiPlug',

'b0:7f:b9:a6:47:4d': 'NetgearArloCamera',

'b0:c5:54:03:c7:09': 'D-LinkDCS-5009LCamera',

'b4:79:a7:22:f9:fc': 'WinkHub',

'c0:56:27:53:09:6d': 'BelkinNetcam',

'c8:db:26:02:bb:bb': 'LogitechHarmonyHub',

'cc:b8:a8:ad:4d:04': 'AugustDoorbellCam',

'd0:03:4b:39:12:e3': 'AppleTV(4thGen)',

'd0:52:a8:63:47:9e': 'SamsungSmartThingsHub',

'd0:73:d5:12:84:d1': 'LIFXVirtualBulb',

'd4:90:9c:cc:62:42': 'AppleHomePod',

'd8:f7:10:c2:29:be': 'HarmonKardonInvoke',

'e4:71:85:25:ce:ec': 'SecurifiAlmond',

'e8:b2:ac:af:62:0f': 'iPad',

'f4:5e:ab:5e:c0:23': 'CasetaWirelessHub',

'f4:f2:6d:ce:9a:5d': 'GoogleOnHub',

IoT devices captures MAC and Device names

'00:17:88:24:76:ff': 'Hue-Device',

'00:1a:22:03:cb:be': 'MAXGateway',

'00:1a:22:05:c4:2e': 'HomeMaticPlug',

'00:24:e4:24:80:2a': 'Withings',

'00:b5:6d:06:08:ba': 'unknown',

'1c:5f:2b:aa:fd:4e': 'D-LinkDevice',

'20:f8:5e:ca:91:52': 'Aria',

'24:77:03:7c:ea:dc': 'unknown',

'28:b2:bd:c3:41:79': 'unknown',

'38:0b:40:ef:85:41': 'unknown',

'50:c7:bf:00:c7:03': 'TP-LinkPlugHS110',

'50:c7:bf:00:fc:a3': 'TP-LinkPlugHS100',

'3c:49:37:03:17:db': 'EdnetCam',

'3c:49:37:03:17:f0': 'EdnetCam',

'5c:cf:7f:06:d9:02': 'iKettle2',

'5c:cf:7f:07:ae:fb': 'SmarterCoffee',

'6c:72:20:c5:17:5a': 'D-LinkWaterSensor',

```
'74:da:38:23:22:7b': 'EdimaxPlug2101W',
'74:da:38:4a:76:49': 'EdimaxPlug1101W',
'74:da:38:80:79:fc': 'EdimaxCam',
'74:da:38:80:7a:08': 'EdimaxCam',
'84:18:26:7b:5f:6b': 'Lightify',
'90:8d:78:a8:e1:43': 'D-LinkSensor',
'90:8d:78:a9:3d:6f': 'D-LinkSwitch',
'90:8d:78:dd:0d:60': 'D-LinkSiren',
'94:10:3e:34:0c:b5': 'WeMoSwitch',
'94:10:3e:35:01:c1': 'WeMoSwitch',
'94:10:3e:41:c2:05': 'WeMoInsightSwitch',
'94:10:3e:42:80:69': 'WeMoInsightSwitch',
'94:10:3e:cd:37:65': 'WeMoLink',
'ac:cf:23:62:3c:6e': 'EdnetGateway',
'b0:c5:54:1c:71:85': 'D-LinkDayCam',
'b0:c5:54:25:5b:0e': 'D-LinkCam',
'bc:f5:ac:f4:c0:9d': 'unknown'}
# specify which dataset you want to create (training, validation and testing) .
files_add=train;file_name="Aalto_train_IoTDevID.csv"
files_add=validation;file_name="Aalto_validation_IoTDevID.csv"
files_add=test;file_name="Aalto_test_IoTDevID.csv"
def shannon(data):
    LOG_BASE = 2
   # We determine the frequency of each byte
   # in the dataset and if this frequency is not null we use it for the
   # entropy calculation
```

```
    dataSize = len(data)
    ent = 0.0
    freq={}
    for c in data:
        if c in freq:
            freq[c] += 1
        else:
            freq[c] = 1
   # to determine if each possible value of a byte is in the list
    for key in freq.keys():
        f = float(freq[key])/dataSize
        if f > 0: # to avoid an error for log(0)
            ent = ent + f * math.log(f, LOG_BASE)
    return -ent
def pre_entropy(payload):
    characters=[]
    for i in payload:
            characters.append(i)
    return shannon(characters)
            def port_class(port):
    port_list=[0,53,67,68,80,123,443,1900,5353,49153]
    if port in port_list:
        return port_list.index(port)+1
    elif 0 <= port <= 1023:
        return 11
    elif  1024 <= port <= 49151 :
```

```
        return 12
    elif 49152 <=port <= 65535 :
        return 13
    else:
        return 0
def port_1023(port):
    if 0 <= port <= 1023:
        return port
    elif  1024 <= port <= 49151 :
        return 2
    elif 49152 <=port <= 65535 :
        return 3
    else:
        return 0
header="pck_size,Ether_type,LLC_dsap,LLC_ssap,LLC_ctrl,EAPOL_version,EAPOL_type,EA
POL_len,IP_version,IP_ihl,IP_tos,IP_len,IP_flags,IP_Z,IP_MF,IP_id,IP_chksum,IP_DF,IP_frag,
IP_ttl,IP_proto,IP_options,IP_add_count,ICMP_type,ICMP_code,ICMP_chksum,ICMP_id,ICM
P_seq,ICMP_ts_ori,ICMP_ts_rx,ICMP_ts_tx,ICMP_ptr,ICMP_reserved,ICMP_length,ICMP_ne
xthopmtu,ICMP_unused,TCP_seq,TCP_ack,TCP_dataofs,TCP_reserved,TCP_flags,TCP_FIN,T
CP_SYN,TCP_RST,TCP_PSH,TCP_ACK,TCP_URG,TCP_ECE,TCP_CWR,TCP_window,TC
P_chksum,TCP_urgptr,TCP_options,UDP_len,UDP_chksum,DHCP_options,BOOTP_op,BOOT
P_htype,BOOTP_hlen,BOOTP_hops,BOOTP_xid,BOOTP_secs,BOOTP_flags,BOOTP_sname,
BOOTP_file,BOOTP_options,DNS_length,DNS_id,DNS_qr,DNS_opcode,DNS_aa,DNS_tc,DN
S_rd,DNS_ra,DNS_z,DNS_ad,DNS_cd,DNS_rcode,DNS_qdcount,DNS_ancount,DNS_nscount,
DNS_arcount,sport_class,dport_class,sport23,dport23,sport_bare,dport_bare,TCP_sport,TCP_dp
ort,UDP_sport,UDP_dport,payload_bytes,entropy,Label,MAC,Folder,Session\n"


#header="pck_size,Ether_type,LLC_dsap,LLC_ssap,LLC_ctrl,EAPOL_version,EAPOL_type,E
APOL_len,IP_version,IP_ihl,IP_tos,IP_len,IP_flags,IP_frag,IP_ttl,IP_proto,IP_options,IP_add_c
ount,ICMP_type,ICMP_code,ICMP_seq,ICMP_ts_ori,ICMP_ts_rx,ICMP_ts_tx,ICMP_gw,ICM
P_ptr,ICMP_reserved,ICMP_length,ICMP_nexthopmtu,ICMP_unused,TCP_dataofs,TCP_reserv
```

ed,TCP_flags,TCP_window,TCP_urgptr,UDP_len,BOOTP_op,BOOTP_htype,BOOTP_hlen,BOOTP_hops,BOOTP_secs,BOOTP_flags,DNS_length,DNS_qr,DNS_opcode,DNS_aa,DNS_tc,DNS_rd,DNS_ra,DNS_z,DNS_ad,DNS_cd,DNS_rcode,DNS_qdcount,DNS_ancount,DNS_nscount,DNS_arcount,sport,dport,entropy,Label,MAC\n"

#flags

#TCP

FIN = 0x01

SYN = 0x02

RST = 0x04

PSH = 0x08

ACK = 0x10

URG = 0x20

ECE = 0x40

CWR = 0x80

#IP

Z = 0x00

MF= 0x01

DF= 0x02

ipf=[]

tcpf=[]

import time

degistir=""

dst_ip_list={}

Ether_adresses=[]

IP_adresses=[]

label_count=0

```
filename="aalto.csv"
ths = open(filename, "w")
ths.write(header)
 for numero,i in enumerate (files_add):
    #header=header
    #ths.write(header)
    filename=str(i)
    filename=filename.replace("\\","/")
    #x = filename.rfind("/")
    filename=filename.split("/")
        #break
    pkt = rdpcap(i)
    #print("\n",numero,"/",len(files_add),"========"+ i[8:]+"========\n" )
    print("\n",numero,"/",len(files_add))
    sayaç=len(pkt)//20
    for jj, j in enumerate (pkt):
        try:
            if jj%sayaç==0:
                    sys.stdout.write("\r[" + "=" * int(jj//sayaç) +  " " * int((sayaç*20 - jj)// sayaç) + "]" +
str(5*jj//sayaç) + "%")
                    sys.stdout.flush()
        except:pass
        if j.haslayer(ARP):
            continue
        else:
            ts=j.time
```

```
try:pck_size=j.len
except:pck_size=0
if j.haslayer(Ether):
    if j[Ether].dst not in Ether_adresses:
        Ether_adresses.append(j[Ether].dst)
    if j[Ether].src not in Ether_adresses:
        Ether_adresses.append(j[Ether].src)
    Ether_dst=j[Ether].dst#Ether_adresses.index(j[Ether].dst)+1
    Ether_src=j[Ether].src#Ether_adj[Ether].dstresses.index(j[Ether].src)+1
    Ether_type=j[Ether].type
else:
    Ether_dst=0
    Ether_src=0
    Ether_type=0
if j.haslayer(ARP):
    ARP_hwtype=j[ARP].hwtype
    ARP_ptype=j[ARP].ptype
    ARP_hwlen=j[ARP].hwlen
    ARP_plen=j[ARP].plen
    ARP_op=j[ARP].op
    ARP_hwsrc=j[ARP].hwsrc
    ARP_psrc=j[ARP].psrc
    ARP_hwdst=j[ARP].hwdst
    ARP_pdst=j[ARP].pdst

    if j[ARP].hwsrc not in Ether_adresses:
```

```
        Ether_adresses.append(j[ARP].hwsrc)
    if j[ARP].psrc not in IP_adresses:
        IP_adresses.append(j[ARP].psrc)
    if j[ARP].hwdst not in Ether_adresses:
        Ether_adresses.append(j[ARP].hwdst)
    if j[ARP].pdst not in IP_adresses:
        IP_adresses.append(j[ARP].pdst)
    ARP_hwsrc=j[ARP].hwsrc#Ether_adresses.index(j[ARP].hwsrc)+1
    ARP_psrc=j[ARP].psrc#IP_adresses.index(j[ARP].psrc)+1
    ARP_hwdst=j[ARP].hwdst#Ether_adresses.index(j[ARP].hwdst)+1
    ARP_pdst=j[ARP].pdst#IP_adresses.index(j[ARP].pdst)+1
else:
    ARP_hwtype=0
    ARP_ptype=0
    ARP_hwlen=0
    ARP_plen=0
    ARP_op=0
    ARP_hwsrc=0
    ARP_psrc=0
    ARP_hwdst=0
    ARP_pdst=0
 if j.haslayer(LLC):
    LLC_dsap=j[LLC].dsap
    LLC_ssap=j[LLC].ssap
    LLC_ctrl=j[LLC].ctrl
else:
```

```
    LLC_dsap=0
    LLC_ssap=0
    LLC_ctrl=0
if j.haslayer(EAPOL):
    EAPOL_version=j[EAPOL].version
    EAPOL_type=j[EAPOL].type
    EAPOL_len=j[EAPOL].len
else:
    EAPOL_version=0
    EAPOL_type=0
    EAPOL_len=0
if j.haslayer(IP):
    IP_Z = 0
    IP_MF= 0
    IP_DF= 0
    IP_version=j[IP].version
    IP_ihl=j[IP].ihl
    IP_tos=j[IP].tos
    IP_len=j[IP].len
    IP_id=j[IP].id
    IP_flags=j[IP].flags

    IP_frag=j[IP].frag
    IP_ttl=j[IP].ttl
    IP_proto=j[IP].proto
    IP_chksum=j[IP].chksum
```

```
#if j[IP].options!=0:
IP_options=j[IP].options
if "IPOption_Router_Alert"   in str(IP_options):
   IP_options=1
else:IP_options=0
if j[Ether].src not in dst_ip_list:
   dst_ip_list[j[Ether].src]=[]
   dst_ip_list[j[Ether].src].append(j[IP].dst)
elif j[IP].dst not in dst_ip_list[j[Ether].src]:
   dst_ip_list[j[Ether].src].append(j[IP].dst)
IP_add_count=len(dst_ip_list[j.src])
#if IP_flags not in ipf: ipf.append(IP_flags)
if IP_flags & Z:IP_Z = 1
if IP_flags & MF:IP_MF = 1
if IP_flags & DF:IP_DF = 1
#if "Flag" in str(IP_flags):
   #IP_flags=str(IP_flags)
   #temp=IP_flags.find("(")
   #IP_flags=int(IP_flags[6:temp-1])
 if j[IP].src not in IP_adresses:
   IP_adresses.append(j[IP].src)
if j[IP].dst  not in IP_adresses:
   IP_adresses.append(j[IP].dst)
```

```
    IP_src=j[IP].src#IP_adresses.index(j[IP].src)+1
    IP_dst=j[IP].dst#IP_adresses.index(j[IP].dst)+1
else:
    IP_Z = 0
    IP_MF= 0
    IP_DF= 0
    IP_version=0
    IP_ihl=0
    IP_tos=0
    IP_len=0
    IP_id=0
    IP_flags=0
    IP_frag=0
    IP_ttl=0
    IP_proto=0
    IP_chksum=0
    IP_src=0
    IP_dst=0
    IP_options=0
    IP_add_count=0

if j.haslayer(ICMP):
    ICMP_type=j[ICMP].type
    ICMP_code=j[ICMP].code
    ICMP_chksum=j[ICMP].chksum
    ICMP_id=j[ICMP].id
```

```
    ICMP_seq=j[ICMP].seq
    ICMP_ts_ori=j[ICMP].ts_ori
    ICMP_ts_rx=j[ICMP].ts_rx
    ICMP_ts_tx=j[ICMP].ts_tx
    ICMP_gw=j[ICMP].gw
    ICMP_ptr=j[ICMP].ptr
    ICMP_reserved=j[ICMP].reserved
    ICMP_length=j[ICMP].length
    ICMP_addr_mask=j[ICMP].addr_mask
    ICMP_nexthopmtu=j[ICMP].nexthopmtu
    ICMP_unused=j[ICMP].unused
else:
    ICMP_type=0
    ICMP_code=0
    ICMP_chksum=0
    ICMP_id=0
    ICMP_seq=0
    ICMP_ts_ori=0
    ICMP_ts_rx=0
    ICMP_ts_tx=0
    ICMP_gw=0
    ICMP_ptr=0
    ICMP_reserved=0
    ICMP_length=0
    ICMP_addr_mask=0
    ICMP_nexthopmtu=0
```

```
    ICMP_unused=0
if j.haslayer(TCP):
    TCP_FIN = 0
    TCP_SYN = 0
    TCP_RST = 0
    TCP_PSH = 0
    TCP_ACK = 0
    TCP_URG = 0
    TCP_ECE = 0
    TCP_CWR = 0
    TCP_sport=j[TCP].sport
    TCP_dport=j[TCP].dport
    TCP_seq=j[TCP].seq
    TCP_ack=j[TCP].ack
    TCP_dataofs=j[TCP].dataofs
    TCP_reserved=j[TCP].reserved
    TCP_flags=j[TCP].flags

    TCP_window=j[TCP].window
    TCP_chksum=j[TCP].chksum
    TCP_urgptr=j[TCP].urgptr
    TCP_options=j[TCP].options
    TCP_options= str(TCP_options).replace(",","-")
    if TCP_options!="0":
        TCP_options=1
    else:
```

```
        TCP_options=0
    #if TCP_flags not in tcpf:
        #tcpf.append(TCP_flags)
    #print(TCP_options)
    if TCP_flags & FIN:TCP_FIN = 1
    if TCP_flags & SYN:TCP_SYN = 1
    if TCP_flags & RST:TCP_RST = 1
    if TCP_flags & PSH:TCP_PSH = 1
    if TCP_flags & ACK:TCP_ACK = 1
    if TCP_flags & URG:TCP_URG = 1
    if TCP_flags & ECE:TCP_ECE = 1
    if TCP_flags & CWR:TCP_CWR = 1
    #print(TCP_flags)
    #if "Flag" in str(TCP_flags):
        #TCP_flags=str(TCP_flags)
        #temp=TCP_flags.find("(")
        #TCP_flags=int(TCP_flags[6:temp-1])

else:
    TCP_sport=0
    TCP_dport=0
    TCP_seq=0
    TCP_ack=0
    TCP_dataofs=0
    TCP_reserved=0
    TCP_flags=0
```

```
    TCP_window=0
    TCP_chksum=0
    TCP_urgptr=0
    TCP_options=0
    TCP_options=0
    TCP_FIN = 0
    TCP_SYN = 0
    TCP_RST = 0
    TCP_PSH = 0
    TCP_ACK = 0
    TCP_URG = 0
    TCP_ECE = 0
    TCP_CWR = 0
if j.haslayer(UDP):
    UDP_sport=j[UDP].sport
    UDP_dport=j[UDP].dport
    UDP_len=j[UDP].len
    UDP_chksum=j[UDP].chksum
else:
    UDP_sport=0
    UDP_dport=0
    UDP_len=0
    UDP_chksum=0

if j.haslayer(DHCP):
    DHCP_options=str(j[DHCP].options)
```

```
            DHCP_options=DHCP_options.replace(",","-")
            if "message" in DHCP_options:
                x = DHCP_options.find(")")
                DHCP_options=int(DHCP_options[x-1])


        else:
            DHCP_options=0
         if j.haslayer(BOOTP):
            BOOTP_op=j[BOOTP].op
            BOOTP_htype=j[BOOTP].htype
            BOOTP_hlen=j[BOOTP].hlen
            BOOTP_hops=j[BOOTP].hops
            BOOTP_xid=j[BOOTP].xid
            BOOTP_secs=j[BOOTP].secs
            BOOTP_flags=j[BOOTP].flags
                                                                    #if "Flag" in
str(BOOTP_flags):BOOTP_flags=str(BOOTP_flags)temp=BOOTP_flags.find("(")
BOOTP_flags=int(BOOTP_flags[6:temp-1])
            BOOTP_ciaddr=j[BOOTP].ciaddr
            BOOTP_yiaddr=j[BOOTP].yiaddr
            BOOTP_siaddr=j[BOOTP].siaddr
            BOOTP_giaddr=j[BOOTP].giaddr
            BOOTP_chaddr=j[BOOTP].chaddr
            BOOTP_sname=str(j[BOOTP].sname)
            if BOOTP_sname!="0":
                BOOTP_sname=1
            else:
```

```
        BOOTP_sname=0
    BOOTP_file=str(j[BOOTP].file)
    if BOOTP_file!="0":
        BOOTP_file=1
    else:
        BOOTP_file=0
    BOOTP_options=str(j[BOOTP].options)
    BOOTP_options=BOOTP_options.replace(",","-")
    if BOOTP_options!="0":
        BOOTP_options=1
    else:
        BOOTP_options=0
else:
    BOOTP_op=0
    BOOTP_htype=0
    BOOTP_hlen=0
    BOOTP_hops=0
    BOOTP_xid=0
    BOOTP_secs=0
    BOOTP_flags=0
    BOOTP_ciaddr=0
    BOOTP_yiaddr=0
    BOOTP_siaddr=0
    BOOTP_giaddr=0
    BOOTP_chaddr=0
    BOOTP_sname=0
```

```
    BOOTP_file=0
    BOOTP_options=0
if j.haslayer(DNS):
    DNS_length=j[DNS].length
    DNS_id=j[DNS].id
    DNS_qr=j[DNS].qr
    DNS_opcode=j[DNS].opcode
    DNS_aa=j[DNS].aa
    DNS_tc=j[DNS].tc
    DNS_rd=j[DNS].rd
    DNS_ra=j[DNS].ra
    DNS_z=j[DNS].z
    DNS_ad=j[DNS].ad
    DNS_cd=j[DNS].cd
    DNS_rcode=j[DNS].rcode
    DNS_qdcount=j[DNS].qdcount
    DNS_ancount=j[DNS].ancount
    DNS_nscount=j[DNS].nscount
    DNS_arcount=j[DNS].arcount
    DNS_qd=str(j[DNS].qd).replace(",","-")
    if DNS_qd!="0":
       DNS_qd=1
    else:
       DNS_qd=0
    DNS_an=str(j[DNS].an).replace(",","-")
    if DNS_an!="0":
```

```
        DNS_an=1
    else:
        DNS_an=0
    DNS_ns=str(j[DNS].ns).replace(",","-")
    if DNS_ns!="0":
        DNS_ns=1
    else:
        DNS_ns=0
    DNS_ar=str(j[DNS].ar).replace(",","-")
    if DNS_ar!="0":
        DNS_ar=1
    else:
        DNS_ar=0
else:
    DNS_length=0
    DNS_id=0
    DNS_qr=0
    DNS_opcode=0
    DNS_aa=0
    DNS_tc=0
    DNS_rd=0
    DNS_ra=0
    DNS_z=0
    DNS_ad=0
    DNS_cd=0
    DNS_rcode=0
```

```
    DNS_qdcount=0
    DNS_ancount=0
    DNS_nscount=0
    DNS_arcount=0
    DNS_qd=0
    DNS_an=0
    DNS_ns=0
    DNS_ar=0
pdata=[]
if "TCP" in j:
    pdata = (j[TCP].payload)
if "Raw" in j:
    pdata = (j[Raw].load)
elif "UDP" in j:
    pdata = (j[UDP].payload)
elif "ICMP" in j:
    pdata = (j[ICMP].payload)
pdata=list(memoryview(bytes(pdata)))

if pdata!=[]:
    entropy=shannon(pdata)
else:
    entropy=0
payload_bytes=len(pdata)

sport_class=port_class(TCP_sport+UDP_sport)
```

dport_class=port_class(TCP_dport+UDP_dport)

sport23=port_1023(TCP_sport+UDP_sport)

dport23=port_1023(TCP_dport+UDP_dport)

sport_bare=TCP_sport+UDP_sport

dport_bare=TCP_dport+UDP_dport#port_class(TCP_dport+UDP_dport)

label=MAC_list[j.src]

Mac=j.src

if label=="unknown" and j.dst in ['3c:49:37:03:17:db', '3c:49:37:03:17:f0', '5c:cf:7f:06:d9:02', '5c:cf:7f:07:ae:fb'] :

label=MAC_list[j.dst] #Both outgoing and incoming packets were added because the packet numbers of these devices were very low.

Mac=j.dst

line=[pck_size,

Ether_type,

LLC_dsap,

LLC_ssap,

LLC_ctrl,

EAPOL_version,

EAPOL_type,

EAPOL_len,

IP_version,

IP_ihl,

IP_tos,

IP_len,

IP_flags,

IP_Z,

IP_MF,

IP_id,

IP_chksum,

IP_DF ,

IP_frag,

IP_ttl,

IP_proto,

IP_options,

IP_add_count,

ICMP_type,

ICMP_code,

ICMP_chksum,

ICMP_id,

ICMP_seq,

ICMP_ts_ori,

ICMP_ts_rx,

ICMP_ts_tx,

ICMP_ptr,

ICMP_reserved,

ICMP_length,

#ICMP_addr_mask,

ICMP_nexthopmtu,

ICMP_unused,

TCP_seq,

TCP_ack,

TCP_dataofs,

TCP_reserved,

TCP_flags,

TCP_FIN,

TCP_SYN,

TCP_RST,

TCP_PSH,

TCP_ACK,

TCP_URG,

TCP_ECE,

TCP_CWR ,

TCP_window,

TCP_chksum,

TCP_urgptr,

TCP_options,

UDP_len,

UDP_chksum,

DHCP_options,

BOOTP_op,

BOOTP_htype,

BOOTP_hlen,

BOOTP_hops,

BOOTP_xid,

BOOTP_secs,

BOOTP_flags,

BOOTP_sname,

BOOTP_file,

BOOTP_options,

DNS_length,

DNS_id,

DNS_qr,

DNS_opcode,

DNS_aa,

DNS_tc,

DNS_rd,

DNS_ra,

DNS_z,

DNS_ad,

DNS_cd,

DNS_rcode,

DNS_qdcount,

DNS_ancount,

DNS_nscount,

DNS_arcount,

sport_class,

dport_class,

sport23,

dport23,

sport_bare,

dport_bare,

TCP_sport,

TCP_dport,

UDP_sport,

```
            UDP_dport,
            payload_bytes,
            entropy,
            label,
            Mac,filename[2],filename[3][:-5]]

            #print(line)
            line=str(line).replace("[","")
            line=str(line).replace("]","")
            #line=str(line).replace("\',","-")
            line=str(line).replace(", ",",")
            line=str(line).replace("\'","")
            line=str(line).replace("None","0")
            if label!="unknown":
                ths.write(str(line)+"\n")

            #kk=line.split(",")
            #print(len(kk))
            #if len(kk)==112:
            #ths.write(line+"\n")

            #else:print(line)
        print("  - ",filename[2],"-",filename[3])

ths.close()
```

```
filename="Protocol.csv"


ths = open(filename, "w")
ths.write("Protocol\n")
for ii,i in enumerate(files_add):
      command="tshark -r "+i+" -T fields -e _ws.col.Protocol -E header=n -E separator=, -E quote=d -E occurrence=f > temp.csv"
   os.system(command)


   with open("temp.csv", "r") as file:
      while True:
         line=file.readline()
         if line=="":break
            if  "ARP" not in line:# this line eliminates the headers of CSV files and incomplete streams .
            ths.write(str(line))
         else:
            continue


   print("   {}  /  {}".format(ii,len(files_add)))
   os.remove("temp.csv")
ths.close()
filename="Protocol.csv"


ths = open(filename, "w")
ths.write("Protocol\n")
for ii,i in enumerate(files_add):
```

```
    command="tshark -r "+i+" -T fields -e _ws.col.Protocol -E header=n -E separator=, -E quote=d -E occurrence=f > temp.csv"
   os.system(command)


   with open("temp.csv", "r") as file:
      while True:
         line=file.readline()
         if line=="":break
             if  "ARP" not in line:# this line eliminates the headers of CSV files and incomplete streams .
            ths.write(str(line))
         else:
            continue


   print("  {}  /  {}".format(ii,len(files_add)))
   os.remove("temp.csv")
ths.close()
df1=pd.read_csv("aalto.csv")
#del df1["Protocol"]
df2=pd.read_csv("Protocol.csv")
df1["Protocol"]=df2["Protocol"]
df1.to_csv(file_name,index=None)
IP_flags = {'0': 1, '<Flag 0 ()>': 2, '<Flag 2 (DF)>': 3, '<Flag 1 (MF)>': 4}
TCP_flags = {'0': 1, '<Flag 2 (S)>': 2, '<Flag 18 (SA)>': 3, '<Flag 16 (A)>': 4, '<Flag 24 (PA)>': 5, '<Flag 25 (FPA)>': 6, '<Flag 17 (FA)>': 7, '<Flag 4 (R)>': 8, '<Flag 20 (RA)>': 9, '<Flag 194 (SEC)>': 10, '<Flag 1 (F)>': 11, '<Flag 152 (PAC)>': 12, '<Flag 144 (AC)>': 13,'<Flag 82 (SAE)>':14,'<Flag 49 (FAU)>':15}
BOOTP_flags = {'0': 1, '<Flag 0 ()>': 2, '<Flag 32768 (B)>': 3, 0: 1}
```

Protocol = {'EAPOL': 1, 'DHCP': 2, 'DNS': 3, 'TCP': 4, 'HTTP': 5, 'ICMP': 6, 'MDNS': 7, 'IGMPv3': 8, 'SSDP': 9, 'NTP': 10, 'HTTP/XML': 11, 'UDP': 12, 'SSLv2': 13, 'TLSv1': 14, 'ADwin Config': 15, 'TLSv1.2': 16, 'ICMPv6': 17, 'HTTP/JSON': 18, 'XID': 19, 'TFTP': 20, 'NXP 802.15.4 SNIFFER': 21, 'IGMPv2': 22, 'A21': 23, 'STUN': 24, 'Gearman': 25, '? KNXnet/IP': 26, 'UDPENCAP': 27, 'ESP': 28, 'SSL': 29, 'NBNS': 30, 'SIP': 31, 'BROWSER': 32, 'SABP': 33, 'ISAKMP': 34, 'CLASSIC-STUN': 35, 'Omni-Path': 36, 'XMPP/XML': 37, 'ULP': 38, 'TFP over TCP': 39, 'AX4000': 40, 'MIH': 41, 'DHCPv6': 42, 'TDLS': 43, 'RTMP': 44, 'TCPCL': 45, 'IPA': 46, 'GQUIC': 47, '0x86dd': 48, 'DB-LSP-DISC': 49, 'SSLv3': 50, 'LLMNR': 51, 'FB_ZERO': 52, 'OCSP': 53, 'IPv4': 54, 'STP': 55, 'SSH': 56, 'TLSv1.1': 57, 'KINK': 58, 'MANOLITO': 59, 'PKTC': 60, 'TELNET': 61, 'RTSP': 62, 'HCrt': 63, 'MPTCP': 64, 'S101': 65, 'IRC': 66, 'AJP13': 67, 'PMPROXY': 68, 'PNIO': 69, 'AMS': 70, 'ECATF': 71, 'LLC': 72, 'TZSP': 73,'RSIP':74,'SSHv2':75

,'DIAMETER':76

,'BFD Control':77

,'ASAP':78

,'DISTCC':79

,'DISTCC ':79

,'LISP':80

,'WOW':81

,'DTLSv1.0':82

,'SNMP':83

,'SMB2':84

,'SMB':85

,'NBSS':86

,'UDT':87,'HiQnet':88

,'POWERLINK/UDP':89

,'RTP':90

,'WebSocket':91

,'NAT-PMP':92

,'RTCP':93,'Syslog':94

,'Portmap':95

,'OpenVPN':96

,'BJNP':97

,'RIPv1':98

,'MAC-Telnet':99

,'ECHO':100

,'ASF':101

,'DAYTIME':102

,'SRVLOC':103

,'KRB4':104

,'CAPWAP-Control':105

,'XDMCP':106

,'Chargen':107

,'RADIUS':108

,'L2TP':109

,'DCERPC':110

,'KPASSWD':111

,'H264':112

,'FTP':113

,'FTP-DATA':114

,'ENIP':115

,'RIPv2':116

,'ICP':117,

"BACnet-APDU":118,

"IAX2":119,

"RX":120,

"HTTP2":121,

"SIP/SDP":122,

"TIME":123,

"Elasticsearch":124,

"RSL":125,

"TPCP":126,

"IPv6": 127 }

Folder= {'Aria': 'Aria', 'D-LinkCam': 'D-LinkCam', 'D-LinkDayCam': 'D-LinkDayCam', 'D-LinkDoorSensor': 'D-LinkDoorSensor', 'D-LinkHomeHub': 'D-LinkHomeHub', 'D-LinkSensor': 'D-LinkSensor', 'D-LinkSiren': 'D-LinkSiren', 'D-LinkSwitch': 'D-LinkSwitch', 'D-LinkWaterSensor': 'D-LinkWaterSensor', 'EdimaxCam1': 'EdimaxCam', 'EdimaxCam2': 'EdimaxCam', 'EdimaxPlug1101W': 'EdimaxPlug1101W', 'EdimaxPlug2101W': 'EdimaxPlug2101W', 'EdnetCam1': 'EdnetCam', 'EdnetCam2': 'EdnetCam', 'EdnetGateway': 'EdnetGateway', 'HomeMaticPlug': 'HomeMaticPlug', 'HueBridge': 'HueBridge', 'HueSwitch': 'HueSwitch', 'iKettle2': 'iKettle2', 'Lightify': 'Lightify', 'MAXGateway': 'MAXGateway', 'SmarterCoffee': 'SmarterCoffee', 'TP-LinkPlugHS100': 'TP-LinkPlugHS100', 'TP-LinkPlugHS110': 'TP-LinkPlugHS110', 'WeMoInsightSwitch': 'WeMoInsightSwitch', 'WeMoInsightSwitch2': 'WeMoInsightSwitch', 'WeMoLink': 'WeMoLink', 'WeMoSwitch': 'WeMoSwitch', 'WeMoSwitch2': 'WeMoSwitch', 'Withings': 'Withings'}

df=pd.read_csv(file_name)

df=df.replace({"IP_flags": IP_flags})

df=df.replace({"TCP_flags": TCP_flags})

df=df.replace({"BOOTP_flags": BOOTP_flags})

df=df.replace({"Protocol": Protocol})

df=df.replace({"Folder": Folder})

del df["Label"]

df["Label"]=df["Folder"]

del df["Folder"]

del df["Session"]

df1=pd.read_csv("Aalto_train_IoTDevID.csv")

```
df2=pd.read_csv("Aalto_validation_IoTDevID.csv")
frames = [df1, df2]
result = pd.concat(frames)
result.to_csv("Aalto_BIG_train_IoTDevID.csv",index=None)
```

FEATURE SELECTION USING GENETIC ALGORITHM

```
cols=feature=[#'IP_id',
#'ICMP_chksum',
#'ICMP_id',
#'TCP_seq',
#'TCP_ack',
#'TCP_chksum',
# 'UDP_chksum',
# 'DNS_id',
# 'BOOTP_xid',
'sport_class',
'dport_class',
#'sport23',
#'dport23',
#'sport_bare',
# 'dport_bare',
# 'TCP_sport',
# 'TCP_dport',
# 'UDP_sport',
# 'UDP_dport',
'pck_size',
'Ether_type',
```

'LLC_ssap',

'LLC_ctrl',

'EAPOL_version',

'EAPOL_type',

'EAPOL_len',

'IP_version',

'IP_ihl',

'IP_tos',

'IP_len',

'IP_flags',

'IP_DF',

'IP_ttl',

'IP_proto',

'IP_options',

'ICMP_type',

'ICMP_code',

'ICMP_seq',

'TCP_dataofs',

'TCP_flags',

'TCP_FIN',

'TCP_SYN',

'TCP_RST',

'TCP_PSH',

'TCP_ACK',

'TCP_window',

'TCP_options',

'UDP_len',

'DHCP_options',

'BOOTP_op',

'BOOTP_htype',

'BOOTP_hlen',

'BOOTP_secs',

'BOOTP_flags',

'BOOTP_sname',

'BOOTP_file',

'BOOTP_options',

'DNS_qr',

'DNS_aa',

'DNS_rd',

'DNS_ra',

'DNS_rcode',

'DNS_qdcount',

'DNS_ancount',

'DNS_nscount',

'DNS_arcount',

'payload_bytes',

'entropy',

'Protocol',

#"MAC",

'Label']

test='./Aalto_validation_IoTDevID.csv'

```
train='./Aalto_train_IoTDevID.csv'

df = pd.read_csv(train,usecols=cols)#,header=None )

X_train =df[df.columns[0:-1]]
#X_train=np.array(X_train)
df[df.columns[-1]] = df[df.columns[-1]].astype('category')
y_train=df[df.columns[-1]].cat.codes
df = pd.read_csv(test,usecols=cols)#,header=None )

X_test =df[df.columns[0:-1]]
#X_test=np.array(X_test)
df[df.columns[-1]] = df[df.columns[-1]].astype('category')
y_test=df[df.columns[-1]].cat.codes
 print(X_train.shape,
X_test.shape,
y_train.shape,
y_test.shape ,)

#training a logistics regression model
logmodel = DecisionTreeClassifier()
results=[]

#print("Accuracy = "+ str(accuracy_score(y_test,predictions)))
for i in range(100):
    logmodel.fit(X_train,y_train)
```

```
    predictions = logmodel.predict(X_test)
    results.append(f1_score(y_test,predictions,average= "macro"))
print ('%-30s %-30s' % ("MEAN","STD"))
print ('%-30s %-30s' % (np.mean(results),np.std(results)))
#defining various steps required for the genetic algorithm
# GA adapted from https://datascienceplus.com/genetic-algorithm-in-machine-learning-using-python/
def initilization_of_population(size,n_feat):
    population = []
    for i in range(size):
        chromosome = np.ones(n_feat,dtype=np.bool)
        chromosome[:int(0.3*n_feat)]=False
        np.random.shuffle(chromosome)
        population.append(chromosome)
    return population

def fitness_score(population):
    scores = []
    for chromosome in population:
        logmodel.fit(X_train.iloc[:,chromosome],y_train)
        predictions = logmodel.predict(X_test.iloc[:,chromosome])
        scores.append(f1_score(y_test,predictions,average= "macro"))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds,:][::-1])
```

```
def selection(pop_after_fit,n_parents):
    population_nextgen = []
    for i in range(n_parents):
        population_nextgen.append(pop_after_fit[i])
    return population_nextgen

def crossover(pop_after_sel):
    population_nextgen=pop_after_sel
    for i in range(len(pop_after_sel)):
        child=pop_after_sel[i]
        child[3:7]=pop_after_sel[(i+1)%len(pop_after_sel)][3:7]
        population_nextgen.append(child)
    return population_nextgen

def mutation(pop_after_cross,mutation_rate):
    population_nextgen = []
    for i in range(0,len(pop_after_cross)):
        chromosome = pop_after_cross[i]
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j]= not chromosome[j]
        population_nextgen.append(chromosome)
    #print(population_nextgen)
    return population_nextgen

def generations(size,n_feat,n_parents,mutation_rate,n_gen,X_train,
```

```
                    X_test, y_train, y_test):

    best_chromo= []
    best_score= []
    population_nextgen=initilization_of_population(size,n_feat)
    for i in range(n_gen):
        second=time.time()
        scores, pop_after_fit = fitness_score(population_nextgen)
        #print(scores[:2])
        zaman=time.time()-second
        print ('%-30s %-30s %-30s' % (np.mean(scores),np.std(scores),zaman))

        pop_after_sel = selection(pop_after_fit,n_parents)
        pop_after_cross = crossover(pop_after_sel)
        population_nextgen = mutation(pop_after_cross,mutation_rate)
        best_chromo.append(pop_after_fit[0])
        best_score.append(scores[0])
    return best_chromo,best_score
print ('%-30s %-30s %-30s' % ("MEAN","STD","TIME"))
 chromo,score=generations(size=200,n_feat=52,n_parents=120,mutation_rate=0.005,
                n_gen=100,X_train=X_train,X_test=X_test,y_train=y_train,y_test=y_test)
logmodel.fit(X_train.iloc[:,chromo[-1]],y_train)
predictions = logmodel.predict(X_test.iloc[:,chromo[-1]])
print("F1 Score score after genetic algorithm is= "+str(f1_score(y_test,predictions,average=
"macro")))
results=[]
```

```
for i in range(10):
    logmodel.fit(X_train.iloc[:,chromo[-1]],y_train)
    predictions = logmodel.predict(X_test.iloc[:,chromo[-1]])
    results.append(f1_score(y_test,predictions,average= "macro"))
print ('%-30s %-30s' % ("MEAN","STD"))
print ('%-30s %-30s' % (np.mean(results),np.std(results)))
sonuç=[]
for j in chromo:
    temp=X_train.iloc[:,j]
    temp=list(temp.columns)
    temp.append("Label")
    sonuç.append(temp)
print(sonuç)
results=[]
for i in range(10):
    logmodel.fit(X_train.iloc[:,chromo[-1]],y_train)
    predictions = logmodel.predict(X_test.iloc[:,chromo[-1]])
    results.append(f1_score(y_test,predictions,average= "macro"))
print ('%-30s %-30s' % ("MEAN","STD"))
print ('%-30s %-30s' % (np.mean(results),np.std(results)))
test='./Aalto_test_IoTDevID.csv'
train='./Aalto_train_IoTDevID.csv'
sayac=1
output_csv=dataset+str(sayac)+"_"+str(step)+"_"+str(mixed)+".csv"
features=[['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len',
'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code',
'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK',
```

'TCP_window', 'DHCP_options', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'ICMP_type', 'TCP_flags', 'TCP_FIN', 'TCP_window', 'TCP_options', 'UDP_len', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_secs', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_nscount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ctrl', 'EAPOL_type', 'IP_version', 'IP_tos', 'IP_len', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_flags', 'TCP_SYN', 'TCP_PSH', 'TCP_ACK', 'TCP_window', 'UDP_len', 'DHCP_options', 'BOOTP_op', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_qdcount', 'DNS_nscount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'ICMP_type', 'TCP_flags', 'TCP_FIN', 'TCP_window', 'TCP_options', 'UDP_len', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_nscount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs',

'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST',

'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_len', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_len', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_file', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_version', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen',

'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_arcount', 'sport_class', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_RST', 'TCP_ACK', 'TCP_window',

'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_PSH', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_DF',

'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_ihl', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_file', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_file', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_version', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_file', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_nscount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy',

'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_DF', 'IP_ttl', 'ICMP_type', 'ICMP_seq', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_rd', 'DNS_ra', 'DNS_rcode', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_flags', 'TCP_SYN', 'TCP_ACK', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_file', 'DNS_qr', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_PSH', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_nscount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_version', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl',

'EAPOL_version', 'EAPOL_len', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen', 'BOOTP_file', 'DNS_qr', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_hlen',

'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_len', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_options', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_ancount', 'DNS_nscount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_DF', 'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_aa', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'LLC_ssap', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_version', 'IP_ihl', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'ICMP_type', 'ICMP_code', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_PSH', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_htype', 'BOOTP_sname', 'BOOTP_file', 'DNS_qr', 'DNS_aa', 'DNS_rd', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label'], ['pck_size', 'Ether_type', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'EAPOL_len', 'IP_tos', 'IP_flags', 'IP_ttl', 'IP_proto', 'IP_options', 'ICMP_type', 'ICMP_seq', 'TCP_dataofs', 'TCP_flags', 'TCP_FIN', 'TCP_SYN', 'TCP_RST', 'TCP_ACK', 'TCP_window', 'DHCP_options', 'BOOTP_op', 'BOOTP_hlen', 'BOOTP_secs', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_ra', 'DNS_rcode', 'DNS_qdcount', 'DNS_ancount', 'DNS_arcount', 'dport_class', 'payload_bytes', 'entropy', 'Protocol', 'Label']]

step=1

mixed=False

dataset="./Aalto/False/"#dataset[2:-1]+"_"+str(step)

ml_list={"DT" :DecisionTreeClassifier()}

cm=False

```
repetition=10
for sayac,feature in enumerate(features):
    output_csv="./100/"+str(sayac)+"_"+str(len(feature))+".csv"
    dataset=str(sayac)+"__"+str(len(feature))
    feature.insert(0,"MAC")
    ML(train,test,output_csv,feature,step,mixed,dataset,ml_list,cm,repetition)
# these  result list taken from GA
all_features_score_aalto=0.7003946487780672
mean_of_gean_aalto=[
0.658976323,
0.690342384,
0.70377347,
0.710583201,
0.715113088,
0.716330202,
0.715825834,
0.717101207,
0.714730845,
0.717147627,
0.719827763,
0.71687345,
0.719555657,
0.71784551,
0.720511176,
0.71761989,
0.7208948,
```

0.719915251,

0.719071419,

0.717713946,

0.71852689,

0.719481632,

0.720086608,

0.720659253,

0.717996341,

0.720695689,

0.719423281,

0.719209018,

0.720199103,

0.719667945,

0.719187511,

0.720041875,

0.71853992,

0.720302949,

0.718784064,

0.720059881,

0.720123043,

0.717933308,

0.719474679,

0.720819512,

0.720453079,

0.719996107,

0.719966414,

0.717600793,

0.71963478,

0.720071005,

0.719987193,

0.720115493,

0.719696026,

0.721087154,

0.717968351,

0.719526983,

0.720521227,

0.719682923,

0.718161014,

0.71961584,

0.720972553,

0.71862179,

0.718258233,

0.718730554,

0.719514412,

0.720331836,

0.720606497,

0.720477354,

0.720010626,

0.717763388,

0.718967923,

0.718510293,

0.720149418,

0.719662958,

0.719015127,

0.719849898,

0.718018871,

0.721156234,

0.71992728,

0.719043396,

0.720337345,

0.719655518,

0.719187358,

0.720663234,

0.72010119,

0.720624521,

0.720547255,

0.718880147,

0.718824711,

0.7192739,

0.7199886,

0.719052036,

0.720159135,

0.717845337,

0.720058593,

0.720912641,

0.71953248,

0.720038212,

0.720384589,

```
0.72093063,
0.719072245,
0.720627628,
0.71839229,
0.718327975]
def find_the_way(path,file_format):
    files_add = []
    # r=root, d=directories, f = files
    for r, d, f in os.walk(path):
        for file in f:
            if file_format in file:
                files_add.append(os.path.join(r, file))
    return files_add
name_list=find_the_way('./100/','.csv')
name_list

flag=1
for i in name_list:
    df = pd.read_csv(i)
    col=i[6:-4]
    temp=pd.DataFrame(df.mean(),columns=[col])
    if flag:
        std=temp
        flag=0
    else:
        std[col]=temp[col]
```

```
aalto=std.T
aalto=aalto.sort_values(by=[' F1-score'])
aalto


aalto.to_csv("mean_100.csv")
best_scores_aalto=[all_features_score_aalto]
best_score_aalto=all_features_score_aalto
for i in aalto[' F1-score']:
    if i > best_score_aalto:
        best_score_aalto=i
    best_scores_aalto.append(best_score_aalto)


best_scores_aalto=best_scores_aalto[:-1]
sns.set_style("whitegrid")
graph_name="100feature selection merge using genetic algorithm.pdf"
my_xticks=list(range(len(aalto)))
import matplotlib.pylab as pylab
params = {'legend.fontsize': 'x-large',
          'figure.figsize': (10, 5),
         'axes.labelsize': 'x-large',
         'axes.titlesize':'x-large',
         'xtick.labelsize':'x-large',
         'ytick.labelsize':'x-large'}
pylab.rcParams.update(params)
#plt.figure(figsize=(10,10))
```

#plt.plot(my_xticks,tt['Acc'], linestyle='--', marker='.', color='b',label= "Separate Train & Test acc")

#plt.plot(my_xticks,cv['Acc'], linestyle='--', marker='.', color='r',label= "10-Fold CV acc")

plt.plot(my_xticks,aalto[' F1-score'], linestyle='', marker='.', color='b',label= "Generation Average Score")

plt.plot(my_xticks,best_scores_aalto, linestyle='-', marker='', color='g',label= "Best Score")

#plt.plot(my_xticks,mean_of_gean, linestyle='-', marker='', color='r',label= "Best Score")

#plt.plot(my_xticks,cv[' F1-score'], linestyle='-', marker='o', color='b',label= "10-Fold CV F1")

plt.axhline(all_features_score_aalto , color='r', label= "All Features Score")## hepsi

#plt.axhline(0.7382699395282875 , color='g', label= "F1 score using all features")

plt.title("Feature Selection Process Using GA in Aalto Dataset")

plt.legend(numpoints=1)

#plt.annotate(s='', xy=(89,0.70), xytext=(89,0.72), arrowprops=dict(arrowstyle='->'))

#plt.arrow(89,0.70, 89,0.72, head_width=0.1)

#plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")

#plt.arrow(x=89, y=0.650, dx=0, dy=0.11, width=.1)

plt.ylabel("F1 Score")

plt.xlabel("Generation Number")

plt.xticks(rotation=90)

plt.ylim([0.69, 0.7250])

plt.savefig(graph_name,bbox_inches='tight',format="pdf")#, dpi=400)

chosen_Feature_set=['pck_size', 'Ether_type', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_ihl', 'IP_tos', 'IP_len', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_code', 'TCP_dataofs', 'TCP_FIN', 'TCP_ACK', 'TCP_window', 'UDP_len', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_qdcount', 'dport_class', 'payload_bytes', 'entropy', 'Label']

ALGORITHM SELECTION

import warnings

```
warnings.filterwarnings("ignore")
%matplotlib inline
from numpy import array
from random import random
from sklearn import metrics
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB#57
from sklearn.naive_bayes import GaussianNB#52
from sklearn.naive_bayes import MultinomialNB#56
from sklearn.naive_bayes import CategoricalNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import Normalizer
from sklearn.svm import SVC
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import shuffle
import csv
import math
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import sklearn
import time

def target_name(name):
    df = pd.read_csv(name,usecols=["Label"])
    target_names=sorted(list(df["Label"].unique()))
    return target_names
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier

# Modify ml_list to combine RandomForest and KNN classifiers
ml_list = {
    "Combined": VotingClassifier(estimators=[
                ("RF", RandomForestClassifier(bootstrap=True, criterion="gini", max_depth=18.0, max_features=8, min_samples_split=9, n_estimators=96)),
                 ("KNN", KNeighborsClassifier(algorithm='brute', leaf_size=41, n_neighbors=48, weights='distance'))
    ], voting="hard")
```

```
}
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
altime=0
#def most_frequent(List):
#    return max(set(List), key = List.count)
def most_frequent(List):
    occurence_count = Counter(List)
        occurence_count={k: v for k, v in sorted(occurence_count.items(), key=lambda item: item[1],reverse=True)}
    big=list(occurence_count.values())
    big=big.count(big[0])
    return list(occurence_count.keys())[np.random.randint(big)]



def split(a, n):
    k, m = divmod(len(a), n)
    return (a[i * k + min(i, m):(i + 1) * k + min(i + 1, m)] for i in range(n))
def create_exception(df):
    exception_list=[]
    dominant_mac=[]
    for i in df['aggregated'].unique():
        k=df[df['aggregated']==i]
        for ii in ['MAC']:
```

```
            hist = {}
            for x in k[ii].values:
                hist[x] = hist.get(x, 0) + 1
            hist=dict(sorted(hist.items(), key=lambda item: item[1],reverse=True))
            temp=next(iter(hist))
            if temp not in dominant_mac:
                dominant_mac.append(temp)
            else:
                exception_list.append(temp)
    return exception_list

def merged(m_test,predict,step,mixed):
    second=time.time()
    mac_test=[]
    for q in m_test.index:
        mac_test.append(m_test[q])

    d_list=sorted(list(m_test.unique()))
    devices={}
    for q in d_list:
        devices[q]=[]


    new_y=[0]*len(m_test)

    for q,qq in enumerate (mac_test):
```

```
            devices[qq].append(q)
        for q in devices:
            a = [devices[q][j:j + step] for j in range(0, len(devices[q]), step)]
            for qq in a:
                step_list=[]
                for qqq in qq:
                    step_list.append(predict[qqq])
                add=most_frequent(list(step_list))
                for qqq in qq:
                    new_y[qqq]=add
        results=pd.DataFrame(m_test)
        results["aggregated"]=new_y
        results["normal"]=predict

        #MIXED METHOD
        if mixed:
            exception=create_exception(results)
            for q in exception:
                results.loc[results.MAC == q, 'aggregated'] = results['normal']
        return results["aggregated"].values,time.time()-second
    def score(altime,train_time,test_time,predict,y_test,class_based_results,i,cv,dname,ii):
        precision=[]
        recall=[]
        f1=[]
        accuracy=[]
        total_time=[]
```

```
    kappa=[]
    accuracy_b=[]

    rc=sklearn.metrics.recall_score(y_test, predict,average= "macro")
    pr=sklearn.metrics.precision_score(y_test, predict,average= "macro")
    f_1=sklearn.metrics.f1_score(y_test, predict,average= "macro")
    report = classification_report(y_test, predict, target_names=target_names,output_dict=True)
    cr = pd.DataFrame(report).transpose()
    if class_based_results.empty:
        class_based_results =cr
    else:
        class_based_results = class_based_results.add(cr, fill_value=0)
    precision.append(float(pr))
    recall.append(float(rc))
    f1.append(float(f_1))
    accuracy_b.append(balanced_accuracy_score( y_test,predict))
    accuracy.append(accuracy_score(y_test, predict))

    kappa.append(round(float(sklearn.metrics.cohen_kappa_score(y_test, predict,
    labels=None, weights=None, sample_weight=None)),15))
      print ('%-15s %-3s %-3s %-6s  %-5s %-5s %-5s %-5s %-8s %-5s %-8s %-8s%-8s%-8s' %
(dname,i,cv,ii[0:6],str(round(np.mean(accuracy),2)),str(round(np.mean(accuracy_b),2)),
        str(round(np.mean(precision),2)), str(round(np.mean(recall),2)),str(round(np.mean(f1),4)),

str(round(np.mean(kappa),2)),str(round(np.mean(train_time),2)),str(round(np.mean(test_time),2)
),str(round(np.mean(test_time)+np.mean(train_time),2)),str(round(np.mean(altime),2))))
```

```
lines=(str(dname)+","+str(i)+","+str(cv)+","+str(ii)+","+str(round(np.mean(accuracy),15))+","+s
tr(round(np.mean(accuracy_b),15))+","+str(round(np.mean(precision),15))+","+
str(round(np.mean(recall),15))+","+str(round(np.mean(f1),15))+","+str(round(np.mean(kappa),1
5))+","+str(round(np.mean(train_time),15))+","+str(round(np.mean(test_time),15))+","+str(altim
e)+"\n")
    return lines,class_based_results


def ML(loop1,loop2,output_csv,cols,step,mixed,dname):


    ths = open(output_csv, "w")
        ths.write("Dataset,T,CV,ML  algorithm,Acc,b_Acc,Precision,  Recall  ,  F1-score,  kappa
,tra-Time,test-Time,Al-Time\n")



    from sklearn.metrics import balanced_accuracy_score
    from sklearn.preprocessing import Normalizer


    for ii in ml_list:
        print ('%-15s %-3s %-3s %-6s  %-5s %-5s %-5s %-5s %-8s %-5s %-8s %-8s%-8s%-8s'%
                        ("Dataset","T","CV","ML  alg","Acc","b_Acc","Prec",  "Rec"  ,  "F1",  "kap"
,"tra-T","test-T","total","al-time"))
                                    class_based_results=pd.DataFrame()#""      #pd.DataFrame(0,
index=np.arange((len(target_names)+3)), columns=["f1-score","precision","recall","support"])
        cm=pd.DataFrame()
        cv=0
        if ii in ["GB","SVM"]: #for slow algorithms.
            repetition=100
        else:
```

```
    repetition=100
if ii in ["MLP"]: #for slow algorithms.
    repetition=10


for i in range(repetition):
     #TRAIN
    df = pd.read_csv(loop1,usecols=cols)
    try:df=df.replace({"Protocol": Protocol})
    except:pass
    m_train=df["MAC"]
    del df["MAC"]
    X_train =df[df.columns[0:-1]]
    X_train=np.array(X_train)
    df[df.columns[-1]] = df[df.columns[-1]].astype('category')
    y_train=df[df.columns[-1]].cat.codes
     #TEST
    df = pd.read_csv(loop2,usecols=cols)
    try:df=df.replace({"Protocol": Protocol})
    except:pass
    df = shuffle(df)
    m_test=df["MAC"]
    del df["MAC"]
    X_test =df[df.columns[0:-1]]
    X_test=np.array(X_test)
    df[df.columns[-1]] = df[df.columns[-1]].astype('category')
    y_test=df[df.columns[-1]].cat.codes
```

```
        results_y=[]
        cv+=1
        results_y.append(y_test)
        #machine learning algorithm is applied in this section
        clf = ml_list[ii]#choose algorithm from ml_list dictionary
        second=time.time()
        clf.fit(X_train, y_train)
        train_time=(float((time.time()-second)) )
        second=time.time()
        predict =clf.predict(X_test)
        test_time=(float((time.time()-second)) )
        if step==1:
            altime=0

lines,class_based_results=score(altime,train_time,test_time,predict,y_test,class_based_results,i,cv,dname,ii)
        else:
            predict,altime=merged(m_test,predict,step,mixed)

lines,class_based_results=score(altime,train_time,test_time,predict,y_test,class_based_results,i,cv,dname,ii)
        ths.write (lines)
        df_cm = pd.DataFrame(confusion_matrix(y_test, predict))
        if cm.empty:
            cm =df_cm
        else:
            cm = cm.add(df_cm, fill_value=0)
```

```
        class_based_results=class_based_results/repetition
        #print(class_based_results)
        class_based_results.to_csv("class_based_results.csv")
        if True:  # Change this line to 'if True:' to enable heatmap generation
            cm = cm // repetition
            graph_name = output_csv + ii + "_confusion matrix.pdf"
            plt.figure(figsize=(40, 28))
                sns.heatmap(cm, xticklabels=target_names, yticklabels=target_names, annot=True, fmt='g')
            plt.savefig(graph_name, bbox_inches='tight')  # , dpi=400)
            plt.show()
    #print(cm)
            print("\n\n\n")
ths.close()
feature= ['pck_size', 'Ether_type', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_ihl', 'IP_tos', 'IP_len', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_code', 'TCP_dataofs', 'TCP_FIN', 'TCP_ACK', 'TCP_window', 'UDP_len', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_qdcount', 'dport_class', 'payload_bytes', 'entropy',
"MAC",
'Label']
from sklearn.ensemble import VotingClassifier
test = 'Aalto_test_IoTDevID.csv'
train = 'Aalto_BIG_train_IoTDevID.csv'
dataset = "./Aalto/"
step = 1
mixed = False
sayac = 2
```

```
output_csv = dataset + str(sayac) + "_" + str(step) + "_" + str(mixed) + "100_knn_RF_combined.csv"

target_names = target_name(test)

ML(train, test, output_csv, feature, step, mixed, dataset[2:-1] + "_" + str(step))
```

PERFORMANCE EVALUATION

```
from sklearn.metrics import accuracy_score

from sklearn.metrics import average_precision_score

from sklearn.metrics import balanced_accuracy_score

from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

from sklearn.model_selection import RandomizedSearchCV

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import BernoulliNB#57

from sklearn.naive_bayes import GaussianNB#52

from sklearn.naive_bayes import MultinomialNB#56

from sklearn.naive_bayes import CategoricalNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural_network import MLPClassifier

from sklearn.preprocessing import Normalizer

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.utils import shuffle

import csv

import math

import matplotlib.pyplot as plt
```

```
from collections import Counter
import numpy as np
import os
import pandas as pd
import seaborn as sns
import sklearn
import time
def target_name(name):
    df = pd.read_csv(name)
    df=df.replace({"Label": new_labels})
    target_names=sorted(list(df[df.columns[-1]].unique()))
    return target_names
def folder(f_name): #this function creates a folder.
    try:
        if not os.path.exists(f_name):
            os.makedirs(f_name)
    except OSError:
        print ("Tthe folder could not be created!")


def find_the_way(path,file_format):
    files_add = []
    # r=root, d=directories, f = files
    for r, d, f in os.walk(path):
        for file in f:
            if file_format in file:
                files_add.append(os.path.join(r, file))
```

```
    return files_add
altime=0
#def most_frequent(List):
#    return max(set(List), key = List.count)
def most_frequent(List):
    occurence_count = Counter(List)
        occurence_count={k: v for k, v in sorted(occurence_count.items(), key=lambda item: item[1],reverse=True)}
    big=list(occurence_count.values())
    big=big.count(big[0])
    return list(occurence_count.keys())[np.random.randint(big)]
def split(a, n):
    k, m = divmod(len(a), n)
    return (a[i * k + min(i, m):(i + 1) * k + min(i + 1, m)] for i in range(n))
def create_exception(df):
    exception_list=[]
    dominant_mac=[]
    for i in df['aggregated'].unique():
        k=df[df['aggregated']==i]
        for ii in ['MAC']:
            hist = {}
            for x in k[ii].values:
                hist[x] = hist.get(x, 0) + 1
            hist=dict(sorted(hist.items(), key=lambda item: item[1],reverse=True))
            temp=next(iter(hist))
            if temp not in dominant_mac:
```

```
            dominant_mac.append(temp)
        else:
            exception_list.append(temp)
    return exception_list
def merged(m_test,predict,step,mixed):
    second=time.time()
    mac_test=[]
    for q in m_test.index:
        mac_test.append(m_test[q])

    d_list=sorted(list(m_test.unique()))
    devices={}
    for q in d_list:
        devices[q]=[]
    new_y=[0]*len(m_test)
    for q,qq in enumerate (mac_test):
        devices[qq].append(q)
    for q in devices:
        a = [devices[q][j:j + step] for j in range(0, len(devices[q]), step)]
        for qq in a:
            step_list=[]
            for qqq in qq:
                step_list.append(predict[qqq])
            add=most_frequent(list(step_list))
            for qqq in qq:
                new_y[qqq]=add
```

```
    results=pd.DataFrame(m_test)
    results["aggregated"]=new_y
    results["normal"]=predict

    #MIXED METHOD
    if mixed:
        exception=create_exception(results)
        for q in exception:
            results.loc[results.MAC == q, 'aggregated'] = results['normal']
    return results["aggregated"].values,time.time()-second
def score(altime,train_time,test_time,predict,y_test,class_based_results,i,cv,dname,ii):
    precision=[]
    recall=[]
    f1=[]
    accuracy=[]
    total_time=[]
    kappa=[]
    accuracy_b=[]
    rc=sklearn.metrics.recall_score(y_test, predict,average= "macro")
    pr=sklearn.metrics.precision_score(y_test, predict,average= "macro")
    f_1=sklearn.metrics.f1_score(y_test, predict,average= "macro")
    report = classification_report(y_test, predict, target_names=target_names,output_dict=True)
    cr = pd.DataFrame(report).transpose()
    if class_based_results.empty:
        class_based_results =cr
    else:
```

```
        class_based_results = class_based_results.add(cr, fill_value=0)
    precision.append(float(pr))
    recall.append(float(rc))
    f1.append(float(f_1))
    accuracy_b.append(balanced_accuracy_score( y_test,predict))
    accuracy.append(accuracy_score(y_test, predict))
    kappa.append(round(float(sklearn.metrics.cohen_kappa_score(y_test, predict,
    labels=None, weights=None, sample_weight=None)),15))
      print ('%-15s %-3s %-3s %-6s  %-5s %-5s %-5s %-5s %-8s %-5s %-8s %-8s%-8s%-8s' %
(dname,i,cv,ii[0:6],str(round(np.mean(accuracy),2)),str(round(np.mean(accuracy_b),2)),
        str(round(np.mean(precision),2)), str(round(np.mean(recall),2)),str(round(np.mean(f1),4)),

str(round(np.mean(kappa),2)),str(round(np.mean(train_time),2)),str(round(np.mean(test_time),2)
),str(round(np.mean(test_time)+np.mean(train_time),2)),str(round(np.mean(altime),2))))

lines=(str(dname)+","+str(i)+","+str(cv)+","+str(ii)+","+str(round(np.mean(accuracy),15))+","+s
tr(round(np.mean(accuracy_b),15))+","+str(round(np.mean(precision),15))+","+
str(round(np.mean(recall),15))+","+str(round(np.mean(f1),15))+","+str(round(np.mean(kappa),1
5))+","+str(round(np.mean(train_time),15))+","+str(round(np.mean(test_time),15))+","+str(altim
e)+"\n")
    return lines,class_based_results
from sklearn import tree
import graphviz
from graphviz import render
def ciz(name,model,feature_names,target_names):


    dot_data = tree.export_graphviz(model, out_file=None,
                                    feature_names=feature_names,
                                    class_names=target_names,
```

```
                    filled=True)


    # Draw graph
    #graph = graphviz.Source(dot_data)
    graph = graphviz.Source(dot_data,format='pdf')
    name=name[:-4]
    graph.render(name, view=True)
def ML(loop1,loop2,output_csv,cols,step,mixed,dname):
    maxy=0
    ths = open(output_csv, "w")
        ths.write("Dataset,T,CV,ML  algorithm,Acc,b_Acc,Precision,  Recall  ,  F1-score,  kappa
,tra-Time,test-Time,Al-Time\n"
    from sklearn.metrics import balanced_accuracy_score
    from sklearn.preprocessing import Normalizer
    for ii in ml_list:
        print ('%-15s %-3s %-3s %-6s  %-5s %-5s %-5s %-5s %-8s %-5s %-8s %-8s%-8s%-8s'%
                        ("Dataset","T","CV","ML  alg","Acc","b_Acc","Prec",  "Rec"  ,  "F1",  "kap"
,"tra-T","test-T","total","al-time"))
                                        class_based_results=pd.DataFrame()#""      #pd.DataFrame(0,
index=np.arange((len(target_names)+3)), columns=["f1-score","precision","recall","support"])
        cm=pd.DataFrame()
        cv=0
        if ii in ["GB","SVM"]: #for slow algorithms.
            repetition=10
        else:
            repetition=100
        for i in range(repetition):
```

```
#TRAIN
df = pd.read_csv(loop1,usecols=cols)
df=df.replace({"Label": new_labels})
m_train=df["MAC"]
del df["MAC"]
feature_names=df.columns
feature_names=feature_names[0:-1]
X_train =df[df.columns[0:-1]]
X_train=np.array(X_train)
df[df.columns[-1]] = df[df.columns[-1]].astype('category')
y_train=df[df.columns[-1]].cat.codes

#TEST
df = pd.read_csv(loop2,usecols=cols)
df=df.replace({"Label": new_labels})
df = shuffle(df, random_state=42)
m_test=df["MAC"]
del df["MAC"]
X_test =df[df.columns[0:-1]]
X_test=np.array(X_test)
df[df.columns[-1]] = df[df.columns[-1]].astype('category')
y_test=df[df.columns[-1]].cat.codes
results_y=[]
cv+=1
results_y.append(y_test)
#machine learning algorithm is applied in this section
```

```
        clf = ml_list[ii]#choose algorithm from ml_list dictionary
        second=time.time()
        clf.fit(X_train, y_train)
        train_time=(float((time.time()-second)) )
        second=time.time()
        predict =clf.predict(X_test)
        test_time=(float((time.time()-second)) )
        if step==1:
            altime=0

lines,class_based_results=score(altime,train_time,test_time,predict,y_test,class_based_results,i,cv,dname,ii)
        else:
            predict,altime=merged(m_test,predict,step,mixed)

lines,class_based_results=score(altime,train_time,test_time,predict,y_test,class_based_results,i,cv,dname,ii)
        ths.write (lines)
        f1=sklearn.metrics.f1_score(y_test, predict,average= "macro")
        if maxy<f1:
            maxy=f1
            chosen=clf
        df_cm = pd.DataFrame(confusion_matrix(y_test, predict))
        if cm.empty:
            cm =df_cm
        else:
            cm = cm.add(df_cm, fill_value=0)
```

```
        class_based_results=class_based_results/repetition
        print(class_based_results)
        class_based_results.to_csv("class_based_results.csv")
        if True:
            cm=cm//repetition
            cm.to_csv("cm_13.csv")
            graph_name=output_csv+ii+"_confusion matrix.pdf"
            plt.figure(figsize = (10,7))
                    sns.heatmap(cm,xticklabels=target_names, yticklabels=target_names, annot=True, fmt='g')
            plt.savefig(graph_name,bbox_inches='tight')#, dpi=400)
            plt.show()
            #print(cm)
            print("\n\n\n")
     ths.close()
feature= ['pck_size', 'Ether_type', 'LLC_ctrl', 'EAPOL_version', 'EAPOL_type', 'IP_ihl', 'IP_tos', 'IP_len', 'IP_flags', 'IP_DF', 'IP_ttl', 'IP_options', 'ICMP_code', 'TCP_dataofs', 'TCP_FIN', 'TCP_ACK', 'TCP_window', 'UDP_len', 'DHCP_options', 'BOOTP_hlen', 'BOOTP_flags', 'BOOTP_sname', 'BOOTP_file', 'BOOTP_options', 'DNS_qr', 'DNS_rd', 'DNS_qdcount', 'dport_class', 'payload_bytes', 'entropy',
"MAC", 'Label']
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier


# Modify ml_list to combine RandomForest and KNN classifiers
ml_list = {
    "Combined": VotingClassifier(estimators=[
```

```
            ("RF", RandomForestClassifier(bootstrap=True, criterion="gini", max_depth=18.0, max_features=8, min_samples_split=9, n_estimators=96)),
             ("KNN", KNeighborsClassifier(algorithm='brute', leaf_size=41, n_neighbors=48, weights='distance'))
   ], voting="hard")
}

dataset="./Aalto combined labels/"
folder(dataset)
new_labels={ 'D-LinkSensor':"D-LinkSensors",
'D-LinkSiren':"D-LinkSensors",
'D-LinkSwitch':"D-LinkSensors",
'D-LinkWaterSensor':"D-LinkSensors",
'EdimaxPlug1101W':"Edimax",
'EdimaxPlug2101W':"Edimax",
'TP-LinkPlugHS100':"TP-LinkPlugHS",
'TP-LinkPlugHS110':"TP-LinkPlugHS",
"HueBridge":'Hue-Device',
"HueSwitch":'Hue-Device',
'WeMoLink':"WeMos",
'WeMoSwitch':"WeMos", 'WeMoInsightSwitch':"WeMos"     }

def target_name(name):
   df = pd.read_csv(name)
   df=df.replace({"Label": new_labels})
   target_names=sorted(list(df[df.columns[-1]].unique()))
   return target_names
```

```
def target_name(name):
    df = pd.read_csv(name)
    df=df.replace({"Label": new_labels})
    target_names=sorted(list(df[df.columns[-1]].unique()))
    return target_names
test='Aalto_test_IoTDevID.csv'
train='Aalto_BIG_train_IoTDevID.csv'
dataset="./Aalto combined labels/"
folder(dataset)
mixed=False
step=1
sayac=1
output_csv=dataset+str(sayac)+"_"+str(step)+"_"+str(mixed)+".csv"
target_names=target_name(test)
ML(train,test,output_csv,feature,step,mixed,dataset[2:-1]+"_"+str(step))
```

CHAPTER 5

SNAPSHOTS

	pck_size	Ether_type	LLC_dsap	LLC_ssap	LLC_ctrl	EAPOL_version	EAPOL_type	EAPOL_len	IP_version	IP_ihl	...	dport_bare	TCP_sport
0	117	34958	0	0	0	1	3	117	0	0	...	0	0
1	95	34958	0	0	0	1	3	95	0	0	...	0	0
2	328	2048	0	0	0	0	0	0	4	5	...	67	0
3	328	2048	0	0	0	0	0	0	4	5	...	67	0
4	328	2048	0	0	0	0	0	0	4	5	...	67	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19927	313	2048	0	0	0	0	0	0	4	5	...	80	49154
19928	262	2048	0	0	0	0	0	0	4	5	...	80	49154
19929	40	2048	0	0	0	0	0	0	4	5	...	80	49154
19930	40	2048	0	0	0	0	0	0	4	5	...	80	49154
19931	40	2048	0	0	0	0	0	0	4	5	...	80	49154

Fig 1: Feature extraction process

- Fig 1 shows the feature extraction process from the PCAP dataset which contains 19932 rows and 97 features. This contains the metadata of the packets communicating between the IoTDevices.

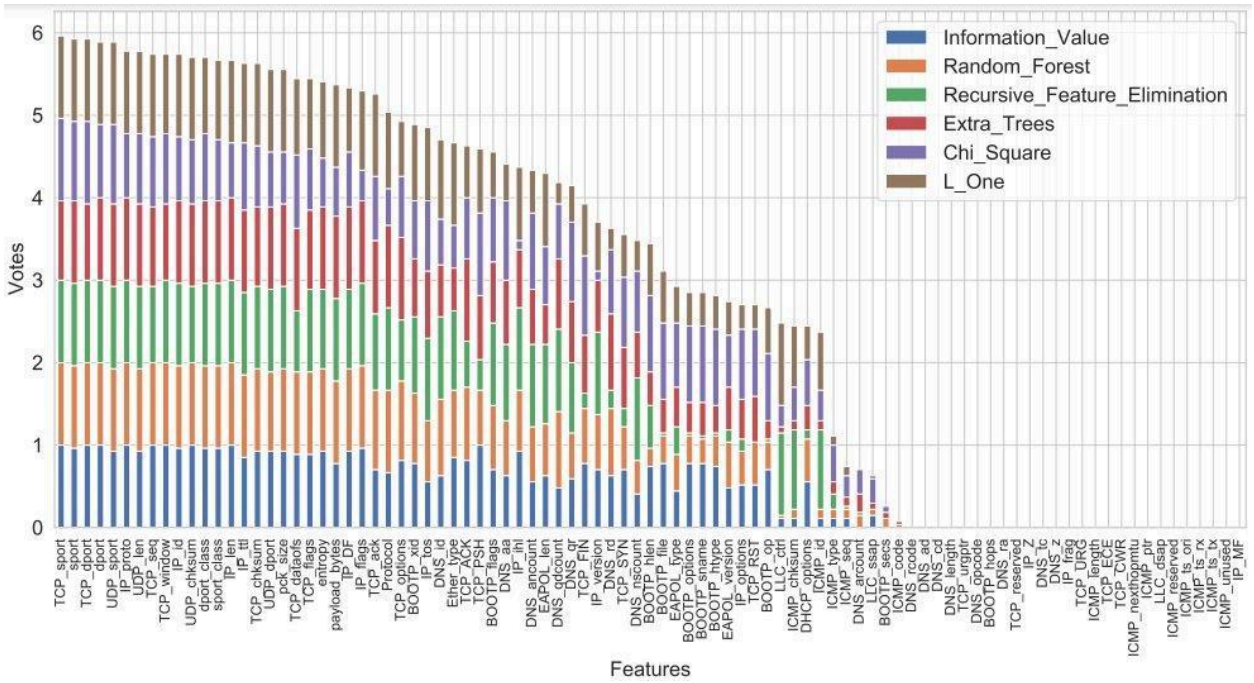


Fig 2: Output of voting classification

- The figure 2 shows the voting classification process of the features. Total of 6 algorithms are used to determine the maximum number of votes for each feature, and here TCP_sport has the maximum number of votes.

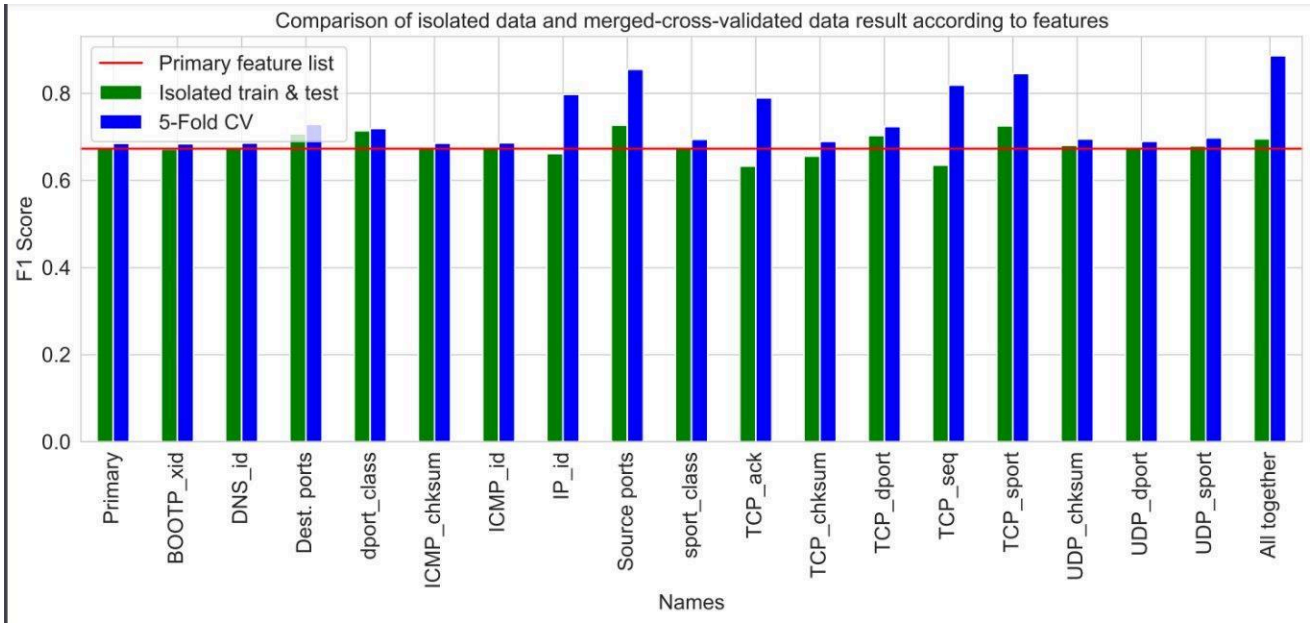


Fig 3: Comparison of isolated data and merged cross-validated data result according to the features

- This bar graph compares the F1 scores of different features using isolated data and merged-cross-validated data. The x-axis lists various features, and the y-axis represents the F1 score. Each feature has three bars showing its performance in different scenarios: Primary feature list, Isolated train & test, and 5-Fold CV.

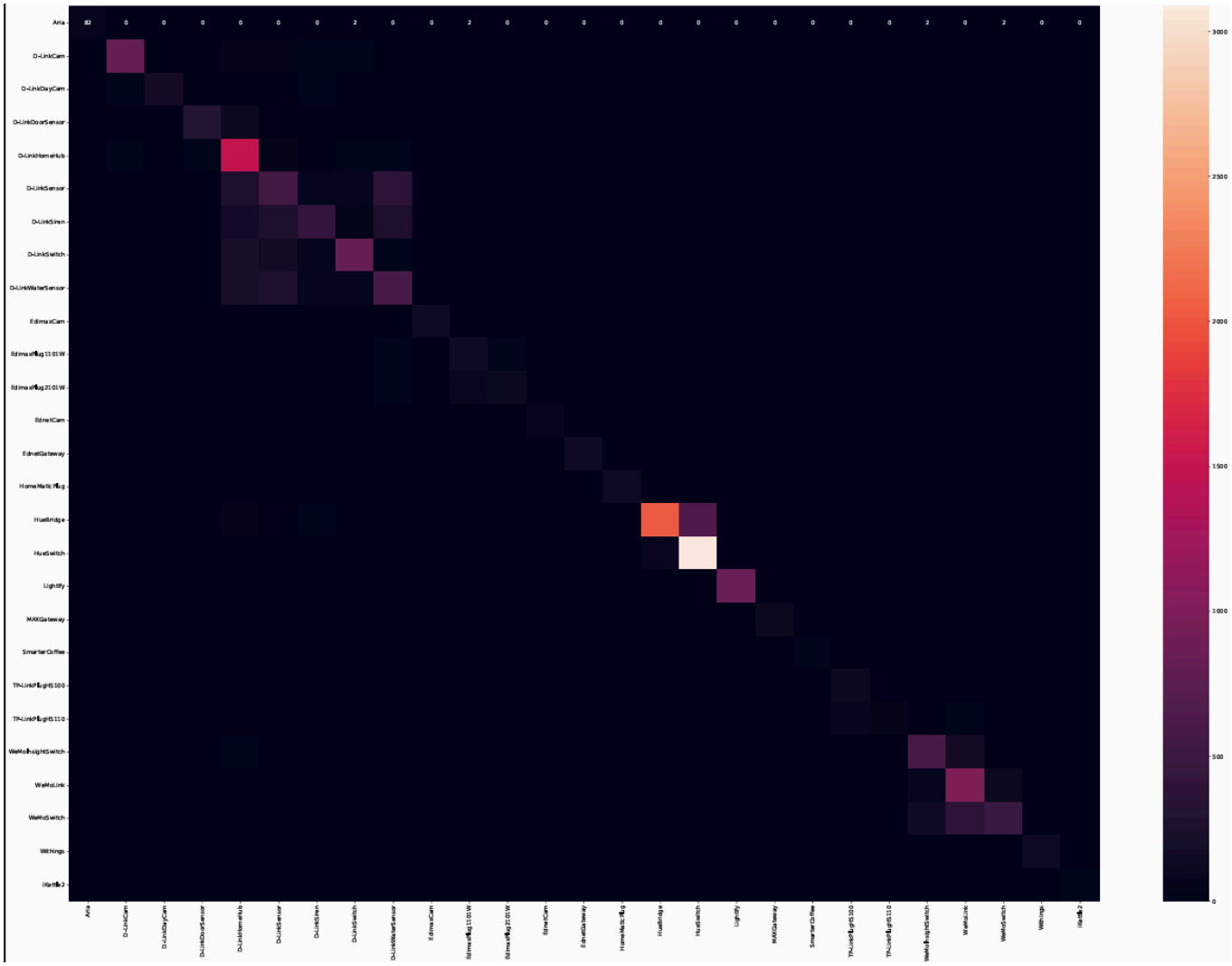


Fig 4: Confusion matrix of KNN and RF Combined (Aalto)

- Fig 4 shows the heatmap generated with the help of a combined algorithm in the Aalto dataset. Helps to find correlation between the features of the packets, and identify the patterns involved in it. In this HueBridge has a higher correlation than any other features.

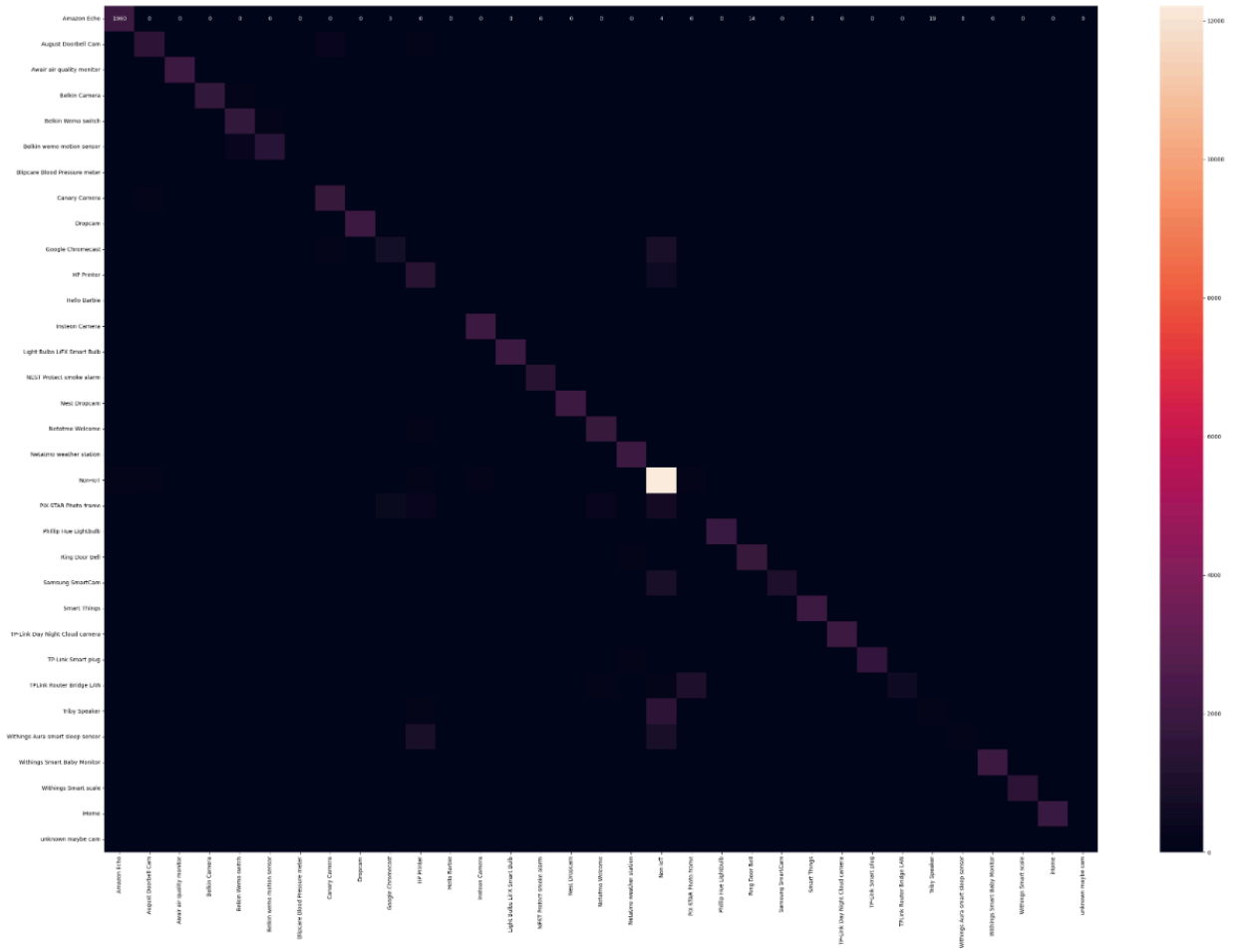


Fig 5: Confusion matrix of KNN and GB Combined (UNSW)

- Fig 5 shows the heatmap generated with the help of a combined algorithm in the UNSW dataset. Helps to find correlation between the features of the packets, and identify the patterns involved in it. Non_IoT has a higher correlation than any other features.

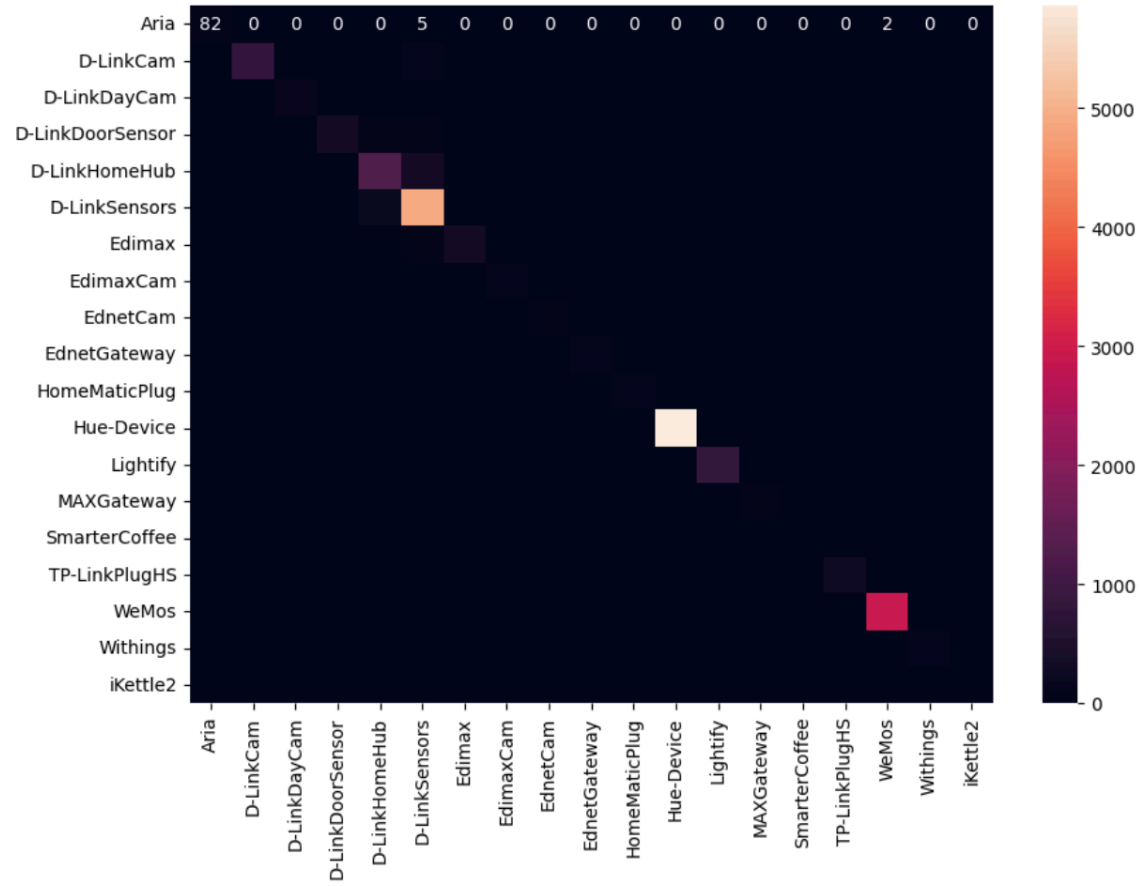


Fig 6: Confusion matrix for Performance evaluation (Combined datasets)

- Fig 6 shows the heatmap generated with the help of a combined algorithm in the Combined dataset. Helps to find correlation between the features of the packets, and identify the patterns involved in it. In this Hue-Device has a higher correlation than any other features.

CHAPTER 6

EFFICIENCY OF PROPOSED METHOD

In the existing method, altogether six algorithms were utilized, among which the Decision Tree Classifier (DT) achieved an accuracy of 70.5%. To surpass this performance, we employed an ensemble method with a different combination of algorithms. This amalgamation enhances resilience against noise and outliers present within the datasets, while also adapting to the heterogeneity of IoT devices. This flexibility makes it a preferred method for improving accuracy. Through various combinations, we discovered that the combination of Random Forest and KNN yielded a higher accuracy of 72% compared to the existing model.

CONCLUSION AND FUTURE WORK

The presented approach accommodates devices without MAC or IP addresses by initially combining packets and then employing a machine learning model. Machine learning utilizes aggregation to consider packet-level classification and, if available, IP or MAC addresses. Decision trees, chosen for their balance between inference time and predictive performance, are favored for real-time monitoring. Future endeavors entail developing an Intrusion Detection System (IDS) integrated with IoTDevID for threat detection, alongside an SDN-based management system. The goal is to deliver a robust IoT security solution applicable in practical scenarios. Subsequent development efforts will focus on augmenting the dataset with more data and a broader range of device types. Real-time implementation facilitates swift identification upon network connection, while merging with existing security systems enhances their functionality. Evaluation will encompass diverse network scenarios, addressing privacy concerns and threats such as botnets and denial-of-service attacks. User-friendly solutions aim to aid network managers and security experts in identifying suspicious devices effectively.

CHAPTER 7

REFERENCES

1. M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT sentinel: Automated device-type identification for security enforcement in IoT," in 37th Int. Conf. DCS. IEEE, 2017.
2. M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow," in 2016 11th Int. Conf. on ARES. IEEE, 2016, pp. 147–156.
3. M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow," in 2016 11th Int. Conf. on ARES. IEEE, 2016, pp. 147–156.
4. O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based IoT deployments," in Symp. on Security and Privacy. IEEE, 2019, pp. 1362–1380.
5. B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Iotsense: Behavioral fingerprinting of IoT devices," arXiv preprint arXiv:1804.03852, 2018.
6. A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "De tecting volumetric attacks on lot devices via sdn-based monitoring of mud activity," in 2019 ACM Symp. on SDN Research, 2019, pp. 36–48.
7. A. Aksoy and M. H. Gunes, "Automated IoT device identification using network traffic," in ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE, 2019, pp. 1–7.
8. F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, "Machine learning in iot security: Current solutions and future challenges," IEEE Com. Surveys & Tutorials, vol. 22, no. 3, pp. 1686–1721, 2020.
9. D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of things security: A top-down survey," Computer Networks, 2018.
10. B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," Journal of Network and Computer Applications, vol. 84, pp. 25–37, 2017.
11. M. Ozay, I. Esnaola, F. T. Y. Vural, S. R. Kulkarni, and H. V. Poor, "Machine learning methods for attack detection in the smart grid," IEEE transactions on neural networks and learning systems, vol. 27, no. 8, 2015.
12. "Internet try report Available: of things 2026," market 2019, size, accessed: growth IoT 2020-04-07. indus [Online].

https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307

13. N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, “Network intrusion detection for IoT security based on learning tech niques,” IEEE Communications Surveys & Tutorials, 2019.
14. M. Esmalifalak, L. Liu, N. Nguyen, R. Zheng, and Z. Han, “Detecting stealthy false data injection using machine learning in smart grid,” IEEE Systems Journal, vol. 11, no. 3, pp. 1644–1652, 2014.
15. A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, “Classifying IoT devices in smart environments using network traffic characteristics,” IEEE-TMC, 2018.