LAB REPORT

INTRUSION DETECTION SYSTEM IN INTERNET OF VEHICLES USING TREE BASED ALGORITHMS

Submitted by:

(802232006) ANISH THAKUR
(802232007) ANSH SINGH
(802232015) DEEPAK SHARMA
(802232021) HARJOT SINGH

ME 1st YEAR

CS1

Submitted to:

Dr. Seema Bawa

Professor (CSE)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala

JAN - MAY 2023

CERTIFICATE

This is to certify that the project report on, **"COVID-19 PREDICTION BY CNN"** being submitted by **ANISH THAKUR (802232006) ,ANSH SINGH (802232007),** DEEPAK SHARMA (802232015), HARJOT SINGH (802232021) computer science and Engineering, Thapar Institute of Engineering and Technology, Patiala for the fulfillment of the course requirement of **"MACHINE LEARNING (PCS206L)**is a bonafide record of work carried out by using conformity with the rules and regulations of the institute. The results presented in this report have not been submitted, in part or full, to any other University or Institute for the award of any degree or diploma.

Dated: **MAY 01, 2023**

ANISH THAKUR (802232006)
ANSH SINGH (802232007)
DEEPAK SHARMA (802232015)
HARJOT SINGH (802232021)

ACKNOWLEDGEMENT

The success and the outcome of this project required a lot of guidance and assistance and we are extremely privileged to have got this all along the completion of our project. A special gratitude to madam **Dr. Seema Bawa** and lab cordinator **Divisha Mam** whose contribution in stimulating suggestions and encouragement, helped us to coordinate our project as well as writing this report.

Our report would have been impossible without the guidance and help from our teachers, our institution, lab assistants and all the other members of our department. Last but not the least we would like to thank our classmates who have made valuable comment suggestions which gave us an inspiration to improve our assignment. We also thank our classmates for their help directly and indirectly to complete our assignment.

Signature

TABLE OF CONTENTS

ABSTRACT

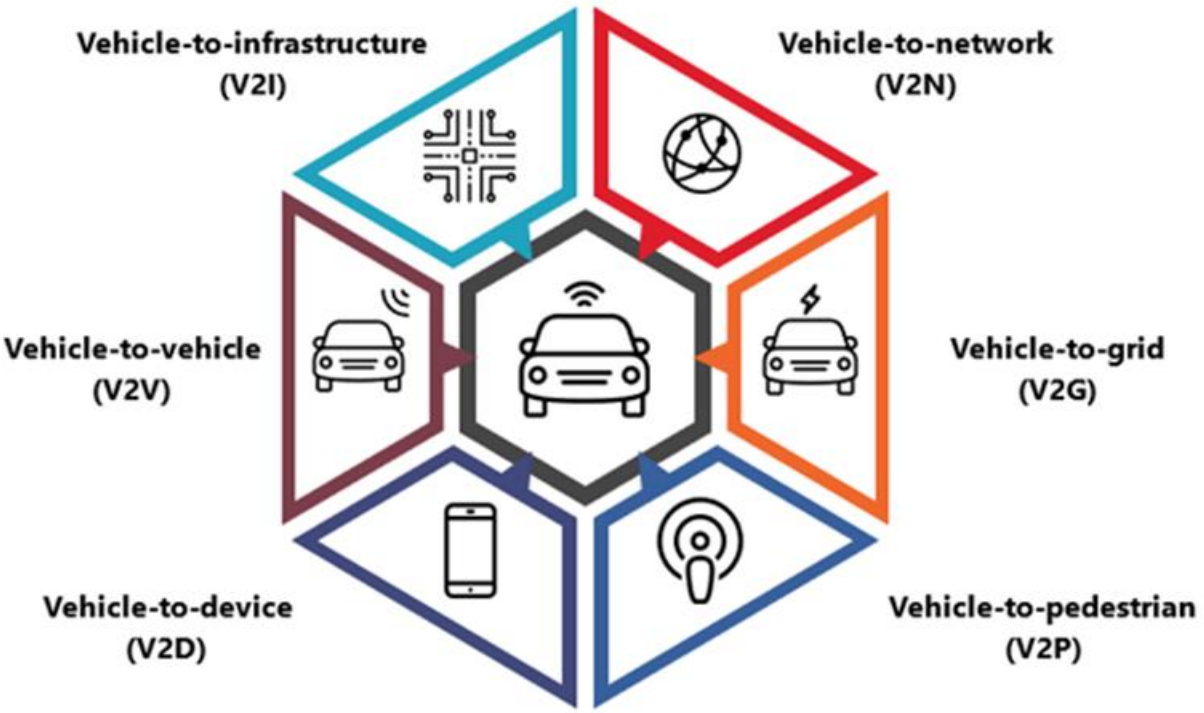
The use of autonomous vehicles (AVs) is a promising technology in Intelligent Transportation Systems (ITSs) to improve safety and driving efficiency. Vehicle-to-everything (V2X) technology enables communication among vehicles and other infrastructures. However, AVs and

Internet of Vehicles (IoV) are vulnerable to different types of cyber-attacks such as denial of service, spoofing, and sniffing attacks. In this paper, an intelligent intrusion detection system (IDS) is proposed based on tree-structure machine learning models. The results from the implementation of the proposed intrusion detection system on standard data sets indicate that the system has the ability to identify various cyber-attacks in the AV networks. Furthermore, the proposed ensemble learning and feature selection approaches enable the proposed system to achieve high

1. INTRODUCTION

With more vehicles, devices, and infrastructures involved, the conventional vehicular ad hoc networks (VANETs) are gradually evolving into the Internet of Vehicles (IoV) In intelligent transportation systems (ITSs), VANETs enable wireless communications between vehicles and devices, then transform the vehicles and devices into wireless routers or mobile nodes [2].

Autonomous vehicles (AV) technology is a fast-paced technology which provides an ideal solution to reduce traffic collisions and related costs.



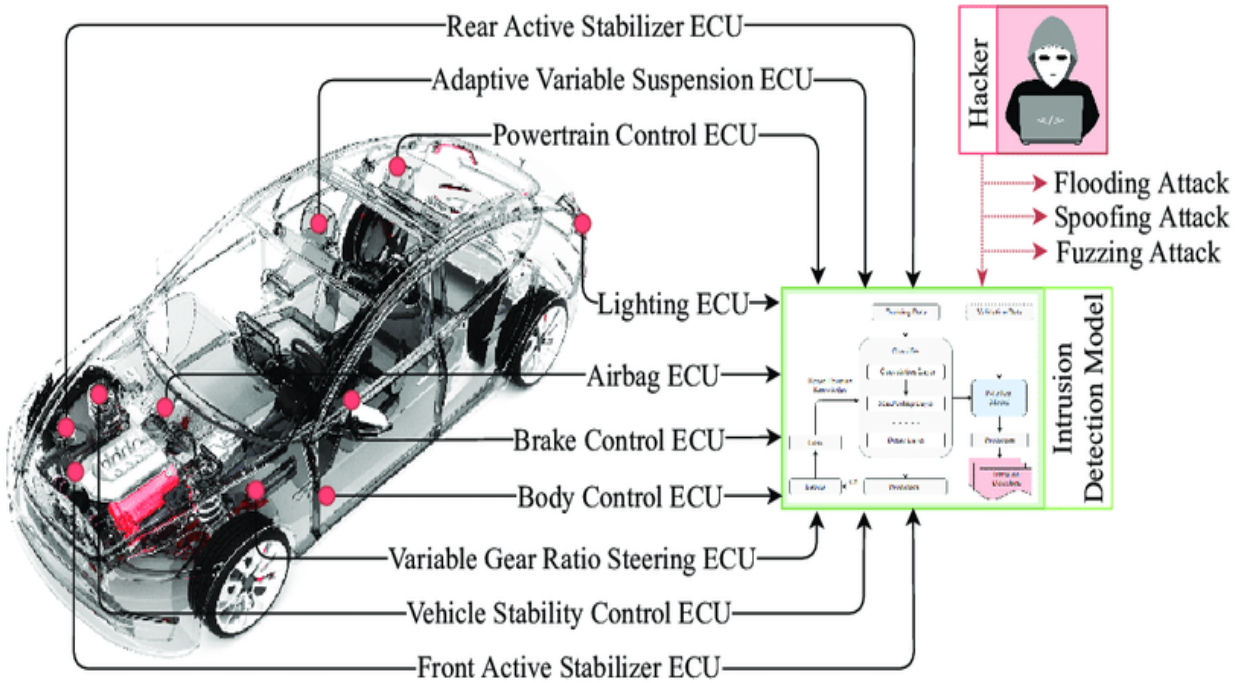
Vehicle-to-everything (V2X) technology is at the core of AV, consisting of

- Vehicle-to-Vehicle (V2V),
- Vehicle-to-Pedestrian (V2P),
- Vehicle-to-Infrastructure (V2I),
- Vehicle-to-Network (V2N) technologies

to enable local and wide area cellular network communications between vehicles, pedestrians, and infrastructures. V2X technology, with the means of wireless communications, aims to involve more Internet of things (IoT) devices.

1.1 NETWORK THREATS

However, some of these devices lack security mechanisms such as firewalls and gateways . AVs are prone to network threats with severe consequences because attacking or maliciously controlling the vehicles on the road poses serious threat to human lives.



1.1.1 EXTERNAL COMMUNICATION THREATS

The potential networking threats include the following attacks. A common type of attack is

- **Denial of services (DoS) attacks** launched by sending a large number of irrelevant messages or requests to occupy the node and take control of the resources of the vehicle
- The attackers perform spoofing attacks such as **GPS spoofing** to masquerade as legitimate users and provide the nodes with fake GPS information
- **Sniffing attacks** such as port scan attacks is another type of attack, which is launched to obtain confidential or sensitive data of the vehicles systems and users
- **brute-force attacks** are launched by attackers to crack passwords in vehicle networks
- To intrude into the web interface of vehicles or servers, web attacks including **SQL injection attacks** and **cross-site scripting (XSS)** can also be implemented by the attackers [

COMMON ATTACK TYPES ON EXTERNAL VEHICULAR NETWORKS

Attack Type	Description and IoV Scenarios
DoS	Send a large number of requests to exhaust the compromised nodes' resources, causing vehicle unavailability or accidents.
GPS Spoofing	Masquerade as authorized IoV users to provide a node with false information, like false geographic information, therefore causing fake evidence, event delay, or property losses.
Jamming	Jam signals to prevent legitimate IoV devices from communicating with connected vehicles.
Sniffing	Capture vehicular network packets to steal confidential or sensitive information of vehicles, users, or enterprises.
Brute-force	Crack passwords in vehicle systems to take control of vehicles or machines and perform malicious actions.
Botnets	Infect multiple connected vehicles and IoV devices with Bot viruses to breach them and launch other attacks.
Infiltration	Traverse the compromised vehicle systems and create a backdoor for future attacks.
Web Attack	Hack IoV servers or web interfaces of connected vehicles to gain confidential information or perform malicious actions.

1.1.2 INTRA-VEHICLE COMMUNICATION THREATS

Apart from external communication threats, AVs are also prone to intra-vehicle communication attacks. Controller Area Network (CAN)is a bus communication protocol which enables in-vehicle communications between all Electronic Control Units (ECUs).

It provides an efficient error detection mechanism for stable transmission and can reduce wiring cost, weight, and complexity . However, all the ECUs communicate with each other through the CAN bus, which makes the ECUs vulnerable to various attacks if the CAN bus is compromised. In CAN bus communications, attackers can inject malicious messages to monitor the network traffic or launch other hostile attacks and the nodes will inevitably deal with the messages without validating their origins.

The message injection attacks on CAN bus can be classified by their aims.

- **DoS and spoofing attacks** can also be launched on the CAN bus to occupy the resources or provide malicious information such as gear and RPM (revolutions per minute) information.
- **fuzzy attacks** are another common type of attack launched on CAN bus by injecting arbitrary messages to cause the vehicles to show unintended states or malfunction

All the above vulnerabilities and threats call for a robust protection system that can repel possible attacks that pose threats to intra-vehicle and external communications in AV systems.

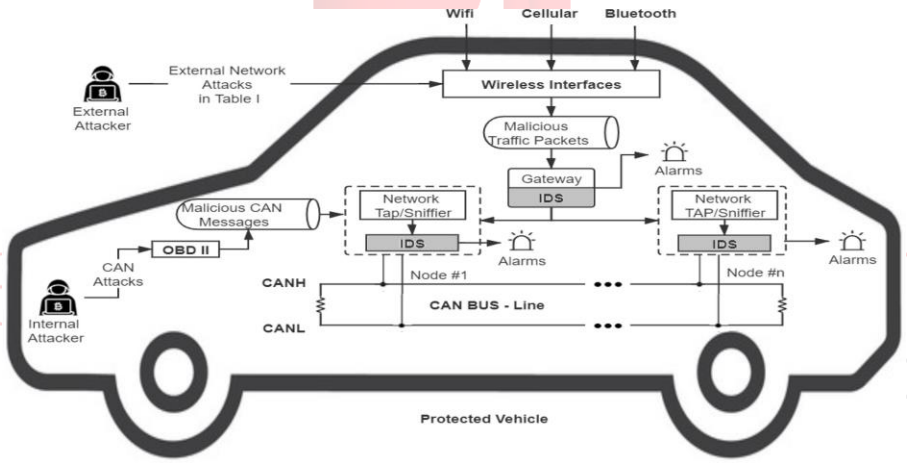
ROADMAP

Intrusion detection systems (IDSs) are an effective security mechanism to identify the abnormal information and attacks through the network traffic data during the communication between vehicles and other devices. Intrusion detection is often considered as a classification problem; machine learning (ML) methods have been widely used to develop IDSs. Attack detection that can be applied to not only Controller Area Network (CAN) bus of AVs but also on general IoVs.

ALGORITHMS

The proposed IDS utilizes tree-based ML algorithms including

- decision tree (DT),
- random forest (RF),
- extra trees (ET),
- Extreme Gradient Boosting (XGBoost).



A qualified IDS needs to not only achieve a **high detection rate**, but also have a **low computational cost.** Therefore, an ensemble learning model, namely stacking, is used to improve accuracy and the feature selection methods are also implemented to reduce computational time. The performance of the proposed IDS is evaluated using multiple standard open-source data sets with results showing high accuracy in detecting intrusions.

CONTRIBUTION

This report makes the following contributions:

- Surveys the vulnerabilities and potential attacks in CAN and AV networks

- Proposes an intelligent IDS for both AV and general networks by using the tree structure ML and ensemble learning methods.
- Comprehensive framework to prepare network traffic data for purpose of IDS development.
- Proposes an averaging **feature selection method** using **tree structure ML models** to improve the efficiency of the proposed IDS and to perform an analysis of network attributes and attacks for network monitoring uses.

This report is organized as follows:

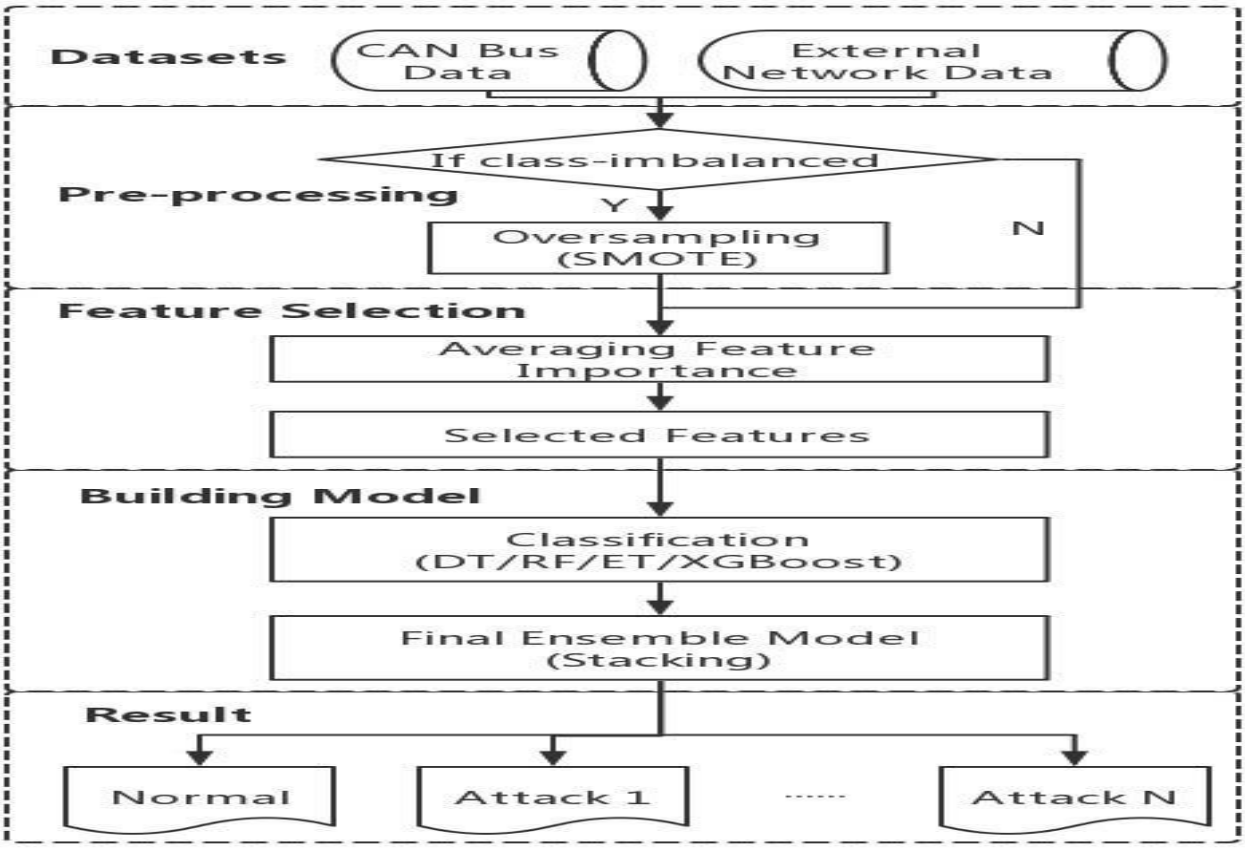
- presents the system overview of the proposed IDS and its architecture.
- discusses the proposed IDS framework in detail.
- provides the results, performance comparison and feature analysis.

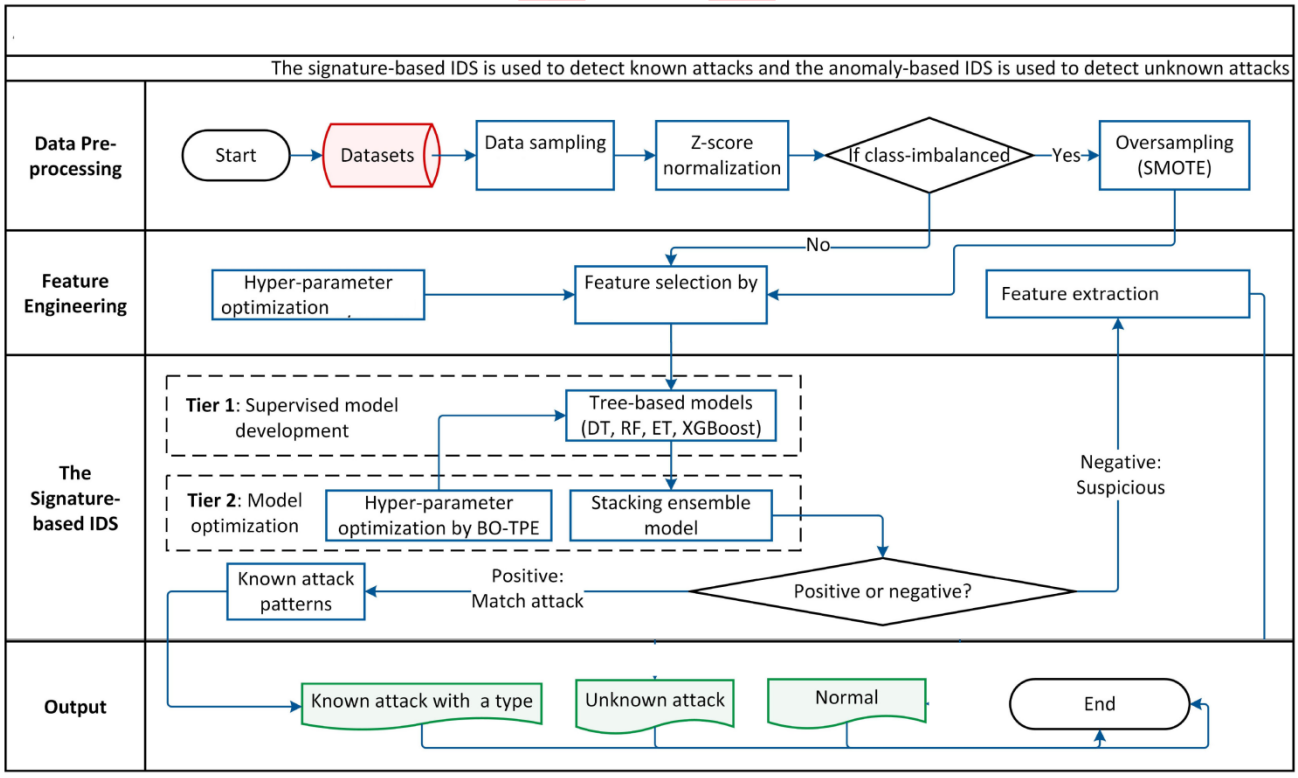
PROBLEM STATEMENT

Due to the fact that the AV systems are vulnerable to many types of network threats through different communication mediums, a comprehensive IDS system is proposed to be implemented for both intra-vehicle and external communication networks. This is done to better protect not only the vehicle components but also all the IoT devices involved in the entire IoV.

The proposed IDS should be able to detect various common intrusions launched on CAN bus and external networks. The main attacks to be considered in this work include DoS attacks on both the intra-vehicle and external communication networks, fuzzy attacks and spoofing attacks on the CAN-bus, sniffing, brute force and web attacks launched on the external networks. The IDS should have a high detection rate to effectively identify most of the attacks and low computational time to improve efficiency.

2.. IDS SYSTEM OVERVIEW AND ARCHITECTURE





To detect threats on the CAN bus and secure it, the IDS can be placed on the top of the CAN bus to process every transmitted message and ensure the nodes are not compromised

3 THE PROPOSED IDS FRAMEWORK

3.1 Library Used

```
import glob
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_recall_fscore_support, f1_score
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from xgboost import plot_importance
from zipfile import ZipFile
```

3.2. DATA PRE-PROCESSING

The first step to develop an IDS is to collect sufficient amount of network traffic data under both the normal state and the abnormal state caused by different types of attacks. The data can be collected by the packet sniffers, but they should have suitable network attributes, or named network features, for the purpose of IDS development.

Preprocessing

```
[ ] ds_path = f"{path}/MachineLearningCVE"
    df = pd.concat([pd.read_csv(f, sep=',') for f in glob.glob(ds_path + "/*.csv")],ignore_index=True)
    df.head()
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...
0	80	38308	1	1	6	6	6	6	6.000000	0.000000	...
1	389	479	11	5	172	326	79	0	15.636364	31.449238	...
2	88	1095	10	6	3150	3150	1575	0	315.000000	632.561635	...
3	389	15206	17	12	3452	6660	1313	0	203.058823	425.778474	...
4	88	1092	9	6	3150	3152	1575	0	350.000000	694.509719	...

5 rows × 79 columns

```
[ ] df.shape
```

```
(2830743, 79)
```

```
[ ] df.isnull().sum().sum()
```

```
1358
```

External networks, since they belong to general networks and are prone to various regular network threats, the data with more network attributes should be collected to develop an effective IDS that can detect various types of cyber-attacks.

```
BENIGN                          2273097
DoS Hulk                         231073
PortScan                         158930
DDoS                             128027
DoS GoldenEye                     10293
FTP-Patator                        7938
SSH-Patator                        5897
DoS slowloris                      5796
DoS Slowhttptest                   5499
Bot                                1966
Web Attack  Brute Force           1507
Web Attack  XSS                    652
Infiltration                        36
Web Attack  Sql Injection           21
Heartbleed                          11
Name: Label, dtype: int64
```

```
df['Label'] = df['Label'].replace(['DoS Hulk', 'DDoS', 'DoS GoldenEye', 'DoS slowloris', 'DoS Slowhttptest', 'Heartbleed'], 'DoS')
df['Label'] = df['Label'].replace(['Web Attack  Brute Force', 'Web Attack  XSS', 'Web Attack  Sql Injection'], 'WebAttack')
df['Label'] = df['Label'].replace(['FTP-Patator', 'SSH-Patator'], 'BruteForce')
```

```
df['Label'].value_counts()
```

```
BENIGN          2273097
DoS              380699
PortScan         158930
BruteForce        13835
WebAttack          2180
Bot                1966
Infiltration         36
Name: Label, dtype: int64
```

```
df_minor = df[(df['Label']=='WebAttack')|(df['Label']=='Bot')|(df['Label']=='Infiltration')]
df_BENIGN = df[(df['Label']=='BENIGN')]
df_BENIGN = df_BENIGN.sample(n=None, frac=0.01, replace=False, weights=None, random_state=None, axis=0)
df_DoS = df[(df['Label']=='DoS')]
df_DoS = df_DoS.sample(n=None, frac=0.05, replace=False, weights=None, random_state=None, axis=0)
df_PortScan = df[(df['Label']=='PortScan')]
df_PortScan = df_PortScan.sample(n=None, frac=0.05, replace=False, weights=None, random_state=None, axis=0)
df_BruteForce = df[(df['Label']=='BruteForce')]
df_BruteForce = df_BruteForce.sample(n=None, frac=0.2, replace=False, weights=None, random_state=None, axis=0)
```

```
df_s = pd.concat([df_BENIGN, df_DoS, df_PortScan, df_BruteForce, df_minor], ignore_index=True)
df_s
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	min_seg_size_forward	Acti Me
0	53	527	1	1	61	89	61	61	61.00	0.000000	...	20	0
1	443	85623762	16	21	3180	6972	2781	0	198.75	690.827764	...	20	164164
2	53	193	2	2	46	46	23	23	23.00	0.000000	...	32	0
3	53	47513	2	2	74	444	37	37	37.00	0.000000	...	32	0
4	53	24000	2	2	70	306	35	35	35.00	0.000000	...	32	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
56656	8080	1030558	3	3	0	18	0	0	0.00	0.000000	...	28	0
56657	8080	1008819	3	3	0	18	0	0	0.00	0.000000	...	28	0
56658	8080	1032755	3	3	0	18	0	0	0.00	0.000000	...	28	0
56659	8080	1045087	3	3	0	18	0	0	0.00	0.000000	...	28	0
56660	8080	997161	3	3	0	18	0	0	0.00	0.000000	...	28	0

56661 rows × 79 columns

```
df_s = df_s.sort_index()
```

```
df_s['Label'].value_counts()
```

```
BENIGN          22731
DoS             19035
PortScan         7946
BruteForce       2767
WebAttack        2180
Bot              1966
Infiltration       36
Name: Label, dtype: int64
```

Most of the regular network attributes such as

- packet length,
- data transfer rate,

- throughput,
- inter-arrival time,
- flags of TCP and their counts, segment size, and active/idle time

should be considered . However, the computational complexity of the proposed IDS may increase dramatically due to the high data dimensionality. Thus, further feature analysis should be done for the external network data. The collected network data would be pre-processed after a few steps to be better suited for IDS development purpose. Firstly, the data can be encoded with one-hot-vector because it has a certain threshold to help separate normal data and anomalies .

ML training is often more efficient with normalized data . Thus, each feature with numerical values is set to the range of 0.0 to 1.0, and each value after normalization xn is indicated as:

where x is the original value, max and min are the maximum and minimum values of each feature.

LabelEncoder to encode categorical variables.

```
[ ] labelencoder = LabelEncoder()                                    # Initializes a LabelEncoder object from scikit
    df.iloc[:, -1] = labelencoder.fit_transform(df.iloc[:, -1])      # Encodes the last column (assumed to be the ta
    X = df.drop(['Label'],axis=1).values                             # Assigns the feature variables (all columns ex
    y = df.iloc[:, -1].values.reshape(-1,1)                          # Assigns the target variable (last column) to
    y = np.ravel(y)                                                  # Converts y to a 1D array using ravel(). This
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2, random_state = 0,stratify = y)
```

```
<ipython-input-23-cb095e7eca90>:2: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the
  df.iloc[:, -1] = labelencoder.fit_transform(df.iloc[:, -1])
```

```
[ ] X_train.shape
```

```
(45328, 78)
```

```
pd.Series(y_train).value_counts()                    #Converts the y_train numpy array to a pandas Series.
 #output is   pandas Series where the index is the unique values in y_train and the values are the number of occurrences of
```

```
0    18184
3    15228
5     6357
2     2213
6     1744
1     1573
4       29
dtype: int64
```

3.2.1 OVERSAMPLING TECHNIQUE

To overcome the issue of class-imbalanced data which often results in a low anomaly detection rate, **random oversampling** and **(SMOTE)** can be used to generate more data in the minority classes that do not have enough data.

Addressing class imbalance in training data - SMOTE

```
[ ]  smote = SMOTE(sampling_strategy={4:1500})                    # r
     X_train, y_train = smote.fit_resample(X_train, y_train)      # A

                          # X_train and y_train arrays will be updated with the
                          # This can be useful for addressing class imbalance ir
```

```
[ ]  pd.Series(y_train).value_counts()
```

```
0    18184
3    15228
5     6357
2     2213
6     1744
1     1573
4     1500
dtype: int64
```

The basic strategy of random oversampling is to simply make multiple copies of the samples to increase samples in the minority classes. However, the random oversampling method can easily result in over-fitting because the information learned would be very specific instead of generic. On the other hand, the SMOTE algorithm analyzes the minority classes and generates new samples based on them by using the idea of K nearest neighbors. Therefore, SMOTE can generate high-quality samples and is used for the minority classes in the proposed system.

3.3. The proposed ML approaches

In the proposed system, to detect various cyber-attacks, developing the IDS can be considered as a multi-classification problem, and machine learning algorithms are widely used to solve such classification problems .

The selected ML algorithms are based on tree structure, including

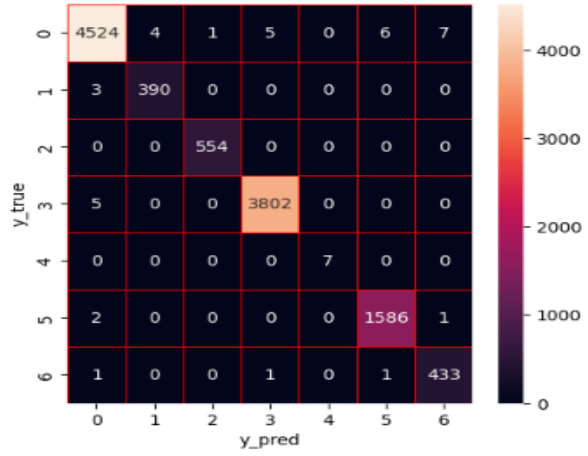
3.3.1 **Decision tree (DT)** is a common classification method based on divide and conquer strategy. A DT comprises decision nodes and leaf nodes, and they represent a decision test over one of the features, and the result class, respectively.

```
dt = DecisionTreeClassifier(random_state = 0)                                    # creates a
dt.fit(X_train,y_train)                                                          # fits the
dt_score=dt.score(X_test,y_test)                                                 # calculates
y_predict=dt.predict(X_test)                                                     # predict th
y_true=y_test                                                                    # actual lab
print('Accuracy of DT: '+ str(dt_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of DT: '+(str(precision)))
print('Recall of DT: '+(str(recall)))
print('F1-score of DT: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)                                            # CONFUSION M
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of DT: 0.9967351980940616
Precision of DT: 0.9967433779773336
Recall of DT: 0.9967351980940616
F1-score of DT: 0.9967370334972472
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      4547
           1       0.99      0.99      0.99       393
           2       1.00      1.00      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      1.00      1.00         7
           5       1.00      1.00      1.00      1589
           6       0.98      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       0.99      1.00      1.00     11333
weighted avg       1.00      1.00      1.00     11333
```




```
dt_train=dt.predict(X_train)
dt_test=dt.predict(X_test)
```

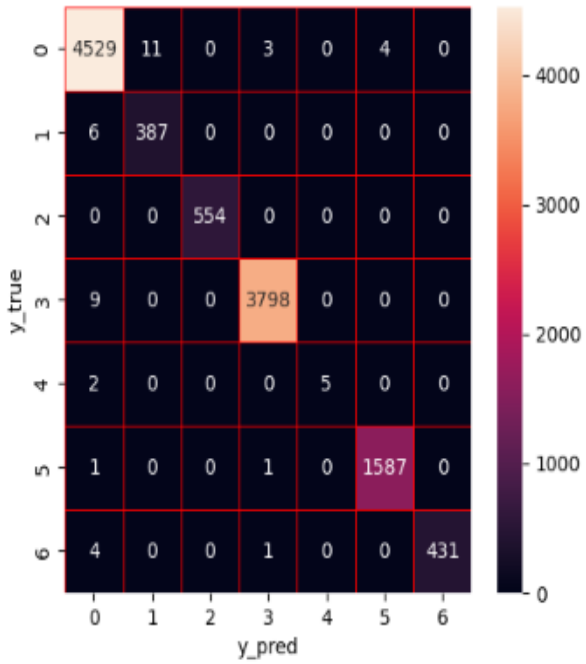
- 3.3.2 **Random forest (RF)** is an ensemble learning classifier based on the majority voting rule that the class with the highest votes by decision trees is selected to be the classification result.

```
rf = RandomForestClassifier(random_state = 0)
rf.fit(X_train,y_train)
rf_score=rf.score(X_test,y_test)
y_predict=rf.predict(X_test)
y_true=y_test
print('Accuracy of RF: '+ str(rf_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of RF: '+(str(precision)))
print('Recall of RF: '+(str(recall)))
print('F1-score of RF: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of RF: 0.9962940086473132
Precision of RF: 0.9963078881710408
Recall of RF: 0.9962940086473132
F1-score of RF: 0.9962833888487207
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4547
           1       0.97      0.98      0.98       393
           2       1.00      1.00      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       1.00      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       0.99      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```



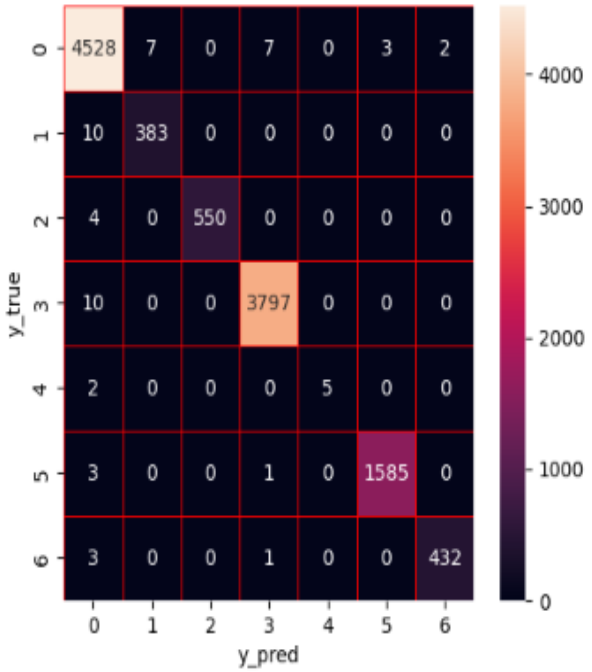
3.3.3 extra trees (ET) model based on a collection of randomized decision trees generated by processing different subsets of data set.

```
et = ExtraTreesClassifier(random_state = 0)
et.fit(X_train,y_train)
et_score=et.score(X_test,y_test)
y_predict=et.predict(X_test)
y_true=y_test
print('Accuracy of ET: '+ str(et_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of ET: '+(str(precision)))
print('Recall of ET: '+(str(recall)))
print('F1-score of ET: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of ET: 0.9953233918644666
Precision of ET: 0.9953255017621429
Recall of ET: 0.9953233918644666
F1-score of ET: 0.9953075685513322
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      4547
           1       0.98      0.97      0.98       393
           2       1.00      0.99      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       1.00      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       1.00      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```



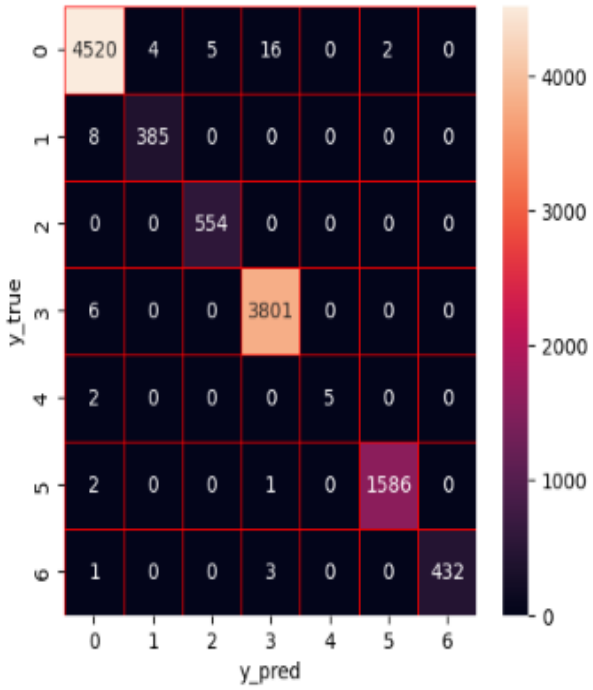
- 3.3.4 **XGBoost** is a ensemble learning algorithm designed for speed and performance improvement by using the gradient descent method to combine many decision trees.

```
xg = xgb.XGBClassifier(n_estimators = 10)
xg.fit(X_train,y_train)
xg_score=xg.score(X_test,y_test)
y_predict=xg.predict(X_test)
y_true=y_test
print('Accuracy of XGBoost: '+ str(xg_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of XGBoost: '+(str(precision)))
print('Recall of XGBoost: '+(str(recall)))
print('F1-score of XGBoost: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of XGBoost: 0.9955881055325156
Precision of XGBoost: 0.9955918224531761
Recall of XGBoost: 0.9955881055325156
F1-score of XGBoost: 0.9955711178626651
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      4547
           1       0.99      0.98      0.98       393
           2       0.99      1.00      1.00       554
           3       0.99      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       1.00      0.99      1.00       436

    accuracy                           1.00     11333
   macro avg       1.00      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```



3.4 OTHER CLASSIFICATION PROBLEMS

K-nearest neighbor (KNN) and **support vector machine (SVM)** are also common for classification problems. For the purpose of model selection, the computational complexity of common supervised ML algorithms is calculated.

Assuming the number of training instances is N, the number of features is P, and the number of trees is T, we have the following approximations.

- **DT is O(N^2P)** * **RF is O($N^2 \sqrt{PT}$).**
- **ET and XGBoost** O(NPT).

On the other hand,

- **KNN's complexity is** O(NP)
- **SVM complexity is** O(N^2P) .

However, unlike KNN andSVM, the proposed four tree-based models, DT, RF, ET, and XGBoost, enable multi-threading to save training time.

Assuming maximum number of participating threads of a computer is M, the time complexity of DT, RF, ET, and XGBoost reduces to $O(N2P/M)$, **O($N^2 \sqrt{PT}$/ M), O(NPT/ M), and O(NPT/ M),**

Therefore, although the original time complexity of the considered algorithms are similar, the four tree structure algorithms have lower computational time due to multi-threading, which is an important reason for choosing these four algorithms.

The other reasons for choosing these algorithms are as follows:

- Most of the tree structure ML models are using ensemble learning so they often show better performance than other single models such as KNN.
- They have the ability to handle non-linear and high dimensional data that the proposed network data belongs to.
- The feature importance calculations are done during the building process of those models, which is beneficial when performing feature selection.

3.5 HYPER-PARAMETERS

It is noteworthy that there are some hyper-parameters of the proposed algorithms that need be tuned to achieve better performance. For the DT algorithm, the split measure function is set to be Gini

impurity, and the **classification and regression trees (CART) model** is then built, which shows better performance than using information gain theory to build an ID3 tree. Assuming that S denotes the set of all sub-trees, CART selects the tree in S that minimizes :

C(S) = ^Ln(S) + _jSj;

where jSj is the cardinality of the tree, _ is a constant

and ^Ln(S) is the empirical risk using the tree S.

Since the deeper tree has more sub-trees, tree depth D is an important parameter of the CART algorithm. For RF and ET, since their results are based on the majority voting of many decision trees, the number of decision trees T is another important parameter affecting the performance. T could also be tuned in XGBoost which is also based on the ensemble of numerous trees. Specifically, XGBoost minimizes the following regularized objective function

$$Obj = -\frac{1}{2}\sum_{j=1}^{t}\frac{G_j^2}{H_j+\lambda} + \gamma t,$$

where G and H represent the sums of the first and second order gradient statistics of the loss function, t is the number of leaves in a decision tree, _ and are the penalty coefficients. Since the gradient statistics are based on the sum of the predicted scores of the T trees, and the number of leaves increases with the increase of the tree depth D, T and D have direct impacts on the objective function value of XGBoost. If the parameters T and D are too small, it leads to **underfitting** and if they are too large, it causes over-fitting and additional computational costs.

The grid search method is utilized to find the optimal values. To be precise, the training of the models starts from a small number of trees and a short depth. Then, these two values are slowly increased with accuracy evaluated until **overfitting,** which is indicated by the dropping of accuracy. Finally, the tree depth D is tuned to 8 and the number of trees T is set to 200. Similarly, the minimum sample split and minimum sample leaf are both tuned from 1 to 10, and finally set to 8 and

3, respectively. parameter tuning process can be further improved by using other optimization techniques.

```
base_predictions_train = pd.DataFrame({                # creates a pandas DataFrame called --base_predictions_train ---that
    'DecisionTree': dt_train.ravel(),                  # contains the predictions made by four different models on the training set.
    'RandomForest': rf_train.ravel(),
    'ExtraTrees': et_train.ravel(),
    'XgBoost': xg_train.ravel(),
    })
base_predictions_train.head(5)
```

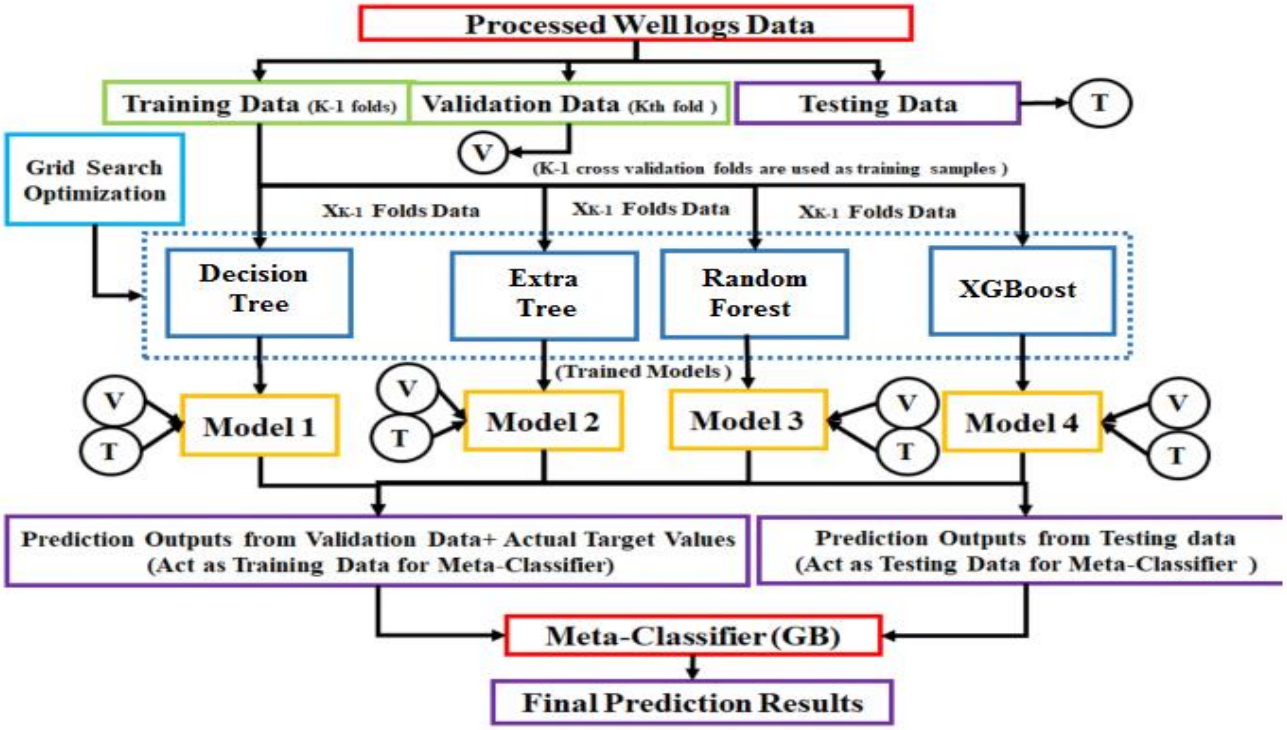
	DecisionTree	RandomForest	ExtraTrees	XgBoost
0	5	5	5	5
1	3	3	3	3
2	5	5	5	5
3	3	3	3	3
4	2	2	2	2

```
[ ] dt_train=dt_train.reshape(-1, 1)
    et_train=et_train.reshape(-1, 1)
    rf_train=rf_train.reshape(-1, 1)
    xg_train=xg_train.reshape(-1, 1)
    dt_test=dt_test.reshape(-1, 1)
    et_test=et_test.reshape(-1, 1)
    rf_test=rf_test.reshape(-1, 1)
    xg_test=xg_test.reshape(-1, 1)
```

```
[ ] x_train = np.concatenate(( dt_train, et_train, rf_train, xg_train), axis=1)    # concatenate  predictions made by the four models  on the t
    x_test = np.concatenate(( dt_test, et_test, rf_test, xg_test), axis=1)         # predictions are stacked horizontally to form a new array
```


3.4. ENSEMBLE LEARNING AND FEATURE SELECTION

For the purpose of further accuracy improvement, an ensemble learning technique**, Stacking**, is implemented. Stacking is a common ensemble method consisting of two layers where the first layer contains a few trained base predictors, and their output serves as the input of a meta-learner in the second layer to build a strong classifier.



- The four trained tree structure algorithms serve as the base models in first layer of stacking ensemble method,
- The singular algorithm with highest accuracy among the four base models is selected to be the **meta-classifier** in the second layer.

To improve the confidence of the selected features, an ensemble feature selection (FS) technique is utilized by calculating the average of feature importance lists generated by the four selected tree-based ML models. They are chosen for feature selection because tree-based algorithms calculate the importance of each feature based on each single tree, and then average the output of the trees to make the result more reliable.

Additionally, different traditional feature selection methods such as

- information gain,
- entropy, Gini coefficient

are utilized in the system by setting different parameters in tree based methods to generate the convincing feature importance. The sum of the total feature importance is 1.0.

To select features, the features are ranked with their importance and each feature is added into the feature list from high importance to low importance till the sum of importance reaches 0.9. The other features with the sum of importance less than 0.1 will be discarded to reduce the computational costs.

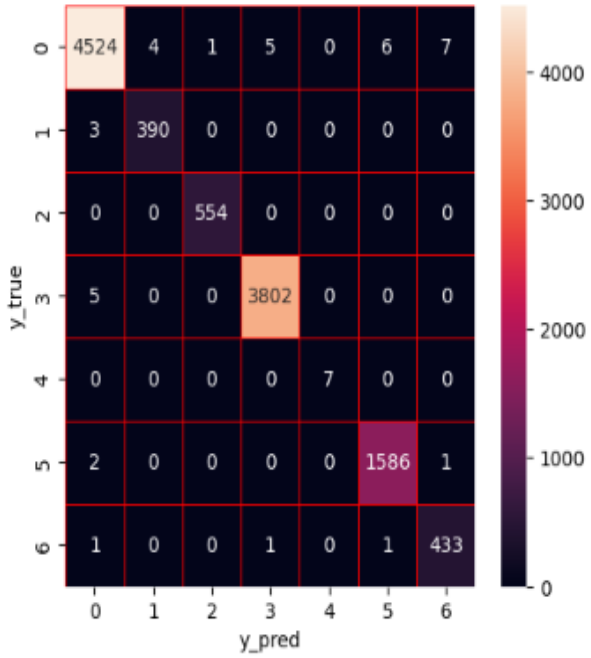
```
[ ] stk = xgb.XGBClassifier().fit(x_train, y_train)                    # variable will contain the trained
                                                                       # used to make predictions on new, u
                                                                       # evaluate the performance of the mo
```

```
y_predict=stk.predict(x_test)
y_true=y_test
stk_score=accuracy_score(y_true,y_predict)
print('Accuracy of Stacking: '+ str(stk_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of Stacking: '+(str(precision)))
print('Recall of Stacking: '+(str(recall)))
print('F1-score of Stacking: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of Stacking: 0.9967351980940616
Precision of Stacking: 0.9967433779773336
Recall of Stacking: 0.9967351980940616
F1-score of Stacking: 0.9967370334972472
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      4547
           1       0.99      0.99      0.99       393
           2       1.00      1.00      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      1.00      1.00         7
           5       1.00      1.00      1.00      1589
           6       0.98      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       0.99      1.00      1.00     11333
weighted avg       1.00      1.00      1.00     11333
```



3.5. VALIDATION METRICS

Each considered data set is split into five subsets and 5- fold cross-validation is implemented to evaluate the proposed models.

1) Accuracy (Acc),
2) detection rate (DR)/recall,
3) false alarm rate (FAR), and
4) F1 score

are the main metrics used to evaluate the proposed method, with their formulas proposed in . The accuracy is the proportion of correctly classified data. However, the data sets may exhibit class imbalance, resulting in a high accuracy of normal data classification but a low attack detection rate. Thus, the detection rate, the ratio between the detected attack data and the total abnormal data, is also calculated for evaluation.

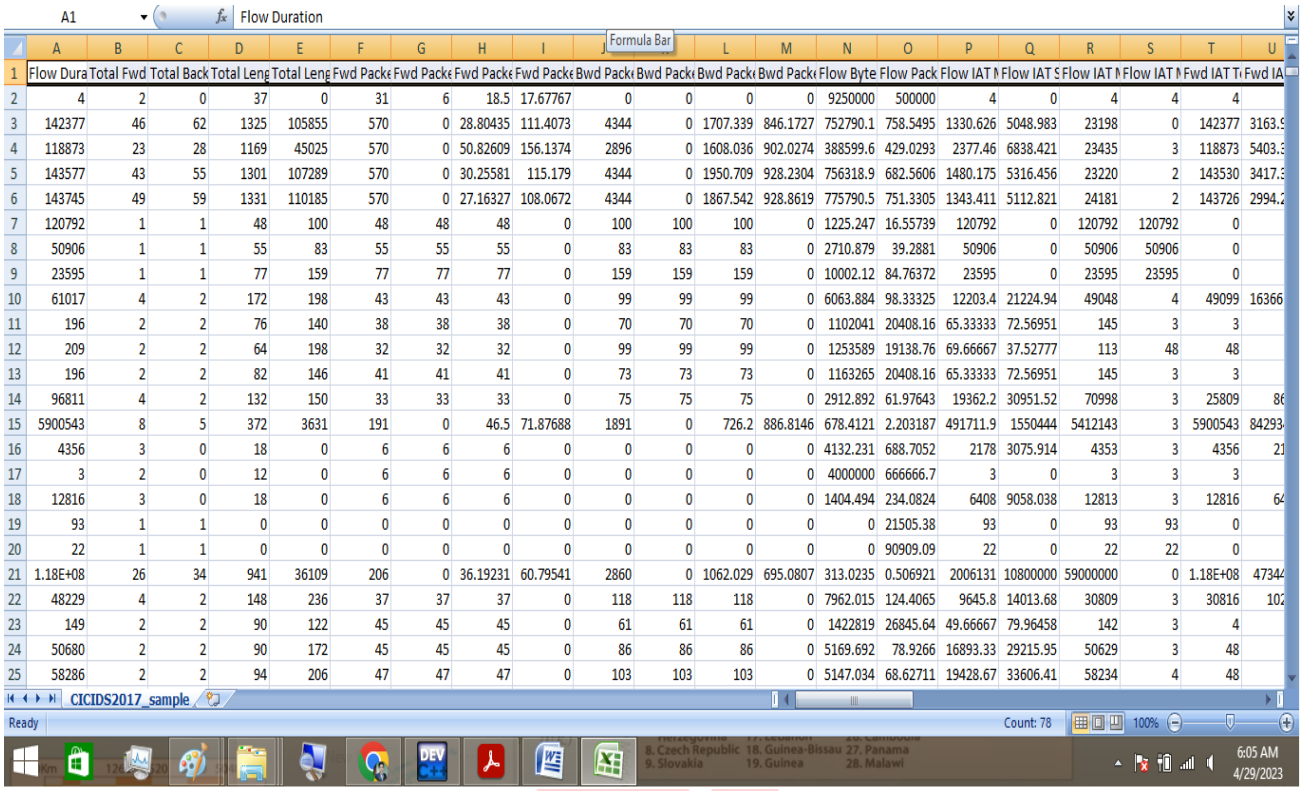
A qualified IDS should have a **high DR t**o ensure most of the attacks can be detected and **a low FAR** to confirm the system does not misreport data for higher DR . In addition, the F1 score considers both the precision and recall by calculating their harmonic average, which can be used to evaluate the overall performance of the methods. Also, the execution time, mainly the model training time, is used to provide insights into the computational performance.

4. PERFORMANCE EVALUATION

4.1. Datasets description

To evaluate the proposed IDS, this work considers two different datasets for the purpose of its implementation in both

intra-vehicle and external networks to build a comprehensive IDS that can also be effective in external communication networks, a standard IDS data set containing the most updated attack scenarios, named ”CICIDS2017” , is considered in this work. The features of the two selected data sets meet our requirements proposed . To prepare better datasets for the IDS development, minor tasks including data combination, missing value removal and new label assignments were done for both datasets based on the methods proposed.



Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) are the most important defense tools against the sophisticated and ever-growing network attacks. Due to the lack of reliable test and validation datasets, anomaly-based intrusion detection approaches are suffering from consistent and accurate performance evolutions.

Our evaluations of the existing eleven datasets since 1998 show that most are out of date and unreliable. Some of these datasets suffer from the lack of traffic diversity and volumes, some do not cover the variety of known attacks, while others anonymize packet payload data, which cannot reflect the current trends. Some are also lacking feature set and metadata.

CICIDS2017 dataset contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using

CICFlowMeter with labeled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols and attack (CSV files). Also available is the extracted features definition.

Generating realistic background traffic was our top priority in building this dataset. We have used our proposed B-Profile system (Sharafaldin, et al. 2016) to profile the abstract behavior of human interactions and generates naturalistic benign background traffic. For this dataset, we built the abstract behavior of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols.

DATA TYPE AND SIZE OF THE CICIDS2017 DATASET

Class label	Number of instances	Class label	Number of instances
BENIGN	2,273,097	Web-Attack	2,180
DoS	380,699	Botnet	1,966
Port-Scan	158,930	Infiltration	36
Brute-Force	13,835	-	-

The data capturing period started at 9 a.m., Monday, July 3, 2017, and ended at 5 p.m. on Friday, July 7, 2017, for a total of 5 days. Monday is the normal day and only includes benign traffic. The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS. They have been executed both morning and afternoon on Tuesday, Wednesday, Thursday and Friday.

In our recent dataset evaluation framework (Gharib et al., 2016), we have identified eleven criteria that are necessary for building a reliable benchmark dataset. None of the previous IDS datasets could cover all of the 11 criteria. In the following, we briefly outline these criteria:

- **Complete Network configuration:** A complete network topology includes Modem, Firewall, Switches, Routers, and presence of a variety of operating systems such as Windows, Ubuntu, and Mac OS X.

- **Complete Traffic:** By having a user profiling agent and 12 different machines in Victim-Network and real attacks from the Attack-Network.

- **Labelled Dataset:** Section 4 and Table 2 show the benign and attack labels for each day. Also, the details of the attack timing will be published on the dataset document.

- **Complete Interaction:** As Figure 1 shows, we covered both within and between internal LAN by having two different networks and Internet communication as well.

- **Complete Capture:** Because we used the mirror port, such as a tapping system, all traffics have been captured and recorded on the storage server.

- **Available Protocols:** Provided the presence of all commonly available protocols, such as HTTP, HTTPS, FTP, SSH an and email protocols.

- **Attack Diversity:** Included the most common attacks based on the 2016 McAfee report, such as Web-based, Brute force, DoS, DDoS, Infiltration, Heart-bleed, Bot, and Scan covered in this dataset.

- **Heterogeneity:** Captured the network traffic from the main Switch and memory dump and system calls from all victim machines, during the execution of the attack.

- **Feature Set:** Extracted more than 80 network flow features from the generated network traffic using CICFlowMeter and delivered the network flow dataset as a CSV file. See our PCAP analyzer and CSV generator.

- **MetaData:** Completely explained the dataset which includes the time, attacks, flows and labels in the published paper.

4.2 . IDS PERFORMANCE ANALYSIS

4.2.1 Feature Extraction

```
[ ]  dt_feature = dt.feature_importances_                                    # extracts the feature importances f
     rf_feature = rf.feature_importances_
     et_feature = et.feature_importances_
     xgb_feature = xg.feature_importances_

[ ]  avg_feature = (dt_feature+rf_feature+et_feature+xgb_feature)/4           # calculate average of model

[ ]  feature=(df.drop(['Label'],axis=1)).columns.values                      # sorts the features in a pandas Dat
     print ("Features sorted by their score:")
     print (sorted(zip(map(lambda x: round(x, 4), avg_feature), feature), reverse=True))

     Features sorted by their score:
     [(0.1095, 'Bwd Packet Length Min'), (0.1084, 'Bwd Packet Length Std'), (0.1016, 'Destination Port'), (0.0596, 'Averag

[ ]  f_list = sorted(zip(map(lambda x: round(x, 4), avg_feature), feature), reverse=True)          # calculates the lengt
     len(f_list)

     78

[ ]  sum = 0                                                                 # calculates the cumulative sum of f
     for i in range(0, len(f_list)):                                         # stores the corresponding feature n
     fs = []
       sum = sum + f_list[i][0]
       fs.append(f_list[i][1])
       if sum >= 0.9:
         break

[ ]  X_fs = df[fs].values

[ ]  X_train, X_test, y_train, y_test = train_test_split(X_fs,y, train_size = 0.8, test_size = 0.2, random_state = 0,stra
     X_train.shape

     (45328, 38)

[ ]  pd.Series(y_train).value_counts()

     0    18184
     3    15228
     5     6357
     2     2213
     6     1744
     1     1573
     4       29
     dtype: int64

[ ]  smote=SMOTE(n_jobs=-1,sampling_strategy={4:1500})
     X_train, y_train = smote.fit_resample(X_train, y_train)
     pd.Series(y_train).value_counts()

     /usr/local/lib/python3.9/dist-packages/imblearn/over_sampling/_smote/base.py:336: FutureWarning: The parameter `n_job
       warnings.warn(
     0    18184
     3    15228
     5     6357
     2     2213
     6     1744
     1     1573
     4     1500
     dtype: int64
```

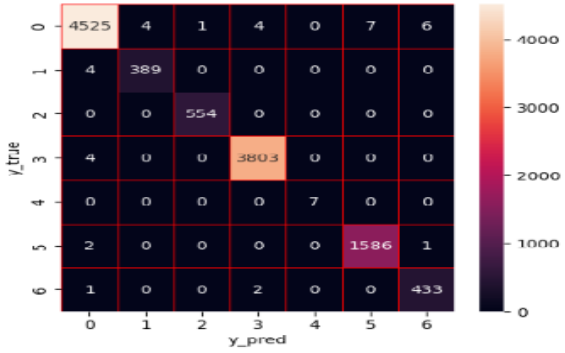

CICIDS2017 data set, as shown in Table , most of the used tree-based algorithms shows 1.8-3.2% higher accuracy, detection rate and F1 score than KNN and SVM except for ET. Hence, only DT, RF, and XGBoost were chosen in the proposed stacking method, and XGBoost was selected to be the meta-classifier of the stacking model.

```
dt = DecisionTreeClassifier(random_state = 0)
dt.fit(X_train,y_train)
dt_score=dt.score(X_test,y_test)
y_predict=dt.predict(X_test)
y_true=y_test
print('Accuracy of DT: '+ str(dt_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of DT: '+(str(precision)))
print('Recall of DT: '+(str(recall)))
print('F1-score of DT: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y true")
plt.show()
```

```
Accuracy of DT: 0.9968234359834113
Precision of DT: 0.9968286852565558
Recall of DT: 0.9968234359834113
F1 score of DT: 0.9968213975913275
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4547
           1       0.99      0.99      0.99       393
           2       1.00      1.00      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      1.00      1.00         7
           5       1.00      1.00      1.00      1589
           6       0.98      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       0.99      1.00      1.00     11333
weighted avg       1.00      1.00      1.00     11333
```

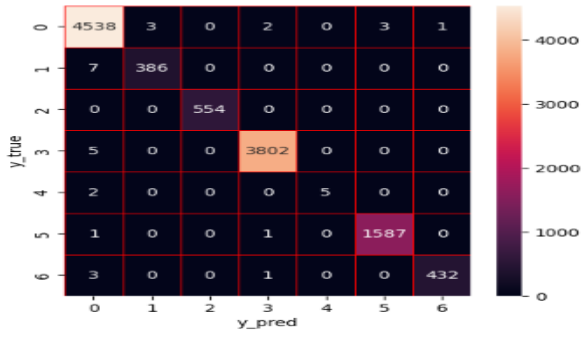



```
rf = RandomForestClassifier(random_state = 0)
rf.fit(X_train,y_train)
rf_score=rf.score(X_test,y_test)
y_predict=rf.predict(X_test)
y_true=y_test
print('Accuracy of RF: '+ str(rf_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of RF: '+(str(precision)))
print('Recall of RF: '+(str(recall)))
print('F1-score of RF: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of RF: 0.9974411012088591
Precision of RF: 0.9974409791398017
Recall of RF: 0.9974411012088591
F1-score of RF: 0.997424571590988
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4547
           1       0.99      0.98      0.99       393
           2       1.00      1.00      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       1.00      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       1.00      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```

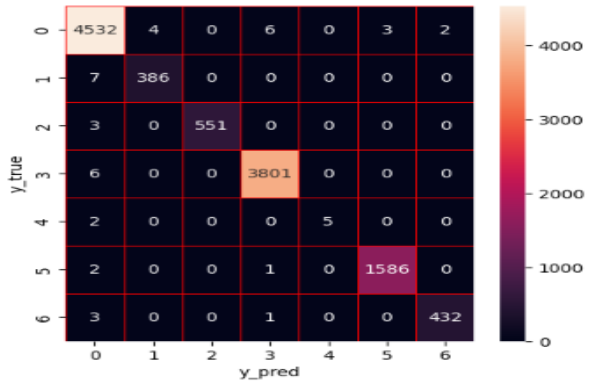


```
et = ExtraTreesClassifier(random_state = 0)
et.fit(X_train,y_train)
et_score=et.score(X_test,y_test)
y_predict=et.predict(X_test)
y_true=y_test
print('Accuracy of ET: '+ str(et_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of ET: '+(str(precision)))
print('Recall of ET: '+(str(recall)))
print('F1-score of ET: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of ET: 0.9964704844260125
Precision of ET: 0.9964708911962491
Recall of ET: 0.9964704844260125
F1-score of ET: 0.9964545865568019
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      4547
           1       0.99      0.98      0.99       393
           2       1.00      0.99      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       1.00      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       1.00      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```

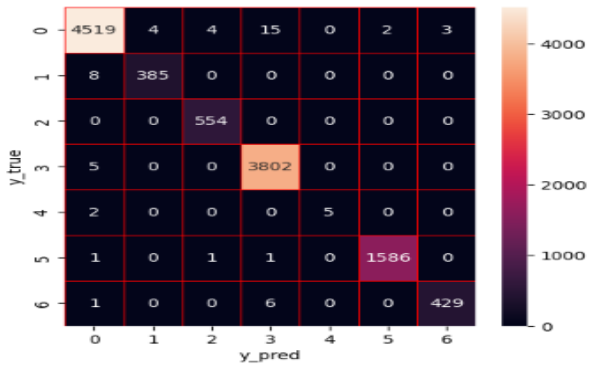



```
xg = xgb.XGBClassifier(n_estimators = 10)
xg.fit(X_train,y_train)
xg_score=xg.score(X_test,y_test)
y_predict=xg.predict(X_test)
y_true=y_test
print('Accuracy of XGBoost: '+ str(xg_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of XGBoost: '+(str(precision)))
print('Recall of XGBoost: '+(str(recall)))
print('F1-score of XGBoost: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of XGBoost: 0.9953233918644666
Precision of XGBoost: 0.9953261388853651
Recall of XGBoost: 0.9953233918644666
F1-score of XGBoost: 0.9953051152662237
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      4547
           1       0.99      0.98      0.98       393
           2       0.99      1.00      1.00       554
           3       0.99      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       0.99      0.98      0.99       436

    accuracy                           1.00     11333
   macro avg       0.99      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```



Although the execution time of stacking is longer than any singular tree-based models, it reaches the highest accuracy among the models (99.86%). After training the data sets with all the features to obtain the highest performance, the feature selection method proposed in Section III-C was implemented to reduce execution time. From Tables, it can be seen that RF and XGBoost are the singular models with best accuracy on CA nintrusion data set and CICIDS2017 data set, respectively. As single base models have lower execution time than the ensemble model, these two singular models were also tested after feature selection.

PERFORMANCE EVALUATION OF IDS ON CICIDS2017 DATASET

Method	Acc (%)	DR (%)	FAR (%)	F1 Score	Execution Time (S)
KNN [5]	96.6	96.4	5.6	0.966	9243.6
SVM [24]	98.01	97.58	1.48	0.978	49953.1
DT	99.72	99.3	0.029	0.998	126.7
RF	98.37	98.29	0.039	0.983	2421.6
ET	93.43	93.35	0.001	0.934	2349.6
XGBoost	99.78	99.76	0.069	0.997	1637.2
Stacking	99.86	99.8	0.012	0.998	4519.3
FS XGBoost	99.7	99.55	0.077	0.996	995.9
FS Stacking	99.82	99.75	0.011	0.997	2774.8

For CICIDS2017 data set, 36 out of 78 features were selected and the accuracy of XGBoost and stacking models only decreased by 0.08% and 0.04% while saving 39.2% and 38.6% of the execution time,respectively, a

Therefore, the tree-based averaging feature selection method enables the IDS to save execution time while maintaining high accuracy.

In order to analyze the features, the proposed feature selection method was tested on the subsets of each attack.

The list of top-3 important features and their corresponding weights of each attack is shown in Table . As Table shows, the destination port can reflect a DoS, brute force, web attack, and botnet attack.

TOP-3 FEATURE IMPORTANCE BY EACH ATTACK

Label	Feature	Weight
DoS	Bwd Packet Length Std	0.1723
	Average Packet Size	0.1211
	Destination Port	0.0785
Port-Scan	Total Length of Fwd Packets	0.3020
	Average Packet Size	0.1045
	PSH Flag Count	0.1019
Brute-Force	Destination Port	0.3728
	Fwd Packet Length Min	0.1022
	Packet Length Variance	0.0859
Web-Attack	Init_Win_bytes_backward	0.2643
	Average Packet Size	0.1650
	Destination Port	0.0616
Botnet	Destination Port	0.2364
	Bwd Packet Length Mean	0.1240
	Avg Bwd Segment Size	0.1104
Infiltration	Total Length of Fwd Packets	0.2298
	Subflow Fwd Bytes	0.1345
	Destination Port	0.1149

The size of packets is another important feature. For example, the average packet size indicates a DoS attack, port scan attack, and web attack. The packet length in the forward direction is related to port scan, brute-force, and infiltration attacks, while the packet length in the backward direction reflects a DoS, web attack, and botnet. In addition, the variance of the length of the packets in both forward and backward directions reflects the brute force attack, and the count of pushing flags indicates port scan attack. After getting the feature importance list of each attack, the IDSs with other aims like designing a dedicated system for detecting a single type of attack can be developed by selecting the important features based on the list. On the other hand, the important features can be selected as key attributes to be monitored by network supervisors. If those attributes change abnormally, the attacks can be detected quickly.

4.4. CONCLUSION

As autonomous vehicles and connected vehicles are vulnerable to various cyber-attacks, IDS is one of the efficient solutions to detect the launched network attacks and secure the vehicle networks. In this report, we presented an IDS based on tree-based machine learning algorithms to detect threats both on CAN bus and external networks.

The SMOTE oversampling method and tree-based averaging feature selection approaches were also introduced to reduce the impact of class imbalance and computational cost. To evaluate the proposed IDS, it was tested on two data sets for both intra-vehicle and external networks.

Model	Accuracy	Acc(with Imp Feature)
`Decision Tree` :	99.67 %	99.68 %
`Random Forest` :	99.62 %	99.74 %
`Extra Trees` :	99.53 %	99.64 %
`XGBoost` :	99.55 %	99.53 %
`Stack` :	**96.63 %**	**99.63 %**

- The results on data set show that the proposed system has 2-3% higher accuracy, detection rate, F1 score and lower false alarm rate than KNN and SVM.
- Moreover, unlike other proposed methods, our work combines all the subsets of the data sets and develops an IDS that can detect various attacks instead of only single type of attack on each run.
- The accuracy of CICIDS2017 data set reaches 99.63.
- Computational time was reduced by 73.7% to 325.6s and by 38.6% to 2774.8s, .

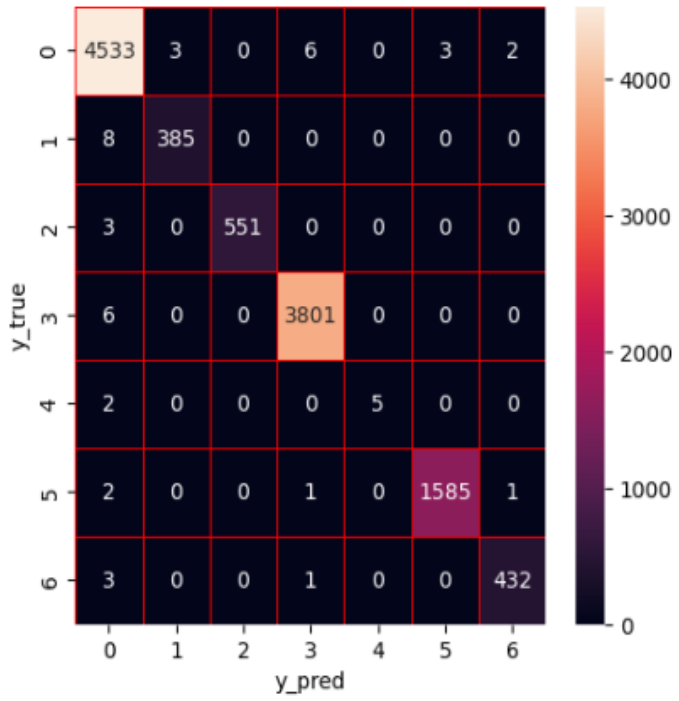
Stack ---> XGBoost

```
[ ] stk = xgb.XGBClassifier().fit(x_train, y_train)
    y_predict=stk.predict(x_test)
    y_true=y_test
    stk_score=accuracy_score(y_true,y_predict)
    print('Accuracy of Stacking: '+ str(stk_score))
    precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weig
    print('Precision of Stacking: '+(str(precision)))
    print('Recall of Stacking: '+(str(recall)))
    print('F1-score of Stacking: '+(str(fscore)))
    print(classification_report(y_true,y_predict))
    cm=confusion_matrix(y_true,y_predict)
    f,ax=plt.subplots(figsize=(5,5))
    sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
    plt.xlabel("y_pred")
    plt.ylabel("y_true")
    plt.show()
```

```
Accuracy of Stacking: 0.9963822465366629
Precision of Stacking: 0.9963830778515008
Recall of Stacking: 0.9963822465366629
F1-score of Stacking: 0.9963656258547763
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      4547
           1       0.99      0.98      0.99       393
           2       1.00      0.99      1.00       554
           3       1.00      1.00      1.00      3807
           4       1.00      0.71      0.83         7
           5       1.00      1.00      1.00      1589
           6       0.99      0.99      0.99       436

    accuracy                           1.00     11333
   macro avg       1.00      0.95      0.97     11333
weighted avg       1.00      1.00      1.00     11333
```



REFERENCES

[1] A. Awang, K. Husain, N. Kamel and S. A¨ıssa, ”Routing in Vehicular Ad-hoc Networks: A Survey on Single- and Cross-Layer Design Techniques, and Perspectives,” in IEEE Access, vol. 5, pp. 9497-9517, 2017.

[2] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, ”An overview of Internet of Vehicles,” China Commun., vol. 11, no. 10, pp. 1-15, 2014.

[3] K. M. Ali Alheeti and K. Mc Donald-Maier, ”Intelligent intrusion detection in external communication systems for autonomous vehicles,” Syst. Sci. Control Eng., vol. 6, no. 1, pp. 48-56, 2018.

[4] H. P. Dai Nguyen and R. Zolt´an, ”The Current Security Challenges of Vehicle Communication in the Future Transportation System,” SISY 2018 - IEEE 16th Int. Symp. Intell. Syst. Informatics, Proc, Subotica, pp. 161-166, 2018.

[5] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, ”Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” no. Cic, pp. 108-116, 2018.

[6] E. Seo, H. M. Song, and H. K. Kim, ”GIDS: GAN based Intrusion Detection System for In-Vehicle Network,” 2018 16th Annu. Conf.
Privacy, Secur. Trust, pp. 1-6, 2018.

[7] H. Lee, S. H. Jeong, and H. K. Kim, ”OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame,” Proc.
15th Annu. Conf. Privacy, Secur. Trust. PST 2017, pp. 57-66, 2017.

[8] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, ”Classification Approach for Intrusion Detection in Vehicle Systems,” Wirel. Eng. Technol., vol. 09, no. 04, pp. 79-94, 2018.

[9] B. Groza and P. Murvay, ”Efficient Intrusion Detection With Bloom Filtering in Controller Area Networks,” IEEE Trans. Inf. Forensics
Security, vol. 14, no. 4, pp. 1037-1051, 2019.

[10] U. E. Larson, D. K. Nilsson and E. Jonsson, ”An approach to specification-based attack detection for in-vehicle networks,” IEEE
Intell. Veh. Symp. Proc, Eindhoven, pp. 220-225, 2008.