Machine Learning Models Applied to Intrusion Detection Systems

CBER710: Capstone Project

Jose Lira
Student ID: 301162762

Agenda

1. Scope
2. Objective of the Research
3. Datasets
4. Program Architecture
5. Algorithm Comparison
6. Shortcomings
7. Conclusion

Objectives

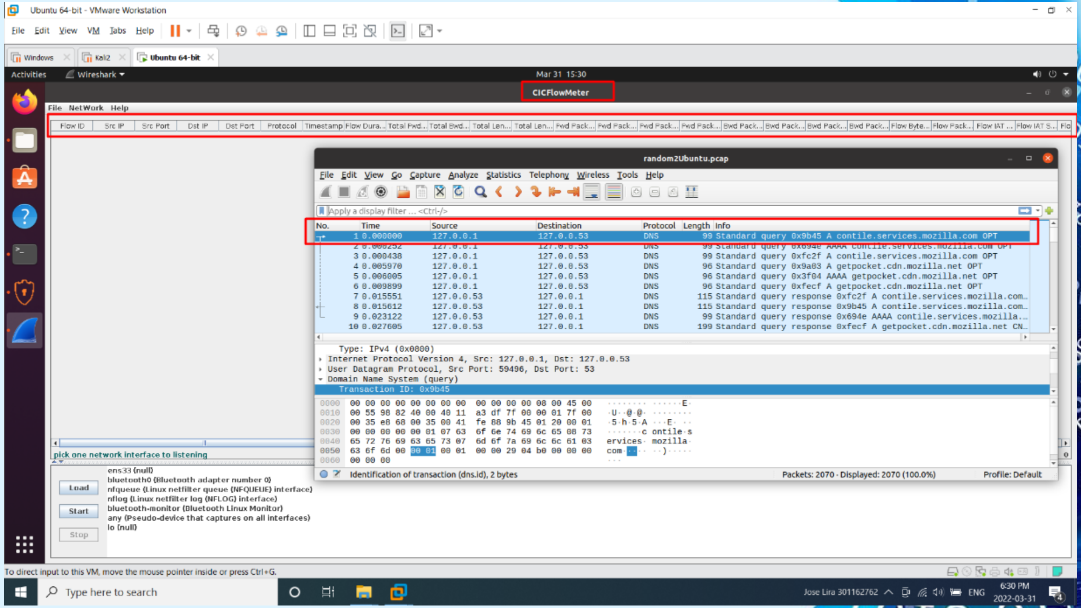
- Compare the leading Artificial Intelligence classification models applied to an Intrusion Detection System.
- Build a didactical tool to graphically show the different capabilities of each Machine Learning algorithm with different datasets.
- Provide a full implementation of advanced algorithms such as Neural Networks and Deep Learning testing different hyperparameters.

Abstract

- This research and implementation pretend to provide the reader with an analysis of the most commonly used Machine Learning models applied to the Intrusion Detection System. In terms of Machine Learning, the IDS problem is defined as a binary classification model that catalogues packets as malicious or not.
- I used two datasets with totally different features and seven Machine Learning models to run the classification and compare the results obtained using Python 3 on a Jupyter notebook.
- Additionally, a Graphical User Interface is developed and implemented from scratch using the Tkinter Python library and integrated with the machine Learning Python code deployed. The GUI was introduced to aid students who are not entirely familiar with the python language and the machine learning libraries

CIC-IDS2017 Dataset

- It was based on a realistic scenario in which two networks were created to emulate real-life conditions: the victim network and the hacker network
  - Brute force
  - Heartbleed
  - Botnet
  - DoS attack
  - SQL Injection
  - XSS (Cross-Site Scripting)
  - Infiltration
- The PCAP file was preprocessed by the tool CICFlow meter to extract 80 features to a CSV file.



UNSW-NB15 Dataset

- UNSW-NB15 Dataset:
  - Backdoors
  - DoS
  - Exploits
  - Generic
  - Reconnaissance
  - Shellcode
  - Worms

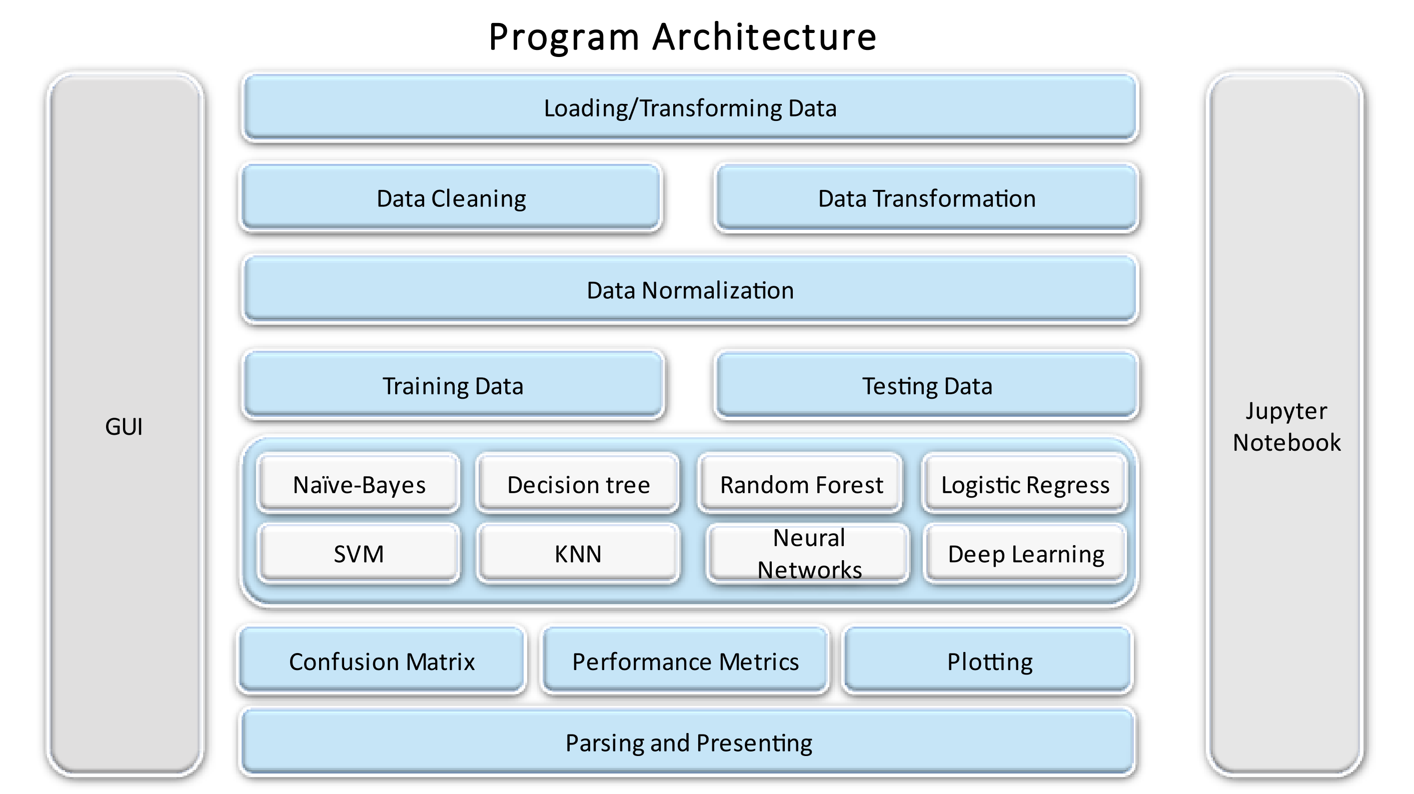
	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service
2	59.166.0.0	1390	149.171.12	53	udp	CON	0.001055	132	164	31	29	0	0	dns
3	59.166.0.0	33661	149.171.12	1024	udp	CON	0.036133	528	304	31	29	0	0	-
4	59.166.0.6	1464	149.171.12	53	udp	CON	0.001119	146	178	31	29	0	0	dns
5	59.166.0.5	3593	149.171.12	53	udp	CON	0.001209	132	164	31	29	0	0	dns
6	59.166.0.3	49664	149.171.12	53	udp	CON	0.001169	146	178	31	29	0	0	dns
7	59.166.0.0	32119	149.171.12	111	udp	CON	0.078339	568	312	31	29	0	0	-
8	59.166.0.6	2142	149.171.12	53	udp	CON	0.001134	132	164	31	29	0	0	dns
9	10.40.182.	0	10.40.182.	0	arp	INT	0	46	0	0	0	0	0	-
10	59.166.0.5	40726	149.171.12	53	udp	CON	0.001126	146	178	31	29	0	0	dns
11	59.166.0.7	12660	149.171.12	53	udp	CON	0.001167	132	164	31	29	0	0	dns
12	10.40.170.	0	10.40.170.	0	arp	INT	0	46	0	0	0	0	0	-
13	10.40.170.	0	10.40.170.	0	arp	INT	0	46	0	0	0	0	0	-
14	10.40.182.	0	10.40.182.	0	arp	INT	0	46	0	0	0	0	0	-
15	59.166.0.1	48847	149.171.12	53	udp	CON	0.001093	132	164	31	29	0	0	dns
16	59.166.0.1	24266	149.171.12	41049	udp	CON	0.001851	528	304	31	29	0	0	-
17	59.166.0.1	10393	149.171.12	44307	udp	CON	0.001749	528	304	31	29	0	0	-
18	59.166.0.2	62539	149.171.12	53	udp	CON	0.001128	132	164	31	29	0	0	dns
19	59.166.0.1	21270	149.171.12	111	udp	CON	0.005153	568	312	31	29	0	0	-
20	59.166.0.1	8989	149.171.12	111	udp	CON	0.004898	568	312	31	29	0	0	-
21	59.166.0.4	49346	149.171.12	53	udp	CON	0.001111	132	164	31	29	0	0	dns
22	175.45.176	21223	149.171.12	32780	udp	INT	0.000021	728	0	254	0	0	0	-
23	175.45.176	23357	149.171.12	80	tcp	FIN	0.240139	918	25552	62	252	2	10	http
24	175.45.176	13284	149.171.12	80	tcp	FIN	2.39039	1362	268	254	252	6	1	http
25	59.166.0.3	4192	149.171.12	53	udp	CON	0.001101	132	164	31	29	0	0	dns
26	59.166.0.2	26872	149.171.12	53	udp	CON	0.001082	132	164	31	29	0	0	dns
27	59.166.0.8	24946	149.171.12	53	udp	CON	0.001122	132	164	31	29	0	0	dns
28	59.166.0.9	5685	149.171.12	53	udp	CON	0.001141	146	178	31	29	0	0	dns
29	59.166.0.0	22848	149.171.12	53	udp	CON	0.001164	146	178	31	29	0	0	dns
30	59.166.0.5	28565	149.171.12	53	udp	CON	0.001127	132	164	31	29	0	0	dns
31	59.166.0.7	46719	149.171.12	53	udp	CON	0.001073	132	164	31	29	0	0	dns
32	59.166.0.8	25100	149.171.12	53	udp	CON	0.001196	146	178	31	29	0	0	dns
33	59.166.0.1	29704	149.171.12	53	udp	CON	0.001101	132	164	31	29	0	0	dns
34	59.166.0.7	36998	149.171.12	53	udp	CON	0.001058	146	178	31	29	0	0	dns
35	59.166.0.7	6438	149.171.12	53	udp	CON	0.0011	132	164	31	29	0	0	dns
36	59.166.0.8	1416	149.171.12	53	udp	CON	0.001126	146	178	31	29	0	0	dns
37	59.166.0.4	35578	149.171.12	53	udp	CON	0.001017	146	178	31	29	0	0	dns
38	59.166.0.4	38902	149.171.12	53	udp	CON	0.001094	132	164	31	29	0	0	dns

	Sload	Dload	Spkts	Dpkts	swin	dwin	stcpb	dtcpb	smeansz	dmeansz	trans_dep	res_bdy_l	Sjit	Djit	Stime	Lti
2	500473.9	621800.9	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
3	87676.09	50480.17	4	4	0	0	0	0	132	76	0	0	9.89101	10.68273	1.42E+09	1
4	521894.5	636282.4	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
5	436724.6	542597.2	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
6	499572.3	609067.6	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
7	43503.23	23896.14	4	4	0	0	0	0	142	78	0	0	29.68222	34.37034	1.42E+09	1
8	465608.5	578483.3	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
9	0	0	1	0	0	0	0	0	46	0	0	0	0	0	1.42E+09	1
10	518650.1	632326.8	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
11	452442.2	562125.1	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
12	0	0	1	0	0	0	0	0	46	0	0	0	0	0	1.42E+09	1
13	0	0	1	0	0	0	0	0	46	0	0	0	0	0	1.42E+09	1
14	0	0	1	0	0	0	0	0	46	0	0	0	0	0	1.42E+09	1
15	483074.1	600182.9	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
16	1711507	985413.3	4	4	0	0	0	0	132	76	0	0	0.656903	0.328339	1.42E+09	1
17	1811321	1042882	4	4	0	0	0	0	132	76	0	0	0.640403	0.280968	1.42E+09	1
18	468085.1	581560.3	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
19	661362.3	363283.5	4	4	0	0	0	0	142	78	0	0	1.890104	1.610554	1.42E+09	1
20	695794.2	382196.8	4	4	0	0	0	0	142	78	0	0	1.780739	1.549507	1.42E+09	1
21	475247.5	590459.1	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
22	1.39E+08	0	2	0	0	0	0	0	364	0	0	0	0	0	1.42E+09	1
23	28050.42	815794.2	12	24	255	255	1.71E+09	1.94E+09	77	1065	1	12026	1170.482	1144.383	1.42E+09	1
24	4233.619	749.6685	14	6	255	255	3.9E+09	2.47E+09	97	45	1	0	18786.71	941.7249	1.42E+09	1
25	479564	595822	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
26	487985.2	606284.7	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
27	470588.2	584670.2	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
28	511831.8	624014.1	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
29	501718.2	611683.8	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
30	468500.5	582076.3	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
31	492078.3	611370	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
32	488294.3	595317.7	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
33	479564	595822	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
34	551984.9	672967.9	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
35	480000	596363.6	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1
36	518650.1	632326.8	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
37	574238	700098.4	2	2	0	0	0	0	73	89	0	0	0	0	1.42E+09	1
38	482632.6	599634.4	2	2	0	0	0	0	66	82	0	0	0	0	1.42E+09	1

UNSW-NB15_1

- The PCAP file was pre-processed with the flow extractor tool Argos to extract the parameters to the CSV file.

Program Architecture



Loading the Datasets

CBER 710 Capstone Project: "Intrusion Detection System Using Machine Learning"

Deep Learning using TensorFlow 2 and Keras

Loading the Libraries

In [1]:
```
# Run on TensorFlow 2.x

from keras.layers.core import Dense, Activation
import gc
from   matplotlib              import pyplot as plt
import numpy as np
import os
import pandas as pd
from   sklearn.ensemble        import RandomForestClassifier
from   sklearn.linear_model    import *
from   sklearn.metrics         import accuracy_score
from   sklearn.metrics         import f1_score
from   sklearn.metrics         import recall_score
from   sklearn.metrics         import precision_score
from   sklearn.model_selection import train_test_split
from   sklearn.naive_bayes     import *
from   sklearn.neighbors       import *
from   sklearn.preprocessing   import StandardScaler
from   sklearn.tree            import *
import tensorflow as tf
from   tensorflow.keras        import layers

# The following lines adjust the granularity of reporting.
pd.options.display.max_rows = 10
pd.options.display.float_format = "{:.2f}".format
%matplotlib inline
%config IPCompleter.greedy=True   # autocompleter of Jupyter notebook using <Tab> key
# tf.keras.backend.set_floatx('float32')
# %tensorflow_version 2.x
```

Loading the Data from the Working Directory

In [2]:
```
# Loading all the datasets in the working directory
mainpath = r"C:\Users\jlira\Training\_Classes\CBER 710 Capstone Project\Data\UNSW-NB15 - CSV Files\Datasets"
path, dirs, ds_file_names = next(os.walk(mainpath))
file_count = len(ds_file_names)
dataset_list = []
print(r"[+] Loading dataset file(s). Please, wait...")
for i in range(file_count):
    fullpath = os.path.join(mainpath, ds_file_names[i])
    dataset_list.append(pd.read_csv(fullpath, encoding='utf8', low_memory=False)) # iso-8859-1 #utf8
    print("File loaded: " + str(ds_file_names[i]))
print("[+] " + str(file_count) + " files(s) uploaded successfully")
```

```
[+] Loading dataset file(s). Please, wait...
File loaded: UNSW-NB15_1.csv
File loaded: UNSW-NB15_2.csv
[+] 2 files(s) uploaded successfully
```

Converting the data to a pandas dataset

In [3]:
```
# Merge all the datasets loaded
data_full = pd.concat(
    dataset_list,
    axis = 0,
    join = "outer",
    ignore_index = False,
    keys = None,
    levels = None,
    names = None,
    verify_integrity = False,
    copy = True,
)
del dataset_list
gc.collect()
data_full.shape
```

Out[3]: (1400002, 49)

Cleaning and Transforming the data

Cleaning the Data before processing

In [8]:
```
# Delete all blank spaces in the columns titles
data_full.columns = data_full.columns.str.replace(' ','')
```

In [9]:
```
# Replacing all nulls with '0's
data_full = data_full.fillna(0)
data_full.shape
```

Out[9]:
```
(1400002, 49)
```

In [10]:
```
# Delete Attack category column: 'attack_cat' (The name of each attack category)
# Nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
data_full.drop(['attack_cat'], axis = 1, inplace = True)
```

In [11]:
```
# Converting IP addresses to integers in both 'srcip' and 'dstip' columns
# Auxiliary fucntion to convert IP to integers
def ip_to_int(ip_ser):
    ips = ip_ser.str.split('.', expand=True).astype(np.int64).values
    mults = np.tile(np.array([24, 16, 8, 0]), len(ip_ser)).reshape(ips.shape)
    return np.sum(np.left_shift(ips, mults), axis=1)

data_full['srcip'] = ip_to_int(data_full.srcip)
data_full['dstip'] = ip_to_int(data_full.dstip)
```

In [12]:
```
# Convert 'dsport' column from object to int
data_full['dsport'] = pd.to_numeric(data_full.dsport, errors='coerce').fillna(0).astype(int)
```

In [13]:
```
# Convert 'sport' column from object to int

data_full['sport'] = pd.to_numeric(data_full.dsport, errors='coerce').fillna(0).astype(int)
```

In [14]:
```
# Replace null values in 'ct_ftp_cmd' to zero
# ct_ftp_cmd = No of flows that has a command in ftp session.

# data_full.loc[pd.to_numeric(data_full['ct_ftp_cmd'], errors='coerce').isnull()]

# Replace with '0's
data_full['ct_ftp_cmd'] = pd.to_numeric(data_full.dsport, errors='coerce').fillna(0).astype(int)
```

In [15]:
```
# Replace categorical columns (strings) with integer values generated in a dictianary (one dictionary per column)
# Create a list of dictionaries for the values of the categorical columns
list_of_dict = []
for col in data_full.columns:
    if(data_full[col].dtype == object):
        values_label = data_full[col].unique()
        values_label_dict ={}
        counter = 0
        for value in values_label:
            if value not in values_label_dict.keys():
                values_label_dict[value] = counter
                counter += 1
        list_of_dict.append(values_label_dict)
        # Replace strings in the iterated column with the integer values taken from the respective generated dictionary
        data_full[col] = [values_label_dict[item] for item in data_full[col]]
        data_full[col].unique()
print(f"[+] Number of categorical columns modified: {len(list_of_dict)}\n")
print(list_of_dict)
```

```
[+] Number of categorical columns modified: 3

[{'udp': 0, 'arp': 1, 'tcp': 2, 'ospf': 3, 'icmp': 4, 'igmp': 5, 'sctp': 6, 'udt': 7, 'sep': 8, 'sun-nd': 9, 'swipe': 10, 'mobi
le': 11, 'pim': 12, 'rtp': 13, 'ipnip': 14, 'ip': 15, 'ggp': 16, 'st2': 17, 'egp': 18, 'cbt': 19, 'emcon': 20, 'nvp': 21, 'ig
p': 22, 'xnet': 23, 'argus': 24, 'bbn-rcc': 25, 'chaos': 26, 'pup': 27, 'hmp': 28, 'mux': 29, 'dcn': 30, 'prm': 31, 'trunk-1':
32, 'xns-idp': 33, 'trunk-2': 34, 'leaf-1': 35, 'leaf-2': 36, 'irtp': 37, 'rdp': 38, 'iso-tp4': 39, 'netblt': 40, 'mfe-nsp': 4
1, 'merit-inp': 42, '3pc': 43, 'xtp': 44, 'idpr': 45, 'tp++': 46, 'ddp': 47, 'idpr-cmtp': 48, 'ipv6': 49, 'il': 50, 'idrp': 51,
'ipv6-frag': 52, 'sdrp': 53, 'ipv6-route': 54, 'gre': 55, 'rsvp': 56, 'mhrp': 57, 'bna': 58, 'esp': 59, 'i-nlsp': 60, 'narp': 6
1, 'ipv6-no': 62, 'tlsp': 63, 'skip': 64, 'ipv6-opts': 65, 'any': 66, 'cftp': 67, 'sat-expak': 68, 'kryptolan': 69, 'rvd': 70,
'ippc': 71, 'sat-mon': 72, 'ipcv': 73, 'visa': 74, 'cpnx': 75, 'cphb': 76, 'wsn': 77, 'pvp': 78, 'br-sat-mon': 79, 'wb-mon': 8
0, 'wb-expak': 81, 'iso-ip': 82, 'secure-vmtp': 83, 'vmtp': 84, 'vines': 85, 'ttp': 86, 'nsfnet-igp': 87, 'dgp': 88, 'tcf': 89,
'eigrp': 90, 'sprite-rpc': 91, 'larp': 92, 'mtp': 93, 'ax.25': 94, 'ipip': 95, 'micp': 96, 'aes-sp3-d': 97, 'encap': 98, 'ether
ip': 99, 'pri-enc': 100, 'gmtp': 101, 'pnni': 102, 'ifmp': 103, 'aris': 104, 'qnx': 105, 'a/n': 106, 'scps': 107, 'snp': 108,
'ipcomp': 109, 'compaq-peer': 110, 'ipx-n-ip': 111, 'vrrp': 112, 'zero': 113, 'pgm': 114, 'iatp': 115, 'ddx': 116, 'l2tp': 117,
'srp': 118, 'stp': 119, 'smp': 120, 'uti': 121, 'sm': 122, 'ptp': 123, 'fire': 124, 'crtp': 125, 'isis': 126, 'crudp': 127, 'sc
copmce': 128, 'sps': 129, 'pipe': 130, 'iplt': 131, 'unas': 132, 'fc': 133, 'ib': 134}, {'CON': 0, 'INT': 1, 'FIN': 2, 'URH':
3, 'REQ': 4, 'ECO': 5, 'RST': 6, 'CLO': 7, 'TXD': 8, 'URN': 9, 'no': 10, 'ACC': 11, 'PAR': 12, 'MAS': 13, 'TST': 14, 'ECR': 1
5}, {'dns': 0, '-': 1, 'http': 2, 'smtp': 3, 'ftp-data': 4, 'ftp': 5, 'ssh': 6, 'pop3': 7, 'snmp': 8, 'ssl': 9, 'irc': 10, 'rad
ius': 11, 'dhcp': 12}]
```

Data Normalization and Training and Testing dataset creation

Processing the Data

In [17]:
```
# Creates "samples" Dataset taken all the columns except the Label
samples = data_full.iloc[:, [i for i in range(0, data_full.shape[1]-1)]].values

# Standardizes the "samples"
samples_standardized = StandardScaler().fit_transform(samples)

# Creates the dataset for the "Label" column
targets = data_full['Label'].values
```

Creating the Testing and Training Datasets

In [19]:
```
# spliting the samples (features) and the targets (Labels) in 70% training and 30% testing
training_samples, testing_samples, training_targets, testing_targets = train_test_split(samples_standardized, targets, test_size=
```

Defining the ML model

Defining the Deep Learning Model

In [24]:
```
# Defining the metrics to calculate in the deep learning model:
METRICS = [
           tf.keras.metrics.BinaryAccuracy(name='accuracy'),
           tf.keras.metrics.Precision(name='precision'),
           tf.keras.metrics.Recall(name="recall"),
           tf.keras.metrics.TruePositives(name="TP"),
           tf.keras.metrics.FalsePositives(name="FP"),
           tf.keras.metrics.TrueNegatives(name="TN"),
           tf.keras.metrics.FalseNegatives(name="FN")
        ]
number_of_labels = 1       # dimensionality of the output space.
input_size       = training_samples.shape[1]
classes          = 1
hidden_neurons   = 3
batch_size       = 1000
epochs           = 30
n_epochs         = 50
learning_rate    = 0.001


model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(hidden_neurons,
                                input_dim=input_size,
                                activation=tf.keras.activations.relu,
                                # kernel_regularizer=tf.keras.regularizers.l2(0.04),
                                name="HiddenLayer1"))

model.add(tf.keras.layers.Dense(classes,
                                input_dim=hidden_neurons,
                                activation=tf.keras.activations.sigmoid,
                                name="Output"))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 HiddenLayer1 (Dense)        (None, 3)                 144

 Output (Dense)              (None, 1)                 4

=================================================================
Total params: 148
Trainable params: 148
Non-trainable params: 0
_________________________________________________________________
```

Running the ML model

Compiling and Training the Deep Learning Model

In [25]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics= METRICS)
```

In [26]:
```
history = model.fit(training_samples,
          training_targets,
          epochs=n_epochs,
          batch_size=batch_size)

loss_and_metrics = model.evaluate(training_samples, training_targets, batch_size=batch_size)
print("The train TP is: "+ str(loss_and_metrics[4]))
print("Metrics: "+ str(loss_and_metrics) )
```

```
981/981 [==============================] - 1s 881us/step - loss: 0.0127 - accuracy: 0.9935 - precision: 0.9194 - recall: 0.96
24 - TP: 50525.0000 - FP: 4431.0000 - TN: 923070.0000 - FN: 1975.0000
Epoch 47/50
981/981 [==============================] - 1s 890us/step - loss: 0.0126 - accuracy: 0.9935 - precision: 0.9203 - recall: 0.96
15 - TP: 50477.0000 - FP: 4370.0000 - TN: 923131.0000 - FN: 2023.0000
Epoch 48/50
981/981 [==============================] - 1s 881us/step - loss: 0.0126 - accuracy: 0.9935 - precision: 0.9203 - recall: 0.96
20 - TP: 50504.0000 - FP: 4376.0000 - TN: 923125.0000 - FN: 1996.0000
Epoch 49/50
981/981 [==============================] - 1s 870us/step - loss: 0.0126 - accuracy: 0.9935 - precision: 0.9208 - recall: 0.96
19 - TP: 50499.0000 - FP: 4344.0000 - TN: 923157.0000 - FN: 2001.0000
Epoch 50/50
981/981 [==============================] - 1s 874us/step - loss: 0.0126 - accuracy: 0.9936 - precision: 0.9212 - recall: 0.96
22 - TP: 50515.0000 - FP: 4324.0000 - TN: 923177.0000 - FN: 1985.0000
981/981 [==============================] - 1s 753us/step - loss: 0.0126 - accuracy: 0.9936 - precision: 0.9346 - recall: 0.94
59 - TP: 49658.0000 - FP: 3474.0000 - TN: 924027.0000 - FN: 2842.0000
The train TP is: 49658.0
Metrics: [0.012592756189405918, 0.9935551285743713, 0.9346156716346741, 0.9458666443824768, 49658.0, 3474.0, 924027.0, 2842.
0]
```

Calculating and Plotting the metrics

Plotting the Metrics

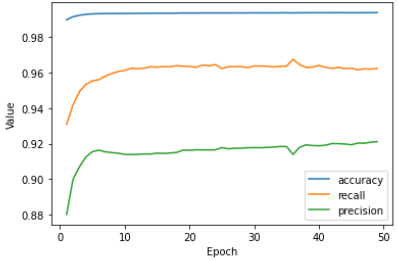
In [27]:

```
#Definition of the plotting function
def plot_curve(epochs, hist, list_of_metrics):
  """Plot a curve of one or more classification metrics vs. epoch."""
  plt.figure()
  plt.xlabel("Epoch")
  plt.ylabel("Value")
  for m in list_of_metrics:
    x = hist[m]
    plt.plot(epochs[1:], x[1:], label=m)
  plt.legend()
print("Defined the plot_curve function.")
```

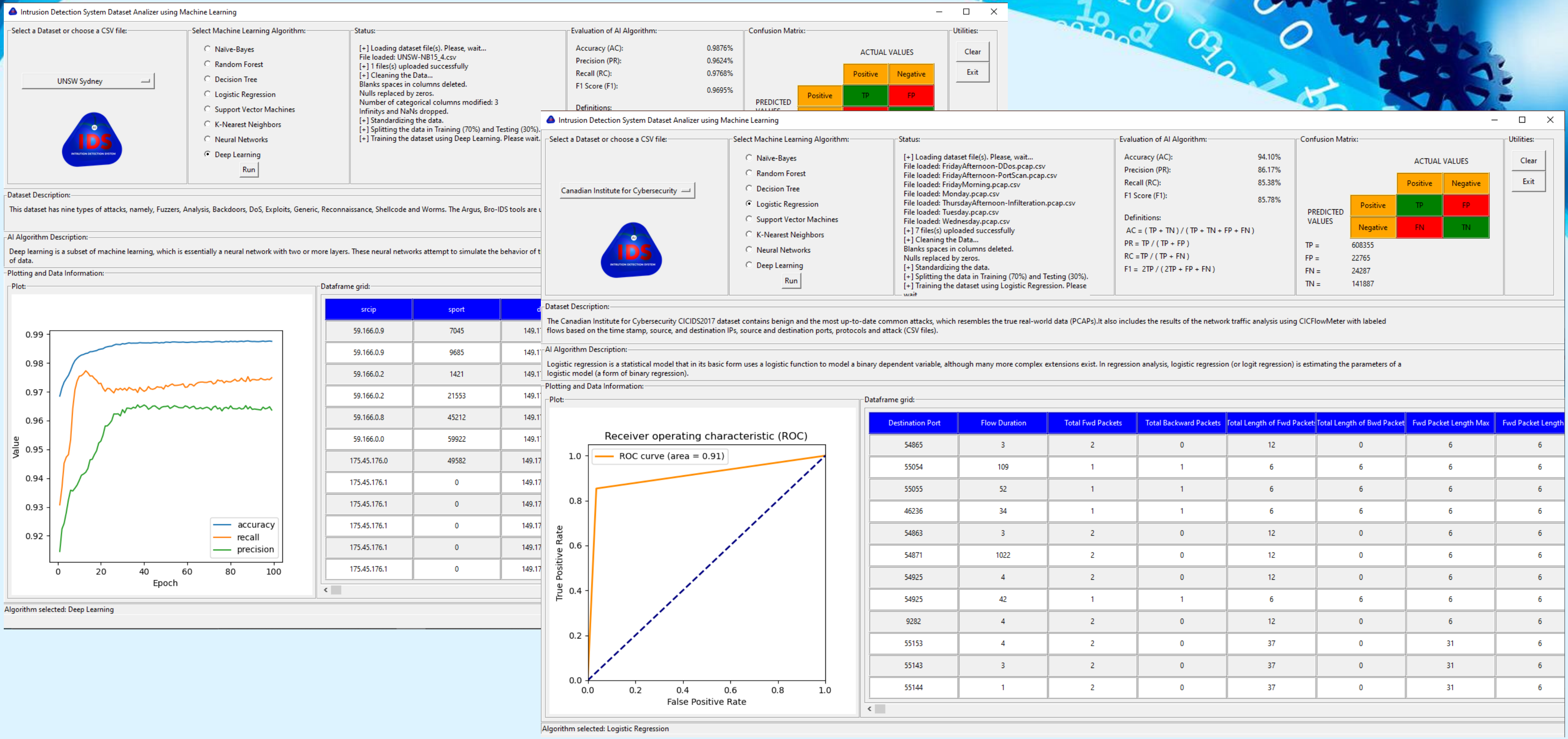
```
Defined the plot_curve function.
```

In [28]:

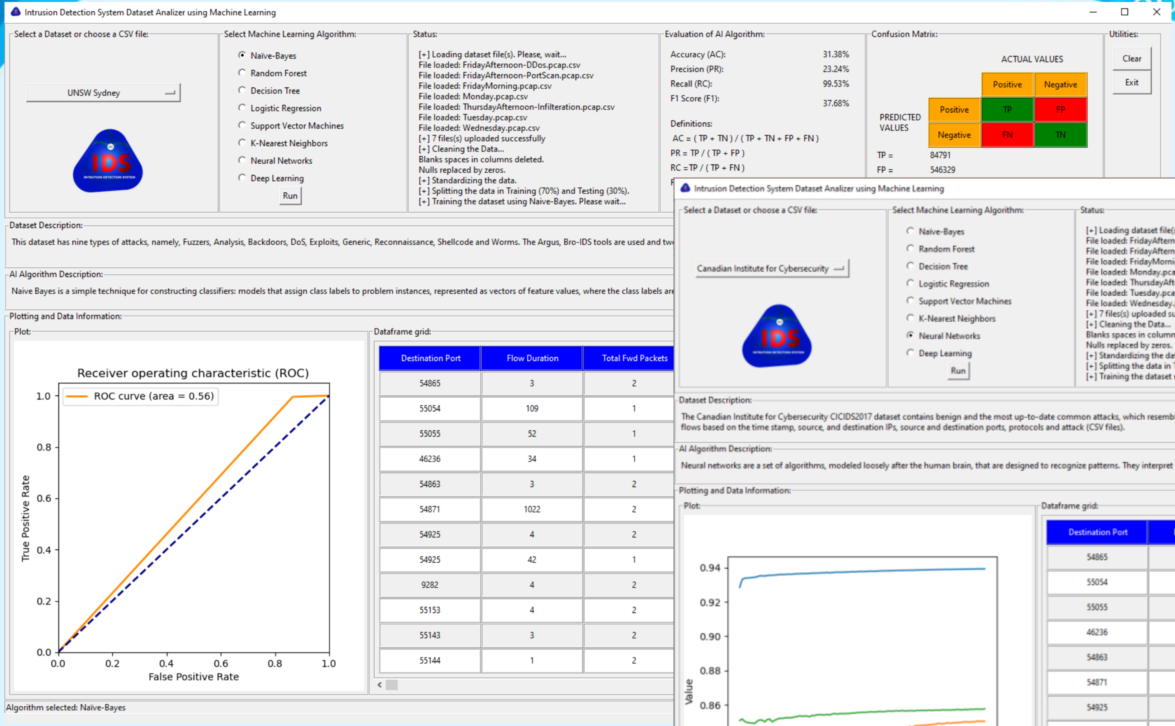
```
# The list of epochs is stored separately from the rest of history.
epochs = history.epoch
# Isolate the classification metric for each epoch.
hist = pd.DataFrame(history.history)
# Plot a graph of the metric(s) vs. epochs.
list_of_metrics_to_plot = ['accuracy', 'recall', 'precision']
plot_curve(epochs, hist, list_of_metrics_to_plot)
```

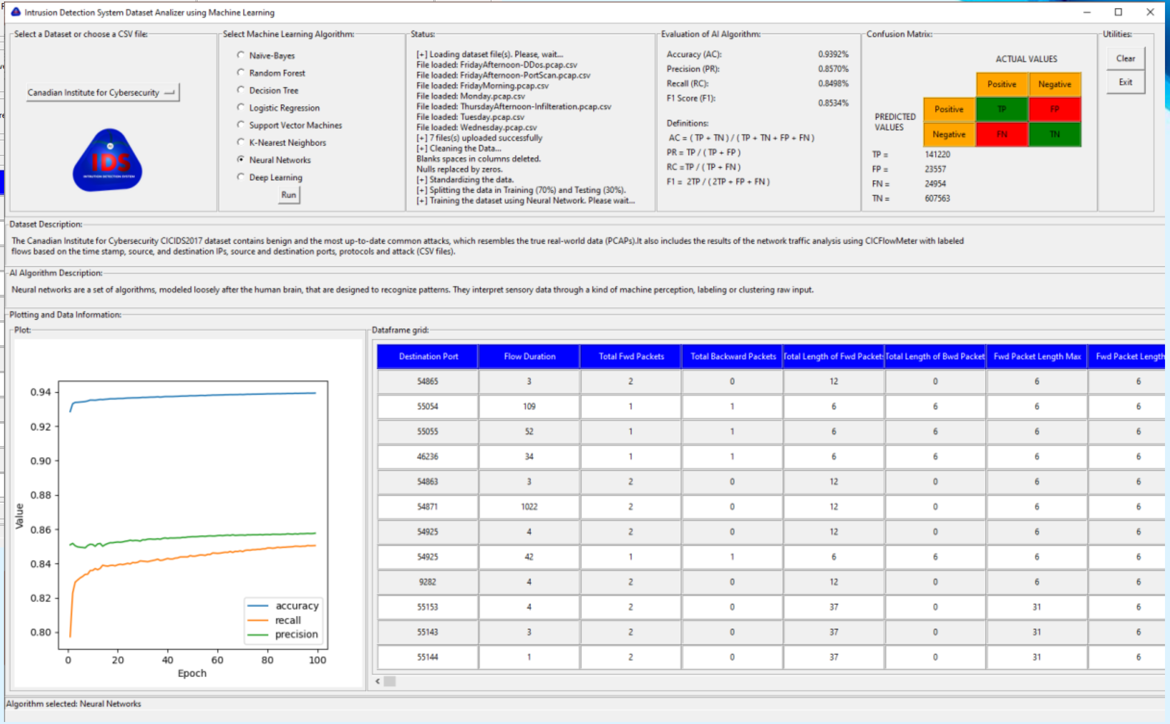


The GUI developed in Python



The GUI developed in Python





Algorithm Comparison: CIC-IDS2017

- Decision tree obtained the best overall performance.
- Deep Neural Network followed was second very close in pefromance to Decision tree (learning rate = 0.001, batch size = 1000, epochs = 50, with no regularization).

Canadian Institute for Cybersecurity: Dataset CIC-IDS2017					
ML Algorithms	**Accuracy**	**Precision**	**Recall**	**F1**	**Timing**
Naïve-Bayes	31.38%	23.24%	99.53%	37.68%	00:00:05
Decision tree	99.87%	99.69%	99.70%	99.69%	00:02:57
Logistic Regression	94.10%	86.17%	85.38%	85.77%	00:06:04
Random Forest	99.90%	99.72%	99.81%	99.77%	00:17:15
KNN	99.57%	99.25%	98.67%	98.96%	05:45:17
Neural Networks (50 epochs)	93.74%	85.61%	84.12%	84.86%	00:02:04
Deep Learning (50 epochs)	97.32%	90.41%	97.50%	93.97%	00:02:29
Neural Networks (100 epochs)	93.96%	85.85%	85.02%	85.43%	00:03:48
Deep Learning (100 epochs)	97.38%	90.35%	97.91%	93.98%	00:05:07

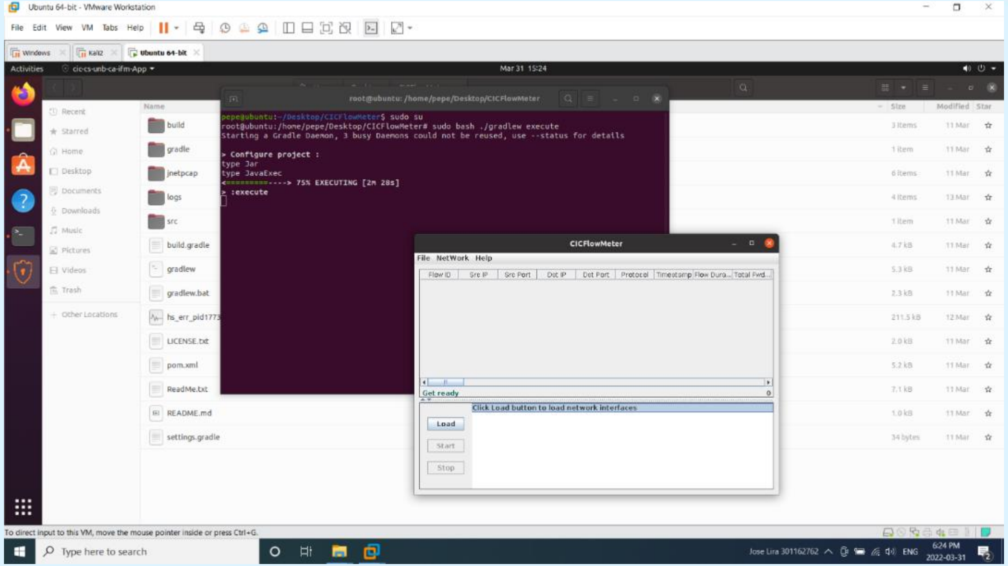
Algorithm Comparison: UNSW-NB15

- Decision tree obtained by far the best overall performance.

University of New South Wales: UNSW-NB15 Dataset					
ML Algorithms	**Accuracy**	**Precision**	**Recall**	**F1**	**Timing**
Naïve-Bayes	98.23%	88.77%	98.52%	93.39%	00:00:03
Decision tree	99.61%	98.51%	98.42%	98.47%	00:00:15
Logistic Regression	98.91%	93.79%	97.86%	95.78%	00:03:34
Random Forest	99.64%	98.83%	98.31%	98.57%	00:05:46
KNN	98.98%	98.15%	93.70%	95.87%	05:02:47
Neural Networks (50 epochs)	98.77%	93.26%	97.31%	95.22%	00:01:50
Deep Learning (50 epochs)	99.05%	95.73%	96.84%	96.28%	00:02:24
Neural Networks (100 epochs)	98.79%	93.17%	97.57%	95.32%	00:03:31
Deep Learning (100 epochs)	99.06%	95.77%	96.81%	96.28%	00:04:45

Short comings

1. Cleaning the Dataset
2. Replicating the Pre-processing of Files in Dataset CIC-IDS2017:
3. Generating my own dataset
4. Programming issues
    1. ML algorithms logic
    2. GUI development



Conclusion

- For the CIC-IDS2017 dataset, the Decision Tree model was overall the best in terms of speed, accuracy, precision, and recall; it was followed closely by the Deep Learning model with 50 epochs (both are suitable for the classification of this dataset).
- The hidden layer added to the Neural Network (Deep Learning) model makes a huge difference in enhancing this dataset's performance.
- For the UNSW-NB15 Dataset, the Decision Tree model was the absolute winner outperforming the Artificial Neurons algorithms and all the other traditional Machine Learning Algorithms in terms of metrics and running speed.
- Running time made a real-life implementation of some of them computationally unfeasible with current hardware (for example, KNN, which took 5h 45min to train the entire dataset CIC-IDS2017 and 5h 02m to train the UNSW-NB15 Dataset).
- It would be beneficial if, for each dataset, the researchers could explain the procedures used during the preprocessing to be reproducible as the datasets available need to be updated by researchers to train with the latest data.
- Updated datasets with more realistic data are needed: Segmented LAN, database with real data flowing (not only DVWA), DDoS coming from different networks, Hybrid Cloud environment, etc.

Life Demonstration

(running the program and modifying parameters life in front of the audience!!)

Thanks!