Project -3, Anomaly Detection

Prateek Baharani 1215199726

*Abstract*— This project aims at understanding the prediction of FeedForward Neural Network model in detecting network attacks. The scope of this project to register attacks based on network packets, that are not normal or similar to known attack packets. In this project we have used KDDTrain.txt data containing normal as well as attack data to train the FNN model and test on a subset of the data. We have observed difference in test accuracy on the basis of train and test subsets.

Keywords— Network Intrusion Detection System, Anomaly detection, Attack model, KDDTrain+.txt training data, Attack types, prediction, accuracy, Sequential classifier, LabelEncoder, OneHotEncoder, sklearn, StandardScaler, confusion matrix.

# Introduction

1. Goal of the project is to understand the prediction of FeedForward Neural Network model in detecting network attacks.
2. This is important in order to flag or register any attack observed in the network, in terms of network packets. This can help in building better Network Intrusion Detection System for known attacks and to an extent new attacks.
3. Used numpy, keras, tenserflow , panda, sklearn libraries, python 3.5 as programming language, Spyder as platform.
4. A better understanding of Anomaly detection tools and how these tools are able to detect anomalies and what all information/data is required to detect an anomaly. Futher to what extent an anomaly detection tool can detect new or exisisting/registered anomalies, based on the training data
5. The first few hours went in understanding the type of model we were going to use. Then developing the basic understanding of libraries, tools, platform that were going to be used in the project. Even before starting the implementation of the project, it was important to first work on a miniproject, with minimalistic goals and expected outcomes. Then making necessary changes to meet the requirements of each part of the project.

# System Models

## System Model

Dell Inspiron, i7 8th Gen, Ubuntu 18.04,

## Software

python 3.5, framework such as keras, tensorflow, Spyder to build machine learning projects and graphviz

## Security Significance

This project aims to better predict the type of packets I.e attack or normal. This is very useful in anomaly based Network Intrusion Detection System, which are able to identify anomalous packets, based on previously learned attack packets as well as normal packets.

However, there are still few challenges which needs to address such as:

* Not all the normal packets can be fed to the IDS, therefore it can have some false positive I.e flagging normal packet as attack packet.
* Flagging new attacks is still difficult, as a result most of the new attack packets are put in false negative.
* Attacks that are a consequence of certain sequence of packets, would be difficult to be highlighted with just the anomaly detection IDS, based on learning through network packets without context.

Possible solution that can be used in order to overcome above mentioned challenges:

* Using stateful machine learning model which is able to learn the attacks based on sequence of packets, such that even if the attack is new, if it follows a questionable sequence, IDS can flag it, also with this, normal packets can be mapped to context I.e stateful, which would be easier of IDS to differentiate and therefore less number of false positive.

# Project Description

## Project Overview

I have completed project 1

## **Tasks: Project-1**

## **Task 1.6.a :** Observing the confusion matrix and prediction of model, when new attack is passed:

**Procedure and Technical details:**

* Taking KDDTrain+.txt as input for training and testing data.
* Parsing data as 2-D matrix, X with rows and columns, each row representing a packet and each column, representing a specific field of a packet.
* Identifying the labels associated to packets, there were two labels, First: Packet type, Second: Difficulty. From this project perspective we have only considered Packet type I.e Normal or Attack, further in Attack, what types of Attack.
* Extracting the label I.e Packet type, into a separate array, for further calculations and parsing.
* Excluding the labels form the data I.e removing the labels columns from X, this is now are training data, which would be used to build the model based on the corresponding label.
* Using LabelEncoder and OneHotEncoder, parsed all the data in X, such that all the strings I.e column 1,2,3 are now converted to integers, such that it could be used to build the model.
* Divided the data in X into following categories:
  + Training data: Normal packets: tagged as ‘normal’, Attack 1: packets tagged as ‘neptune’. Attack 2: packets tagged as ‘teardrop’.
  + Test data: only attack packets. Attack 3: packets tagged as ‘nmap’, Attack 4: packets tagged as ‘smurf’.
  + Relabeling these packets. For the training data, simply labeling normal data as 0 and attack data as 1. storing these labels in an array ‘ylearn’ and all the learning data into ‘Xlearn’. For the test data, seperatly maintaing an array ‘ytest’ to label these test packets as 1, also saving the attack data into seperate 2-D array ‘Xtest’.
* Then for ANN, used StandardScaler for feature scaling using Xlearn and Xtest.
* Initialized FNN using sequential classifier and fitting the ANN to the training data set.
* Passed the test data I.e Xtest to the model in order to get prediction on the Test data.
* Build the confusion matrix on the basis of predicted label and actual label of the test data.
* Plotted the accuracy and loss from the classifier history.
* The confusion matrix is as follows:

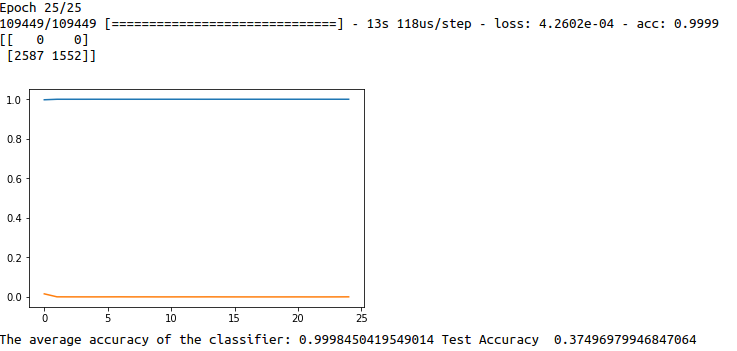
[TN FP]

[FN TP]

* + Confusion Matrix:

[[0 0]

[2587 1552]]

orange indicates loss and blue indicates accuracy learning

**Observation:**

* On passing the new attack type packet I.e A3 and A4 to the model which was trained on A1, A2 and normal packet, the model predicts that the new packet is attack type with Test Accuracy: 0.374969799

## **Task 1.6.b:** Calculating the average accuracy for the model, that it detects the new type of attack to be normal or attack.

* Only attack data was passed as the test data in part 1.
* Using Formula: (TN + TP ) / (TP +TN + FN + FP), we determined that the Test Accuracy is 0.374969799

## **Task 1.6.c**:Identify the differences and similarities between subset (A1,A2 ) and (A3,A4)

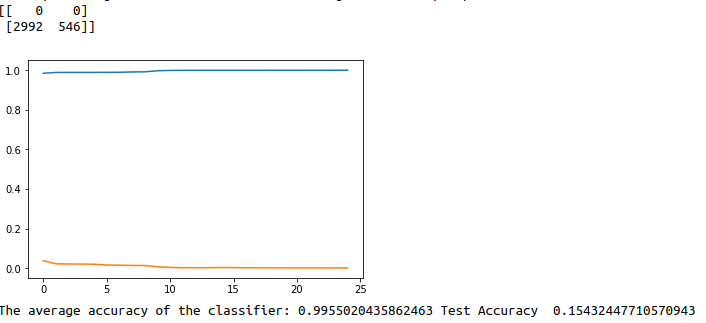
* Attack 1: neptune, Attack 2: teardrop, Attack 3: nmap, Attack 4: smurf.
* Similarity:
  + Both smurf and teardrop are DOS attack, smurf is a DDoS attack while teardrop is a DOS attack.
  + Nmap and smurf
* Differences:
  + Nmap attack is more of a step before the attack, mapping the network and the ports whereas Neptune is Syn Flood attack and thus does not match with other attacks

## **Task 1.6.d**: Verify if changing the training subset, results into better test accuracy

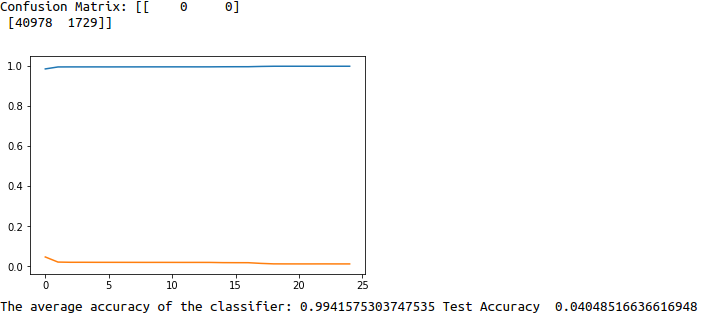
* Attack 1: Neptune, Attack 2: teardrop, Attack 3: nmap, Attack 4: smurf.
* Changing the learning subset from (A1,A2) to (A1, A3) and test subset from (A3,A4) to (A2,A4)
  + The Test Accuracy reduced to 0.1543.
  + Confusion Matrix:

[[ 0 0]

[2992 546]]

orange indicates loss and blue indicates accuracy learning

* Changing the learning subset from (A1,A2) to (A2,A4) and test subset from (A3,A4) to (A1, A3)

orange indicates loss and blue indicates accuracy learning

* + The confusion matrix is:

[[ 0 0]

[40978 1729]]

* + Test Accuracy reduced to 0.0404.
* **Observation:**
  + As per the observation, the test accuracy is different depending on the training subset and test subset.
  + The best accuracy was observed with subset: (A1,A2) as training data and (A3,A4) as test data.

## **Task 1.6.e**: Verify if prediction accuracy was better in last task for a particular subset, explain the similarities or dissimilarities:

* As per the observation the subset with highest accuracy was with (A1,A2) as train data and (A3,A4) as test data.
* This could be because of following reasons:
  + A2 I.e ‘teardrop’ and A4 I.e ‘smurf’ both belong to the same category of attacks I.e DOS attacks.
  + If these two attacks are in the same subset, the model would not be able to use the learning properly.
  + There are no significant similarities b/w nmap and neptune
  + There are no significant similarities b/w smurf and neptune
  + There are no significant similarities b/w teardrop and neptune
  + There are no significant similarities b/w nmap and teardrop
  + There are no significant similarities b/w nmap and smurf

# Discussion and Conclusion

* By removing fields that are not important in the packet from the dataset, we can probably have faster results.
* One of the challenges that we came across was that if the learning and the testing attack data have no similarity among them, then we would get least test accuracy. Thus for demonstration purpose, the attack data in the test and train data were placed strategically.
* At this point we are simply making the model learn about the packets based on the attack type which it was labeled to. As discussed above, if we focus on having stateful learning of packets I.e making the model learn the attacks based on sequence of packets as well, we would have less False postive in terms of attacks detection

##### References

[1] <http://cs231n.github.io/python-numpy-tutorial/>

[2] <https://elitedatascience.com/keras-tutorial-deep-learning-in-python>

[3] <https://keras.io/>

[4] <https://towardsdatascience.com/how-to-build-a-neural-network-with-keras-e8faa33d0ae4>

##### Appendices

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Fri Apr 19 04:47:06 2019

@author: prateek

"""

import pandas as pd

import numpy as np

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

X1 = []

Xtestlist = []

bin\_label= []

bin\_label\_test = []

dataset = pd.read\_csv('~/Desktop/ACNS\_Project3/NSL-KDD/KDDTrain+.txt',header=None)

X = dataset.iloc[:, 0:-2].values

label\_column = dataset.iloc[:, -2].values

labelencoder\_X\_1 = LabelEncoder()

X[:,1] = labelencoder\_X\_1.fit\_transform(X[:,1])

labelencoder\_X\_1 = LabelEncoder()

X[:,2] = labelencoder\_X\_1.fit\_transform(X[:,2])

labelencoder\_X\_1 = LabelEncoder()

X[:,3] = labelencoder\_X\_1.fit\_transform(X[:,3])

#print (X[:,:4])

#onehotencoder = OneHotEncoder(categorical\_features = [1])

#X = onehotencoder.fit\_transform(X).toarray()

onehotencoder = OneHotEncoder(categorical\_features = [1,2,3])

X = onehotencoder.fit\_transform(X).toarray()

X = X[:,1:]

length=len(X)

for i in range(length):

if(label\_column[i]=='normal'):

bin\_label.append(0)

X1.append(X[i,:])

else:

if( label\_column[i]=='neptune' or label\_column[i]=='smurf'):

bin\_label.append(1)

X1.append(X[i,:])

else:

if label\_column[i]=='teardrop' or label\_column[i]=='nmap' :

Xtestlist.append(X[i,:])

bin\_label\_test.append(1)

#print (X[i,:])

Xlearn= np.asarray(X1)

Xtest = np.asarray(Xtestlist)

ylearn= np.asarray(bin\_label)

ytest= np.asarray(bin\_label\_test)

#print (X[:5,:])

sc = StandardScaler()

Xlearn = sc.fit\_transform(Xlearn)

Xtest = sc.transform(Xtest)

#sc = MinMaxScaler(feature\_range = (0, 1))

#Xlearn = sc.fit\_transform(Xlearn)

classifier = Sequential()

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = len(Xlearn[0])))

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))

classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

classifierHistory = classifier.fit(Xlearn, ylearn, batch\_size = 10, epochs = 25)

accavg = np.mean(classifierHistory.history['acc'])

y\_pred = classifier.predict(Xtest)

y\_pred = (y\_pred > 0.9)

cm = confusion\_matrix(ytest, y\_pred)

testacc= (cm[0][0]+cm[1][1])/ (cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])

print ('Confusion Matrix:',cm)

plt.figure()

#plt.plot(classifierHistory.history.)

plt.plot(classifierHistory.history['acc'] )

plt.plot(classifierHistory.history['loss'] )

plt.show()

print ('The average accuracy of the classifier:',accavg, 'Test Accuracy ',testacc)