Student Name: Tiriac Ioana-Raluca  
Email: itiriac@asu.edu  
Submission Date: 6.12.2021  
Class Name and Term: CSE548 Fall B 2021

Machine Learning-Based Anomaly Detection Solutions

# Project Overview

In this lab, we’ll use NSL-KDD dataset and Feed-forward Neural Network (FNN) for network anomaly detection and attack analysis. We will create customized training and testing datasets for several different data analysis scenarios based on the supplied NSL-KDD dataset. Based on the training and testing results of each created scenario, we need to address a few questions regarding ML-based anomaly detection accuracy and efficiency.

# Network Setup

The initial set-up of the virtual infrastructure as I have configured it in VirtualBox for this VM – it is simply connected via NAT.

Graphical user interface, text, application

Description automatically generated

# Software

* -  Ubuntu 18.04 LTS
* -  Python – https://www.python.org/
* -  Anaconda - https://www.anaconda.com/
* -  TensorFlow - https://www.tensorflow.org/
* -  NSL-KDD dataset - https://www.unb.ca/cic/datasets/nsl.html

# Project Description

In this assignment I went through the background labs first to get a good understanding of the lab assessments. In CS-ML-00101 the NSL-KDD Dataset is described and the four datasets it contains, along with features and data distribution. CS-ML-00201 is an ML introduction describing Feed-Forward Neural Network solutions like data pre-processing, building FNN training module, evaluating training results and visualizing FNN training and testing results by using the sklearn and keras libraries. The feed-forward neural trains on the provided attack labels to be able to detect patterns of abnormal traffic in the future. After training, testing data set with similar labels is used to determine the accuracy of the FNN.

As a warm-up to the lab assessments I have run fnn\_sample.py with the Python spyder gui and got the resulting model accuracy & model loss plot and also the confusion matrix.

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

*Create new data modules for anomaly detection*

Went through lab CL-ML-0200 on how to use data processing modules like DataExtractor.py and used this class to generate all needed training & testing data sets for the lab assessment.

For ex. For scenario 1, I ran DataExtractor.py and input to CLI 1,3 (attack 1 for A1 DoS and attack 3 for A3 User 2 Root ) for the training data set and 2,4 for the testing data set and it generated the needed Training-a1-a3.csv & Testing-a2-a4.csv files for Scenario A (SA). I ran *DataExtractor.py* similarly for Scenario B and Scenario C.

Graphical user interface, text

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

*Lab assessment*

1. *(35 points) Using the lab screenshot feature to document ML running results only. Provide sufficient but concise explanations on the generated results for each given training/testing scenarios.*

The accuracy provided by the sample\_fnn.py is the training accuracy and we’re required to provide accuracy of testing. The results of the 3 scenarios A, B, C presented below will show the *testing accuracy* calculated from the confusion matrix using the formula:

*Testing Accuracy = (TP + TN) / (TN + FP + TP + FN)*

**Scenario A**

*Loss=0.0052, Testing Accuracy=11505/15017=****0.7661****, Confusion Matrix: TN=8701, FP=1010, FN=2502, TP=2804*

**Chart, line chart

Description automatically generated**

Text

Description automatically generated screenshot acc = training acc

Explanation: This scenario uses A1 (Ddos) & A3(U2R) attacks as the training dataset for the FNN and the A2() and A4 attacks as the testing dataset, hence having no overlap in terms of attacks between the training and testing dataset. This is the worst performing scenario in terms of testing accuracy (formula described above !). The FNN gets trained on different set of attacks than the ones it gets tested on, so it has a poor performance.

**Scenario B Results:**

*Loss=0.0449, Testing Accuracy=15226/17171=****0.8867****, Confusion Matrix: TN=8840, FP=871, FN=1074, TP=6386*

Chart

Description automatically generated

Text

Description automatically generatedscreenshot accuracy is training accuracy.

Explanation: This scenario uses A1-“DoS” and A2-“Probe” attacks as training and only A1-“DoS” attacks in the test cases. Basically, the FNN gets trained on all the attacks on which it will be tested later on.We expect a much better performance than scenario A in terms of accuracy testing (***0.8867***) and this is in fact the case here!

**Scenario C Results :** *Loss=0.0127, Testing Accuracy=17525/19659=****0.8914****, Confusion Matrix: TN=8872, FP=839, FN=1295, TP=8653*

Chart, line chart

Description automatically generated

Text

Description automatically generated

Explanation: This scenario has the most interesting results because it trains the FNN on 2 types of attacks : A1 and A2 and tests it on A1, A2, A3 having a better performance than the other scenarios even though it introduces testing on a new kind of attack the FNN hasn’t seen during the training phase !

1. *(20 points) Submit the updated fnn\_simple.py and provide sufficient comments to illustrate your updated codes*

Graphical user interface, text

Description automatically generated with medium confidence

**fnn\_sample.py has been modified**: the code that realizes the hot split of the test and training data set has been commented out and instead the file now takes as input from the CLI one of the 3 scenarios, which are lists of predefined .csv file names: each having a training and testing csv that was generated previously for each given Scenario by the DataExtractor.py

Next, I’ve used a data preprocessing module to read the training and testing dataset, after the user enters the wanted scenario:

Graphical user interface, text

Description automatically generated

Everything else up until PART II : Building the FNN , the hot split of the training and testing data set have been commented out :

Graphical user interface, text

Description automatically generated

1. *(45 points) Submit the report to provide in-depth analysis to address the questions given in Task 3.*
2. The Scenario producing the best results in terms of testing accuracy is **Scenario C**. The best results are obtained when training using A1 “DoS” & A2 “Probe” datasets rather than A1 “DoS” & A3 “U2R”. This is most likely because A2 “Probe” dataset is much larger than the A3 “U2R”, A2 holding 9.11% and A3 0.04% of the datasets, which is a representation of the attacks happening on the Internet. Because of that, training with more “Probe” attacks rather than “U2R” ones yield to a better statistical outcome in the FNN.
3. The average testing accuracy between Scenario A, B and C =approx. **0.8481**. When considering only Scenario A and C, = **0.8288**, which is inferior to the average including Scenario B. This shows that when attacks present cases for which the FNN has not been trained for, the average performance sinks.
4. As noticed, when a Scenario is presented with attacks for which it has not been trained, it will have a poor testing accuracy. For Scenario A, A3 & A4 do have a common ground because they are similar, but hold a much smaller percentage of the dataset than A1 & A2 attacks.
5. The A4 R2L and A3 U2R use similar patters – for instance they are mostly using TCP protocol. Another issue is determined by the fact that A3(“U2R”) and A4(“R2L”) attacks are numerically much inferior to A1(“DoS”) and A2(“Probe”) attacks, which reflects negatively on the training and testing accuracy.

# Appendix B: Attached files

* Report
* Fnn\_sample.py

# References

Data preprocessing:

* https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621
* https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html
* https://scikit-learn.org/stable/modules/preprocessing.html

Build ANN:

* https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/