Student Name: Roey Ben Hayun  
Email: [rroeybh@gmail.com](mailto:rroeybh@gmail.com), rbenhayu@asu.edu  
Submission Date: 27.11.2022  
Class Name and Term: CSE548 Summer 2021

Machine-Learning Based Anomaly Detection Solutions

# Project Overview

In this lab, I used NSL-KDD dataset and Feed-forward Neural Network (FNN) for network anomaly detection and attack analysis. I created a customized training and testing datasets for several different data analysis scenarios based on the supplied NSL-KDD dataset.

# Network Setup

# Software

For this project I have used the following ML tools

* Python (anaconda)
* Pandas
* Numpy
* NSL-KDD dataset
* Keras
* Matplotlib

# Project Description

Creating the client and Server/Gateway VM is explain in the supporting docs.

**Preprocessing: Create the different scenarios test and train datasets**

For this project we had to generate data for three different scenarios:

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Scenario A | Scenario B | Scenario C |
| Training | A1, A3, N | A1, A2, N | A1, A2, N |
| Testing | A2, A4, N | A1, N | A1, A2, A3, N |

Where:

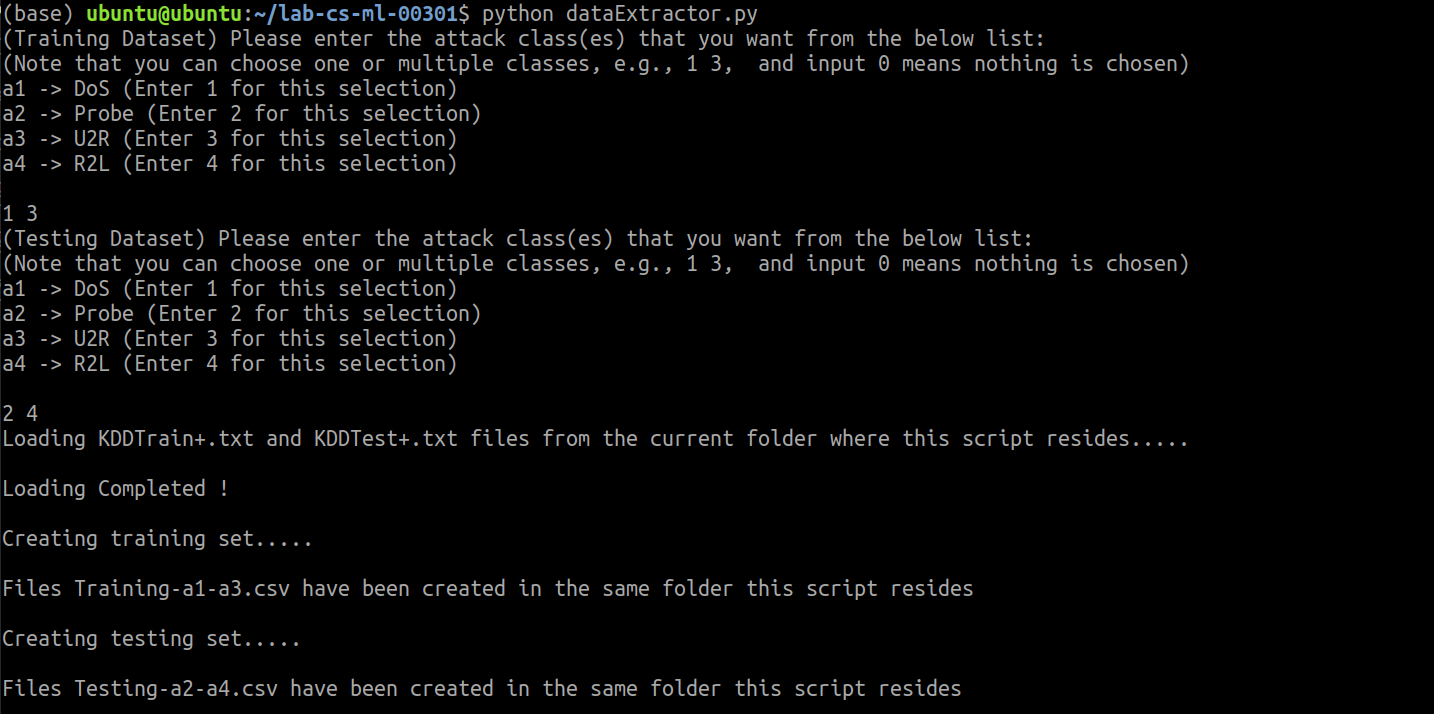
A1: Denial of Service (DoS)

A2: Probe

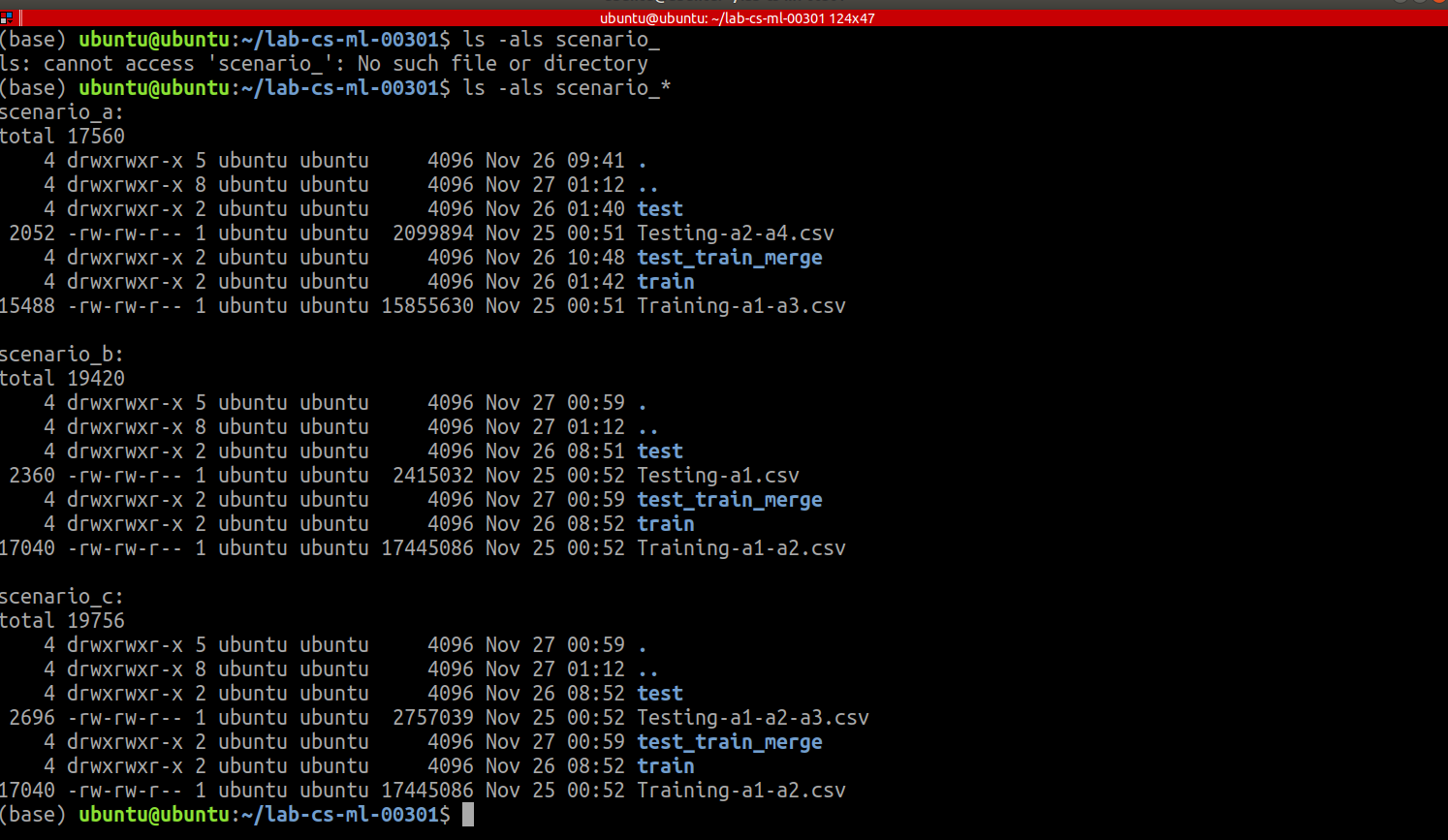
A3: User to Root(U2R)

A4: Remote to Local (R2L).

To create the datasets, I run dataExtreactor.py and specified the combination for each scenario. For example, scenario A, I selected 1 3 for the training dataset and 2 4 for the testing dataset



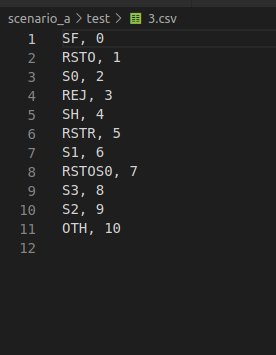
For easy preprocessing I have created the following directory structure After creating the datasets, I moved them to the scenario directory.



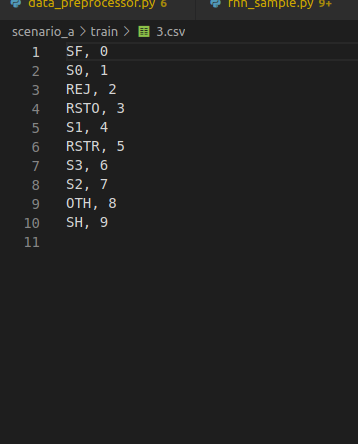
This test/train split, cause a different number of categorical features between the test and train datasets [3]. The implication weas that after the one-hot encoding, X\_train and X\_test had features (column) mismatch. This caused the fnn to fail when trying to predict Test set results.

To validate the categorical features difference, I run the category mapper for scenario A test and train sets. The output of the categoryMapper is 4 files 1,2,3.CSV are the three categorical inputs and 41.csv is the categorical output

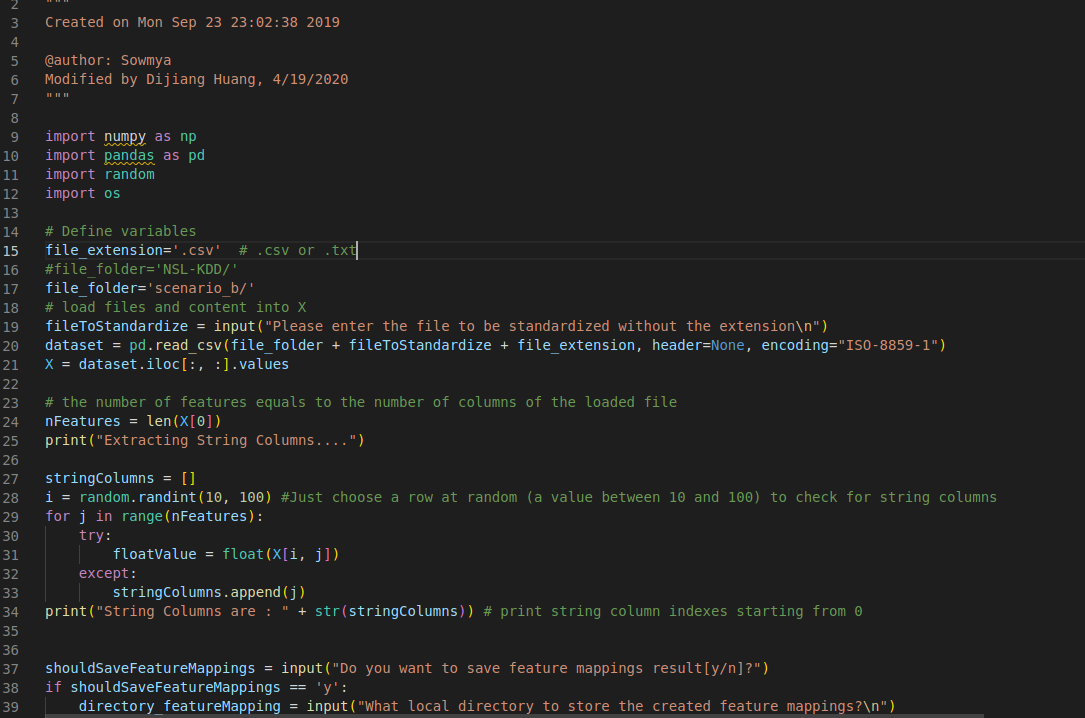
For example, second feature of the test dataset:



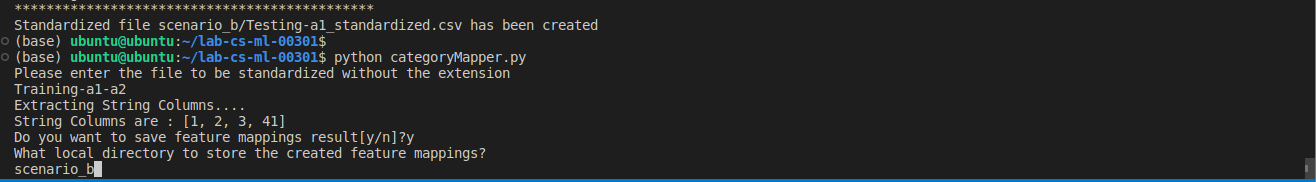
And the second feature of the train dataset:



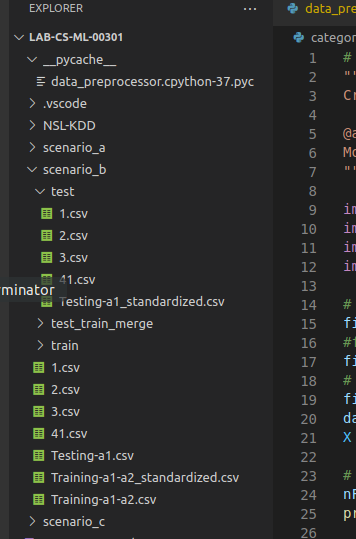
We can see different number of categories for this feature. This means that after the one-hot encoding we will have different number of features (columns). I repeated this process for all the scenarios. Since this was only one time procedure, I manually set the base directory path (for example here, scenario\_b). I moved the files manually to the proper dir locations



Input params for categoryMapper.py



Output



Copy manually CSVs to scenario\_b/train

One way to overcome this, is by taking the common features between the test and train dataset. This can be achieved by using pandas merge with inner join option [1]

The process:

For each categorical column csv path:

Get the test categorical features

Get the train categorical features

Drop the number column for test set

Drop the number column for train set

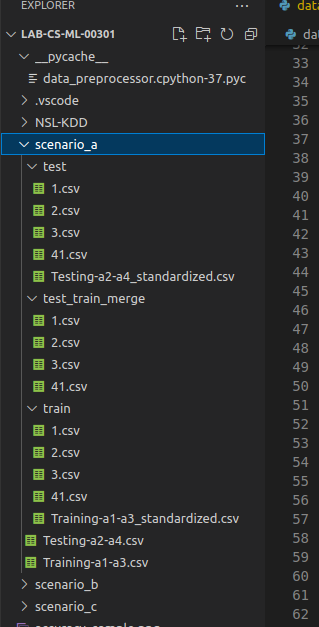
Merge train and test using inner join (take the common)

Insert a column (with shuffled numbers with the max as the number of the features) to the merged file

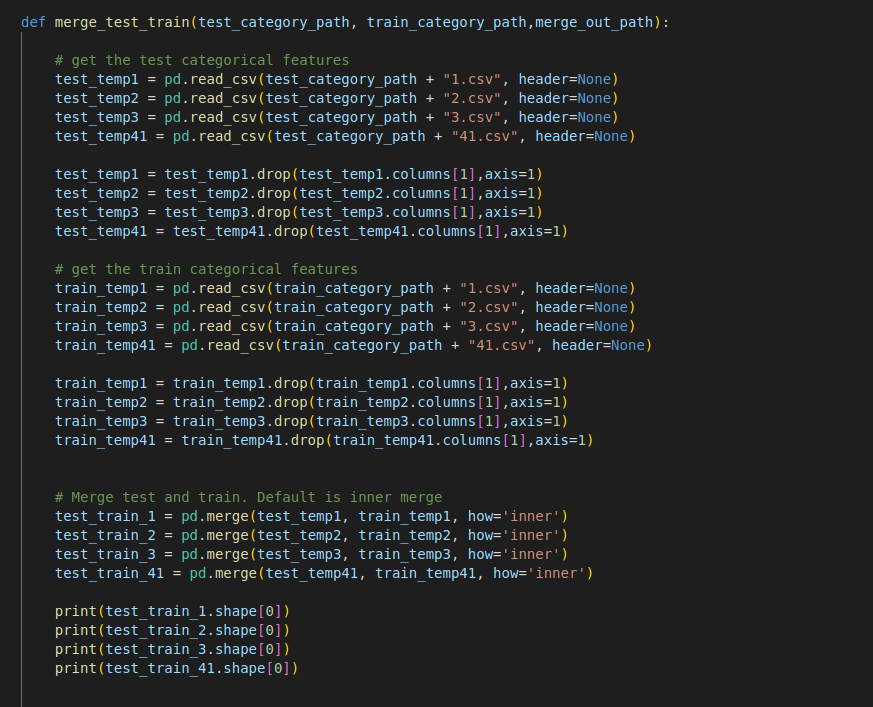
Save the merged file

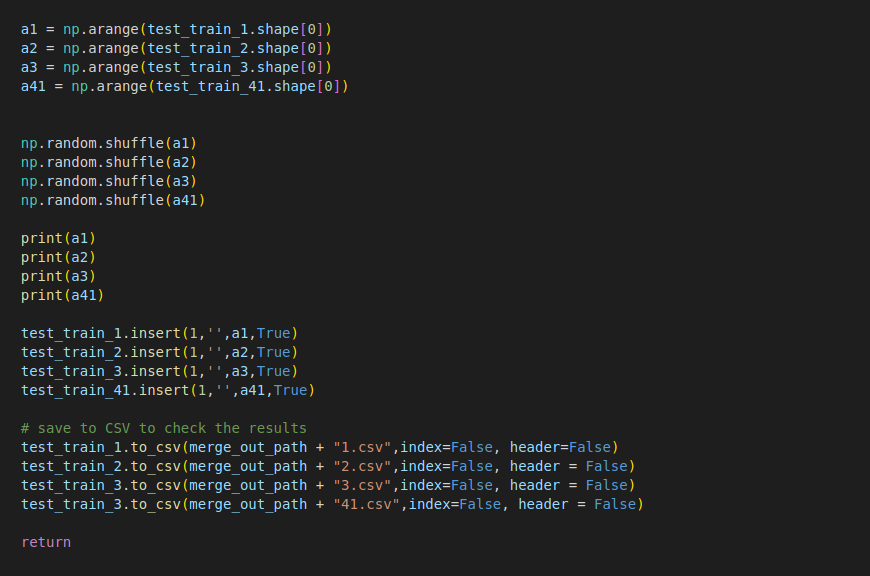
Merged files were saved into the test\_train\_merge in each scenario directory

scenario\_a for example



I added a helper method, merge\_test\_train in data\_preprocessor.py



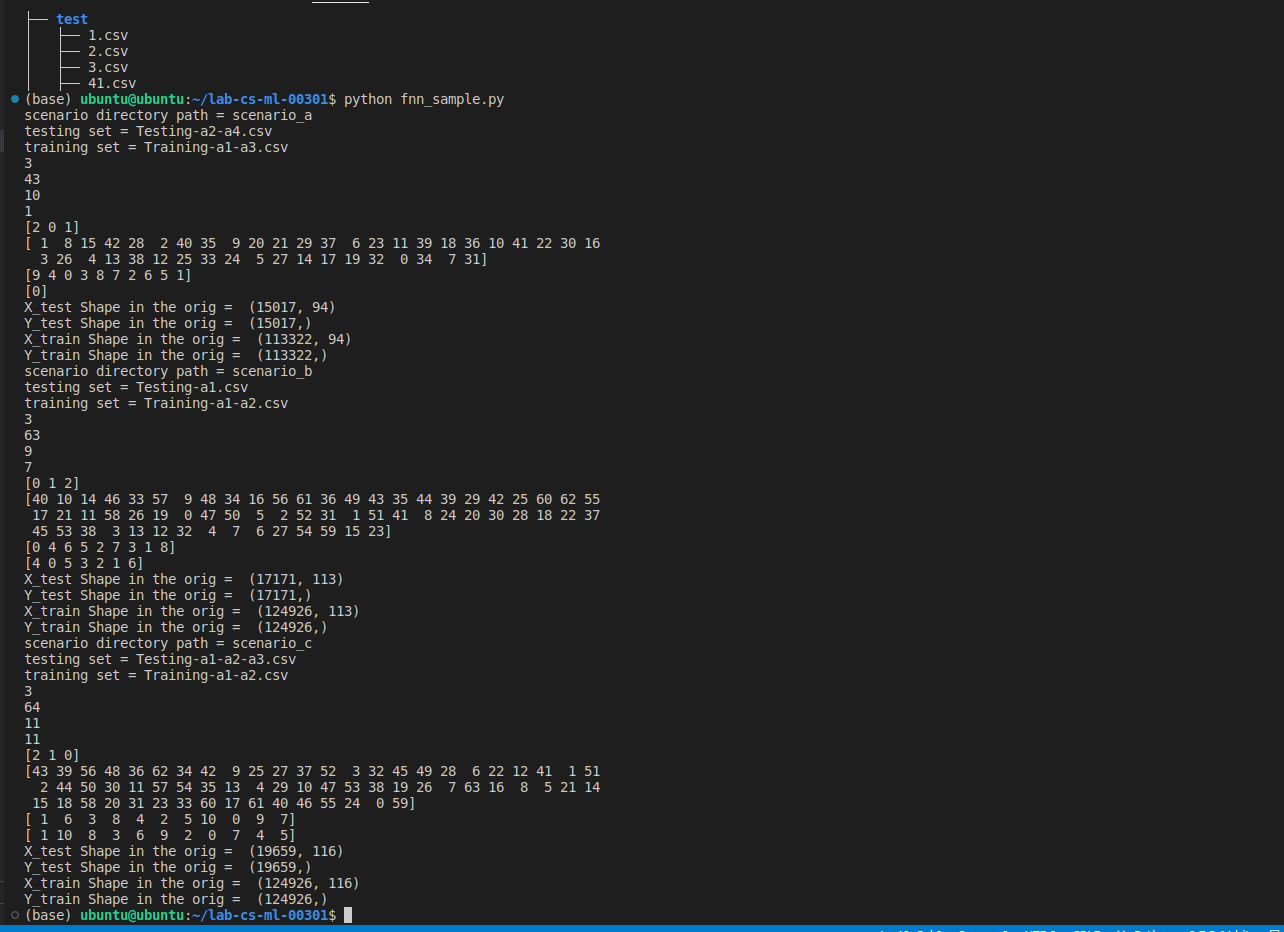


Now, let’s create the merged files. From fnn\_sample.py I called merge\_test\_train (once merged files are ready there is no need to call this method again)

I added a printout to validate train and test have the same number of columns

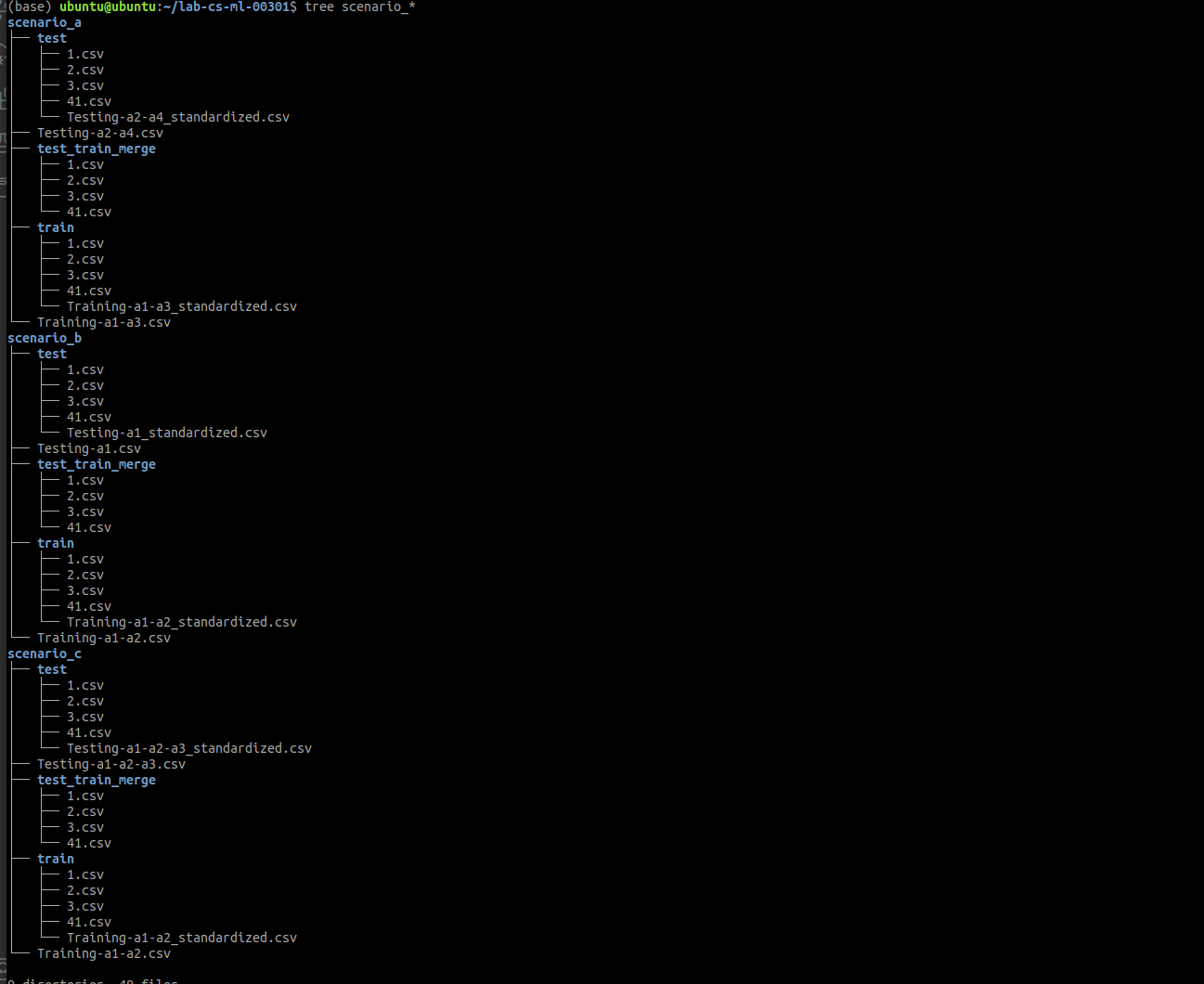


The outpu



You can see the same number of features for train/test for each scenario

Using tree command here is the final directory structure.



Now, let’s set the merged\_files\_ready flag to True.

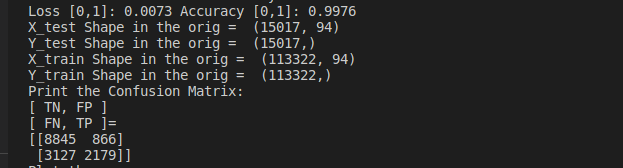
**Anomaly Detection Analysis**

Let’s run the FNN and examine the results for each scenario

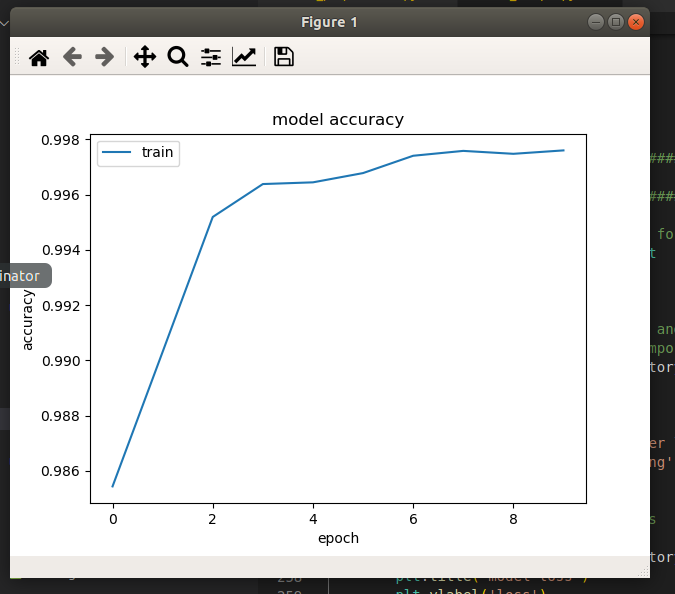
**Scenario A**

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Scenario A | Scenario B | Scenario C |
| Training | A1, A3, N | A1, A2, N | A1, A2, N |
| Testing | A2, A4, N | A1, N | A1, A2, A3, N |

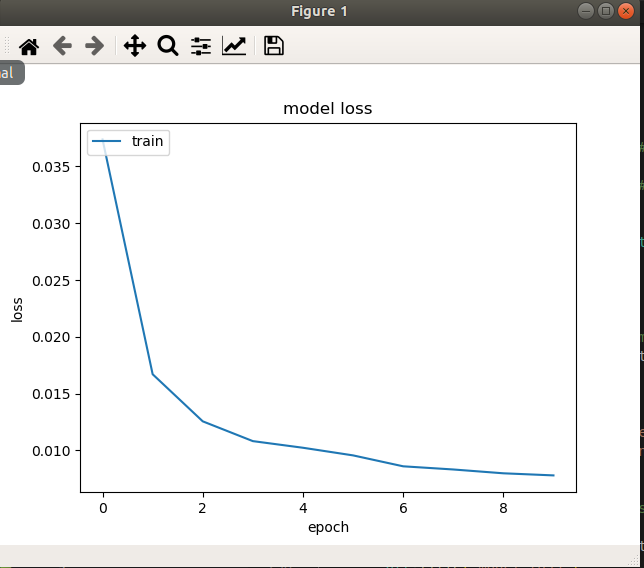
**Train** loss and accuracy and confusion matrix



Model accuracy



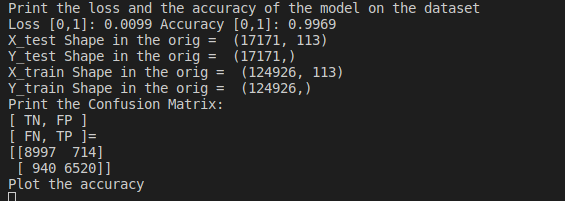
Model loss



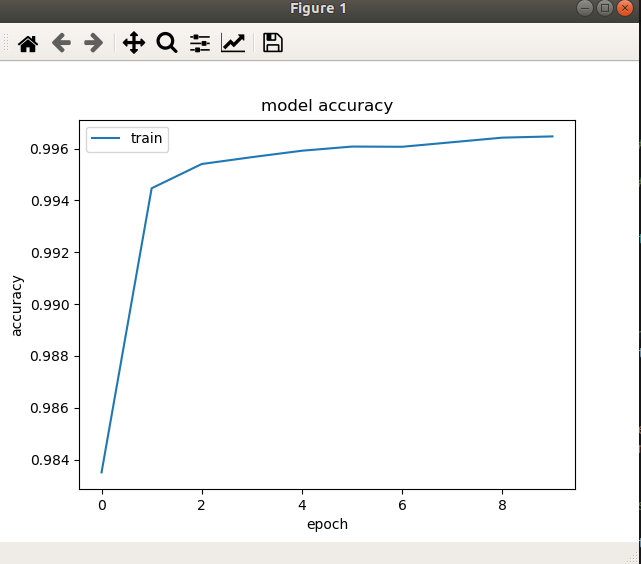
**Scenario B**

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Scenario A | Scenario B | Scenario C |
| Training | A1, A3, N | A1, A2, N | A1, A2, N |
| Testing | A2, A4, N | A1, N | A1, A2, A3, N |

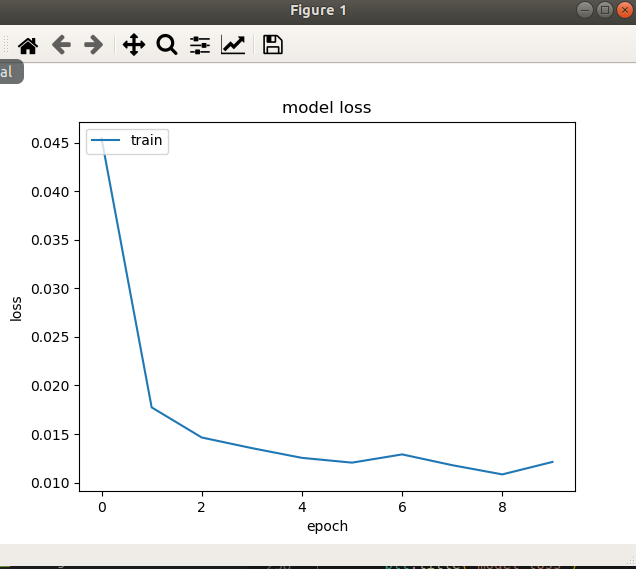
**Train** loss and accuracy and confusion matrix



Model accuracy



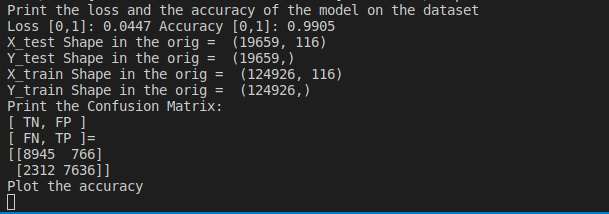
Model loss



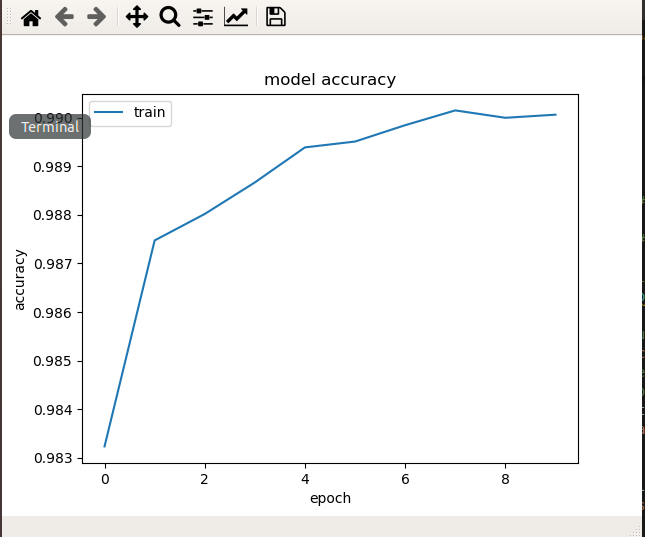
**Scenario C**

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Scenario A | Scenario B | Scenario C |
| Training | A1, A3, N | A1, A2, N | A1, A2, N |
| Testing | A2, A4, N | A1, N | A1, A2, A3, N |

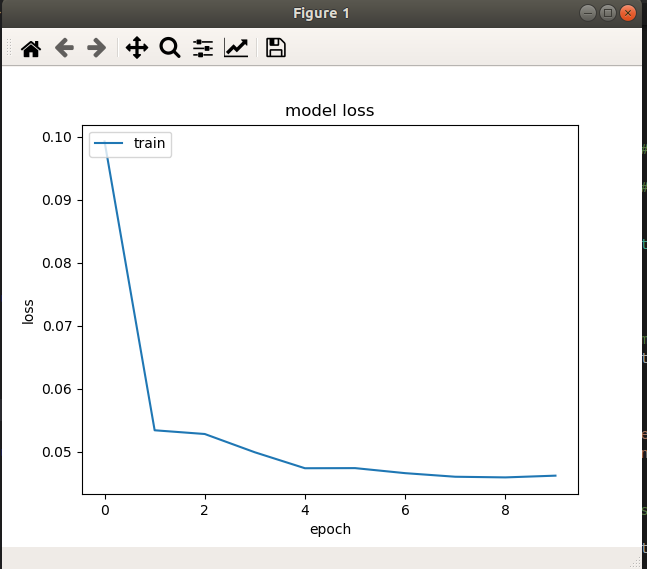
**Train** loss and accuracy and confusion matrix



Model accuracy



Model loss



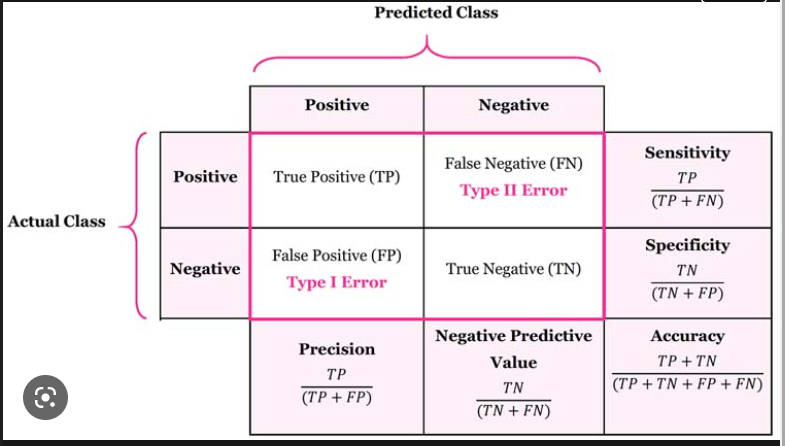
**(1)Test accuracy**

The reported accuracy and loss when running the fnn.py are for the training dataset.

Let’s find the test accuracy.

Accuracy =

Reminder



Now, let’s plug in the numbers for each scenario

Confusion Matrix = [ TN, FP ]

[ FN, TP ]

Scenario A confusion matrix= [ 8845, 866 ]

[ 3127, 2179]

Scenario B confusion matrix= [ 8997, 714]

[ 940, 6520]

Scenario C confusion matrix= [ 8945, 766]

[ 2312, 7636]

|  |  |  |  |
| --- | --- | --- | --- |
| Test results | Scenario A | Scenario B | Scenario C |
| Accuracy | 0.735299993 | 0.903674801 | 0.84343049 |
|  |  |  |  |

We can see here that poor results are expected for scenario is since there is no overlap between the training and testing data

For scenario B we got the best accuracy. We can see the model was trained on both A1 and A2 tested on A1.

For B, we got a better result compared to A since the A1 and A2 were both on train and test. However, ANN was not trained with A3 and was part of the testing dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Scenario A | Scenario B | Scenario C |
| Training | A1, A3, N | A1, A2, N | A1, A2, N |
| Testing | A2, A4, N | A1, N | A1, A2, A3, N |

Where:

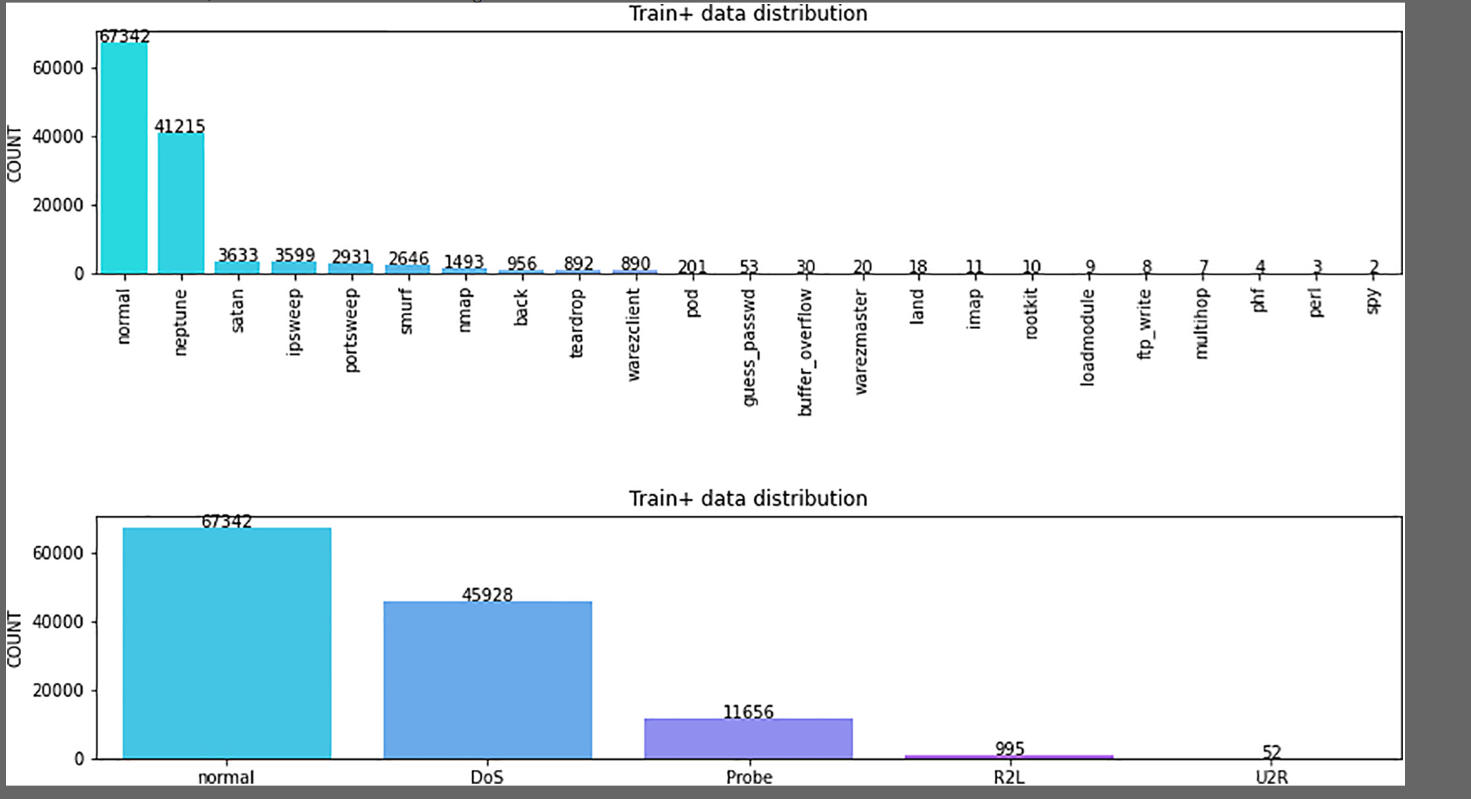
A1: Denial of Service (DoS)

A2: Probe

A3: User to Root(U2R)

A4: Remote to Local (R2L).

Train and data distribution of NSL-KDD dataset



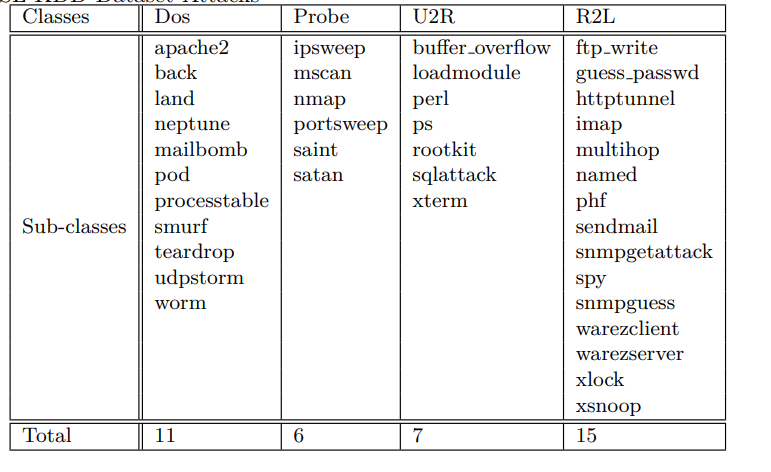
**(2)**

The average accuracy of scenarios A, B and C is 0.827468428 this is less. The average of scenarios A and C

Is 0.789365242. This again strength the argument that when ANN trained with attack type which later will be part of the testing dataset will give better accuracy

**(3)**

There are different attacks associated with each group (A1/A2/A3/A4)



We can see, that the majority of the attacks are belongs to DoS (A1) and Probe (A2). It is safe to reason that a NN model which was trained and tested with the datasets contains the A1 and A2 will yield good results. We can see that not following this guideline, like in Scenario A, will yield poor results.

**(4)**

Let’s take for example, DoS and Probe attacks, from [2]

In a Denial of Service (DoS) attack, the attacker makes some computing or memory resource too busy, or too full, to handle legitimate users’ requests. But before an attacker launches an attack on a given site, the attacker typically probes the victim’s network or host by searching these networks and hosts for open ports. This is done using a sweeping process across the different hosts on a network and within a single host for services that are up by probing the open ports. This is referred to as Probe attacks.

The attacks are not the same, but they have dependencies. This is another explanation for the good accuracy results of Scenario B (and also C).

# Conclusion

We trained and tested ANN using NSL-KDD dataset arranged in different test/train schemas to detect/predict network anomalies.

# Appendix B: Attached files

All project files are under project4 in the following git repo

<https://github.com/roeybenhayun/cse548-advanced_computer_network_security>

I have modified the following sources:

* fnn\_sample.py
* data\_preprocessor.py
* categoryMapper.py

New files:

* Datasets (train/test) for all scenarios
* Datasets (train/test) category mapping CSV for all scenarios
* Datasets (train/test) merged categories mapping CSVs for all scenarios

# References

[1] <https://www.kaggle.com/discussions/getting-started/50008>

[2] <https://www.cs.ucdavis.edu/~vemuri/papers/Detecting%20DoS%20and%20Probe%20Attacks%20using%20PCA.pdf>

[3] <https://medium.com/@vaibhavshukla182/how-to-solve-mismatch-in-train-and-test-set-after-categorical-encoding-8320ed03552f>