

Project Report 4 - Machine Learning-Based Anomaly Detection

Student Name: Mehran Tajbakhsh

Email: mtajbakh@asu.edu

Submission Date: 7/7/2022

Class Name and Term: CSE548 Summer 2022

I. PROJECT OVERVIEW

In this lab I used the FNN algorithm to implement a machine learning-based anomaly detection model for network IDS. In this project, I used the pandas module to implement an ML model and analysis, the Numpy module for number processing, the data_preprocessor module for preprocessing the input data and NSL-KDD as the dataset. I made some changes in the fnn_sample.py code to accept different input as testing and training datasets.

II. NETWORK SETUP

I didn't have anything to do in this section. A ready-to-use VirtualBox VM image with installed tools and software was provided to us. I downloaded it from Google Drive and set up my VM.

III. SOFTWARE

- Ubuntu 18.04 LTS
- Python – <https://www.python.org/>
- Spyder – A GUI of Python - <https://www.spyder-ide.org/>
- Pandas - <https://pandas.pydata.org/>
- NSL-KDD dataset - <https://www.unb.ca/cic/datasets/nsf.html>

IV. PROJECT DESCRIPTION

In this lab, I needed to create three custom datasets based on the NSL-KDD dataset. I used the DataExtractor.py program to create datasets as shown in the following table:

Datasets	Scenario A (S_A)	Scenario B (S_B)	Scenario C (S_C)
Training Dataset	A_1, A_3, N	A_1, A_2, N	A_1, A_2, N
Testing Dataset	A_2, A_4, N	A_1, N	A_1, A_2, A_3, N
Attacks as Training	DoS, U2R	DoS, Probe	DoS, Probe
Attacks as Testing	Probe, R2L	DoS	DoS, Probe, U2R

To create the customized datasets, I ran the DataExtractor.py program three times in the Spyder. I followed the instructions that the program showed on the console and created three different dataset as input for the FNN ML model to detect anomalies in different attacks as shown in the above table.

Task 2 Create Data Modules for Anomaly Detection

A. Create Scenario A dataset (Figure-1):

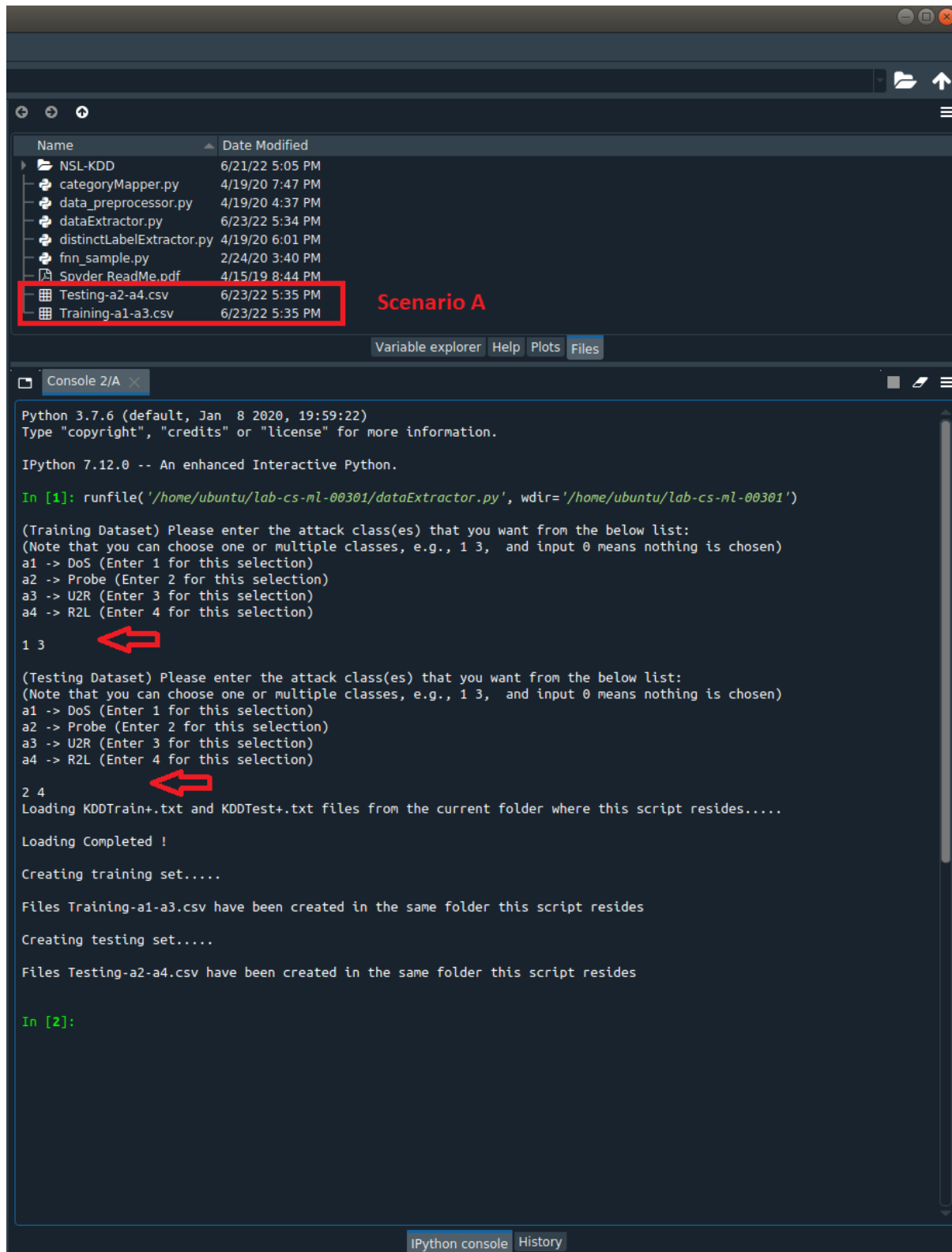


Figure-1 Create scenario A dataset

B. Create Scenario B dataset (Figure-2):

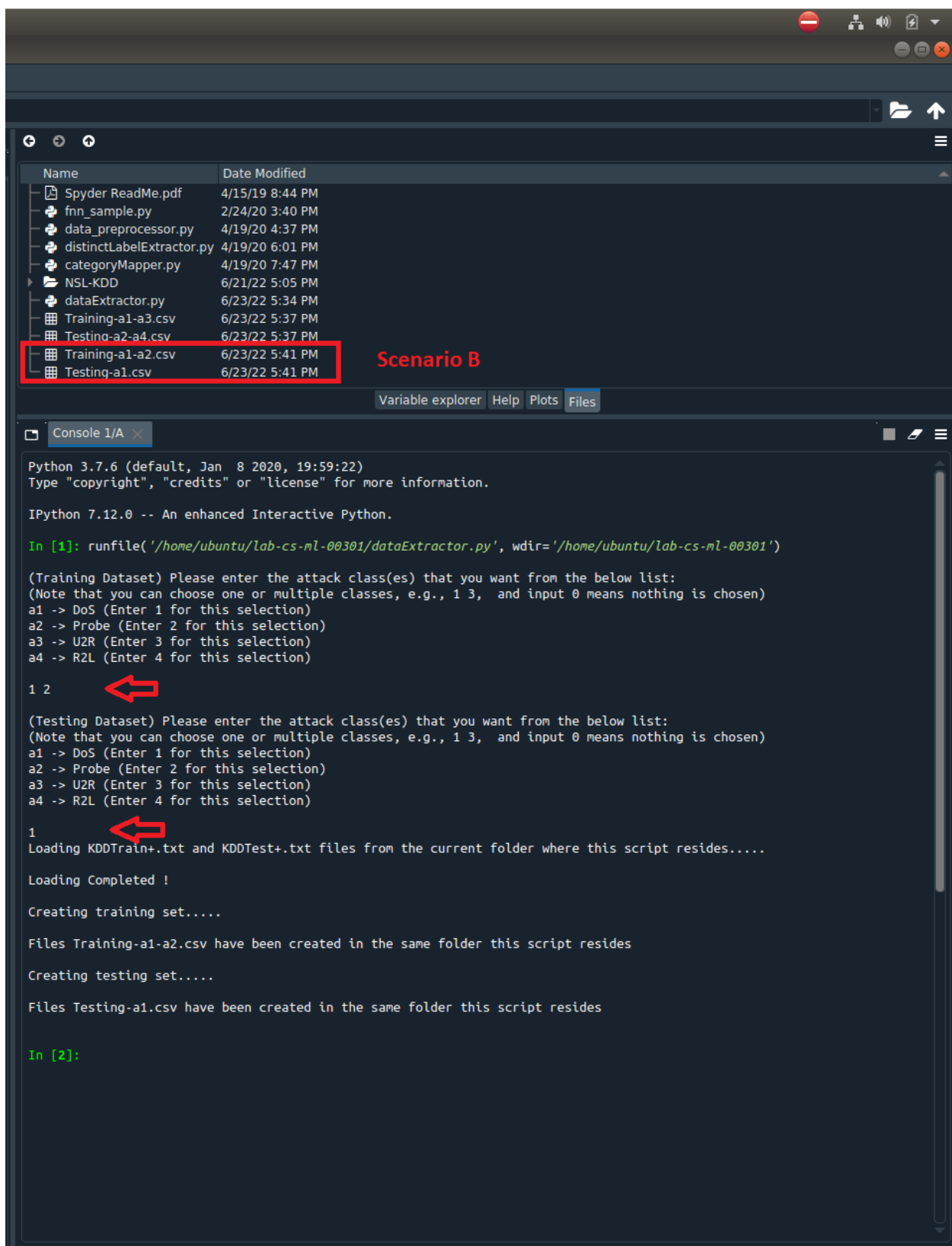


Figure-2 Create scenario B dataset

C. Create Scenario C dataset (Figure-3):

The image shows a file explorer window and a terminal window. The file explorer window displays a list of files and folders. The terminal window shows the execution of a script that creates the Scenario C dataset.

File Explorer:

Name	Date Modified
Spyder ReadMe.pdf	4/15/19 8:44 PM
fnn_sample.py	2/24/20 3:40 PM
data_preprocessor.py	4/19/20 4:37 PM
distinctLabelExtractor.py	4/19/20 6:01 PM
categoryMapper.py	4/19/20 7:47 PM
NSL-KDD	6/21/22 5:05 PM
dataExtractor.py	6/23/22 5:34 PM
Training-a1-a3.csv	6/23/22 5:37 PM
Testing-a2-a4.csv	6/23/22 5:37 PM
Testing-a1.csv	6/23/22 5:41 PM
Training-a1-a2.csv	6/23/22 5:45 PM
Testing-a1-a2-a3.csv	6/23/22 5:45 PM

Terminal Window:

```
Python 3.7.6 (default, Jan 8 2020, 19:59:22)
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: runfile('/home/ubuntu/lab-cs-ml-00301/dataExtractor.py', wdir='/home/ubuntu/lab-cs-ml-00301')

(Training Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3, and input 0 means nothing is chosen)
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1 2

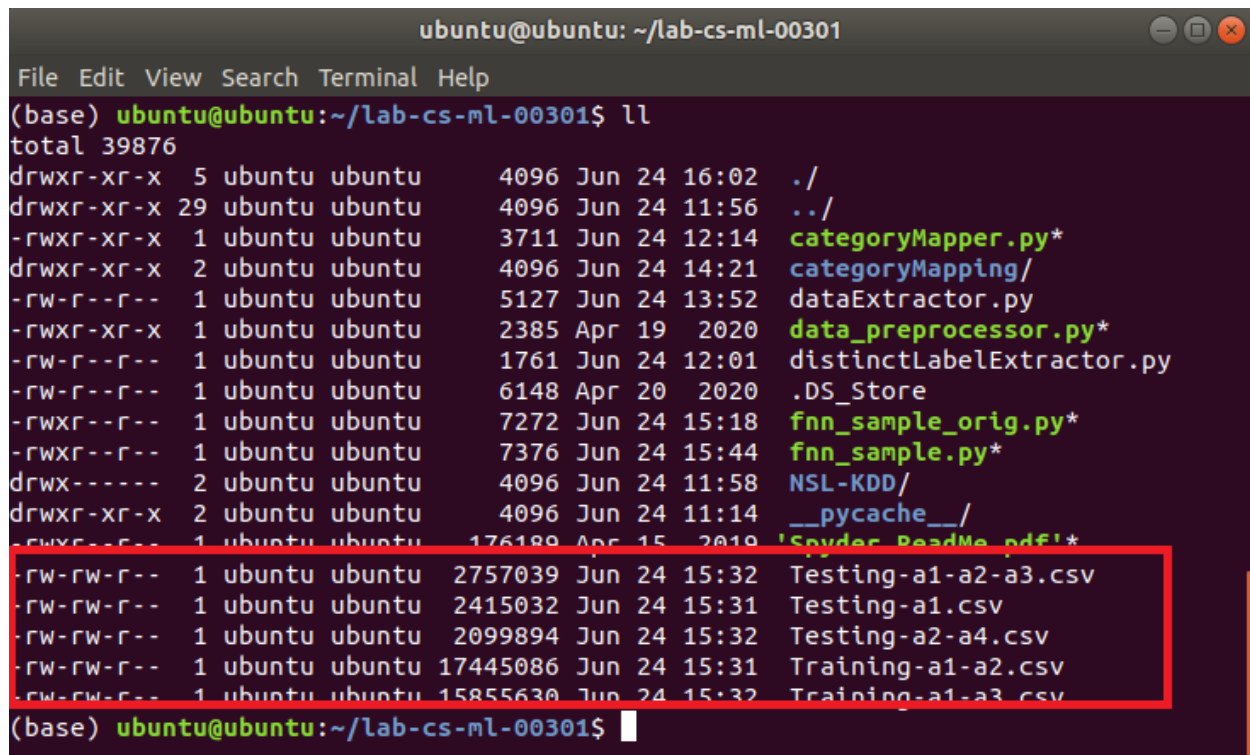
(Testing Dataset) Please enter the attack class(es) that you want from the below list:
(Note that you can choose one or multiple classes, e.g., 1 3, and input 0 means nothing is chosen)
a1 -> DoS (Enter 1 for this selection)
a2 -> Probe (Enter 2 for this selection)
a3 -> U2R (Enter 3 for this selection)
a4 -> R2L (Enter 4 for this selection)

1 2 3
Loading KDDTrain+.txt and KDDTest+.txt files from the current folder where this script resides.....
Loading Completed !
Creating training set.....
Files Training-a1-a2.csv have been created in the same folder this script resides
Creating testing set.....
Files Testing-a1-a2-a3.csv have been created in the same folder this script resides

In [2]:
```

Figure-3 – Create scenario C dataset

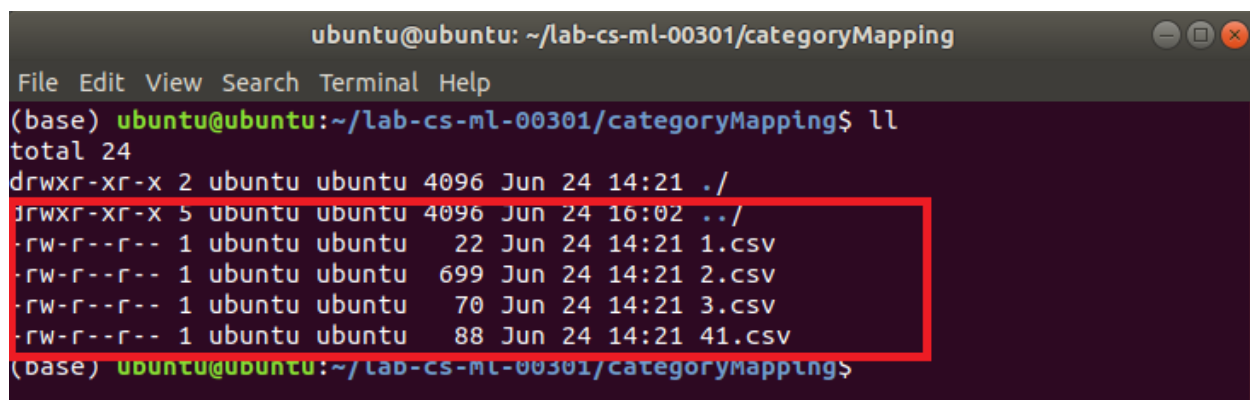
The three dataset files in CSV format are shown in the following screenshot (Figure-4):



```
ubuntu@ubuntu: ~/lab-cs-ml-00301
File Edit View Search Terminal Help
(base) ubuntu@ubuntu:~/lab-cs-ml-00301$ ll
total 39876
drwxr-xr-x  5 ubuntu ubuntu    4096 Jun 24 16:02 ./
drwxr-xr-x 29 ubuntu ubuntu    4096 Jun 24 11:56 ../
-rwxr-xr-x  1 ubuntu ubuntu    3711 Jun 24 12:14 categoryMapper.py*
drwxr-xr-x  2 ubuntu ubuntu    4096 Jun 24 14:21 categoryMapping/
-rw-r--r--  1 ubuntu ubuntu   5127 Jun 24 13:52 dataExtractor.py
-rwxr-xr-x  1 ubuntu ubuntu   2385 Apr 19  2020 data_preprocessor.py*
-rw-r--r--  1 ubuntu ubuntu   1761 Jun 24 12:01 distinctLabelExtractor.py
-rw-r--r--  1 ubuntu ubuntu   6148 Apr 20  2020 .DS_Store
-rwxr--r--  1 ubuntu ubuntu   7272 Jun 24 15:18 fnn_sample_orig.py*
-rwxr--r--  1 ubuntu ubuntu   7376 Jun 24 15:44 fnn_sample.py*
drwx----- 2 ubuntu ubuntu    4096 Jun 24 11:58 NSL-KDD/
drwxr-xr-x  2 ubuntu ubuntu    4096 Jun 24 11:14 __pycache__/
-rwxr--r--  1 ubuntu ubuntu  176188 Apr 15  2019 'Spyder_ReadMe.pdf'*
-rw-rw-r--  1 ubuntu ubuntu  2757039 Jun 24 15:32 Testing-a1-a2-a3.csv
-rw-rw-r--  1 ubuntu ubuntu  2415032 Jun 24 15:31 Testing-a1.csv
-rw-rw-r--  1 ubuntu ubuntu  2099894 Jun 24 15:32 Testing-a2-a4.csv
-rw-rw-r--  1 ubuntu ubuntu  17445086 Jun 24 15:31 Training-a1-a2.csv
-rw-rw-r--  1 ubuntu ubuntu  15855638 Jun 24 15:32 Training-a1-a3.csv
(base) ubuntu@ubuntu:~/lab-cs-ml-00301$
```

Figure-4 Customized dataset's files

I needed to use the data_preprocessor.py module. This module needs labels for category mapping on different levels. In this section, I used the categoryMapper.py program to generate labels based on the data in the datasets. I ran the categoryMapper.py program and entered one of the datasets as the input. This program generated four different CSV files as shown in the following screenshot (Figure-5):



```
ubuntu@ubuntu: ~/lab-cs-ml-00301/categoryMapping
File Edit View Search Terminal Help
(base) ubuntu@ubuntu:~/lab-cs-ml-00301/categoryMapping$ ll
total 24
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 24 14:21 ./
drwxr-xr-x 5 ubuntu ubuntu 4096 Jun 24 16:02 ../
-rw-r--r-- 1 ubuntu ubuntu  22 Jun 24 14:21 1.csv
-rw-r--r-- 1 ubuntu ubuntu  699 Jun 24 14:21 2.csv
-rw-r--r-- 1 ubuntu ubuntu   70 Jun 24 14:21 3.csv
-rw-r--r-- 1 ubuntu ubuntu   88 Jun 24 14:21 41.csv
(base) ubuntu@ubuntu:~/lab-cs-ml-00301/categoryMapping$
```

Figure-5 CSV files for category labels mapping

The content of the category mapping files is shown the following screenshots (Figure-6-10):

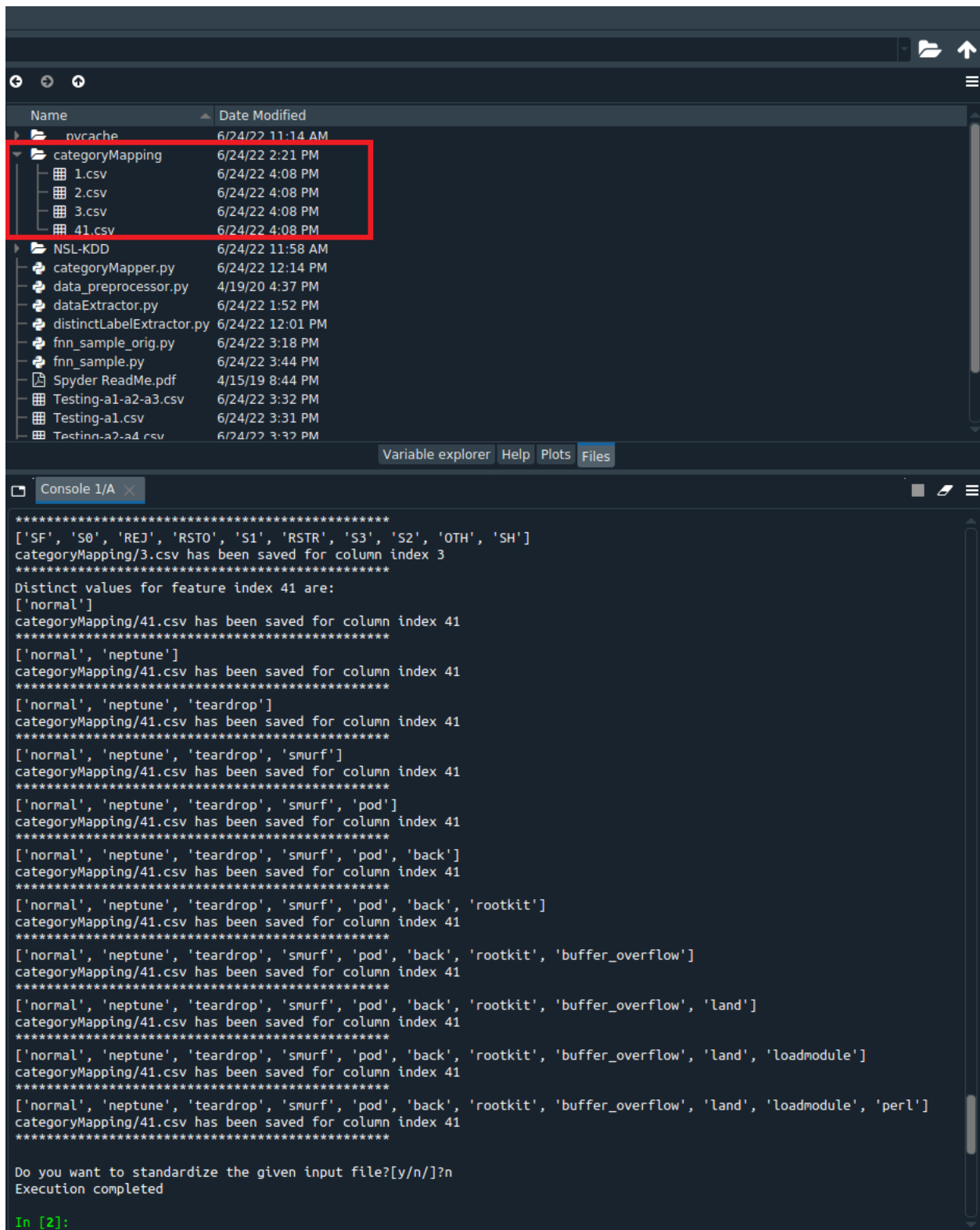


Figure-6 Creating labels in different levels

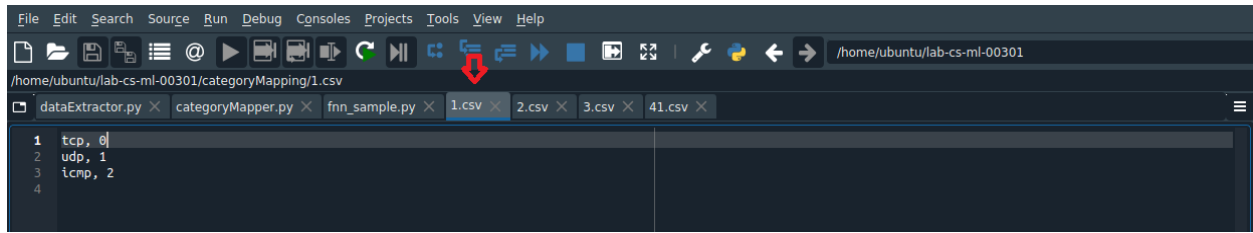


Figure-7 Level 1 category label mapping

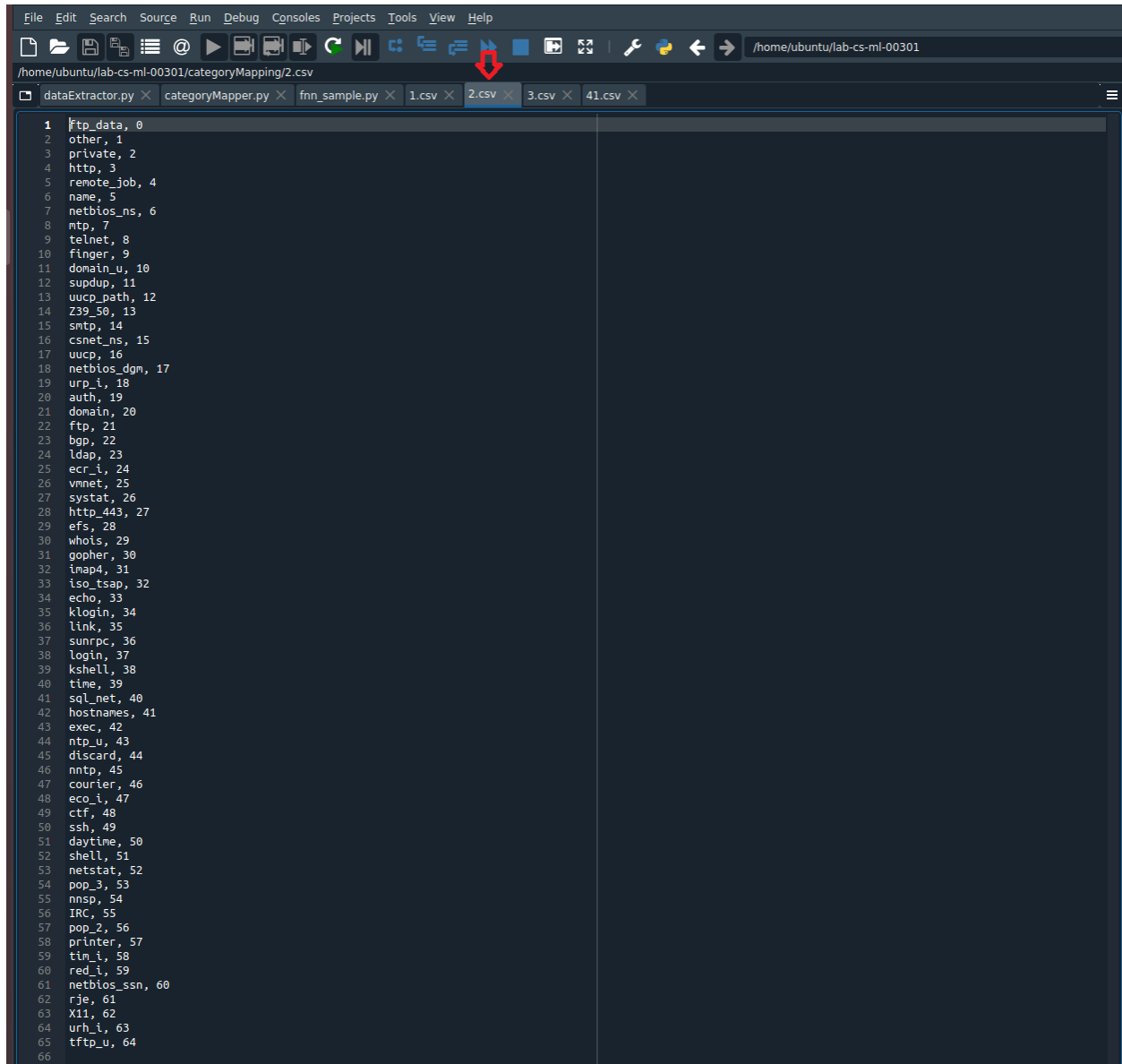


Figure-8 Level 2 category label mapping

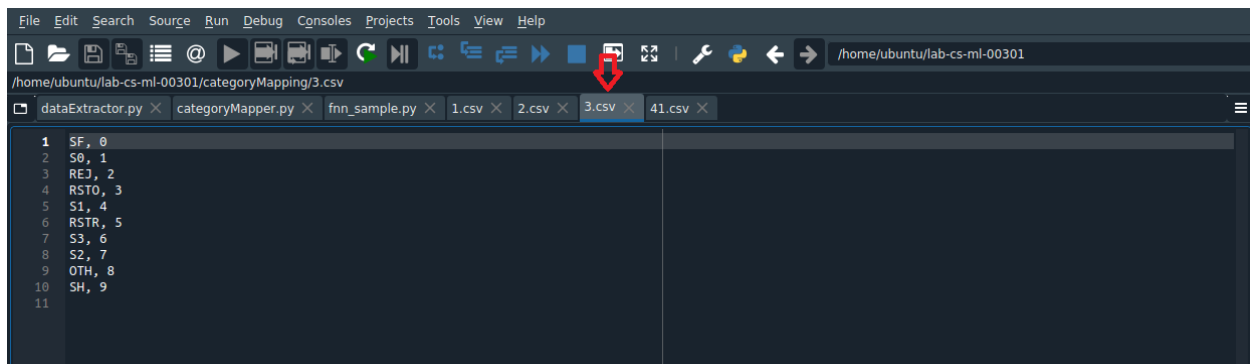


Figure-9 Level 3 category label mapping

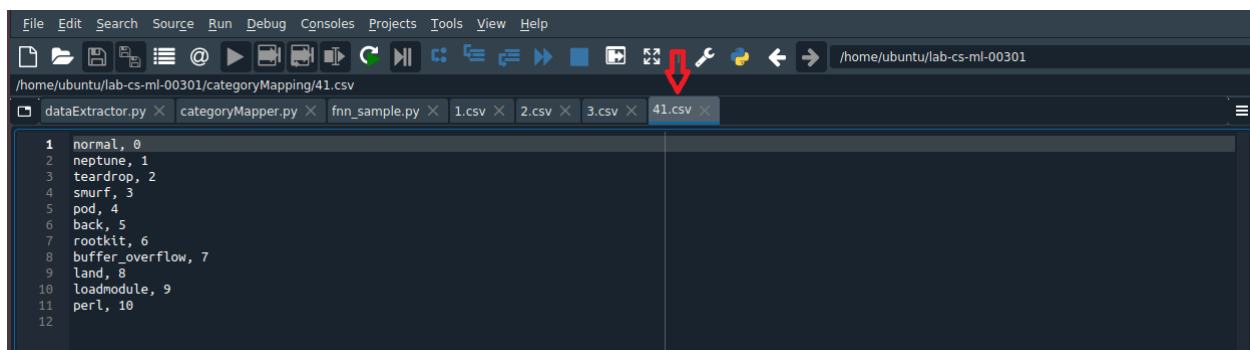


Figure-10 Level 4 category label mapping

Lab assessment

1. 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.

Scenario A:

I initialized the input datasets as follows:

- TrainingData = Training-a1-a3.csv
- TestingData = Testing-a2-a4.csv

I ran fnn_sample.py code and I got the following results (Figure-11):

Confusion Matrix	TN : 8907	FP: 804
	FN: 3147	TP: 2159

Precision: $TP / (TP + FP) \rightarrow 0.7286$

Negative Predictive Value: $TN / (TN + FN) \rightarrow 0.7389$

Recall or Sensitivity: $TP / (TP + FN) \rightarrow 0.4068$

Specificity: $TN / (TN + FP) \rightarrow 0.9172$

The accuracy, calculated using the formula below, shows the accuracy when I applied the FNN model trained dataset to test dataset.

Accuracy: $(TP + TN) / (TP + FP + TN + FN) \rightarrow 0.7369$

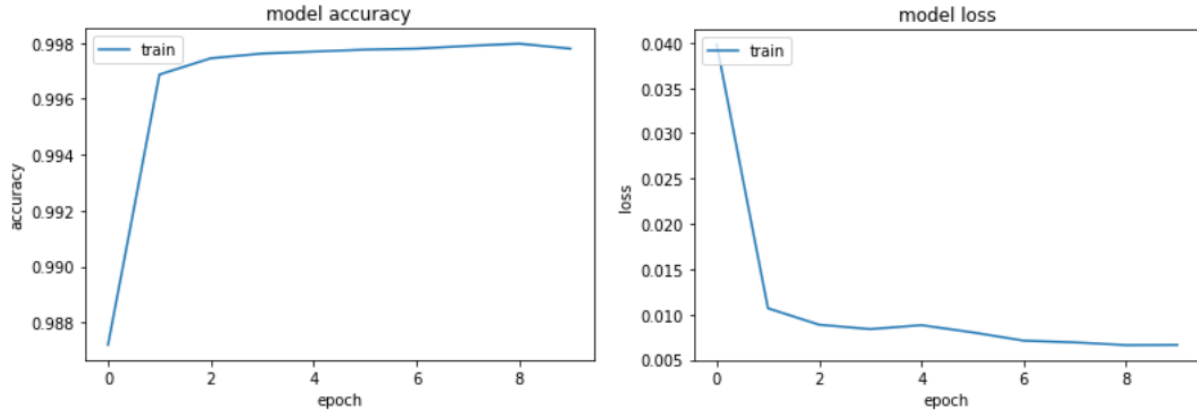
The Accuracy that is shown below, is the accuracy while training the model:

Loss: 0.0058, Accuracy: 0.9981

```
In [4]: runfile('/home/ubuntu/lab-cs-ml-00301/fnn_sample.py', wdir='/home/ubuntu/lab-cs-ml-00301')
Epoch 1/10
WARNING:tensorflow:AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x7f69f84b7440> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x7f69f84b7440> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
11333/11333 [=====] - 44s 4ms/step - loss: 0.0398 - accuracy: 0.9872
Epoch 2/10
11333/11333 [=====] - 44s 4ms/step - loss: 0.0107 - accuracy: 0.9969
Epoch 3/10
11333/11333 [=====] - 45s 4ms/step - loss: 0.0089 - accuracy: 0.9974
Epoch 4/10
11333/11333 [=====] - 44s 4ms/step - loss: 0.0084 - accuracy: 0.9976
Epoch 5/10
11333/11333 [=====] - 45s 4ms/step - loss: 0.0088 - accuracy: 0.9977
Epoch 6/10
11333/11333 [=====] - 46s 4ms/step - loss: 0.0080 - accuracy: 0.9978 - ETA: 7s - loss: 0.0078 - accuracy: 0.9978
Epoch 7/10
11333/11333 [=====] - 45s 4ms/step - loss: 0.0071 - accuracy: 0.9978
Epoch 8/10
11333/11333 [=====] - 45s 4ms/step - loss: 0.0069 - accuracy: 0.9979
Epoch 9/10
11333/11333 [=====] - 45s 4ms/step - loss: 0.0066 - accuracy: 0.9980
Epoch 10/10
11333/11333 [=====] - 44s 4ms/step - loss: 0.0066 - accuracy: 0.9978
3542/3542 [=====] - 12s 3ms/step - loss: 0.0058 - accuracy: 0.9981
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0058 Accuracy [0,1]: 0.9981
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8907 804]
 [3147 2159]]
Plot the accuracy
```

Scenario A

Figure-11 fnn_sample.py – Scenario A



Explanation of Scenario A:

In this scenario, I used the “DoS” and the “U2R” attacks as training datasets and the “Probing” and the “R2L” attacks as testing datasets. Since there is no overlap between the training dataset and testing dataset, I expected this scenario to have the worst performance compared to the other two scenarios.

The accuracy that I computed from the values that I got from confusion matrix confirmed my expectation.

Scenario B:

I initialized the input datasets as follows:

- TrainingData = Training-a1-a2.csv
- TestingData = Testing-a1.csv

I ran `fnn_sample.py` code and I got the following results (Figure-12):

Confusion Matrix	TN : 8856	FP: 855
	FN: 1137	TP: 6323

Precision: $TP / (TP + FP) \rightarrow \mathbf{0.8808}$

Negative Predictive Value: $TN / (TN + FN) \rightarrow \mathbf{0.8862}$

Recall or Sensitivity: $TP / (TP + FN) \rightarrow \mathbf{0.8537}$

Specificity: $TN / (TN + FP) \rightarrow \mathbf{0.9119}$

Accuracy: $(TP + TN) / (TP + FP + TN + FN) \rightarrow \mathbf{0.8839}$

Loss and accuracy values at the end of executing training model:

Loss: 0.0447, Accuracy: 0.9907

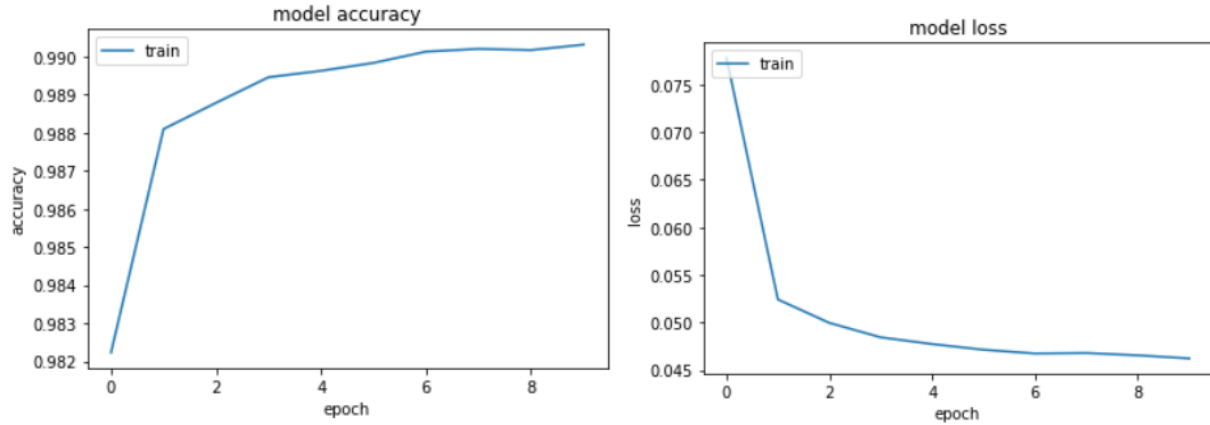
```

To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x7f6a4c8fab90>
and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
12493/12493 [=====] - 55s 4ms/step - loss: 0.0778 - accuracy: 0.9822
Epoch 2/10
12493/12493 [=====] - 54s 4ms/step - loss: 0.0524 - accuracy: 0.9881
Epoch 3/10
12493/12493 [=====] - 78s 6ms/step - loss: 0.0500 - accuracy: 0.9888
Epoch 4/10
12493/12493 [=====] - 131s 10ms/step - loss: 0.0484 - accuracy: 0.9895
Epoch 5/10
12493/12493 [=====] - 55s 4ms/step - loss: 0.0477 - accuracy: 0.9896
Epoch 6/10
12493/12493 [=====] - 57s 5ms/step - loss: 0.0471 - accuracy: 0.9898
Epoch 7/10
12493/12493 [=====] - 55s 4ms/step - loss: 0.0467 - accuracy: 0.9901
Epoch 8/10
12493/12493 [=====] - 54s 4ms/step - loss: 0.0468 - accuracy: 0.9902
Epoch 9/10
12493/12493 [=====] - 54s 4ms/step - loss: 0.0466 - accuracy: 0.9902
Epoch 10/10
12493/12493 [=====] - 53s 4ms/step - loss: 0.0462 - accuracy: 0.9903
WARNING:tensorflow:AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at
0x7f6a4d09e3b0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x7f6a4d09e3b0> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
3904/3904 [=====] - 17s 4ms/step - loss: 0.0447 - accuracy: 0.9907
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0447 Accuracy [0,1]: 0.9907
WARNING:tensorflow:AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at
0x7f6a4ce6a320> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at
0x7f6a4ce6a320> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8856 855]
 [1137 6323]]
Plot the accuracy
Plot the loss

```

Scenario B

Figure-12 fnn_sample.py – Scenario B



Explanation of scenario B:

This scenario uses “DoS” and “Probe” attacks as the training dataset, and “DoS” attack as the testing dataset. In this scenario, we have overlap between the training dataset and the testing dataset. In other words, some attacks in the training dataset have corresponding attack data in the testing dataset.

I expected this scenario to give better results than scenario A. The results than I got from the confusion matrix and the value of accuracy that I computed confirmed my expectation.

Scenario C:

I initialized the input datasets as follows:

- TrainingData = Training-a1-a2.csv
- TestingData = Testing-a1-a2-a3.csv

I ran fnn_sample.py code and I got the following results (Figure-13):

Confusion Matrix	TN : 8880	FP: 831
	FN: 1715	TP: 8233

Precision: $TP / (TP + FP) \rightarrow$ **0.9083**

Negative Predictive Value: $TN / (TN + FN) \rightarrow$ **0.8381**

Recall or Sensitivity: $TP / (TP + FN) \rightarrow$ **0.8276**

Specificity: $TN / (TN + FP) \rightarrow$ **0.9144**

Accuracy: $(TP + TN) / (TP + FP + TN + FN) \rightarrow$ **0.8705**

Loss and accuracy values at the end of executing training model:

Loss: 0.0459, Accuracy: 0.9903

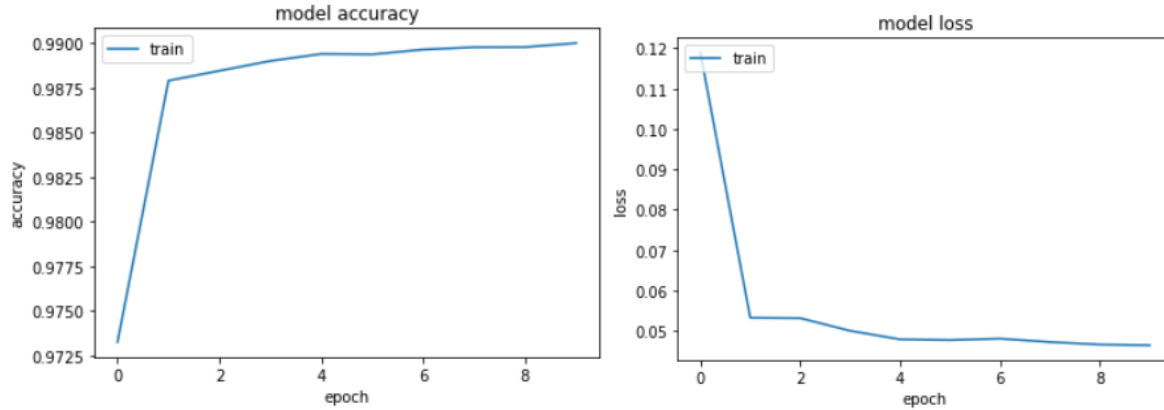
```

and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
12493/12493 [=====] - 55s 4ms/step - loss: 0.1190 - accuracy: 0.9732
Epoch 2/10
12493/12493 [=====] - 62s 5ms/step - loss: 0.0532 - accuracy: 0.9879
Epoch 3/10
12493/12493 [=====] - 62s 5ms/step - loss: 0.0531 - accuracy: 0.9884
Epoch 4/10
12493/12493 [=====] - 61s 5ms/step - loss: 0.0500 - accuracy: 0.9890
Epoch 5/10
12493/12493 [=====] - 60s 5ms/step - loss: 0.0479 - accuracy: 0.9894
Epoch 6/10
12493/12493 [=====] - 60s 5ms/step - loss: 0.0477 - accuracy: 0.9894
Epoch 7/10
12493/12493 [=====] - 85s 7ms/step - loss: 0.0481 - accuracy: 0.9896ETA: 53s - loss: 0.0478 -
accuracy: 0.989510721/12493 [=====] - ETA: 10s - loss: 0.0487 - accuracy: 0.9895
Epoch 8/10
12493/12493 [=====] - 166s 13ms/step - loss: 0.0472 - accuracy: 0.9898
Epoch 9/10
12493/12493 [=====] - 69s 6ms/step - loss: 0.0466 - accuracy: 0.9898
Epoch 10/10
12493/12493 [=====] - 58s 5ms/step - loss: 0.0464 - accuracy: 0.9900
WARNING:tensorflow:AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at
0x7f6a4cb103b0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x7f6a4cb103b0> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
3904/3904 [=====] - 14s 4ms/step - loss: 0.0459 - accuracy: 0.9903
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0459 Accuracy [0,1]: 0.9903
WARNING:tensorflow:AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at
0x7f6a4cb35dd0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at
0x7f6a4cb35dd0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[8880 831]
 [1715 8233]]
Plot the accuracy
Plot the loss

```

Scenario C

Figure-13 fnn_sample.py – Scenario C



Explanation of the scenario C:

This scenario uses the same training dataset that I used in scenario B, but I used the bigger testing dataset. In this case, I had a complete overlap between the training dataset and testing dataset, and I expected this scenario to have better a result in terms of accuracy than scenario B, but there was an attack in the testing dataset (U2R) without a corresponding attack in the training dataset. In this case, our model shows a high variance in training.

“Variance, in the context of Machine Learning, is a type of error that occurs due to a model’s sensitivity to a small fluctuation in the training set¹.”

One of the major characteristics of a high variance model is its potential to over-fit, and over-fitting can reduce the accuracy of the prediction ML model.

Scenario D:

In this section, I ran another scenario based on the datasets that I created. I wanted to confirm the result that I got from scenario B. In this scenario, I used a different training dataset and the same testing dataset with the same overlap between them as in scenario B. I initialized the input datasets as follows:

- TrainingData = Training-a1-a3.csv
- TestingData = Testing-a1.csv

I ran fnn_sample.py code and I got the following results (Figure-14):

Confusion Matrix	TN : 8991	FP: 720
	FN: 2101	TP: 7847

Precision: $TP / (TP + FP) \rightarrow 0.9159$

Negative Predictive Value: $TN / (TN + FN) \rightarrow 0.8105$

Recall or Sensitivity: $TP / (TP + FN) \rightarrow 0.7880$

Specificity: $TN / (TN + FP) \rightarrow 0.9258$

¹ <https://datascience.stackexchange.com/questions/37345/what-is-the-meaning-of-term-variance-in-machine-learning-model>

Accuracy: $(TP + TN) / (TP + FP + TN + FN) \rightarrow 0.8565$

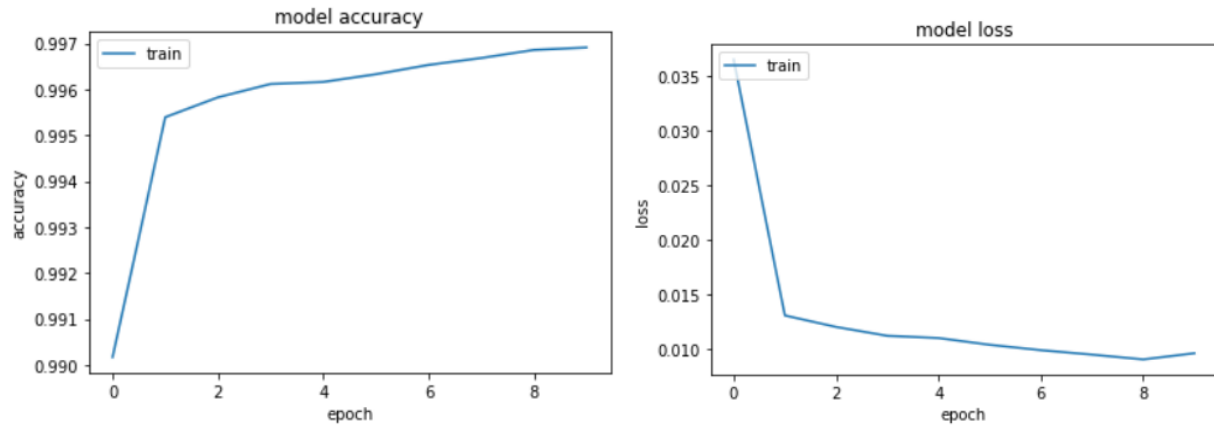
Loss and accuracy values at the end of executing training model:

Loss: 0.0086, Accuracy: 0.9970

```
Please report this to the tensorflow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
11333/11333 [=====] - 56s 5ms/step - loss: 0.0365 - accuracy: 0.9902
Epoch 2/10
11333/11333 [=====] - 52s 5ms/step - loss: 0.0131 - accuracy: 0.9954
Epoch 3/10
11333/11333 [=====] - 52s 5ms/step - loss: 0.0121 - accuracy: 0.9958
Epoch 4/10
11333/11333 [=====] - 51s 5ms/step - loss: 0.0113 - accuracy: 0.9961
Epoch 5/10
11333/11333 [=====] - 52s 5ms/step - loss: 0.0111 - accuracy: 0.9962
Epoch 6/10
11333/11333 [=====] - 47s 4ms/step - loss: 0.0104 - accuracy: 0.9963
Epoch 7/10
11333/11333 [=====] - 46s 4ms/step - loss: 0.0099 - accuracy: 0.9965
Epoch 8/10
11333/11333 [=====] - 46s 4ms/step - loss: 0.0095 - accuracy: 0.9967
Epoch 9/10
11333/11333 [=====] - 46s 4ms/step - loss: 0.0091 - accuracy: 0.9969
Epoch 10/10
11333/11333 [=====] - 81s 7ms/step - loss: 0.0097 - accuracy: 0.9969
WARNING:tensorflow:AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at
0x7f642855a3b0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x7f642855a3b0> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
3542/3542 [=====] - 34s 10ms/step - loss: 0.0086 - accuracy: 0.9970
Print the loss and the accuracy of the model on the dataset
Loss [0,1]: 0.0086 Accuracy [0,1]: 0.9970
WARNING:tensorflow:AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at
0x7f6428267dd0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at
0x7f6428267dd0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause:
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Print the Confusion Matrix:
[ TN, FP ]
[ FN, TP ]=
[[9242  469]
 [1660 5800]]
Plot the accuracy
Plot the loss
```

Scenario D

Figure-14 fnn_sample.py – Scenario D



Explanation of scenario D:

In this scenario, I used the “DoS” and the “U2R” attacks as training datasets and “DoS” attack as the testing dataset. In this scenario, there was overlap between the training dataset and testing dataset. In other words, some attacks in the training dataset had corresponding attack data in the testing dataset.

I expected this scenario to give same results as scenario B, but the results that I got from the confusion matrix and the value of accuracy that I computed show that the accuracy was lower than the accuracy that I computed in scenario B.

When I compared the “Probe” and “U2R” attacks in the training dataset, I realized that the number of records related to the “Probe” attack is much higher than the number of records related to the “U2R” attack in the training dataset, so the larger training dataset (Scenario B) had a positive impact on the learning model in terms of accuracy.

2. Submit the updated `fnn_simple.py` and provide sufficient comments to illustrate your updated codes.

In this section, I changed the `fnn_sample.py` file. I added two variables, `TestingData` and `TrainingData`. I used these variables to get two different input files as testing and training datasets. I used the `data_preprocessor.py` module that was provided to us, and I used it to preprocess the datasets before I used them as input in the `fnn_sample.py` program. I added three lines of code to import the `data_preprocessor` module and used the `get_processed_data` function that was defined inside this module to preprocess the input datasets and put the results in the `x_train`, `x_test`, `y_test`, and `y_train` (Figure-15).


```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Apr 15 19:43:04 2019
4
5 Updated on Wed Jan 29 10:18:09 2020
6
7 @author: created by Sowmya Myneni and updated by Dijiang Huang
8 """
9
10 #####
11 # Part 1 - Data Pre-Processing
12 #####
13
14 # To load a dataset file in Python, you can use Pandas. Import pandas using the line below
15 # import pandas as pd
16 # Import numpy to perform operations on the dataset
17 # import numpy as np
18
19 # Variable Setup
20 # Available datasets: KDDTrain+.txt, KDDTest+.txt, etc. More read Data Set Introduction.html within the NSL-KDD dataset folder
21 # Type the training dataset file name in ''
22 TrainingDataPath='./
23 TrainingData='Training-a1-a3'
24 TestingDataPath='./
25 TestingData='Testing-a2-a4'
26
27 # Batch Size
28 BatchSize=10
29 # Epoche Size
30 NumEpoch=10
31
32 # Import dataset.
33 # Dataset is given in TrainingData variable You can replace it with the file
34 # path such as "C:\Users\...\dataset.csv".
35 # The file can be a .txt as well.
36 # If the dataset file has header, then keep header=0 otherwise use header=None
37 # reference: https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
38
39 import data_preprocessor as dp
40 X_train, y_train = dp.get_processed_data(TrainingData+'.csv', './categoryMapping/', classType = 'binary')
41 X_test, y_test = dp.get_processed_data(TestingData+'.csv', './categoryMapping/', classType= 'binary')
42
43
44 #####
45 # Part 2: Building FNN
46 #####
```

Figure-15 The fnn_sample.py program

I used two different datasets as testing and training datasets, and I used the `get_processed_data` function to encode the datasets. Thus, I converted lines 43 through 93 to comment, because this part of the `fnn_sample.py` program is responsible for encoding the dataset and splitting it into testing and training datasets, and I didn't need this part anymore. I left the remaining part of the `fnn_sample.py` code unchanged (Figure-16):

```
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/ubuntu/lab-cs-ml-00301/fnn_sample.py
dataExtractor.py categoryMapper.py fnn_sample.py*
34 # path such as "C:\Users\...\dataset.csv".
35 # The file can be a .txt as well.
36 # If the dataset file has header, then keep header=0 otherwise use header=None
37 # reference: https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
38
39 import data_preprocessor as dp
40 X_train, y_train = dp.get_processed_data(TrainingData+'.csv', './categoryMapping/', classType = 'binary')
41 X_test, y_test = dp.get_processed_data(TestingData+'.csv', './categoryMapping/', classType= 'binary')
42
43 """
44 dataset = pd.read_csv(TrainingDataPath+TrainingData, header=None)
45 X = dataset.iloc[:, 0:-2].values
46 label_column = dataset.iloc[:, -2].values
47 y = []
48 for i in range(len(label_column)):
49     if label_column[i] == 'normal':
50         y.append(0)
51     else:
52         y.append(1)
53
54 # Convert list to array
55 y = np.array(y)
56
57 # Encoding categorical data (convert letters/words in numbers)
58 # Reference: https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621
59 # The following code work without warning in Python 3.6 or older. Newer versions suggest to use ColumnTransformer
60
61 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
62 le = LabelEncoder()
63 X[:, 1] = le.fit_transform(X[:, 1])
64 X[:, 2] = le.fit_transform(X[:, 2])
65 X[:, 3] = le.fit_transform(X[:, 3])
66 onehotencoder = OneHotEncoder(categorical_features = [1, 2, 3])
67 X = onehotencoder.fit_transform(X).toarray()
68
69 # The following code work Python 3.7 or newer
70 from sklearn.preprocessing import OneHotEncoder
71 from sklearn.compose import ColumnTransformer
72 ct = ColumnTransformer(
73     [['one_hot_encoder', OneHotEncoder(), [1,2,3]]], # The column numbers to be transformed ([1, 2, 3] represents three columns to be transferred)
74     remainder='passthrough' # Leave the rest of the columns untouched
75 )
76 X = np.array(ct.fit_transform(X), dtype=np.float)
77
78 # Splitting the dataset into the Training set and Test set (75% of data are used for training)
79 # reference: https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html
80 from sklearn.model_selection import train_test_split
81 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
82
83
84 # Perform feature scaling. For ANN you can use StandardScaler, for RNNs recommended is
85 # MinMaxScaler.
86 # reference: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
87 # https://scikit-learn.org/stable/modules/preprocessing.html
88 from sklearn.preprocessing import StandardScaler
89 sc = StandardScaler()
90 X_train = sc.fit_transform(X_train) # Scaling to the range [0,1]
91 X_test = sc.fit_transform(X_test)
92 """
93
94 #####
95 # Part 2: Building FNN
96 #####
97
```

Figure-16 The fnn_sample.py program

3. Submit the report to provide in-depth analysis to address the questions given in Task 3.

A. Which scenario produce the most accurate testing results? Observe your model's prediction capability with respect to the change in the attack classes on which it was trained. Then, observe the accuracy of the prediction when trained on these subset of attack types. Observe if the model predicts better when it was trained on a specific subset of attack classes.

Scenario B resulted in the best accuracy (**0.8839**). I created another scenario, Scenario D, with a different training dataset, but I used the same testing dataset that I used in the Scenario B. I found that scenario D resulted in lower accuracy (**0.8565**) when I compared the training datasets between these scenarios. I find out that Probe attack (9.11%) has larger data than U2R attack (0.04%), as happens in the real world on the Internet. Thus, I concluded that a larger training dataset can help to improve the accuracy in my FNN training model.

Attack scenario	Attacks as Training Dataset	Attacks as Testing Dataset	Accuracy
A	DoS, U2R	Probe, R2L	0.7369
B	DoS, Probe	DoS	0.8839
C	DoS, Probe	DoS, Probe, U2R	0.8705
D	DoS, U2R	DoS	0.8565

B. What is the average accuracy that it detects the new class of attacks (in the testing dataset in SA and SC) to be as normal or attack? (Hint: With the model set to perform binary classification, the predicted values of the model can be:

- $0 \rightarrow \text{Normal}$,
- $1 \rightarrow \text{Attack}$.

This prediction is associated with the accuracy of the prediction. Note down the accuracy of the prediction, the prediction is normal or attack, for the above unknown attacks A2 and A4 in SA or A3 in SC).

The average of accuracy between scenario A, B, and C is **0.8319** and the average of accuracy between scenario A and C is **0.8037**. This shows that when I want to test my model (scenario A) with attacks that my model has not been trained for (Probe, R2L), it results in worst accuracy, and on other side, when I used a test dataset with trained attacks (scenario C) in addition to the attack that my model has not been trained for (U2R), then it results in lower accuracy.

Attack scenario	Training Accuracy	Predict Accuracy	True Positive Rate
A	0.9981	0.7369	0.4068
B	0.9907	0.8839	0.8537
C	0.9903	0.8705	0.8276

C. What is the difference between attacks in untrained subset and attacks from the trained dataset? (Hint: For example in SA, how are A1 and A3 different from A2 and A4. Do A1 and A2 belong to same attack category or have similar differences with respect to the normal data? Present your observations can be made by looking at the data for these attacks).

As I demonstrated in the table above, when a scenario use attacks in the testing dataset that the model has not been trained for, then it will perform the worst in terms of accuracy. This result is exactly what I had expected.

D. Does the prediction accuracy relate to the attacks being similar? If so, what is the similarity? (Hint: If the prediction accuracy was found better, look at the data and the attack types for any similarities that would have made the prediction better).

- **DoS** is an attack that tries to shut down traffic flow to and from the target system.
- **Probe** or surveillance is an attack that tries to get information from a network.
- **U2R** is an attack that starts off with a normal user account and tries to gain access to the system or network, as a super-user (root).
- **R2L** is an attack that tries to gain local access to a remote machine.

R2L is conceptually similar to the user-to-root (U2R) attack, but it is more modest in its ultimate ambition. Such an attack is transacted when an attacker sends packets (mostly using TCP or HTTP protocols to send exploit or

crafted illegal packets) to the target host that are intended to disclose vulnerabilities that would enable the attacker to exploit a local user's privileges.

Table CS-ML-00101.2
Statistics of NSL-KDD Dataset Attacks

Dataset	Number of Records					
	Total	Normal	DoS	Prob	U2R	R2L
KDDTrain+20%	25192	13449 (53%)	9234 (37%)	2289 (9.16%)	11 (0.04%)	209 (0.8%)
KDDTrain+	125973	67343 (53%)	45927 (37%)	11656 (9.11%)	52 (0.04%)	995 (0.85%)
KDDTest+	22544	9711 (43%)	7438 (33%)	2421 (11%)	200 (0.9%)	2654 (12.1%)

Another issue is the fact that “U2R” and “R2L” (Group A) attacks had much lower of records than “DoS” and “Probe” (Group B) attacks, which reflects negatively on the training model.

V. APPENDIX A: FILES FOR THE LAB

fnn_sample.py	End of this report.
Lab 4 – Video	https://mehrantaibakhsh.com/cse548/CS-ML-00301.mp4

VI. REFERENCES

Data Preprocessing:

- <https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d>

FNN ML Model:

- <https://link.springer.com/article/10.1007/s42979-022-01031-1>

ANN:

- https://www.researchgate.net/publication/309698316_Performance_analysis_of_NSL-KDD_dataset_using_ANN

fnn_sample.py – source code:

```
# To load a dataset file in Python, you can use Pandas. Import pandas using the line below
# import pandas as pd
# Import numpy to perform operations on the dataset
# import numpy as np

# Variable Setup
# Available datasets: KDDTrain+.txt, KDDTest+.txt, etc. More read Data Set Introduction.html within the NSL-KDD dataset folder
# Type the training dataset file name in ''
TrainingDataPath='./'
TrainingData='Training-al-a3'
TestingDataPath='./'
TestingData='Testing-al'
# Batch Size
BatchSize=10
# Epoche Size
NumEpoch=10

# Import dataset.
# Dataset is given in TrainingData variable You can replace it with the file
# path such as "C:\Users\...\dataset.csv".
# The file can be a .txt as well.
# If the dataset file has header, then keep header=0 otherwise use header=None
# reference: https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/

import data_preprocessor as dp
X_train, y_train = dp.get_processed_data(TrainingData+'.csv', './categoryMapping/', classType = 'binary')
X_test, y_test = dp.get_processed_data(TestingData+'.csv', './categoryMapping/', classType= 'binary')

'''
dataset = pd.read_csv(TrainingDataPath+TrainingData, header=None)
X = dataset.iloc[:, 0:-2].values
label_column = dataset.iloc[:, -2].values
y = []
for i in range(len(label_column)):
    if label_column[i] == 'normal':
        y.append(0)
    else:
        y.append(1)

# Convert list to array
y = np.array(y)

# Encoding categorical data (convert letters/words in numbers)
# Reference: https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621
# The following code work without warning in Python 3.6 or older. Newer versions suggest to use ColumnTransformer

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
X[:, 1] = le.fit_transform(X[:, 1])
X[:, 2] = le.fit_transform(X[:, 2])
X[:, 3] = le.fit_transform(X[:, 3])
onehotencoder = OneHotEncoder(categorical_features = [1, 2, 3])
X = onehotencoder.fit_transform(X).toarray()

# The following code work Python 3.7 or newer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer(
    [('one_hot_encoder', OneHotEncoder(), [1,2,3])],    # The column numbers to be transformed ([1, 2, 3] represents three columns
    remainder='passthrough'                          # Leave the rest of the columns untouched
)
X = np.array(ct.fit_transform(X), dtype=np.float)

# Splitting the dataset into the Training set and Test set (75% of data are used for training)
# reference: https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Perform feature scaling. For ANN you can use StandardScaler, for RNNs recommended is
```

```

# MinMaxScaler.
# referece: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# https://scikit-learn.org/stable/modules/preprocessing.html
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train) # Scaling to the range [0,1]
X_test = sc.fit_transform(X_test)
'''

#####
# Part 2: Building FNN
#####

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
# Reference: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
classifier = Sequential()

# Adding the input layer and the first hidden layer, 6 nodes, input_dim specifies the number of variables
# rectified linear unit activation function relu, reference: https://machinelearningmastery.com/rectified-linear-activation-function/
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = len(X_train[0])))

# Adding the second hidden layer, 6 nodes
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer, 1 node,
# sigmoid on the output layer is to ensure the network output is between 0 and 1
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN,
# Gradient descent algorithm "adam", Reference: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
# This loss is for a binary classification problems and is defined in Keras as "binary_crossentropy", Reference: https://machinelearningmastery.com/binary-crossentropy/
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
# Train the model so that it learns a good (or good enough) mapping of rows of input data to the output classification.
# add verbose=0 to turn off the progress report during the training
# To run the whole training dataset as one Batch, assign batch size: BatchSize=X_train.shape[0]
classifierHistory = classifier.fit(X_train, y_train, batch_size = BatchSize, epochs = NumEpoch)

# evaluate the keras model for the provided model and dataset
loss, accuracy = classifier.evaluate(X_train, y_train)
print('Print the loss and the accuracy of the model on the dataset')
print('Loss [0,1]: %.4f' % (loss), 'Accuracy [0,1]: %.4f' % (accuracy))

#####
# Part 3 - Making predictions and evaluating the model
#####

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.9) # y_pred is 0 if less than 0.9 or equal to 0.9, y_pred is 1 if it is greater than 0.9
# summarize the first 5 cases
#for i in range(5):
#    print('%s => %d (expected %d)' % (X_test[i].tolist(), y_pred[i], y_test[i]))

# Making the Confusion Matrix
# [TN, FP ]
# [FN, TP ]
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('Print the Confusion Matrix:')
print('[' TN, FP ]')
print('[' FN, TP ]=')
print(cm)

```

```
#####
# Part 4 - Visualizing
#####

# Import matplotlib lib libraries for plotting the figures.
import matplotlib.pyplot as plt

# You can plot the accuracy
print('Plot the accuracy')
# Keras 2.2.4 recognizes 'acc' and 2.3.1 recognizes 'accuracy'
# use the command python -c 'import keras; print(keras.__version__)' on MAC or Linux to check Keras' version
plt.plot(classifierHistory.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.savefig('accuracy_sample.png')
plt.show()

# You can plot history for loss
print('Plot the loss')
plt.plot(classifierHistory.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.savefig('loss_sample.png')
plt.show()
```