CALIFORNIA STATE UNIVERSITY, SACRAMENTO (CSUS)

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

CSC 215-01: ARTIFICIAL INTELLIGENCE

FALL 2020

Mini-Project 1 - Modern Low Footprint Cyber Attack Detection

Due at 4.00 pm, Wednesday, September 30, 2020

Submitted by:

Harshitha Onkar (Sac State Id: 220262706)
Gargi Prabhugaonkar (Sac State Id: 301111525)

Table of Contents

Problem Statement

To build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections [1].

We intend to build the models for binary classification and compare the metrics for attacks and normal connections.

We plan to implement the following additional features for enhancement and learning purpose:
1) To model this intrusion detection as a multi-class classifier to detect the type of each intrusion.
2) To create a more balanced dataset using down sampling and oversampling.
3) To perform feature importance analysis to select top-10 most important features and train model on the selected features to compare performance.
4) To model using additional dataset [2].

Methodology

Dataset

The dataset used in this project [1] is a subset of UNSW-NB 15 dataset [3]. The dataset in [1] has 45 features with class label of 0 for normal record and 1 for attack record.

The details of the dataset are as follows

Description	Records	Features
Unprocessed train set	175341	45
Unprocessed test set	82332	45
Preprocessed train set	81173	44
Preprocessed test set	35179	44
Post encoding train set	81173	68
Post encoding test set	35179	67
Final preprocessed train set with common features	81173	66
Final preprocessed test set with common features	35179	66

Table 1: Dataset details

Data preprocessing

The data from test set and train set available at [1] was preprocessed by using the steps below:

1) Read .csv data into data frames by marking any ‘NA’, ’?’, ‘-‘, as NaN.
2) Drop all values marked as NaN.
3) Drop column ‘id’.
4) Drop null values.
5) Identify numeric and categorical columns
6) Perform numeric and categorical encoding on numeric and categorical columns respectively.
7) Identify common features between train set and test set
8) Drop all columns not common to train set and test set
9) Save preprocessed train set and test set for prediction using models.

Models

The preprocessed train and test sets are used to fit the following models for prediction to detect bad connections (intrusions):

1) Nearest Neighbor
2) Support Vector Machine
3) Logistic Regression
4) Fully Connected Neural Networks

Model Training and Evaluation

The following steps are generally followed for model training and evaluation:

1) Import the class for the model
2) Instantiate model
3) Fit the model with training data
4) Predict the response with test data

Experimental Results and Analysis

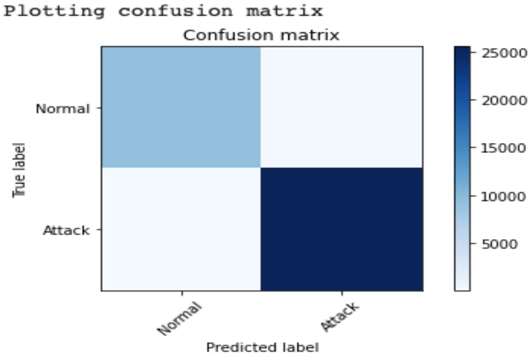
The preprocessed train set is used for fitting the models and the preprocessed test set is used for prediction to detect intrusions.

Model Performance Evaluation

The prediction results for each model are as follows:

1) **Nearest Neighbor (K=3)**

```
Accuracy score: 0.9930071917905569
Precision score: 0.9930511467500288
Recall score: 0.9930071917905569
F1 score: 0.9929824648509145


[[ 9395   230]
 [   16 25538]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9625
           1       0.99      1.00      1.00     25554

    accuracy                           0.99     35179
   macro avg       0.99      0.99      0.99     35179
weighted avg       0.99      0.99      0.99     35179
```

Fig 1.a: Confusion Matrix and performance metrics for Nearest Neighbor model

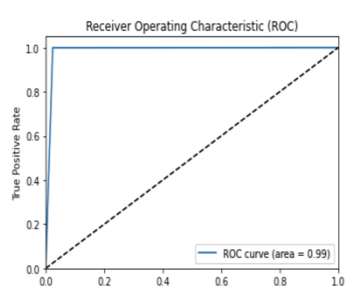
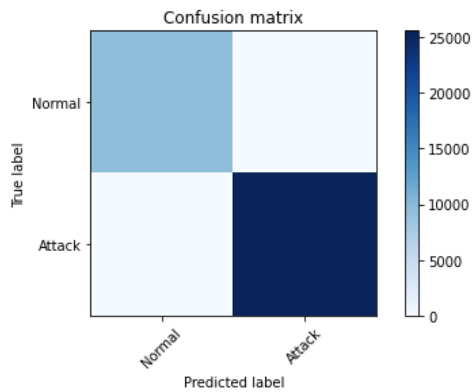


Fig 1.b: ROC Curve plot for Nearest Neighbor model

2) **Support Vector Machine**

```
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0

[[ 9625     0]
 [    0 25554]]

Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

Fig 2.a: Confusion Matrix and performance metrics for Support Vector Machine model

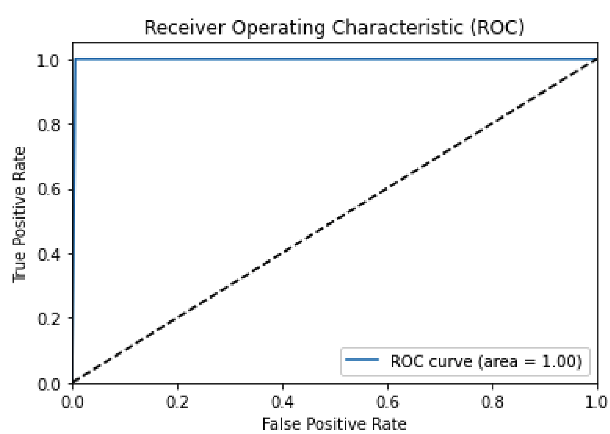
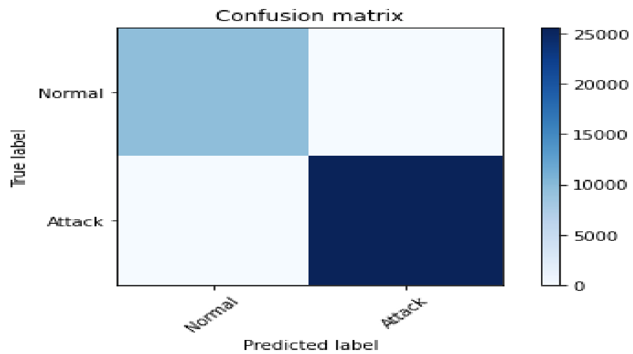


Fig 2.b: ROC Curve plot for Support Vector Machine model

3) **Logistic Regression**

```
Performance Metrics - Logistic Regression
---------------------------------------
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0

[[ 9625     0]
 [    0 25554]]

Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

Fig 3.a: Confusion Matrix and performance metrics for Logistic Regression model

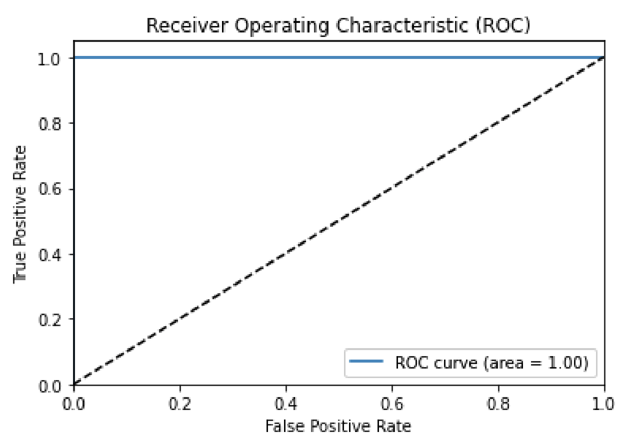
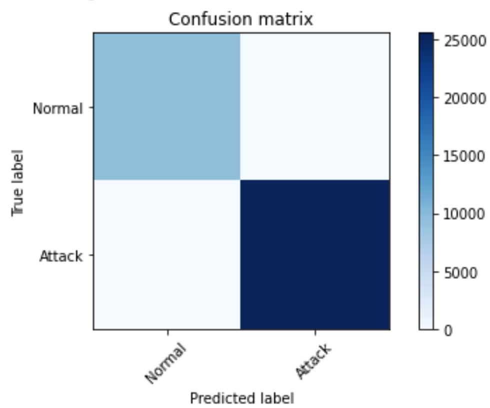


Fig 3.b: ROC Curve plot for Logistic model

4) Fully Connected Neural Networks

```
Performance Metrics for Neural Networks - Binary Classification
---------------------------------------
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0


[[ 9625     0]
 [    0 25554]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

Fig 4.a: Confusion Matrix and performance metrics for Fully Connected Neural Network model

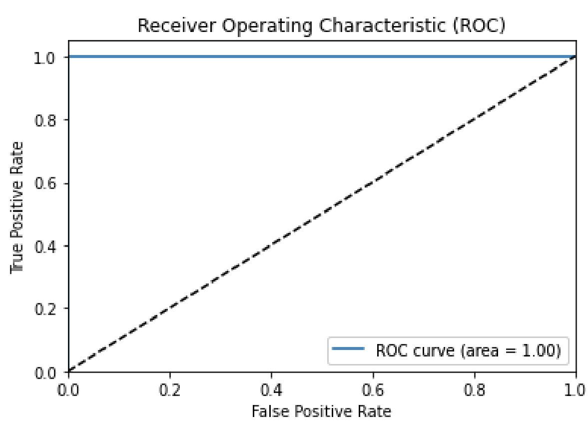


Fig 4.b: ROC Curve plot for Fully Connected Neural Network model

Hyper Parameter Tuning

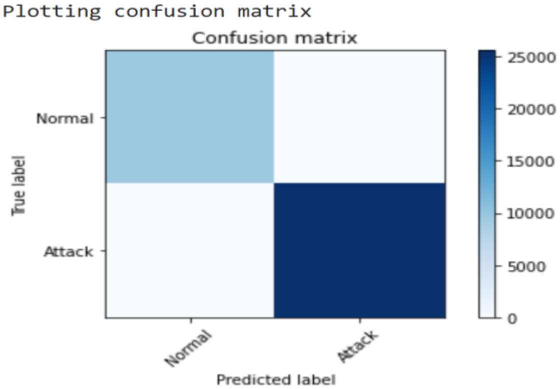
Performance of neural network was evaluated by tuning the following hyper parameters.

CASE 1

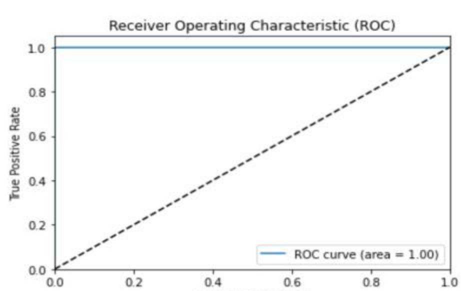
- Optimizer: Adam with default settings
- Activation function: tanh
- Batch size: 128
- Added neurons and layers

Time elapsed (hh:mm:ss.ms) **0:00:48.650198**

```
Performance Metrics for Default ADAM Optimizer
---------------------------------------
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0


[[ 9625     0]
 [    0 25554]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

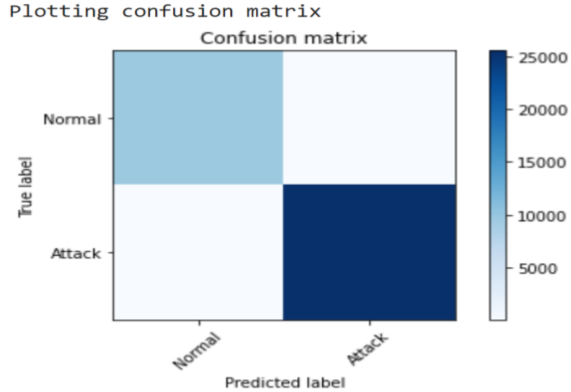


CASE 2

- Optimizer: SGD with default settings
- Activation function: tanh
- Batch size: 128
- Changes to no. of neurons and layers

Time elapsed (hh:mm:ss.ms) **0:01:15.031668**

```
Performance Metrics for Default SGD Optimizer
---------------------------------------
Accuracy score: 0.9990050882628841
Precision score: 0.9990050098564728
Recall score: 0.9990050882628841
F1 score: 0.9990050399237704


[[ 9606    19]
 [   16 25538]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

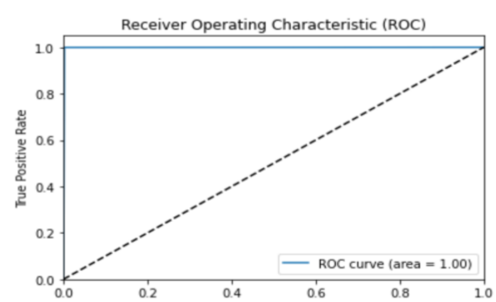


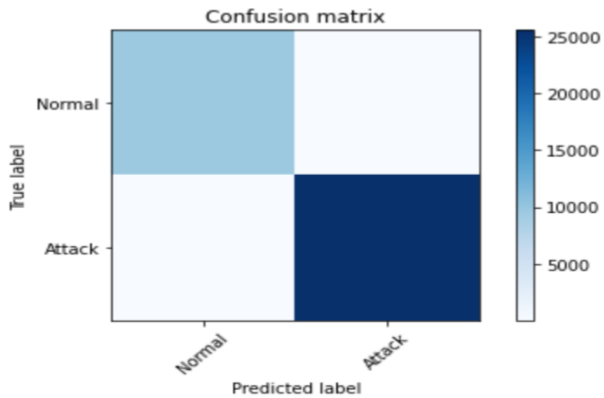
Table 2: Comparison of neural network after performing hyper parameter tuning

CASE 3

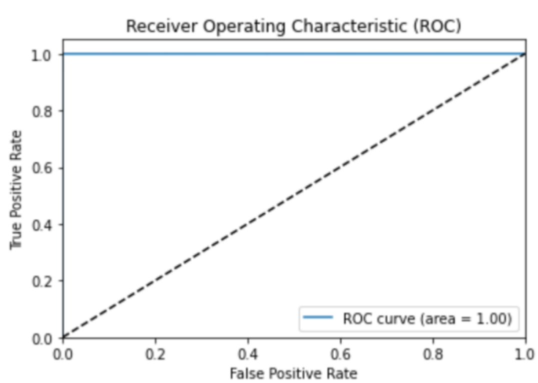
Optimizer: Custom Adam

Time elapsed (hh:mm:ss.ms) 0:01:00.397773

```
Performance Metrics for Custom ADAM optimizers
---------------------------------------
Accuracy score: 0.9995736092555217
Precision score: 0.9995740066201406
Recall score: 0.9995736092555217
F1 score: 0.9995736851158504


[[ 9623     2]
 [   13 25541]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

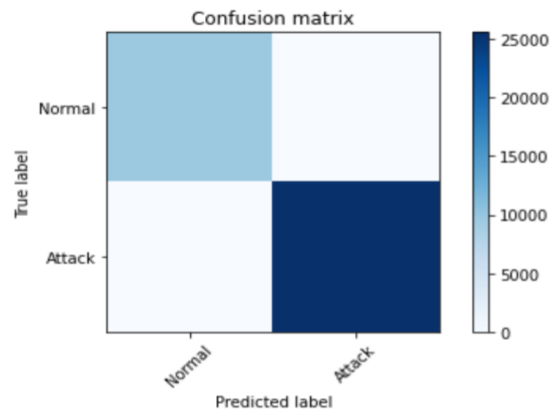


CASE 4

Optimizer: Custom SGD

Time elapsed (hh:mm:ss.ms) 0:02:56.596050

```
Performance Metrics for SGD with Custom Parameters
---------------------------------------
Accuracy score: 0.9994599050569942
Precision score: 0.9994603063319293
Recall score: 0.9994599050569942
F1 score: 0.9994597386093086


[[ 9606    19]
 [    0 25554]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

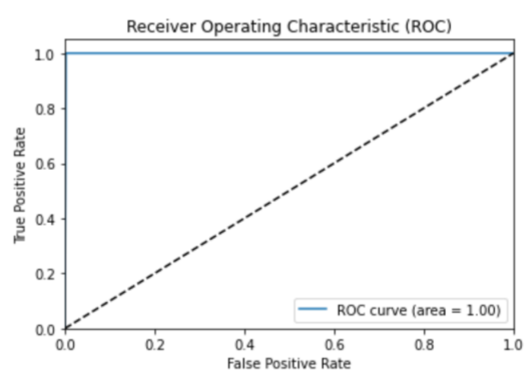


Table 3: Comparison of neural network after performing hyper parameter tuning

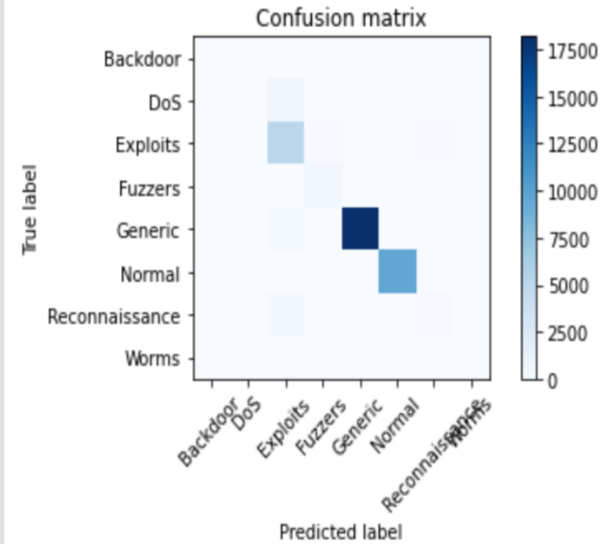
Additional Features

1) Multilabel Classification

- 'Attack_cat' is used as target column which is used to identify type of attack
- Since target column was categorical, it was label encoded and then converted to tensor using to_xy()
- Dataset was unbalanced – More counts for Generic and Normal

```
Attack class category
--------------------------------------
Accuracy score: 0.952755905511811
Precision score: 0.943982189832143
Recall score: 0.952755905511811
F1 score: 0.9442279880040598


[[    0     0     8     2     1     0     0     0]
 [    0    37   616    13    24     1    26     0]
 [    0    27  5069   103     9     0    85     0]
 [    0     1    38   469     0     0    27     0]
 [    0    30   248     3 18173     0     6     0]
 [    0     0     0     0     0  9625     0     0]
 [    0     2   349     2     7     0   144     0]
 [    0     0    32     0     2     0     0     0]]
```

Plotting confusion matrix




```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        11
           1       0.38      0.05      0.09       717
           2       0.80      0.96      0.87      5293
           3       0.79      0.88      0.83       535
           4       1.00      0.98      0.99     18460
           5       1.00      1.00      1.00      9625
           6       0.50      0.29      0.36       504
           7       0.00      0.00      0.00        34

    accuracy                           0.95     35179
   macro avg       0.56      0.52      0.52     35179
weighted avg       0.94      0.95      0.94     35179
```

Fig 5.a: Performance metrics for Multilabel classification model

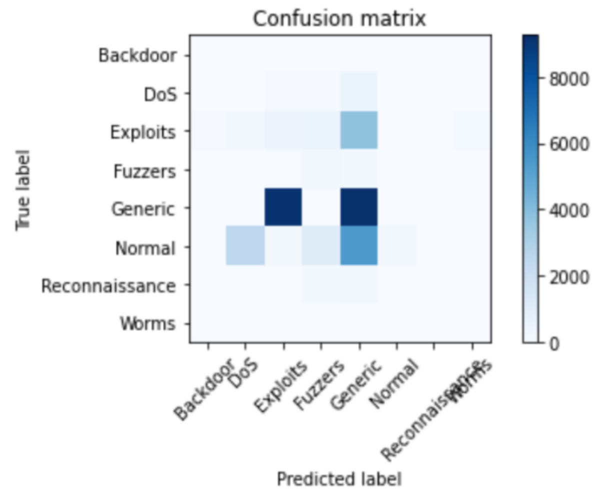
2) Over and Under Sampling

- Scikit- Imblearn functions were used to achieve under and oversampling
- Under Sampling – Near Miss
- Over Sampling – Synthetic Minority Over Sampling Technique (SMOTE), Random Sampler
- Over Sampling was better than Under Sampling due to less record count of minority classes

Under Sampling

```
Performance Evaluation for UnderSampled Dataset
---------------------------------------
Accuracy score: 0.2935558145484522
Precision score: 0.5345315790766723
Recall score: 0.2935558145484522
F1 score: 0.28337418300307227


[[   0    0    1    1    7    0    1    1]
 [   6   24   82   86  490    0    3   26]
 [  89  178  525  570 3771    0   29  131]
 [   0    8   37  271  194    0    0   25]
 [   4    5 9146   37 9254    1    3   10]
 [  18 2545  199 1108 5488  237   19   11]
 [   4   17    7  182  265    0   15   14]
 [   0    0    0   11   21    0    1    1]]
```

```
Plotting confusion matrix
```



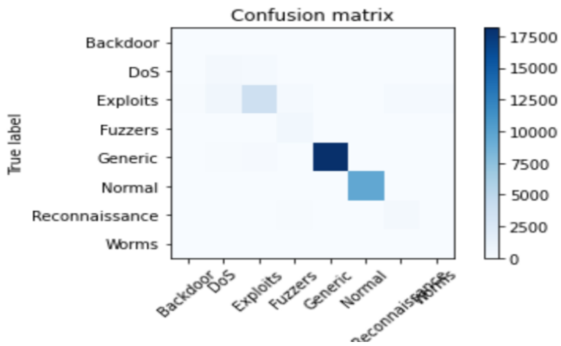

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        11
           1       0.01      0.03      0.01       717
           2       0.05      0.10      0.07      5293
           3       0.12      0.51      0.19       535
           4       0.47      0.50      0.49     18460
           5       1.00      0.02      0.05      9625
           6       0.21      0.03      0.05       504
           7       0.00      0.03      0.01        34

    accuracy                           0.29     35179
   macro avg       0.23      0.15      0.11     35179
weighted avg       0.53      0.29      0.28     35179
```

Fig 6.a: Performance metrics for under sampling

Over Sampling

SMOTE Over Sampler

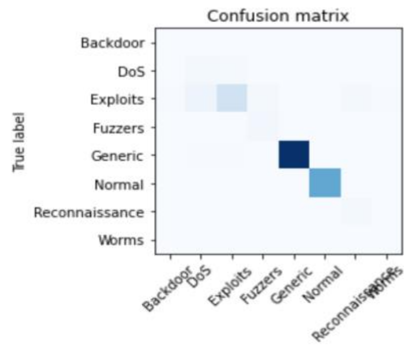
```
Performance Evaluation for OverSampled Dataset
---------------------------------------
Accuracy score: 0.9320048892805367
Precision score: 0.9535320442962062
Recall score: 0.9320048892805367
F1 score: 0.9399572240152725


[[    3     4     1     2     0     0     0     1]
 [   11   325   270    48     2     0    43    18]
 [   76   594  3824   245     3     0   281   270]
 [    1     1     3   520     0     0     6     4]
 [   10   101   158    20 18153     0    10     8]
 [    3     1     5     1     0  9615     0     0]
 [   19     0    52    78     3     0   319    33]
 [    1     1     3     1     0     0     0    28]]


Plotting confusion matrix
```



	precision	recall	f1-score	support
0	0.02	0.27	0.04	11
1	0.32	0.45	0.37	717
2	0.89	0.72	0.80	5293
3	0.57	0.97	0.72	535
4	1.00	0.98	0.99	18460
5	1.00	1.00	1.00	9625
6	0.48	0.63	0.55	504
7	0.08	0.82	0.14	34
accuracy			0.93	35179
macro avg	0.54	0.73	0.58	35179
weighted avg	0.95	0.93	0.94	35179

Random Over Sampler

```
Performance Evaluation for Random Over-Sampler
---------------------------------------
Accuracy score: 0.9248415247733023
Precision score: 0.9519452824201211
Recall score: 0.9248415247733023
F1 score: 0.9338291342546929

[[    2     5     1     2     0     0     1     0]
 [    5   378   221    61    20     2    23     7]
 [  126   899  3529   317     6     4   309   103]
 [    5     4    18   500     0     0     7     1]
 [    8   138   124    23 18160     2     3     2]
 [    0     0     3     0     0  9618     4     0]
 [   56     7    44    50     6     0   333     8]
 [    2     0    14     3     0     0     0    15]]


Plotting confusion matrix
```



	precision	recall	f1-score	support
0	0.01	0.18	0.02	11
1	0.26	0.53	0.35	717
2	0.89	0.67	0.76	5293
3	0.52	0.93	0.67	535
4	1.00	0.98	0.99	18460
5	1.00	1.00	1.00	9625
6	0.49	0.66	0.56	504
7	0.11	0.44	0.18	34
accuracy			0.92	35179
macro avg	0.54	0.67	0.57	35179
weighted avg	0.95	0.92	0.93	35179

Table 4: Performance metrics after using oversampling

3) IoT Dataset

- Modeled as binary classification problem with attack as target column
- Dropped all null and missing values
- pkSeqId column was dropped
- Categorical columns were one hot encoded and Numeric columns were normalized
- For 'saddr' and 'daddr' intersection of test and train datasets were taken and the common columns are one hot encoded

```
Performance Metrics - Neural Nets for IoT Dataset
---------------------------------------
Accuracy score: 0.9998712478683989
Precision score: 0.9998712644456466
Recall score: 0.9998712478683989
F1 score: 0.9998141638036828


[[      6     94]
 [      0 729985]]


Plotting confusion matrix
```

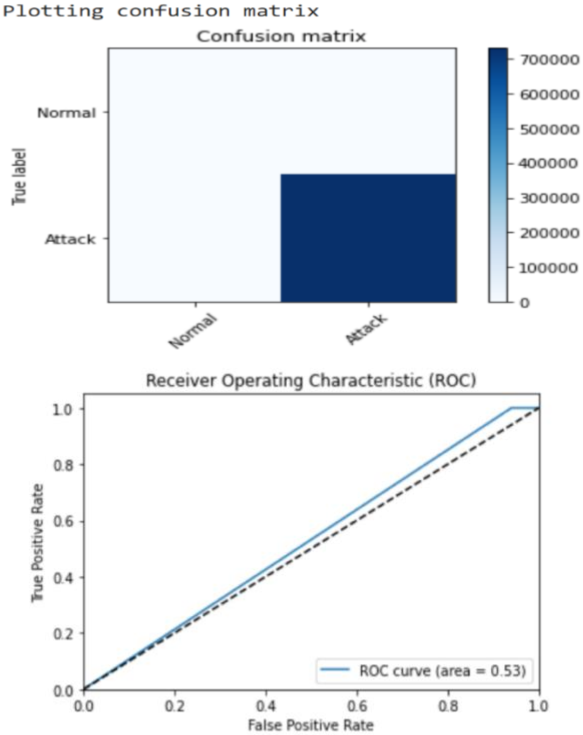


Fig 7.a: Confusion Matrix and performance metrics for IoT dataset using fully connected neural network model

Additional Features

2) Feature Importance Analysis

A large number of features can increase the training time for the models [4]. They may also take a large amount of system memory [5]. We use feature selection to reduce the number of features used for training the fully connected neural network.

We use the filter method to compute the relationship between features and the target variable and filter the features based on the computed scores. This method allows us to compute the importance of feature and select the most important features to train the model.

As the output variable for the model is categorical, we have a Classification predictive modeling problem [5]. Based on the input and output variables present in the dataset, we use univariate statistical measures to identify most important features.

The scikit-learn library provides implementation for useful statistical measures [5]. We use ANOVA: f_classif() statistical measure based on the output variable of our model [6]. Further, the library also provides methods for filtering the features. We use SelectKBest to select top 10 variables.

Experimental Results and Analysis

	Feature	Scores
62	attack_cat-Normal	inf
28	ct_state_ttl	6.752095e+04
6	sttl	4.151726e+04
61	attack_cat-Generic	2.504474e+04
43	state-INT	2.384177e+04
41	state-CON	1.125093e+04
27	ct_srv_src	9.046051e+03
37	ct_srv_dst	8.985007e+03
31	ct_dst_sport_ltm	7.660008e+03
32	ct_dst_src_ltm	7.103948e+03

Fig 8.a: Top-10 most important features using F-Test

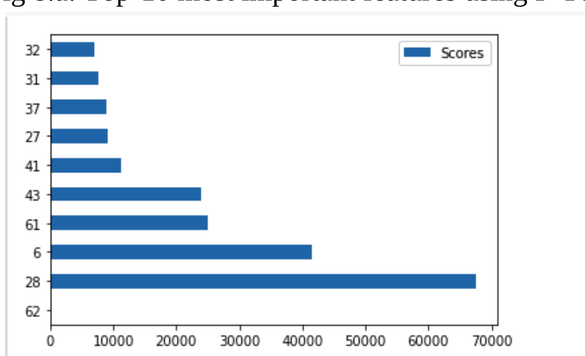
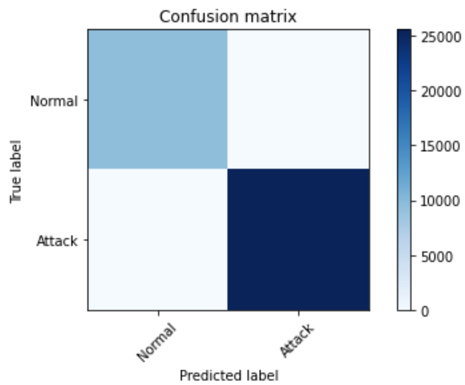


Fig 8.b: Graphical representation of scores of top-10 most important features using F-Test

```
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0


[[ 9625     0]
 [    0 25554]]


Plotting confusion matrix
```



	precision	recall	f1-score	support
0	1.00	1.00	1.00	9625
1	1.00	1.00	1.00	25554
accuracy			1.00	35179
macro avg	1.00	1.00	1.00	35179
weighted avg	1.00	1.00	1.00	35179

Fig 8.c: Confusion Matrix and performance metrics for Fully Connected Neural Network model with 65 features

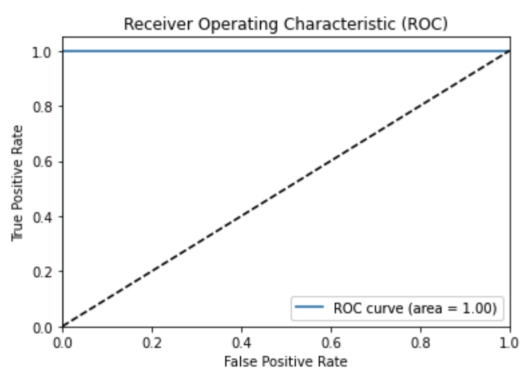
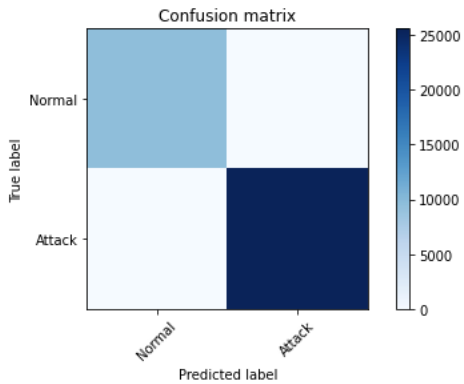


Fig 8.d: ROC Curve plot for Fully Connected Neural Network model with 65 features

```
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0


[[ 9625     0]
 [    0 25554]]


Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9625
           1       1.00      1.00      1.00     25554

    accuracy                           1.00     35179
   macro avg       1.00      1.00      1.00     35179
weighted avg       1.00      1.00      1.00     35179
```

Fig 8.e: Confusion Matrix and performance metrics for Fully Connected Neural Network model with top 10 most important features

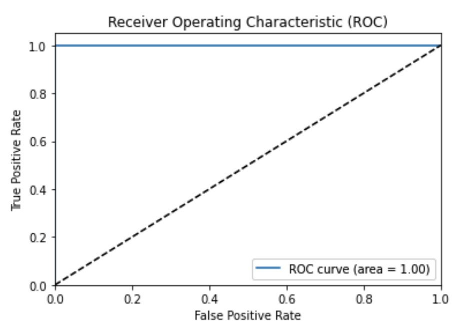


Fig 8.f: ROC Curve plot for Fully Connected Neural Network model with top 10 most important features

```
Model training time using Original All-feature Dataset: 0:01:36.239688
Model training time using Selected Top-10 Best Dataset: 0:01:00.065537
```

Fig 9.g: Comparison of prediction time using 65 features vs 10 most important selected features

Task Division and Project Reflection

Task Division

Data Preprocessing	Harshitha, Gargi
Logistic Regression	Harshitha
Nearest Neighbor	Gargi
Support Vector Machine	Gargi
Fully Connected Neural Network and Hyper Parameter Tuning	Harshitha
Additional Features - Multi-class classification problem	Harshitha
Additional Features - Under and Over Sampling	Harshitha
Additional Features - Feature important analysis	Gargi
Additional Features – IoT Dataset	Harshitha

Table 5: Task Division details

Project Reflection

Multiple challenges were encountered during the project implementation which allowed to learn new techniques and tools.

1. **Virtual Collaboration:** Since the project must be carried out virtually, communication and collaboration are challenging. With the help of collaborative tools like Google Colab, Git and Zoom we were able to connect and coordinate better.
2. **Data Preprocessing:** Data Preprocessing is a major portion of any AI or Machine learning. Understanding the nature and context of each column and how we process huge datasets is key to better model performance.
3. **Saving Preprocessed data:** Since AI programs are process intensive, it is better to save preprocessed data to csv and reuse it later instead of preprocessing the data.
4. **Code Reusability:** Data preprocessing and plot lib functions are used across many notebooks. To achieve clean code, the above functions are put in a separate helper file and imported across notebooks rather than having it in each notebook.
5. **Fine tuning hyper parameters for models:** Fine tuning hyper parameters require a lot of permutation and combination to get it right. Changes should be small and only a few parameters must be changed at once step.
6. **Feature importance analysis:** Feature selection can be done in different ways. Identifying and implementing the method suitable for the given type of problem statement is important. Feature selection reduces the model training time and can make significant difference when a large number of features are present in the dataset.

Works Cited

[1] "CSC 215 Mini-Project 1 resources," [Online]. Available: https://drive.google.com/open?id=1-jiDgzfbnTzyF5MZDGzictIKjnJQ39lj.

[2] "The BoT-IoT Dataset," [Online]. Available: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php.

[3] "The UNSW-NB15 Dataset Description," [Online]. Available: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/.

[4] "Why, How and When to apply Feature Selection," [Online]. Available: https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2.

[5] "How to Choose a Feature Selection Method For Machine Learning," [Online]. Available: https://analyticsweek.com/content/how-to-choose-a-feature-selection-method-for-machine-learning/.

[6] "Sklearn feature selection," [Online]. Available: https://analyticsweek.com/content/how-to-choose-a-feature-selection-method-for-machine-learning/.