CSC 215-01 Artificial Intelligence (Fall 2019)

Mini-Project 1: Modern Low Footprint Cyber Attack Detection

Team Members:

Ayushi Vadwala (220234041)

Jeet Shah (220267750)

I. Problem Statement:

In this project, we aim to build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections

II. Methodology:

- Data Pre-processing to gain better accuracy.
- Feature normalization using Z-score.
- Applied Logistic Regression, Nearest Neighbor, Support Vector Machine and Fully-Connected Neural Networks
- Applying the models and generating classification report and ROC curve to compare their performance.
- Parameter tuning to compare model with different parameters.

Step 1: Read file with the training set and testing set.

```
import pandas as pd

train_df = pd.read_csv('data/UNSW_NB15_training-set_csc215.csv')
test_df = pd.read_csv('data/UNSW_NB15_test-set_csc215.csv')
```

Step 2: Drop the columns which are not needed for further analysis by analyzing training and testing dataset. By analyzing we found the columns with missing values and columns which do not have impact on to be predicted column.

```
train_df.drop(['id','service','attack_cat', 'swin'], axis=1, inplace=True)
test_df.drop(['id','service','attack_cat', 'swin'], axis=1, inplace=True)
```

Step 3: Detecting OUTLIERS in dataset

- As the values of column "proto" has more values of 'tcp', 'udp' and 'unas' in both training and testing dataset compared to others, others might not have more impact on target compare to 'tcp', 'udp' and 'unas' or they may be OUTLIERS. So, combine all other values except 'tcp', 'udp' and 'unas' into value called others and drop rows of that values.
- As the values of the column “state” attribute mismatch in both train and test dataset, it might create issues while encoding them using one-hot encoding because of shape mismatch. Hence, we will be keeping the top 4 states in the data and will remove the rest. THEY ARE OUTLIERS.

Step 4: One-hot Encoding on categorical features

```
encode_text_dummy(train_df_v1,"proto")    # encoding first before you call to_xy()
encode_text_dummy(train_df_v1,"state")

encode_text_dummy(test_df_v1,"proto")    # encoding first before you call to_xy()
encode_text_dummy(test_df_v1,"state")
```

Step 5: More data preprocessing using Correlation between each feature and target

- Correlation is obtained by dividing the covariance of the two variables by the product of their standard deviations.
- Correlation(x,y) = (Covariance(x,y)) / Sx * Sy

$$= (E[x*y] - E[x]*E[y]) / Sx * Sy$$

where, E = Total sum, Sx = Standard Deviation of x, Sy = Standard Deviation of y

- We have used pandas defined corr() to find correlation and it returns series
- The correlation of a variable with itself is 1. So, which column has correlation value near to 1 has more impact on target column. Hence, we will keep columns which has correlation greater than 0.05 and will drop others.

```
import statistics

correlation_list = train_df_v1[train_df_v1.columns[:]].corr()['label']

#med = statistics.median(correlation_list)
correlation_list = [col for col in correlation_list.index if correlation_list[col] < 0.05]
print(correlation_list)
```

```
['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit', 'stcpb', 'dtcp
b', 'dwin', 'smean', 'dmean', 'trans_depth', 'response_body_len', 'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'is_sm_i
ps_ports', 'proto-tcp', 'state-CON', 'state-FIN', 'state-REQ']
```

```
train_df_v2 = train_df_v1.drop(correlation_list, axis = 1)
test_df_v2 = test_df_v1.drop(correlation_list, axis = 1)
```

```
train_df_v2.shape
```

```
(155230, 19)
```

Step 6: Splitting the data into X and y

```
X_train = train_df_v2.drop(['label'], axis = 1)
y_train = train_df_v2.loc[:, ['label']]

X_test = test_df_v2.drop(['label'], axis = 1)
y_test = test_df_v2.loc[:, ['label']]
```

Step 7: Feature Normalization using Z-Score

- Z-score (X) = (X - MEAN) / Standard Deviation

- Z-score is the number of standard deviations from the mean a data point is. Z-score range falls between -3 (Standard Deviation) to 3 (Standard Deviation).
- We can Normalize data using **Min-Max Scaling** too but for this data, It is taking too long that's why we used Z-score.

```
X_train = X_train.astype('float')
X_test = X_test.astype('float')
```

```
for col in X_train.columns:
        encode_numeric_zscore(X_train,col)

for col in X_test.columns:
        encode_numeric_zscore(X_test,col)
```

Step 8: Balancing the dataset using DownSampling

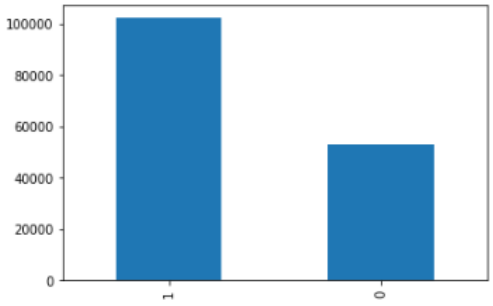
- Downsampling is the process of reducing the sampling size of the data. As per the 1st figure, We can see that data is not balanced for the target value of '0' and '1'. So, make dataset balanced we will do Downsampling and keep minimum rows among both values.

- There are more ways to do downsampling but we have used random downsampling using numpy's **random.shuffle** method to shuffle data and keep minimum rows among both target value.

```
target = 'label'
```

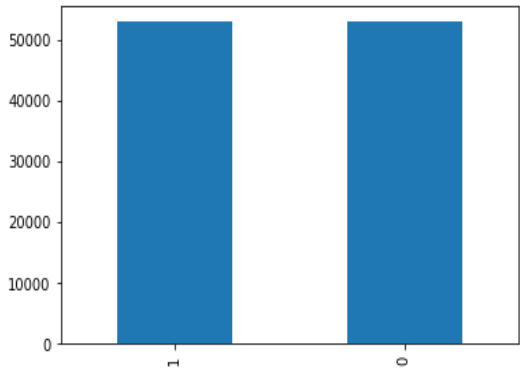
```
y_train[target].value_counts().plot(kind = 'bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x400b6b3390>
```




```
index_label0 = y_train[y_train[target] == 0].index.values
index_label1 = y_train[y_train[target] == 1].index.values

labels_min_rows = min(y_train[target].value_counts())

np.random.shuffle(index_label0)
np.random.shuffle(index_label1)

index_shuffled_label0, index_shuffled_label1 = list(index_label0), list(index_label1)

indices_ = index_shuffled_label0[:labels_min_rows] + index_shuffled_label1[:labels_min_rows]
```

- **Downsampled Data**



Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train_ds.values, y_train_ds.values)

# predict the response for new observations
y_pred = logreg.predict(X_test.values)
y_pred
```

Classification report for Logistic Regression

```
from sklearn import metrics

print(metrics.classification_report(y_test, y_pred))
```

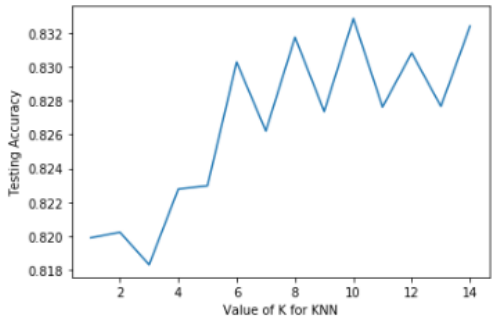
```
              precision    recall  f1-score   support

           0       0.95      0.54      0.69     35942
           1       0.70      0.97      0.82     40080

    accuracy                           0.77     76022
   macro avg       0.83      0.76      0.75     76022
weighted avg       0.82      0.77      0.76     76022
```

Nearest Neighbor

```
#KNN starts here

from sklearn.neighbors import KNeighborsClassifier
k_range = range(1, 15)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    # fit the model with data
    knn.fit(X_train_ds.values, y_train_ds.values)
    # predict the response for new observations
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

print(scores)
# instantiate the model (using the value K=10)
```

```
[0.8198942411407224, 0.820223093315093, 0.8183025966167689, 0.8227749861882087, 0.8229722974928311, 0.8302859698508327, 0.82
62082028886375, 0.8317592275920128, 0.8273657625424219, 0.832864170897898, 0.8276288442819184, 0.8308252874168004, 0.8276683
065428428, 0.832416931940754]
```

```
Text(0, 0.5, 'Testing Accuracy')
```



We tried to locate the best value for k and for K=10 we get the best accuracy.

Classification report for KNN

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.78      0.82     35942
           1       0.82      0.89      0.85     40080

    accuracy                           0.83     76022
   macro avg       0.84      0.83      0.83     76022
weighted avg       0.84      0.83      0.83     76022
```

SVM (Support Vector Machine)

```
from sklearn import svm
from sklearn import metrics

#create svm classifier
cls = svm.SVC(kernel = 'rbf', gamma = 0.5)

#train
cls.fit(X_train_ds,y_train_ds)

#predict
y_pred=cls.predict(X_test)

y_pred
```

Classification report for SVM

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.72      0.81     35942
           1       0.79      0.94      0.86     40080

    accuracy                           0.84     76022
   macro avg       0.85      0.83      0.83     76022
weighted avg       0.85      0.84      0.84     76022
```

Neural Networks

```
import io
import requests
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
save_path="./dnn/"

model = Sequential()

#Tuning Hyperparameters to achieve best model and to improve performance
model.add(Dense(50, input_dim=X_train_ds.shape[1], activation='relu'))
model.add(Dense(25,activation='relu'))
model.add(Dense(15,activation='relu'))
#model.add(Dense(16,activation='relu'))
model.add(Dense(y_train_ds1.shape[1],activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-4, patience=10, verbose=2, mode='auto')

#checkpointer = ModelCheckpoint(filepath="/best_weights.hdf5", verbose=0, save_best_only=True) # save best model

model.fit(X_train_ds, y_train_ds1, validation_data=(X_test,y_test_tf), callbacks=[monitor], verbose=2, epochs=100)

#model.load_weights('/best_weights.hdf5') # load weights from best model
model.save(os.path.join(save_path,"best_weights.hdf5"))
```

```
105944/105944 - 9s - loss: 0.1516 - acc: 0.9229 - val_loss: 0.3017 - val_acc: 0.8521
Epoch 11/100
105944/105944 - 8s - loss: 0.1512 - acc: 0.9236 - val_loss: 0.3139 - val_acc: 0.8420
Epoch 12/100
105944/105944 - 8s - loss: 0.1506 - acc: 0.9236 - val_loss: 0.2905 - val_acc: 0.8529
Epoch 13/100
105944/105944 - 8s - loss: 0.1499 - acc: 0.9238 - val_loss: 0.3127 - val_acc: 0.8400
Epoch 14/100
105944/105944 - 9s - loss: 0.1498 - acc: 0.9231 - val_loss: 0.3518 - val_acc: 0.8235
Epoch 15/100
105944/105944 - 10s - loss: 0.1494 - acc: 0.9238 - val_loss: 0.3313 - val_acc: 0.8142
Epoch 16/100
105944/105944 - 8s - loss: 0.1490 - acc: 0.9235 - val_loss: 0.3411 - val_acc: 0.8361
Epoch 17/100
105944/105944 - 8s - loss: 0.1485 - acc: 0.9245 - val_loss: 0.3221 - val_acc: 0.8528
Epoch 18/100
105944/105944 - 8s - loss: 0.1482 - acc: 0.9238 - val_loss: 0.3160 - val_acc: 0.8409
Epoch 19/100
105944/105944 - 8s - loss: 0.1477 - acc: 0.9251 - val_loss: 0.3687 - val_acc: 0.8306
Epoch 00019: early stopping
```

Classification Report for neural network

```
print(metrics.classification_report(y_test, y_pred))
```

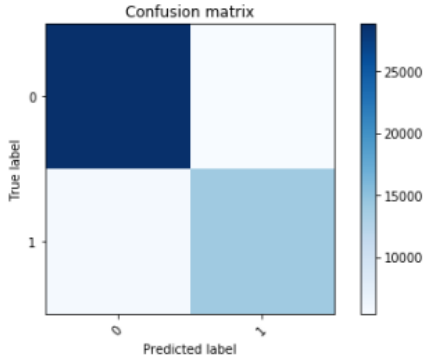
```
              precision    recall  f1-score   support

           0       0.95      0.68      0.79     35942
           1       0.77      0.97      0.86     40080

    accuracy                           0.83     76022
   macro avg       0.86      0.82      0.82     76022
weighted avg       0.85      0.83      0.83     76022
```

III. Experimental Results and Analysis

Logistic regression

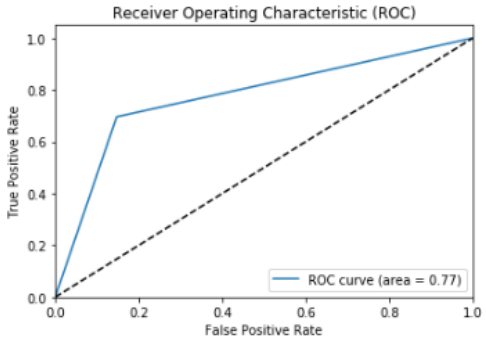
Confusion Matrix

```
[[28818  5383]
 [ 5668 14022]]
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

         0.0       0.84      0.84      0.84     34201
         1.0       0.72      0.71      0.72     19690

    accuracy                           0.79     53891
   macro avg       0.78      0.78      0.78     53891
weighted avg       0.79      0.79      0.79     53891
```

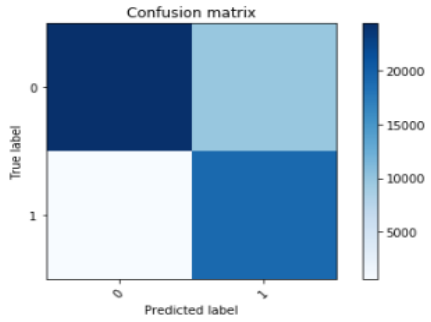
ROC Curve



KNN

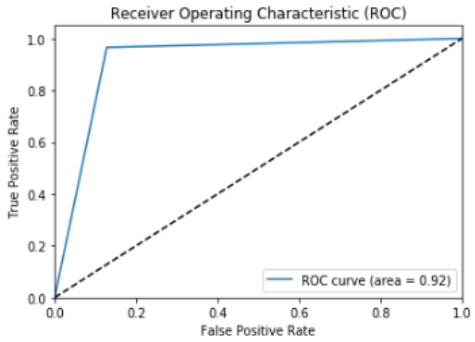
Confusion Matrix

```
[[24396  9805]
 [  587 19103]]
Plotting confusion matrix
```



	precision	recall	f1-score	support
0.0	0.98	0.71	0.82	34201
1.0	0.66	0.97	0.79	19690
accuracy			0.81	53891
macro avg	0.82	0.84	0.81	53891
weighted avg	0.86	0.81	0.81	53891

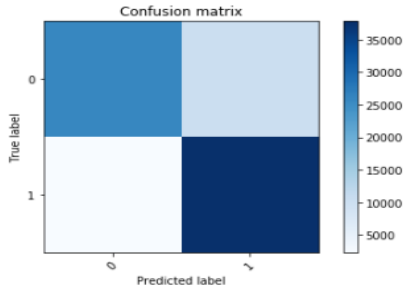
ROC Curve



SVM

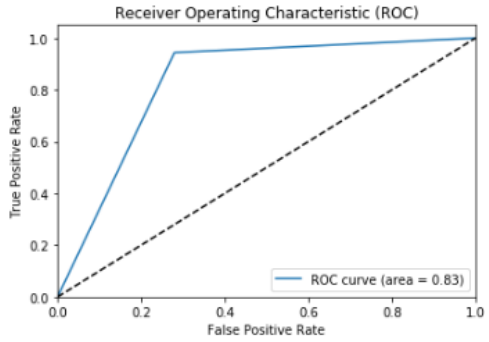
Confusion Matrix

```
[[25890 10052]
 [ 2255 37825]]
Plotting confusion matrix
```



	precision	recall	f1-score	support
0	0.92	0.72	0.81	35942
1	0.79	0.94	0.86	40080
accuracy			0.84	76022
macro avg	0.85	0.83	0.83	76022
weighted avg	0.85	0.84	0.84	76022

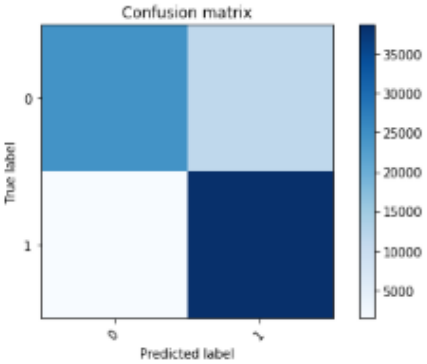
ROC curve



Neural Network

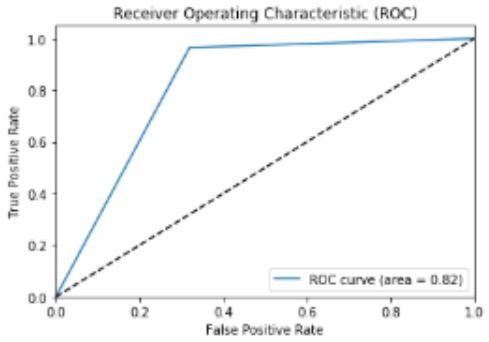
Confusion Matrix

```
[[24457 11485]
 [ 1392 38688]]
Plotting confusion matrix
```



	precision	recall	f1-score	support
0	0.95	0.68	0.79	35942
1	0.77	0.97	0.86	40080
accuracy			0.83	76022
macro avg	0.86	0.82	0.82	76022
weighted avg	0.85	0.83	0.83	76022

ROC curve



Comparison of all the algorithms

	Accuracy	F1 Score	Recall	Precision
Logistic regression	0.79	0.76	0.77	0.82
KNN	0.83	0.83	0.83	0.84
SVM	0.84	0.84	0.84	0.85
Neural Network	0.83	0.82	0.83	0.85

Neural Network

Hidden Dense layer	Neurons	Activation	Optimizer	Accuracy	F1 score [0,1]	Recall [0,1]	Precision [0,1]
3	50, 25, 15	'Relu'	'Adam'	0.86	0.83,0.87	0.75,0.95	0.94,0.81
3	72,47,31	'Relu'	'sgd'	0.83	0.79,0.85	0.67,0.96	0.94,0.77
4	128,64,32,16	'Relu'	'Adam'	0.82	0.78,0.84	0.69,0.93	0.90,0.77
3	128,64,32	'Sigmoid'	'sgd'	0.80	0.75,0.84	0.63,0.96	0.93,0.74
4	50,37,23,16	'Sigmoid'	'sgd'	0.81	0.77,0.84	0.67,0.93	0.90,0.76
3	60,45,32	'tanh'	'Adam'	0.84	0.81,0.86	0.74,0.93	0.91,0.80
4	120,85,40,16	'tanh'	'Adam'	0.82	0.78,0.85	0.69,0.94	0.91,0.77
2	120,85	'tanh'	'sgd'	0.85	0.83,0.86	0.79,0.90	0.88,0.82

IV. Task Division and Project Reflection

Ayushi Vadwala:

- Tried Extra Trees Classifier to find most important features.
- Did Correlation analysis
- Worked on removing the unnecessary features
- Applied Logistic Regression and Nearest Neighbor by finding best K value and SVM
- Worked on Additional Feature

Jeet Shah:

- Tried Linear Regression method to find important features.
- Downsampling of data
- Worked on removing the unnecessary features
- Worked on SVM and Fully-Connected Neural Networks with parameter tuning
- Worked on Additional Feature

V. Challenges

1. Mismatch in train and test set after categorical encoding.

As the values of the proto & state occurring feature mismatch in both train and test dataset, it could generate problems while encoding them using one-hot encoding. We therefore thought that they must be OUTLIERS in dataset. So, we print counts for each value in proto & state column. And the result was only 3 to 4 values had covered most of the rows of dataset. Other value's counts were so small compared to other 4 values. Therefore, we kept the top 3 values of proto and top 4 of states in the dataset and droped the remaining values.

And after droping that rows we trained our model and got better accuracy. That's how we were came to know about OUTLIERS that are making data too noisy and they were making negative impact in predicting by solving feature mismatch.

2. **Selection of most important features in dataset.**
   We tried several statistical methods to find first few important features
   Recursive feature elimination was taking long time to execute. SelectKBest method doesn't accept negative numeric values as input. We tried extra trees classifier but didn't get proper results. Then we worked on Correlation analysis and got better results.
3. **More Preprocessing of dataset for best accuracy.**
   First of all, We were removing the duplicate rows that are available in dataset but it does not making model's performance better. Moreover, It was reducing accuracy of predicting the target. So, we thought of Downsampling of target column instead of dropping duplicate rows. So, we used numpy's random.shuffle method to randomly shuffling data for DownSampling. After that, we kept rows which has minimum target value and dropped the rest from train dataset. And we got better performance and accuracy than previous one.
4. **Finding the best features which has actual impact on target.**
   With the help of correlation analysis between each feature and target, we found out which feature has more impact on target and which has not. And we dropped feature which has not more impact on target while keeping the rest.
5. **Deciding the number of layers and nodes in the neural network.**
   We tried several different models and some of them are shown in the result analysis to compare and find the best suitable model for the dataset.

VI. Additional Feature implementation

1) **Among all the features, identify the top 10 important features**
   Using correlation, we can find the top 10 features which has more correlation coefficient compared to others. Advantage of doing so is we can reduce the training time and there is less chance of overfitting.
2) **Intrusion Detection as a Multi-class classification problem to detect the type of each intrusion**
   For Multi-class classification problem, we did not downsample and removed 'label' attribute instead of 'attack_cat'. Rest of the process for neural network is same. The accuracy of this model turned out to be 0.71.
3) **Create a more balanced dataset to train your model**
   For the implementation of this feature we did down sampling of the data so that our data is not biased for any one specific value of 'Label'.