

# Machine Learning for the Detection of Network Attacks

Analyse the machine learning algorithms on the [CICIDS 2017 Dataset] for clasification of network attacks. (<https://www.unb.ca/cic/datasets/ids-2017.html>):

- Support Vector Machine (Naive Bayes)
- Decision Tree
- Naive Bayes
- K Means Clustering
- K Nearest Neighbours

## Import required libraries.

In [1]:

```
import glob
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sn
import time

from numpy import array

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import mutual_info_classif

from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import completeness_score, homogeneity_score, v_measure_score

from sklearn.model_selection import train_test_split
```

# Loading the dataset

The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS.

Datasets is available in 8 different csv files.

- Monday-WorkingHours.pcap\_ISCX.csv
- Tuesday-WorkingHours.pcap\_ISCX.csv
- Wednesday-workingHours.pcap\_ISCX.csv
- Thursday-WorkingHours-Morning-WebAttacks.pcap\_ISCX.csv
- Thursday-WorkingHours-Afternoon-Infiltration.pcap\_ISCX.csv
- Friday-WorkingHours-Morning.pcap\_ISCX.csv
- Friday-WorkingHours-Afternoon-PortScan.pcap\_ISCX.csv
- Friday-WorkingHours-Afternoon-DDos.pcap\_ISCX.csv

8 different csv files of cids dataset needs to be concatenated into a single csv file.

```
In [2]: # # path to the all 8 files of CICIDS dataset.
# path = './datasets'
# all_files = glob.glob(path + "/*.csv")

# # concatenate the 8 files into 1.
# dataset = pd.concat((pd.read_csv(f) for f in all_files))
```

```
In [3]: # # saving the combined dataset to disk named cids.csv
# dataset.to_csv('cids')
```

```
In [4]: dataset=pd.read_csv('cids.csv')
```

```
In [5]: # Dimenions of dataset.
print(dataset.shape)
```

(2827876, 79)

```
In [6]: # column names as per dataset.

col_names = ["Destination_Port",
             "Flow_Duration",
             "Total_Fwd_Packets",
             "Total_Backward_Packets",
             "Total_Length_of_Fwd_Packets",
             "Total_Length_of_Bwd_Packets",
             "Fwd_Packet_Length_Max",
             "Fwd_Packet_Length_Min",
             "Fwd_Packet_Length_Mean",
             "Fwd_Packet_Length_Std",
             "Bwd_Packet_Length_Max",
             "Bwd_Packet_Length_Min",
             "Bwd_Packet_Length_Mean",
             "Bwd_Packet_Length_Std",
```

```
"Flow_Bytes_s",
"Flow_Packets_s",
"Flow_IAT_Mean",
"Flow_IAT_Std",
"Flow_IAT_Max",
"Flow_IAT_Min",
"Fwd_IAT_Total",
"Fwd_IAT_Mean",
"Fwd_IAT_Std",
"Fwd_IAT_Max",
"Fwd_IAT_Min",
"Bwd_IAT_Total",
"Bwd_IAT_Mean",
"Bwd_IAT_Std",
"Bwd_IAT_Max",
"Bwd_IAT_Min",
"Fwd_PSH_Flags",
"Bwd_PSH_Flags",
"Fwd_URG_Flags",
"Bwd_URG_Flags",
"Fwd_Header_Length",
"Bwd_Header_Length",
"Fwd_Packets_s",
"Bwd_Packets_s",
"Min_Packet_Length",
"Max_Packet_Length",
"Packet_Length_Mean",
"Packet_Length_Std",
"Packet_Length_Variance",
"FIN_Flag_Count",
"SYN_Flag_Count",
"RST_Flag_Count",
"PSH_Flag_Count",
"ACK_Flag_Count",
"URG_Flag_Count",
"CWE_Flag_Count",
"ECE_Flag_Count",
"Down_Up_Ratio",
"Average_Packet_Size",
"Avg_Fwd_Segment_Size",
"Avg_Bwd_Segment_Size",
"Fwd_Header_Length",
"Fwd_Avg_Bytes_Bulk",
"Fwd_Avg_Packets_Bulk",
"Fwd_Avg_Bulk_Rate",
"Bwd_Avg_Bytes_Bulk",
"Bwd_Avg_Packets_Bulk",
"Bwd_Avg_Bulk_Rate",
"Subflow_Fwd_Packets",
"Subflow_Fwd_Bytes",
"Subflow_Bwd_Packets",
"Subflow_Bwd_Bytes",
"Init_Win_bytes_forward",
"Init_Win_bytes_backward",
"act_data_pkt_fwd",
"min_seg_size_forward",
"Active_Mean",
"Active_Std",
"Active_Max",
"Active_Min",
"Idle_Mean",
```

```

        "Idle_Std",
        "Idle_Max",
        "Idle_Min",
        "Label"
    ]

```

In [7]: *# Max rows and columns to be shown in print console*

```

pd.options.display.max_columns= 200
pd.options.display.max_rows= 200

```

In [8]: *# Assigning the column names.*  
*# first 5 records in the dataset.*

```

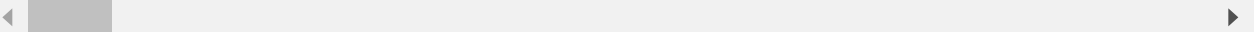
dataset.columns = col_names

dataset.head(5)

```

Out[8]:

	Destination_Port	Flow_Duration	Total_Fwd_Packets	Total_Backward_Packets	Total_Length_of_Fwd_Pack
0	0	54865	3	2	
1	1	55054	109	1	
2	2	55055	52	1	
3	3	46236	34	1	
4	4	54863	3	2	



In [9]: *# check whether there is any categorical column are not if it is there it is to be encoded*

```

dataset.dtypes

```

Out[9]:

Destination_Port	int64
Flow_Duration	int64
Total_Fwd_Packets	int64
Total_Backward_Packets	int64
Total_Length_of_Fwd_Packets	int64
Total_Length_of_Bwd_Packets	int64
Fwd_Packet_Length_Max	int64
Fwd_Packet_Length_Min	int64
Fwd_Packet_Length_Mean	int64
Fwd_Packet_Length_Std	float64
Bwd_Packet_Length_Max	float64
Bwd_Packet_Length_Min	int64
Bwd_Packet_Length_Mean	int64
Bwd_Packet_Length_Std	float64
Flow_Bytes_s	float64
Flow_Packets_s	float64
Flow_IAT_Mean	float64
Flow_IAT_Std	float64
Flow_IAT_Max	float64
Flow_IAT_Min	int64
Fwd_IAT_Total	int64
Fwd_IAT_Mean	int64
Fwd_IAT_Std	float64
Fwd_IAT_Max	float64
Fwd_IAT_Min	int64

Bwd_IAT_Total	int64
Bwd_IAT_Mean	int64
Bwd_IAT_Std	float64
Bwd_IAT_Max	float64
Bwd_IAT_Min	int64
Fwd_PSH_Flags	int64
Bwd_PSH_Flags	int64
Fwd_URG_Flags	int64
Bwd_URG_Flags	int64
Fwd_Header_Length	int64
Bwd_Header_Length	int64
Fwd_Packets_s	int64
Bwd_Packets_s	float64
Min_Packet_Length	float64
Max_Packet_Length	int64
Packet_Length_Mean	int64
Packet_Length_Std	float64
Packet_Length_Variance	float64
FIN_Flag_Count	float64
SYN_Flag_Count	int64
RST_Flag_Count	int64
PSH_Flag_Count	int64
ACK_Flag_Count	int64
URG_Flag_Count	int64
CWE_Flag_Count	int64
ECE_Flag_Count	int64
Down_Up_Ratio	int64
Average_Packet_Size	int64
Avg_Fwd_Segment_Size	float64
Avg_Bwd_Segment_Size	float64
Fwd_Header_Length	float64
Fwd_Avg_Bytes_Bulk	int64
Fwd_Avg_Packets_Bulk	int64
Fwd_Avg_Bulk_Rate	int64
Bwd_Avg_Bytes_Bulk	int64
Bwd_Avg_Packets_Bulk	int64
Bwd_Avg_Bulk_Rate	int64
Subflow_Fwd_Packets	int64
Subflow_Fwd_Bytes	int64
Subflow_Bwd_Packets	int64
Subflow_Bwd_Bytes	int64
Init_Win_bytes_forward	int64
Init_Win_bytes_backward	int64
act_data_pkt_fwd	int64
min_seg_size_forward	int64
Active_Mean	float64
Active_Std	float64
Active_Max	int64
Active_Min	int64
Idle_Mean	float64
Idle_Std	float64
Idle_Max	int64
Idle_Min	int64
Label	object

dtype: object

## Remove repeated columns, (NaN,Null,Infinite) values.

```
In [10]: # Removing the duplicate columns (Header_Length is repeated)
dataset = dataset.loc[:, ~dataset.columns.duplicated()]
```

```
dataset.shape
```

```
Out[10]: (2827876, 78)
```

```
In [11]: # check if there are any Null values
dataset.isnull().any().any()
```

```
Out[11]: False
```

```
In [12]: # Replace Inf values with NaN
dataset = dataset.replace([np.inf, -np.inf], np.nan)

# Drop all occurrences of NaN
dataset = dataset.dropna()

# Double check these are all gone
dataset.isnull().any()
```

```
Out[12]: Destination_Port      False
Flow_Duration                 False
Total_Fwd_Packets             False
Total_Backward_Packets        False
Total_Length_of_Fwd_Packets    False
Total_Length_of_Bwd_Packets    False
Fwd_Packet_Length_Max          False
Fwd_Packet_Length_Min          False
Fwd_Packet_Length_Mean         False
Fwd_Packet_Length_Std          False
Bwd_Packet_Length_Max          False
Bwd_Packet_Length_Min          False
Bwd_Packet_Length_Mean         False
Bwd_Packet_Length_Std          False
Flow_Bytes_s                   False
Flow_Packets_s                 False
Flow_IAT_Mean                  False
Flow_IAT_Std                   False
Flow_IAT_Max                   False
Flow_IAT_Min                   False
Fwd_IAT_Total                  False
Fwd_IAT_Mean                   False
Fwd_IAT_Std                    False
Fwd_IAT_Max                    False
Fwd_IAT_Min                    False
Bwd_IAT_Total                  False
Bwd_IAT_Mean                   False
Bwd_IAT_Std                    False
Bwd_IAT_Max                    False
Bwd_IAT_Min                    False
Fwd_PSH_Flags                  False
Bwd_PSH_Flags                  False
Fwd_URG_Flags                  False
Bwd_URG_Flags                  False
Fwd_Header_Length              False
Bwd_Header_Length              False
Fwd_Packets_s                  False
Bwd_Packets_s                  False
Min_Packet_Length              False
Max_Packet_Length              False
Packet_Length_Mean             False
Packet_Length_Std              False
```

Packet_Length_Variance	False
FIN_Flag_Count	False
SYN_Flag_Count	False
RST_Flag_Count	False
PSH_Flag_Count	False
ACK_Flag_Count	False
URG_Flag_Count	False
CWE_Flag_Count	False
ECE_Flag_Count	False
Down_Up_Ratio	False
Average_Packet_Size	False
Avg_Fwd_Segment_Size	False
Avg_Bwd_Segment_Size	False
Fwd_Avg_Bytes_Bulk	False
Fwd_Avg_Packets_Bulk	False
Fwd_Avg_Bulk_Rate	False
Bwd_Avg_Bytes_Bulk	False
Bwd_Avg_Packets_Bulk	False
Bwd_Avg_Bulk_Rate	False
Subflow_Fwd_Packets	False
Subflow_Fwd_Bytes	False
Subflow_Bwd_Packets	False
Subflow_Bwd_Bytes	False
Init_Win_bytes_forward	False
Init_Win_bytes_backward	False
act_data_pkt_fwd	False
min_seg_size_forward	False
Active_Mean	False
Active_Std	False
Active_Max	False
Active_Min	False
Idle_Mean	False
Idle_Std	False
Idle_Max	False
Idle_Min	False
Label	False

dtype: bool

## Analysing the attacks in dataset

```
In [13]: # Distribution of Dataset
dataset['Label'].value_counts()
```

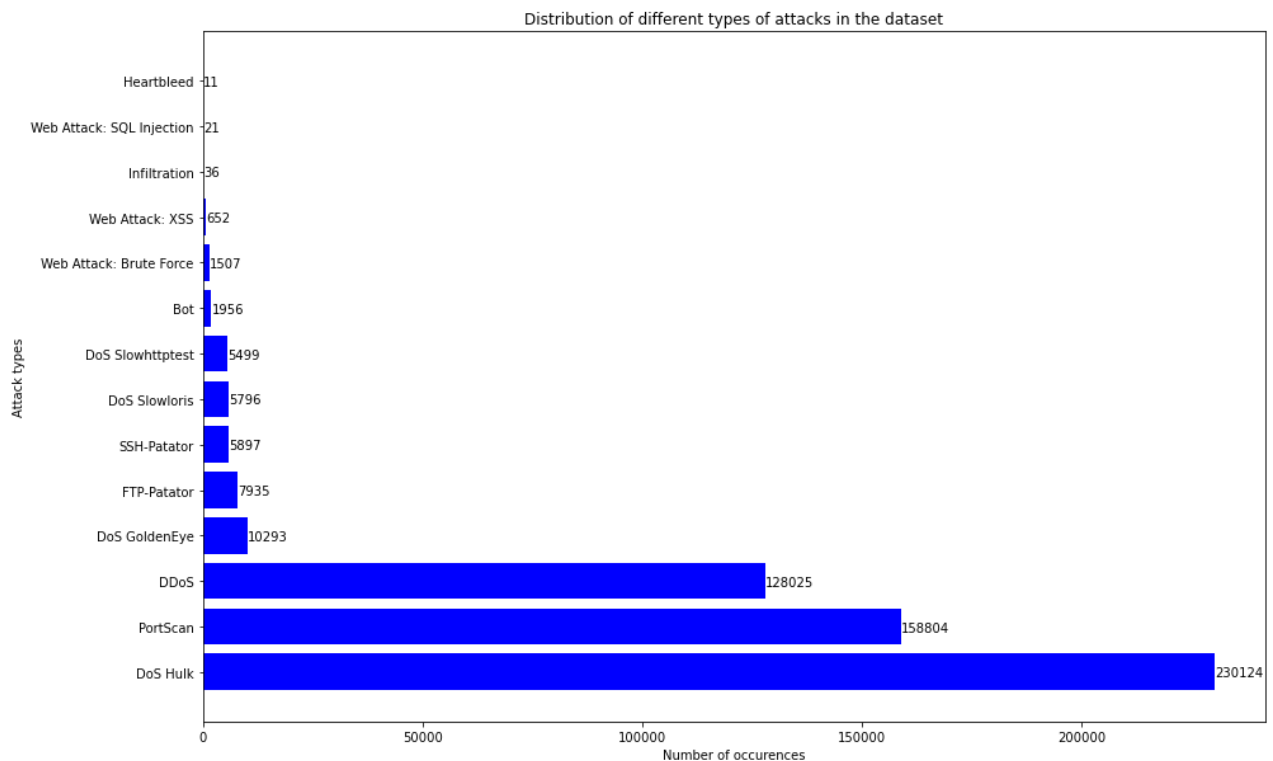
```
Out[13]: BENIGN                2271320
DoS Hulk                    230124
PortScan                   158804
DDoS                       128025
DoS GoldenEye              10293
FTP-Patator                 7935
SSH-Patator                 5897
DoS slowloris              5796
DoS Slowhttptest           5499
Bot                        1956
Web Attack  Brute Force    1507
Web Attack  XSS             652
Infiltration                36
Web Attack  Sql Injection   21
Heartbleed                  11
Name: Label, dtype: int64
```

```
In [14]: # Plotting the distribution of attacks in the dataset
```

```
plt.figure(figsize=(15,10))

attack = ('DoS Hulk', 'PortScan', 'DDoS', 'DoS GoldenEye', 'FTP-Patator', 'SSH-Patator',
          'DoS Slowhttptest', 'Bot', 'Web Attack: Brute Force', 'Web Attack: XSS', 'Inf
y_pos = np.arange(len(attack))
amount = dataset['Label'].value_counts()[1:]
plt.barh(y_pos, amount, align='center', color='blue' )
plt.yticks(y_pos, attack)
plt.title('Distribution of different types of attacks in the dataset')
plt.xlabel('Number of occurrences')
plt.ylabel('Attack types')
for i, v in enumerate(amount):
    plt.text(v + 3, i-0.1, str(v))

plt.show()
```



```
In [15]: # There are only 11, 21, and 36 instances of Heartbleed, SQL injection and infiltration
# Remove 'Heartbleed', 'Web attack Sql Injection', 'Infiltration' as it's negligible.

dataset = dataset.replace(['Heartbleed', 'Web Attack ⚠ Sql Injection', 'Infiltration'])
dataset = dataset.dropna()
dataset['Label'].value_counts()
```

```
Out[15]: BENIGN                2271320
DoS Hulk                230124
PortScan                158804
DDoS                    128025
DoS GoldenEye           10293
FTP-Patator              7935
SSH-Patator              5897
DoS slowloris           5796
DoS Slowhttptest        5499
Bot                      1956
Web Attack ⚠ Brute Force    1507
```



Web Attack ⚡ XSS  
 Name: Label, dtype: int64

652

```
In [16]: # Labelling Web Attack ⚡ Brute Force as Brute Force
# Labelling Web Attack ⚡ XSS as XSS

dataset.loc[dataset.Label == 'Web Attack ⚡ Brute Force', ['Label']] = 'Brute Force'
dataset.loc[dataset.Label == 'Web Attack ⚡ XSS', ['Label']] = 'XSS'

In [17]: # Creating a attack column, containing binary labels for normal and attack to apply bin

dataset['Attack'] = np.where(dataset['Label'] == 'BENIGN', 'Normal' , 'Attack')

In [18]: # Grouping attack labels in attack category as in dataset description for multi-class c

attack_group = {'BENIGN': 'benign',
                'DoS Hulk': 'dos',
                'PortScan': 'probe',
                'DDoS': 'ddos',
                'DoS GoldenEye': 'dos',
                'FTP-Patator': 'brute_force',
                'SSH-Patator': 'brute_force',
                'DoS slowloris': 'dos',
                'DoS Slowhttptest': 'dos',
                'Bot': 'botnet',
                'Brute Force': 'web_attack',
                'XSS': 'web_attack'}

# Create grouped label column

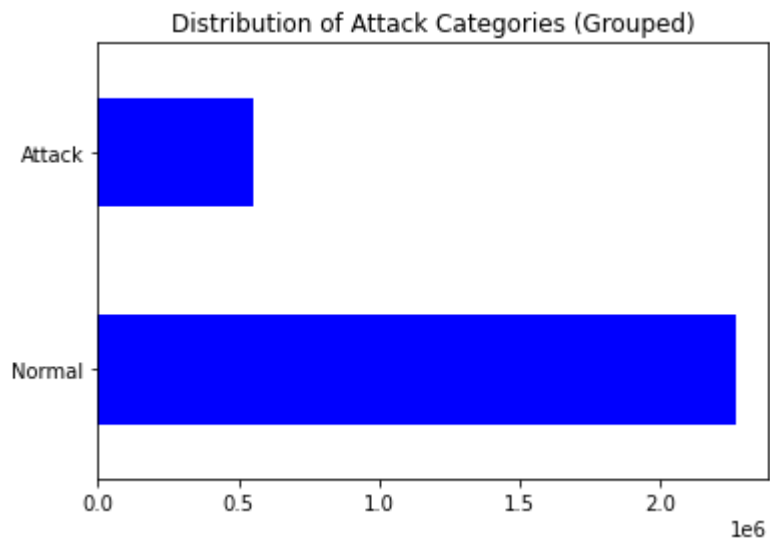
dataset['Label_Category'] = dataset['Label'].map(lambda x: attack_group[x])
dataset['Label_Category'].value_counts()

Out[18]: benign      2271320
dos      251712
probe    158804
ddos     128025
brute_force  13832
web_attack  2159
botnet    1956
Name: Label_Category, dtype: int64

In [19]: # Plotting binary grouped column Attack

train_attacks = dataset['Attack'].value_counts()
train_attacks.plot(kind='barh', color='blue')
plt.title('Distribution of Attack Categories (Grouped)')

Out[19]: Text(0.5, 1.0, 'Distribution of Attack Categories (Grouped)')
```

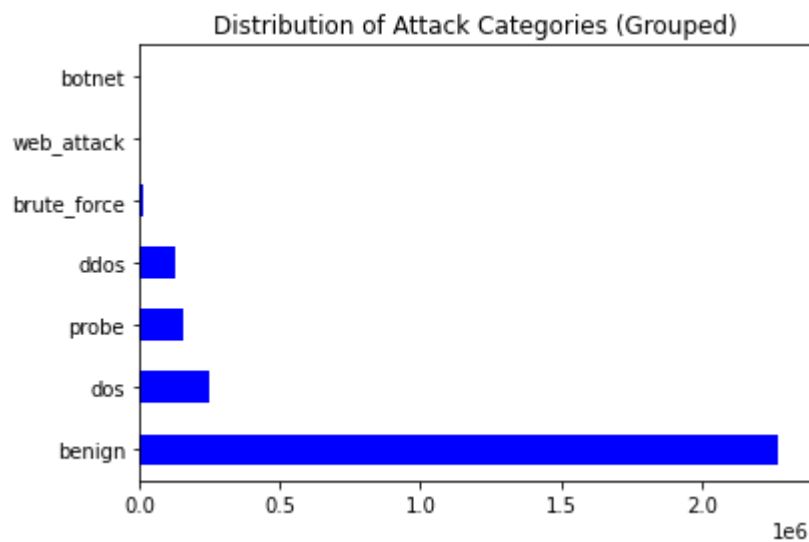


In [20]:

```
# Plotting multi-class grouped column Label_Category

train_attacks = dataset['Label_Category'].value_counts()
train_attacks.plot(kind='barh', color='blue')
plt.title('Distribution of Attack Categories (Grouped)')
```

Out[20]: Text(0.5, 1.0, 'Distribution of Attack Categories (Grouped)')

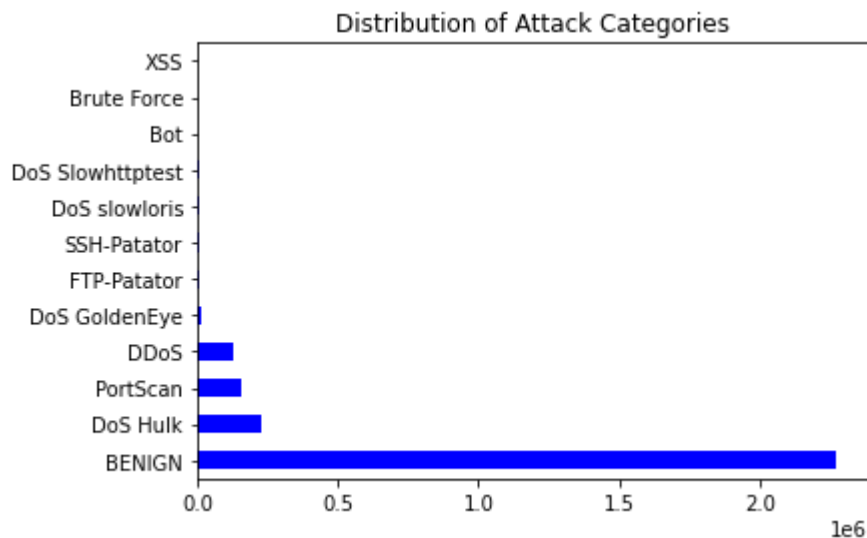


In [21]:

```
# Plotting multi-label column Label

train_attacks = dataset['Label'].value_counts()
train_attacks.plot(kind='barh', color='blue')
plt.title('Distribution of Attack Categories')
```

Out[21]: Text(0.5, 1.0, 'Distribution of Attack Categories')



```
In [22]: print('Total number of all attack classes :', len(dataset.Label.unique()))
print('Total number of attack categories :', len(dataset.Label_Category.unique()))
```

Total number of all attack classes : 12  
Total number of attack categories : 7

## Splitting the dataset

Splitting dataset in 60:20:20 ratio, for training, testing and validation dataset. By stratifying with y label proportions of attacks remain the same throughout the 3 sets.

```
In [23]: # 3 Different Labeling options
attacks = ['Label', 'Label_Category', 'Attack']

# xs=feature vectors, ys=labels
xs = dataset.drop(attacks, axis=1)
ys = dataset[attacks]

# split dataset - stratified
x_train, x_test, y_train, y_test = train_test_split(xs, ys, test_size=0.3, random_state
```

## Removing the columns with single unique values as it has no contribution in classification

```
In [24]: column_names = np.array(list(x_train))
to_drop = []
for x in column_names:
    size = x_train.groupby([x]).size()
    # check for columns that only take one value
    if (len(size.unique()) == 1):
        to_drop.append(x)
to_drop
```

```
Out[24]: ['Fwd_URG_Flags',
          'Fwd_Header_Length',
```

```
'Fwd_Avg_Bytes_Bulk',  
'Fwd_Avg_Packets_Bulk',  
'Fwd_Avg_Bulk_Rate',  
'Bwd_Avg_Bytes_Bulk',  
'Bwd_Avg_Packets_Bulk',  
'Bwd_Avg_Bulk_Rate']
```

```
In [25]: x_train = x_train.drop(to_drop, axis=1)  
x_test = x_test.drop(to_drop, axis=1)  
dataset_copy = dataset.drop(to_drop, axis=1)
```

```
In [26]: x_train.shape
```

```
Out[26]: (1979465, 69)
```

## Data Normalization

Min-max normalization technique is used to normalize the numerical values in dataset.

```
In [27]: # Normalise  
min_max_scaler = MinMaxScaler().fit(x_train)  
  
# Apply normalisation to dataset  
x_train = min_max_scaler.transform(x_train)  
x_test = min_max_scaler.transform(x_test)
```

## Feature Selection

Selecting K-best features by using chi2 scoring function for features

```
In [28]: features = SelectKBest(score_func=chi2, k=x_train.shape[1])  
  
#fit features to the training dataset  
fit = features.fit(x_train, y_train.Label)
```

```
In [29]: # perform selectkbest with k=40  
  
features = SelectKBest(score_func=chi2, k=40)  
fit = features.fit(x_train, y_train.Label)  
  
x_train = fit.transform(x_train)  
x_test = fit.transform(x_test)
```

```
In [30]: new_features = dataset_copy.columns[features.get_support(indices=True)]
```

```
In [31]: print('Number of features selected : ', len(new_features))  
new_features
```

Number of features selected : 40

```
Out[31]: Index(['Destination_Port', 'Flow_Duration', 'Total_Fwd_Packets',
               'Bwd_Packet_Length_Min', 'Bwd_Packet_Length_Mean',
               'Bwd_Packet_Length_Std', 'Flow_Bytes_s', 'Flow_IAT_Std', 'Flow_IAT_Max',
               'Flow_IAT_Min', 'Fwd_IAT_Total', 'Fwd_IAT_Mean', 'Fwd_IAT_Std',
               'Fwd_IAT_Max', 'Fwd_IAT_Min', 'Bwd_IAT_Total', 'Bwd_IAT_Mean',
               'Bwd_IAT_Std', 'Bwd_IAT_Max', 'Bwd_IAT_Min', 'Fwd_PSH_Flags',
               'Bwd_PSH_Flags', 'Bwd_Packets_s', 'Packet_Length_Mean',
               'Packet_Length_Std', 'Packet_Length_Variance', 'FIN_Flag_Count',
               'SYN_Flag_Count', 'RST_Flag_Count', 'ACK_Flag_Count', 'URG_Flag_Count',
               'CWE_Flag_Count', 'Avg_Fwd_Segment_Size', 'Init_Win_bytes_forward',
               'Init_Win_bytes_backward', 'Active_Min', 'Idle_Mean', 'Idle_Std',
               'Idle_Max', 'Idle_Min'],
              dtype='object')
```

```
In [32]: attack = np.array(['BENIGN', 'Bot', 'Brute Force', 'DDoS', 'DoS GoldenEye', 'DoS Hulk',
                           'DoS slowloris', 'FTP-Patator', 'PortScan', 'SSH-Patator', 'XSS'])
        attack_groups = np.array(['benign', 'botnet', 'brute_force', 'ddos', 'dos', 'probe', 'w
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

## Applying Machine Learning classifier models

Each machine learning algorithm is applied in three different categories :

1. On all attack labels (12).
2. Binary Classifier (2).
3. Multi-class Classifier (7).

And then evaluate performance of each algorithm by confusion matrix plot. Evaluate Accuracy, Precision, Recall, F1-score.

## 3. Naive Bayes Classifier

```
In [33]: classifier = MultinomialNB()
```

1. a) On all attack labels.

```
In [34]: # fit the model

start = time.time()
classifier.fit(x_train, y_train.Label)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

Model Training Time is : 9.752284526824951

```
In [35]: # predicting test results of Naive Bayes classifier on all labels.

start = time.time()
y_predict = classifier.predict(x_test)
end = time.time()
testing_time = end - start

print("Model Testing Time is : ", testing_time)
```

Model Testing Time is : 0.12589097023010254

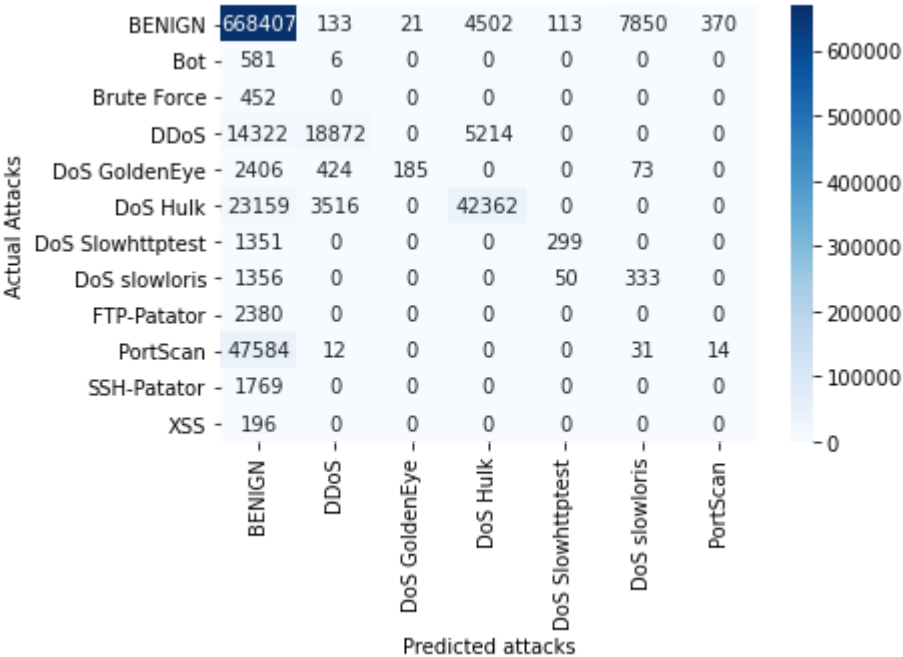
```
In [36]: # Creating confusion matrix for Naive Bayes classifier on all labels.

confusion_nb_1 = pd.crosstab(y_test.Label, y_predict, rownames=['Actual Attacks'], coln

print("Plotting Confusion Matrix of Naive Bayes classifier on all Labels ")

sn.heatmap(confusion_nb_1, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_nb_1
```

Plotting Confusion Matrix of Naive Bayes classifier on all Labels



Out[36]:

Predicted attacks	BENIGN	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	PortScan
Actual Attacks							
BENIGN	668407	133	21	4502	113	7850	370
Bot	581	6	0	0	0	0	0
Brute Force	452	0	0	0	0	0	0
DDoS	14322	18872	0	5214	0	0	0
DoS GoldenEye	2406	424	185	0	0	73	0

Predicted attacks	BENIGN	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS slowloris	PortScan
<b>Actual Attacks</b>							
<b>DoS Hulk</b>	23159	3516	0	42362	0	0	0
<b>DoS Slowhttptest</b>	1351	0	0	0	299	0	0
<b>DoS slowloris</b>	1356	0	0	0	50	333	0
<b>FTP-Patator</b>	2380	0	0	0	0	0	0
<b>PortScan</b>	47584	12	0	0	0	31	14
<b>SSH-Patator</b>	1769	0	0	0	0	0	0
<b>XSS</b>	196	0	0	0	0	0	0

In [37]:

```
# Precision,Recall,F1-score for Naive Bayes classifier on all labels.
```

```
precision, recall, fscore, support = score(y_test.Label, y_predict)
```

```
d = {'attack': attack, 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))

Out[37]:

	attack	precision	recall	fscore
0	BENIGN	0.874921	0.980938	0.924901
1	Bot	0.000000	0.000000	0.000000
2	Brute Force	0.000000	0.000000	0.000000
3	DDoS	0.821844	0.491356	0.615014
4	DoS GoldenEye	0.898058	0.059909	0.112325
5	DoS Hulk	0.813434	0.613613	0.699534
6	DoS Slowhttptest	0.647186	0.181212	0.283144
7	DoS slowloris	0.040183	0.191489	0.066427
8	FTP-Patator	0.000000	0.000000	0.000000
9	PortScan	0.036458	0.000294	0.000583
10	SSH-Patator	0.000000	0.000000	0.000000
11	XSS	0.000000	0.000000	0.000000

In [38]:

```
# Average Accuracy,Precision,Recall,F1-score for Naive Bayes classifier on all labels.
```

```
precision_nb_1, recall_nb_1, fscore_nb_1, support = score(y_test.Label, y_predict, aver
```

```
accuracy_nb_1 = accuracy_score(y_test.Label, y_predict)
print("Accuracy of Naive Bayes classifier on all labels : ", accuracy_nb_1)
```

C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```
_warn_prf(average, modifier, msg_start, len(result))
Accuracy of Naive Bayes classifier on all labels : 0.8610573789139534
```

In [ ]:

In [ ]:

### 1. b) Binary Classifier.

In [39]:

```
# fit the model

start = time.time()
classifier.fit(x_train, y_train.Attack)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)
```

Model Training Time is : 18.767290830612183

In [40]:

```
# predicting test results of Naive Bayes classifier on binary labels.

start = time.time()
y_predict = classifier.predict(x_test)
end = time.time()
testing_time = end - start

print("Model Testing Time is : ", testing_time)
```

Model Testing Time is : 0.0980687141418457

In [41]:

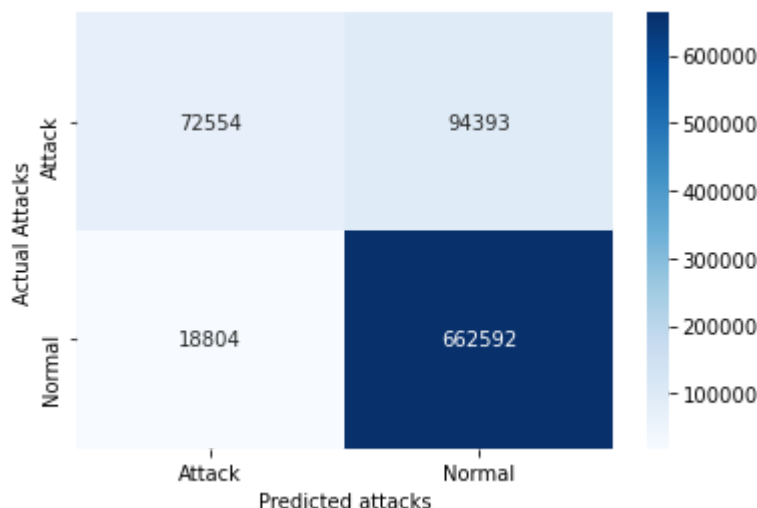
```
# Creating confusion matrix for Naive Bayes classifier on binary labels.

confusion_nb_2 = pd.crosstab(y_test.Attack, y_predict, rownames=['Actual Attacks'], col
print("Plotting Confusion Matrix of Naive Bayes classifier on binary Labels ")

sn.heatmap(confusion_nb_2, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_nb_2
```

Plotting Confusion Matrix of Naive Bayes classifier on binary Labels





Out[41]: **Predicted attacks**   **Attack**   **Normal**

Actual Attacks		
	Attack	Normal
Attack	72554	94393
Normal	18804	662592

```
In [42]: # Precision,Recall,F1-score for Naive Bayes classifier on binary Labels.

precision, recall, fscore, support = score(y_test.Attack, y_predict)
d = {'attack': [0,1], 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

Out[42]:

	attack	precision	recall	fscore
0	0	0.794172	0.434593	0.561770
1	1	0.875304	0.972404	0.921302

```
In [43]: # Average Accuracy,Precision,Recall,F1-score for Naive Bayes classifier on binary Label

precision_nb_2, recall_nb_2, fscore_nb_2, n = score(y_test.Attack, y_predict, average='
accuracy_nb_2 = accuracy_score(y_test.Attack, y_predict)
print("Accuracy of Naive Bayes classifier on binary labels : ", accuracy_nb_2)
```

Accuracy of Naive Bayes classifier on binary labels : 0.8665669428521247

In [ ]:

In [ ]:

1. c) Multi-class Classifier.

```
In [44]: # fit the model
```

```

start = time.time()
classifier.fit(x_train, y_train.Label_Category)
end = time.time()
training_time = end - start

print("Model Training Time is : ", training_time)

```

Model Training Time is : 5.63388204574585

In [45]: *# predicting test results of Naive Bayes classifier on multi-class Labels.*

```

start = time.time()
y_predict = classifier.predict(x_test)
end = time.time()
testing_time = end - start

print("Model Testing Time is : ", testing_time)

```

Model Testing Time is : 0.11755704879760742

In [46]: *# Creating confusion matrix for Naive Bayes classifier on multi-class Labels.*

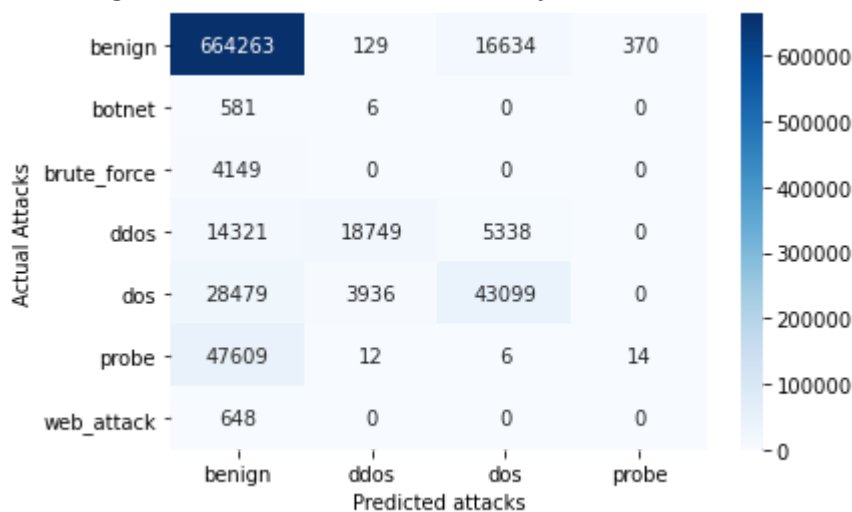
```

confusion_nb_3 = pd.crosstab(y_test.Label_Category, y_predict, rownames=['Actual Attack'])
print("Plotting Confusion Matrix of Naive Bayes classifier on multi-class Labels ")

sn.heatmap(confusion_nb_3, annot=True, cmap= 'Blues', fmt='d')
plt.show()
confusion_nb_3

```

Plotting Confusion Matrix of Naive Bayes classifier on multi-class Labels



Out[46]:

	Predicted attacks	benign	ddos	dos	probe
Actual Attacks					

Actual Attacks	benign	ddos	dos	probe
benign	664263	129	16634	370
botnet	581	6	0	0
brute_force	4149	0	0	0
ddos	14321	18749	5338	0

**Predicted attacks**   **benign**   **ddos**   **dos**   **probe**

**Actual Attacks**

<b>dos</b>	28479	3936	43099	0
<b>probe</b>	47609	12	6	14
<b>web_attack</b>	648	0	0	0

In [47]:

```
# Precision,Recall,F1-score for Naive Bayes classifier on multi-class labels.
```

```
precision, recall, fscore, support = score(y_test.Label_Category, y_predict)
d = {'attack': attack_groups, 'precision': precision, 'recall' : recall, 'fscore': fscore}
results = pd.DataFrame(data=d)
results
```

C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
\_warn\_prf(average, modifier, msg\_start, len(result))

Out[47]:

	<b>attack</b>	<b>precision</b>	<b>recall</b>	<b>fscore</b>
<b>0</b>	benign	0.873973	0.974856	0.921662
<b>1</b>	botnet	0.000000	0.000000	0.000000
<b>2</b>	brute_force	0.000000	0.000000	0.000000
<b>3</b>	ddos	0.821172	0.488154	0.612312
<b>4</b>	dos	0.662277	0.570742	0.613112
<b>5</b>	probe	0.036458	0.000294	0.000583
<b>6</b>	web_attack	0.000000	0.000000	0.000000

In [48]:

```
# Average Accuracy,Precision,Recall,F1-score for Naive Bayes classifier on multi-class
```

```
precision_nb_3, recall_nb_3, fscore_nb_3, n = score(y_test.Label_Category, y_predict, average='weighted')
accuracy_nb_3 = accuracy_score(y_test.Label_Category, y_predict)
print("Accuracy of Naive Bayes classifier on multi-class labels : ", accuracy_nb_3)
```

C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
\_warn\_prf(average, modifier, msg\_start, len(result))  
Accuracy of Naive Bayes classifier on multi-class labels : 0.8559332722731253

In [ ]:

## Results for Naive Bayes:

In [49]:

```
print('Naive Bayes: Precision / Recall / Fscore / Accuracy')

print('All Labels:', precision_nb_1, recall_nb_1, fscore_nb_1, accuracy_nb_1)
```

```
print('Binary Labels:', precision_nb_2, recall_nb_2, fscore_nb_2, accuracy_nb_2)
print('Multi-class Labels:', precision_nb_3, recall_nb_3, fscore_nb_3, accuracy_nb_3)
```

Naive Bayes: Precision / Recall / Fscore / Accuracy

All Labels: 0.3443403606808013 0.20990094123148562 0.22516065133780527 0.8610573789139534

Binary Labels: 0.834738189432076 0.7034983634641163 0.7415362463788386 0.8665669428521247

Multi-class Labels: 0.34198287606723227 0.2905778934659667 0.3068098620024685 0.8559332722731253

In [ ]: