

## **Problem Description:**

The competition task is to build a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, called as intrusions or attacks, and "good" or normal connections. The dataset includes a wide variety of intrusions simulated in a military network environment.

## **What is an INTRUSION DETECTOR?**

Intrusion detector is a software used to detect network intrusions. It protects a computer network from unauthorized users, including perhaps insiders.

## **How the data was collected?**

The 1998 DARPA Intrusion Detection Evaluation Program was prepared and managed by MIT Lincoln Labs. The objective was to survey and evaluate research in intrusion detection. A standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment, was provided. The 1999 KDD intrusion detection contest uses a version of this dataset.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks.

The raw training data was about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was processed into about five million connection records. Similarly, the two weeks of test data yielded around two million connection records.

A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

## **Different Categories of the Attacks:-**

### ***Denial-of-service(DoS) :-***

A Denial-of-Service (DoS) attack is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash.e.g. syn flood;

### ***Remote 2 Local(R2L) attack:-***

Remote to local attack (r2l) has been widely known to be launched by an attacker to gain unauthorized access to a victim machine in the entire network.

### ***User to root attack (U2R) attack:-***

This attack is usually launched for illegally obtaining the root's privileges when legally accessing a local machine. , e.g. guessing password, various "buffer overflow" attacks;

### ***Probing:-***

Probing is an attack in which the hacker scans a machine or a networking device in order to determine weaknesses or vulnerabilities that may later be exploited so as to. compromise the system. e.g., port scanning.

## **References:**

- For determining the performance metric and classifiers used, download PDF:-  
[https://www.researchgate.net/publication/309038723\\_A\\_review\\_of\\_KDD99\\_dataset\\_usage\\_in\\_intrusion\\_detection\\_and\\_machine\\_learning](https://www.researchgate.net/publication/309038723_A_review_of_KDD99_dataset_usage_in_intrusion_detection_and_machine_learning)
- <https://arxiv.org/pdf/1811.05372.pdf>

## **Dataset and Task Description:-**

## Importing necessary Libraries and importing dataset:-

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import MiniBatchKMeans
```

In [3]:

```
features = ['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creations','num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','intrusion_type']
```

In [4]:

```
data = pd.read_csv('kddcup.data_10_percent_corrected', names=features, header=None)
data.head()
```

Out[4]:

```
duration protocol_type service flag src_bytes dst_bytes land wrong_fragment urgent hot ... dst_host_srv_count dst_host_s
```

0	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_s
1	0	tcp	http	SF	239	486	0	0	0	0	...	19	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	29	
3	0	tcp	http	SF	219	1337	0	0	0	0	...	39	
4	0	tcp	http	SF	217	2032	0	0	0	0	...	49	

5 rows × 42 columns

In [5]:

```
print('The no of data points are:',data.shape[0])
print('='*40)
print('The no of features are:',data.shape[1])
print('='*40)
print('Some of the features are:',features[:10])
```

The no of data points are: 494021

=====

The no of features are: 42

=====

Some of the features are: ['duration', 'protocol\_type', 'service', 'flag', 'src\_bytes', 'dst\_bytes', 'land', 'wrong\_fragment', 'urgent', 'hot']

In [6]:

```
output = data['intrusion_type'].values
labels = set(output)
```

In [7]:

```
print('The different type of output labels are:',labels)
print('='*100)
print('No. of different output labels are:', len(labels))
```

The different type of output labels are: {'rootkit.', 'perl.', 'buffer\_overflow.', 'pod.', 'loadmodule.', 'warezclient.', 'smurf.', 'ftp\_write.', 'back.', 'imap.', 'neptune.', 'land.', 'phf.', 'portsweep.', 'spy.', 'ipsweep.', 'satan.', 'normal.', 'guess\_passwd.', 'warezmaster.', 'nmap.', 'teardrop.', 'multihop.'}

=====

No. of different output labels are: 23

## Data Cleaning:-

### Checking for NULL values:-

In [8]:

```
print('Null values in the dataset are: ',len(data[data.isnull().any(1)]))
```

Null values in the dataset are: 0

### Checking for DUPLICATE values:-

In [21]:

```
duplicateRowsDF = data[data.duplicated()]
```

In [22]:

```
duplicateRowsDF.head(5)
```

Out[22]:

duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_hos
2500	0	tcp	http	SF	307	60990	0	0	0	0	...	255
2864	0	tcp	http	SF	351	11485	0	0	0	0	...	255
2977	0	tcp	http	SF	506	2419	0	0	0	0	...	255
3003	0	tcp	http	SF	319	1374	0	0	0	0	...	255
3130	0	tcp	http	SF	234	1212	0	0	0	0	...	255

5 rows × 42 columns

In [9]:

```
data.drop_duplicates(subset=features, keep='first', inplace=True)
data.shape
```

Out[9]:

```
(145586, 42)
```

In [24]:

```
data.to_pickle('data.pkl')
data = pd.read_pickle('data.pkl')
```

## Exploratory Data Analysis:-

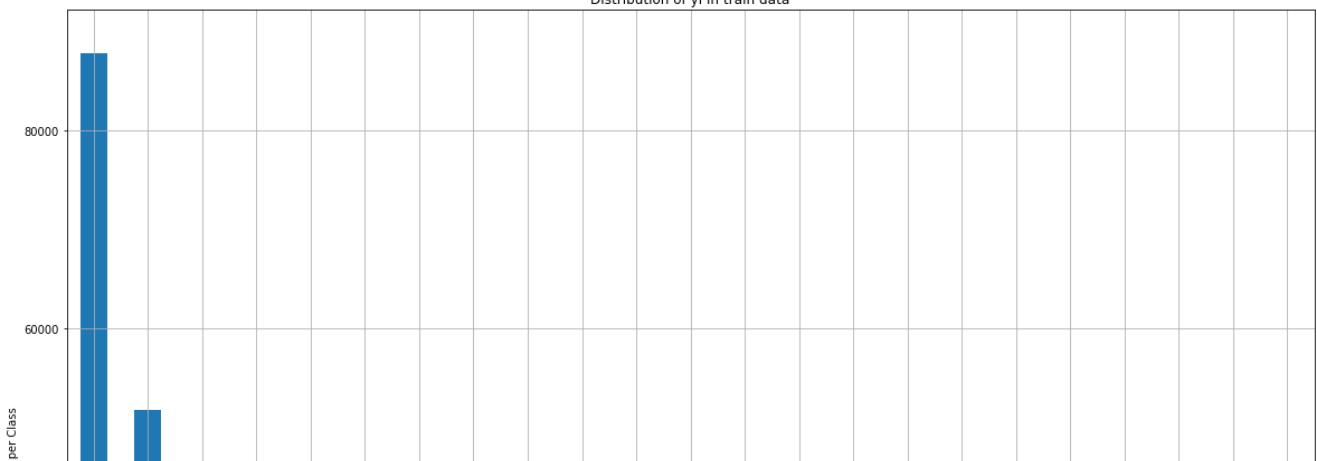
### Distribution of categories in class label:-

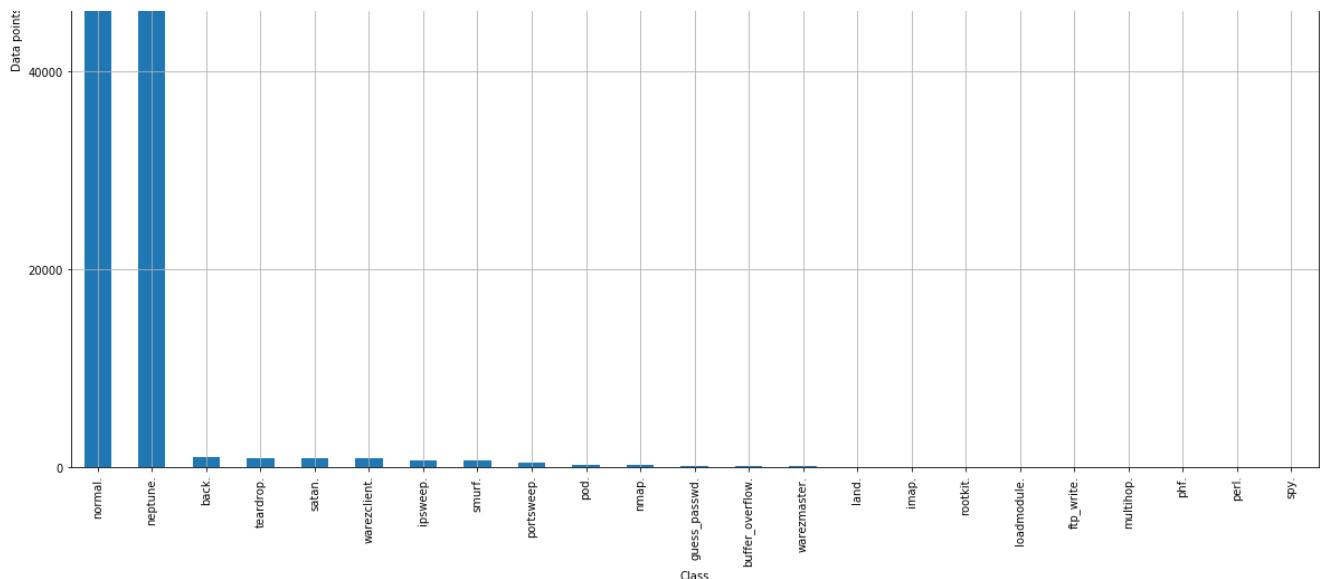
In [14]:

```
plt.figure(figsize=(20,15))
class_distribution = data['intrusion_type'].value_counts()
class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', class_distribution.values[i], '(', np.round((class_distribution.values[i]/data.shape[0]*100), 3), '%)')
```

Distribution of yi in train data





Number of data points in class 1 : 87832 ( 60.33 %)  
 Number of data points in class 2 : 51820 ( 35.594 %)  
 Number of data points in class 3 : 968 ( 0.665 %)  
 Number of data points in class 4 : 918 ( 0.631 %)  
 Number of data points in class 5 : 906 ( 0.622 %)  
 Number of data points in class 6 : 893 ( 0.613 %)  
 Number of data points in class 7 : 651 ( 0.447 %)  
 Number of data points in class 8 : 641 ( 0.44 %)  
 Number of data points in class 9 : 416 ( 0.286 %)  
 Number of data points in class 10 : 206 ( 0.141 %)  
 Number of data points in class 11 : 158 ( 0.109 %)  
 Number of data points in class 12 : 53 ( 0.036 %)  
 Number of data points in class 13 : 30 ( 0.021 %)  
 Number of data points in class 14 : 20 ( 0.014 %)  
 Number of data points in class 15 : 19 ( 0.013 %)  
 Number of data points in class 16 : 12 ( 0.008 %)  
 Number of data points in class 17 : 10 ( 0.007 %)  
 Number of data points in class 18 : 9 ( 0.006 %)  
 Number of data points in class 19 : 8 ( 0.005 %)  
 Number of data points in class 20 : 7 ( 0.005 %)  
 Number of data points in class 21 : 4 ( 0.003 %)  
 Number of data points in class 22 : 3 ( 0.002 %)  
 Number of data points in class 23 : 2 ( 0.001 %)

- Most of the data points are from "normal" (good connections) category which is around 60.33 %.
- In the categories that belong to bad connections, "neptune." (35.594 %) and "back." (0.665 %) have the highest no. of data points.
- Classes "rootkit.", "loadmodule.", "ftp\_write.", "multihop.", "phf.", "perl.", "spy.." have the least no. of data points with less than 10 data points per class.
- The dataset is highly imbalanced, thus we will need to build a model which should be able to classify data points from these low distribution classes accurately.

## Performance metrics for the problem:-

- As the dataset is highly imbalanced, we will need to build a model which should be able to classify the INTRUSION categories accurately.
- We will use the CONFUSION MATRIX as that will help us to determine how well the data points belonging to each of the 23 classes are classified.
- Along with the confusion matrix, we will also calculate precision, recall and weighted f1-score to determine the best model.
- Although not preferred for imbalanced datasets, but we will also display the accuracy score that will give us an estimate of the total no. of correctly classified points.
- Some important evaluation scores that can be added for this problem are the TPR (True Positive rate) and FPR (False Positive

Some important classification scores that can be used for this problem are F1 Score, Precision, Recall, and False Positive Rate (FPR) scores.

For this problem, we want our FPR to be as low as possible. This is because, a "Normal" connection getting dropped because of getting misclassified as a "Bad" connection is less severe compared to a "Bad" connection getting misclassified as a "Normal" connection, which may result in a security threat

## (I) MultiClass Classification Problem :-

### Univariate Analysis:-

#### 1. Duration:-

In [10]:

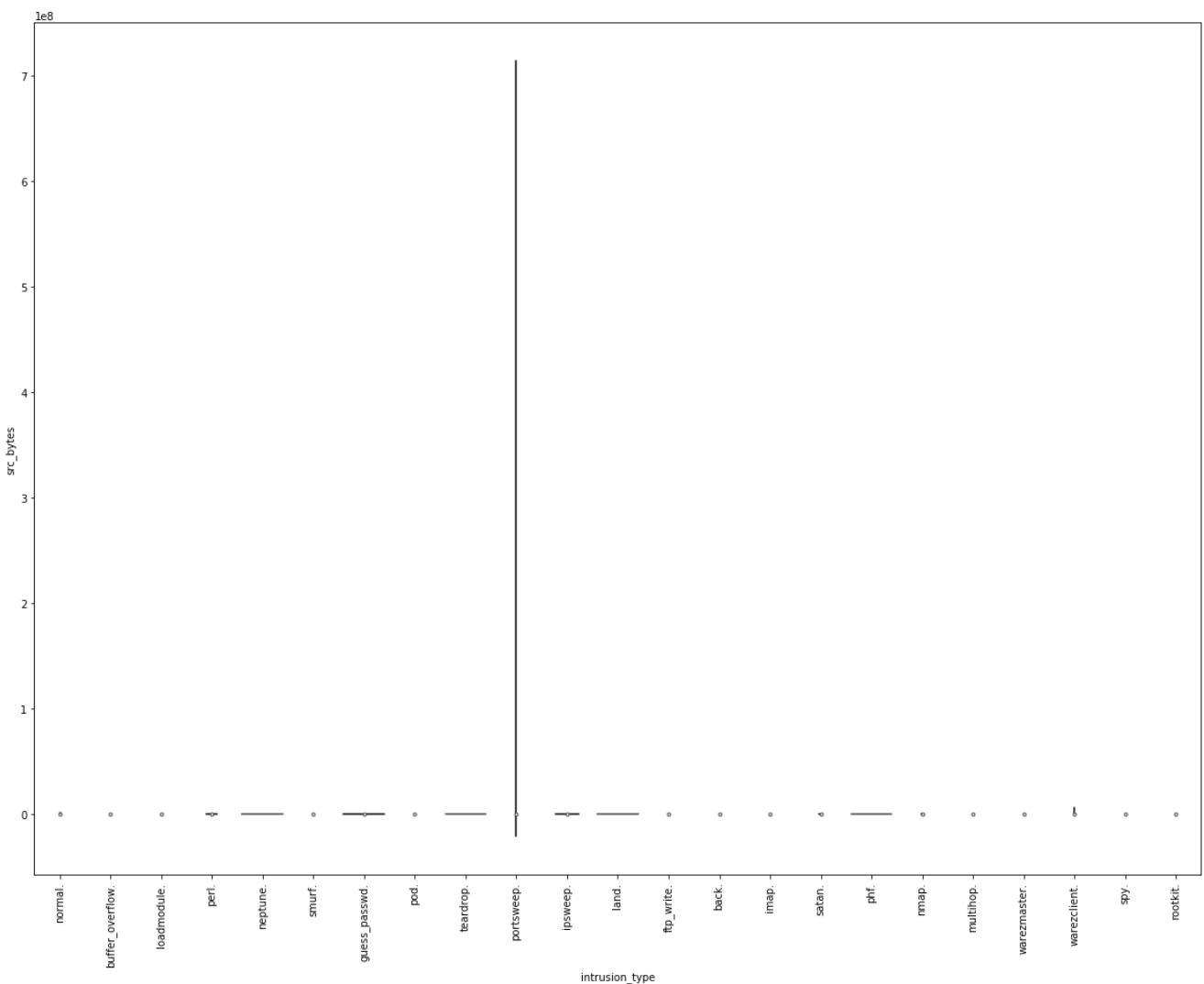
```
import seaborn as sns
```

In [13]:

```
plt.figure(figsize=(20,15))
sns.violinplot(x="intrusion_type", y="src_bytes", data=data)
plt.xticks(rotation=90)
```

Out[13] :

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22]), <a list of 23 Text xticklabel objects>)
```



- Using src\_bytes as a feature for analysis, the intrusion category "portsweep" is distinguishable from rest of the categories.
- As we have relatively high no of classes, the Univariate analysis using boxplots and violin plots do not give us clear and satisfactory results.
- Thus, we can go with pairplots for BiVariate Analysis or we can go with PCA/TSNE to reduce the no. of dimensions and perform Bi/Tri-Variate Analysis.

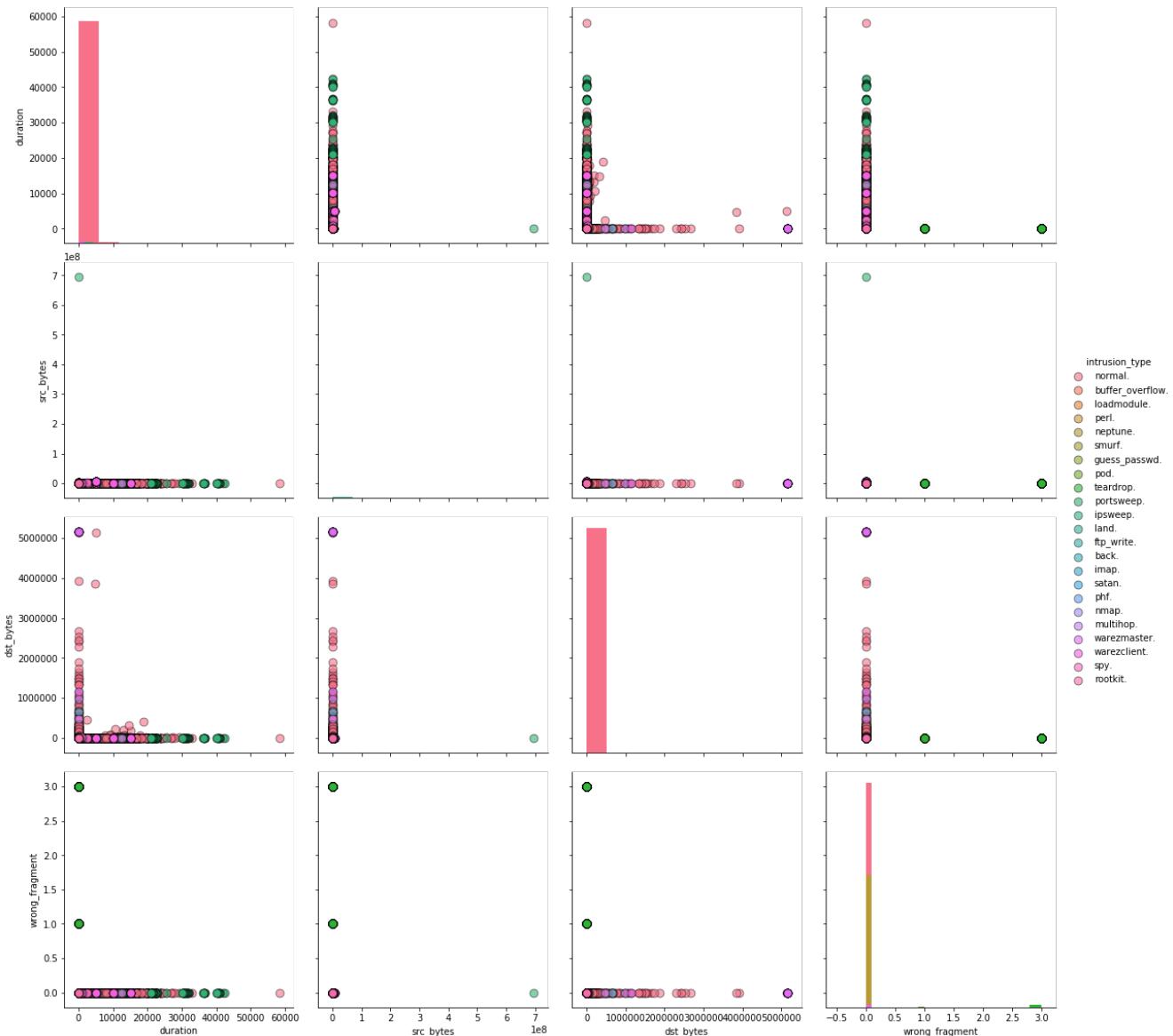
## Pair Plots for Bivariate Analysis:-

In [36]:

```
def pairplot(data, label, features=[]):
    """
    This function creates pairplot taking 4 features from our dataset as default parameters along
    with the output variable
    """
    sns.pairplot(data, hue=label, height=4, diag_kind='hist', vars=features,
                 plot_kws={'alpha':0.6, 's':80, 'edgecolor':'k'})
```

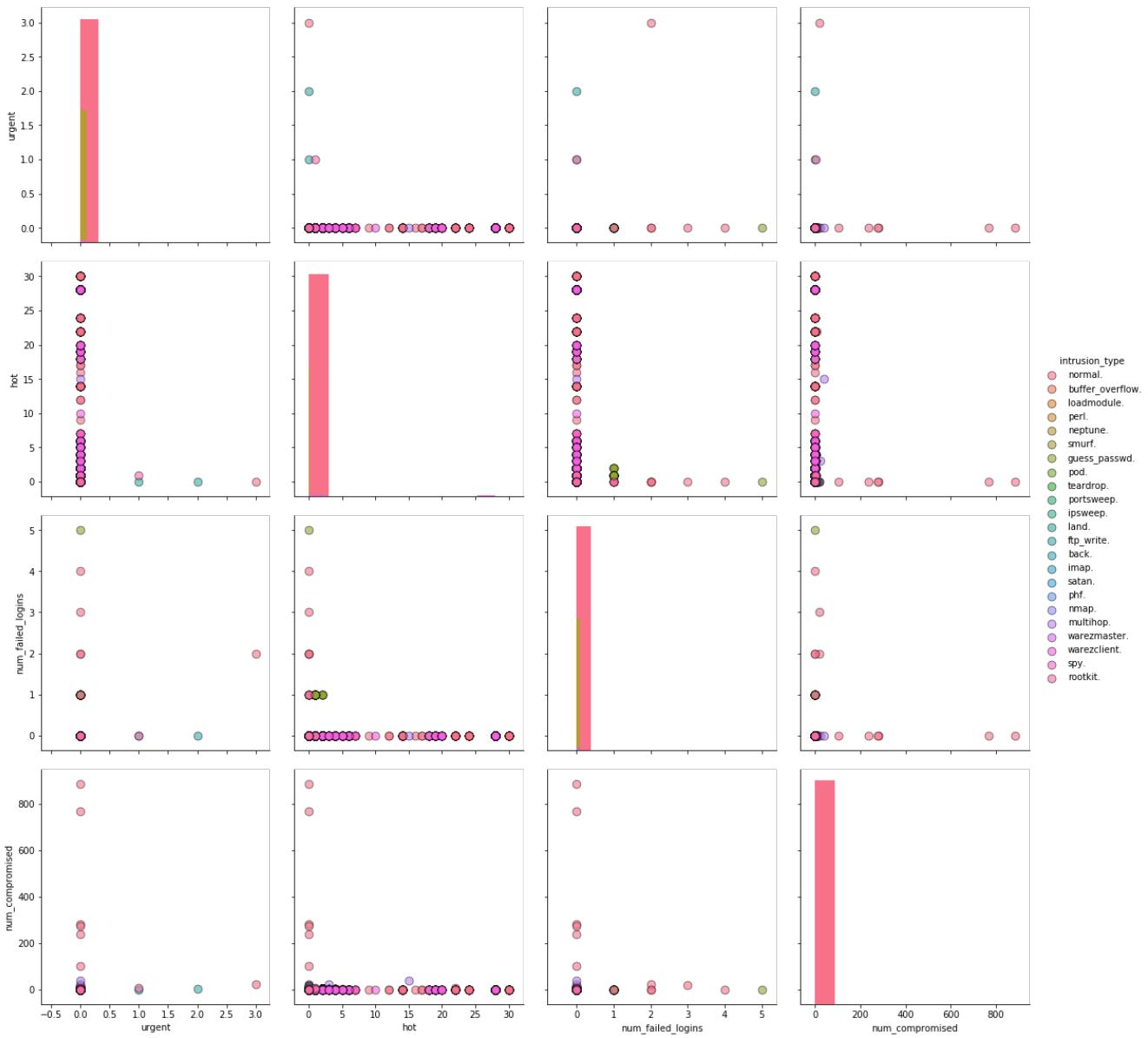
In [15]:

```
pairplot(data, 'intrusion_type', features=['duration', 'src_bytes', 'dst_bytes', 'wrong_fragment'])
```



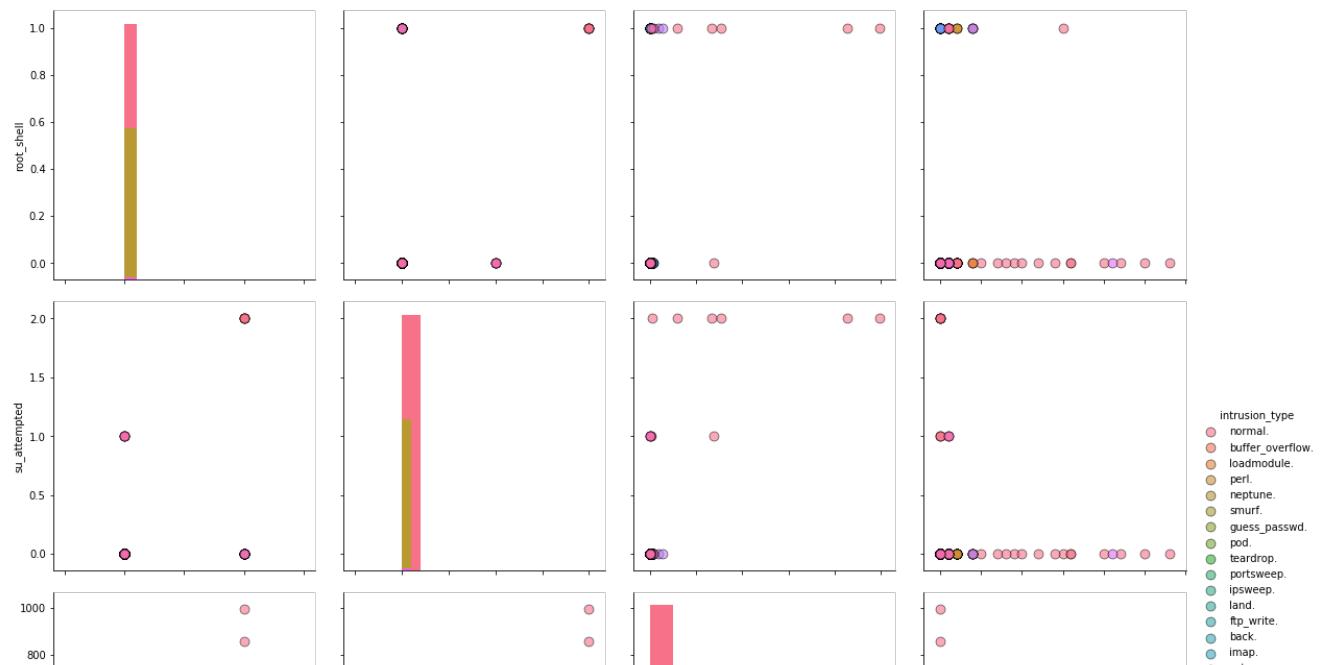
In [16]:

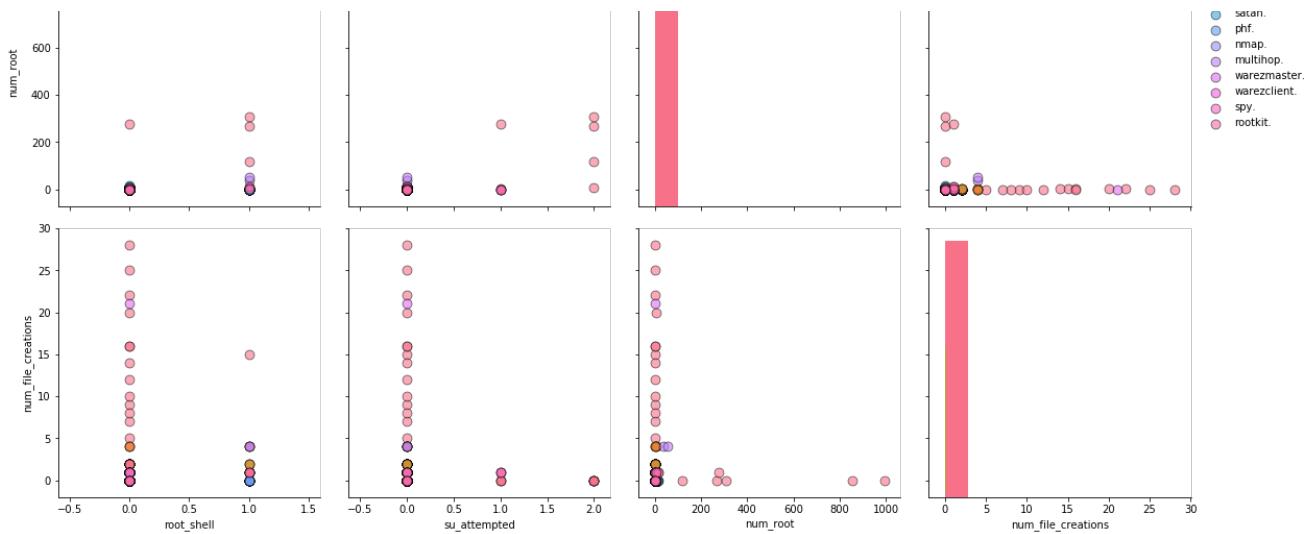
```
pairplot(data, 'intrusion_type', features=['urgent', 'hot', 'num_failed_logins', 'num_compromised'])
```



In [17]:

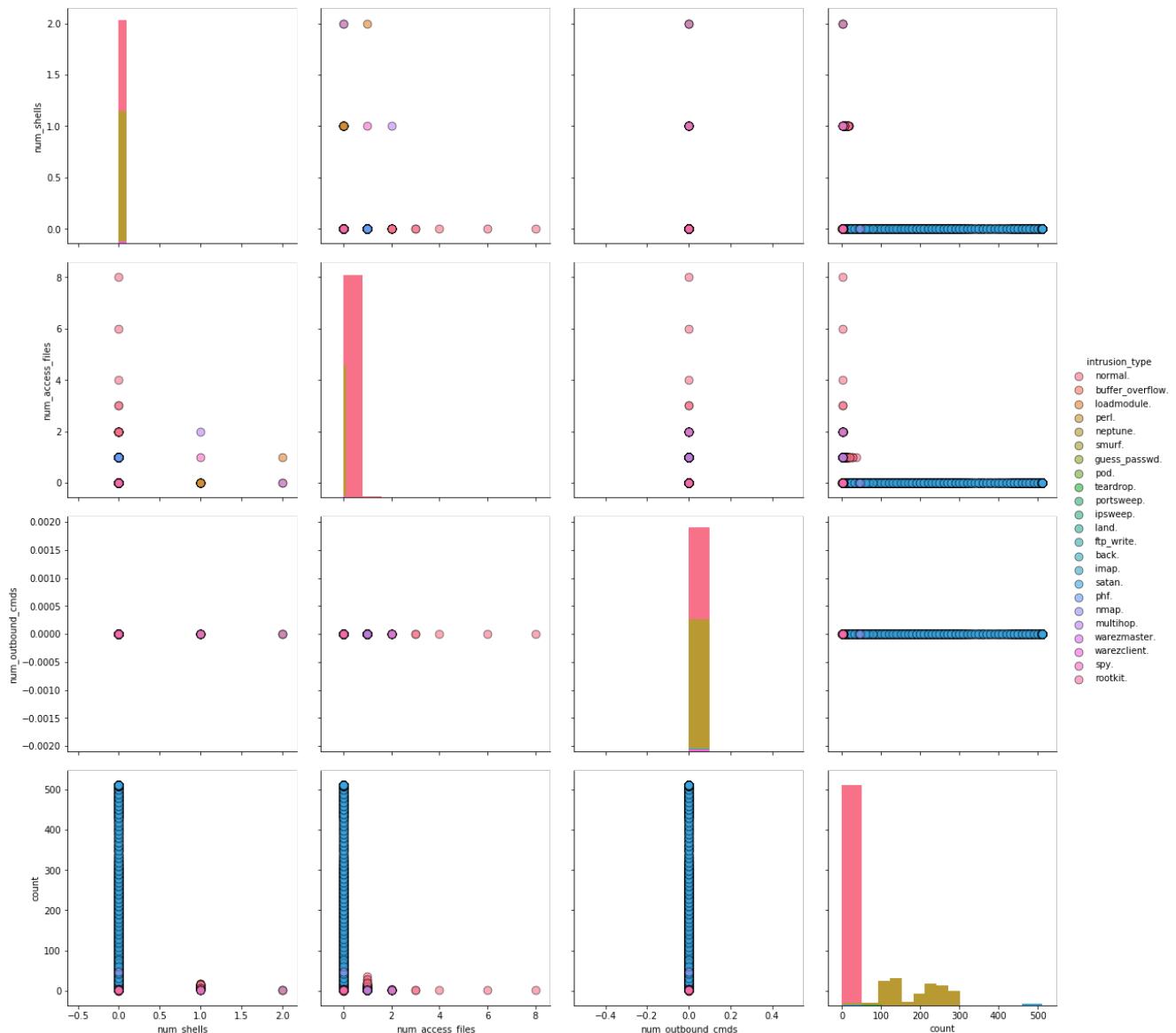
```
pairplot(data, 'intrusion_type',
features=['root_shell','su_attempted','num_root','num_file_creations'])
```





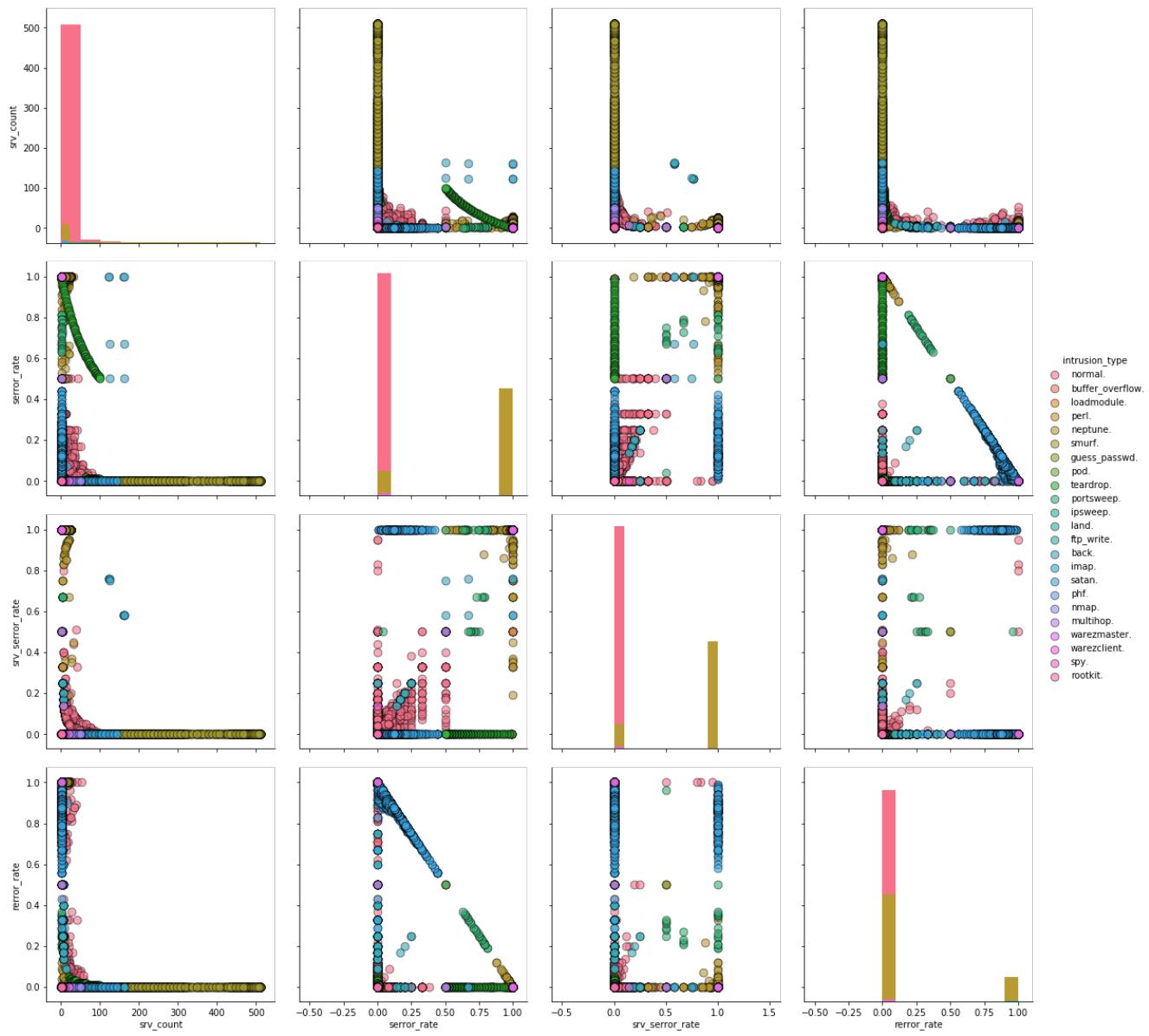
In [18]:

```
pairplot(data, 'intrusion_type',
features=['num_shells','num_access_files','num_outbound_cmds','count'])
```



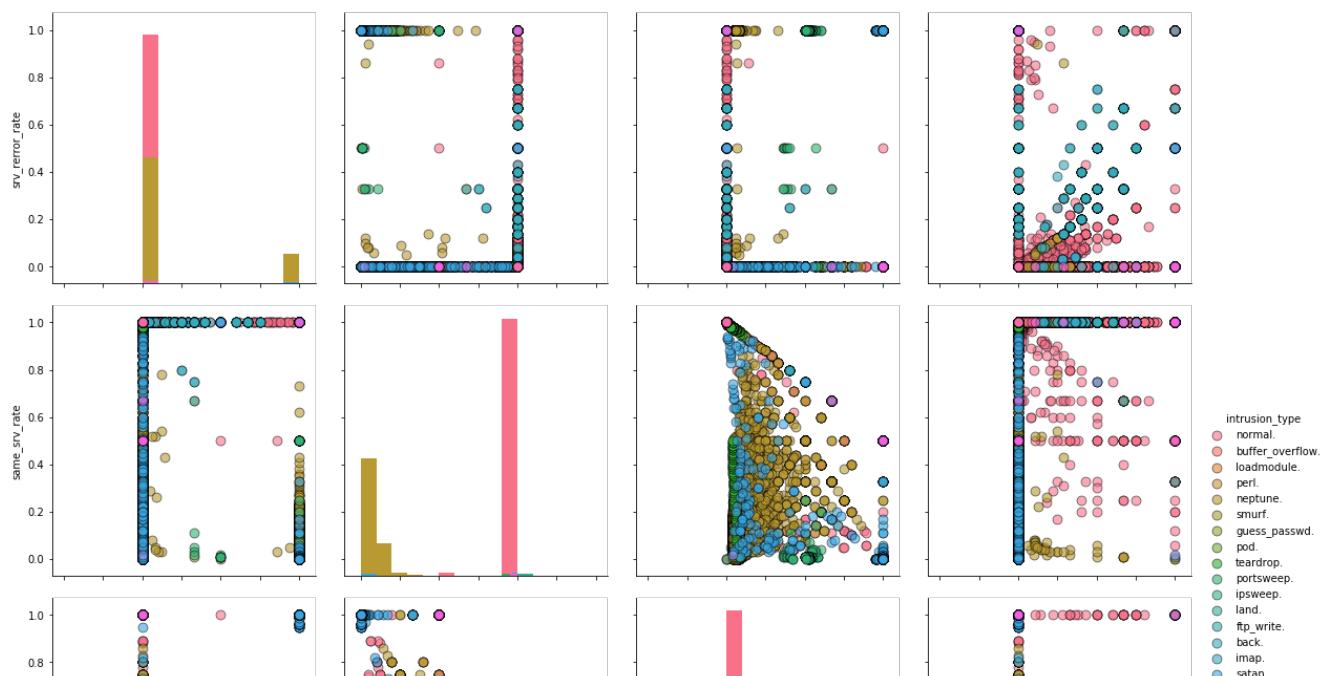
In [19]:

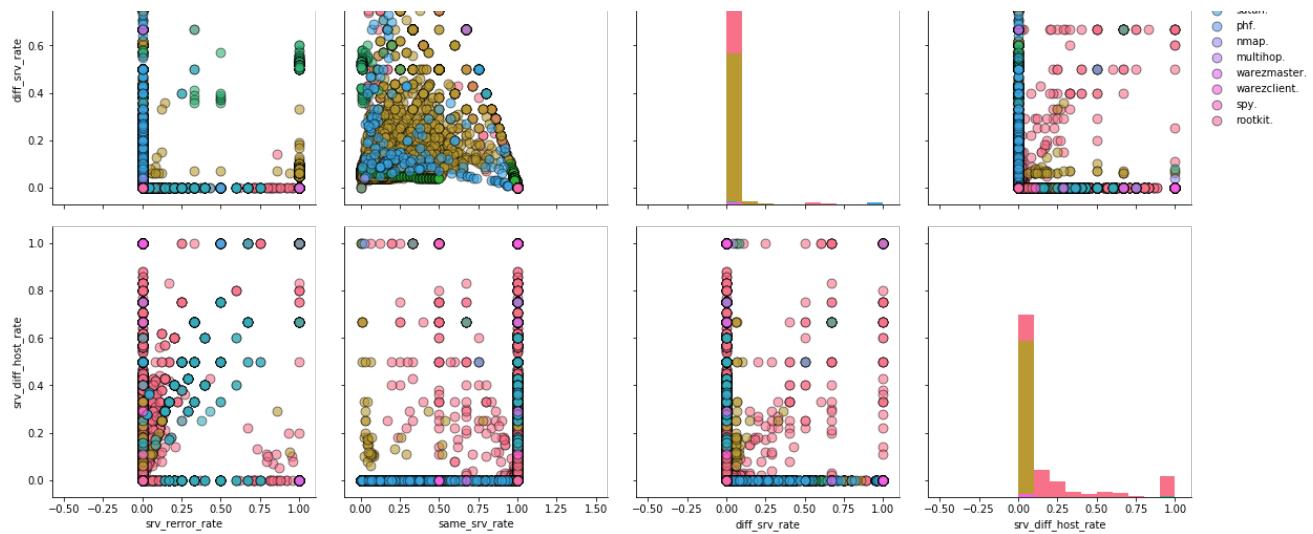
```
pairplot(data, 'intrusion_type',
features=['srv_count','serror_rate','srv_serror_rate','rerror_rate'])
```



In [20]:

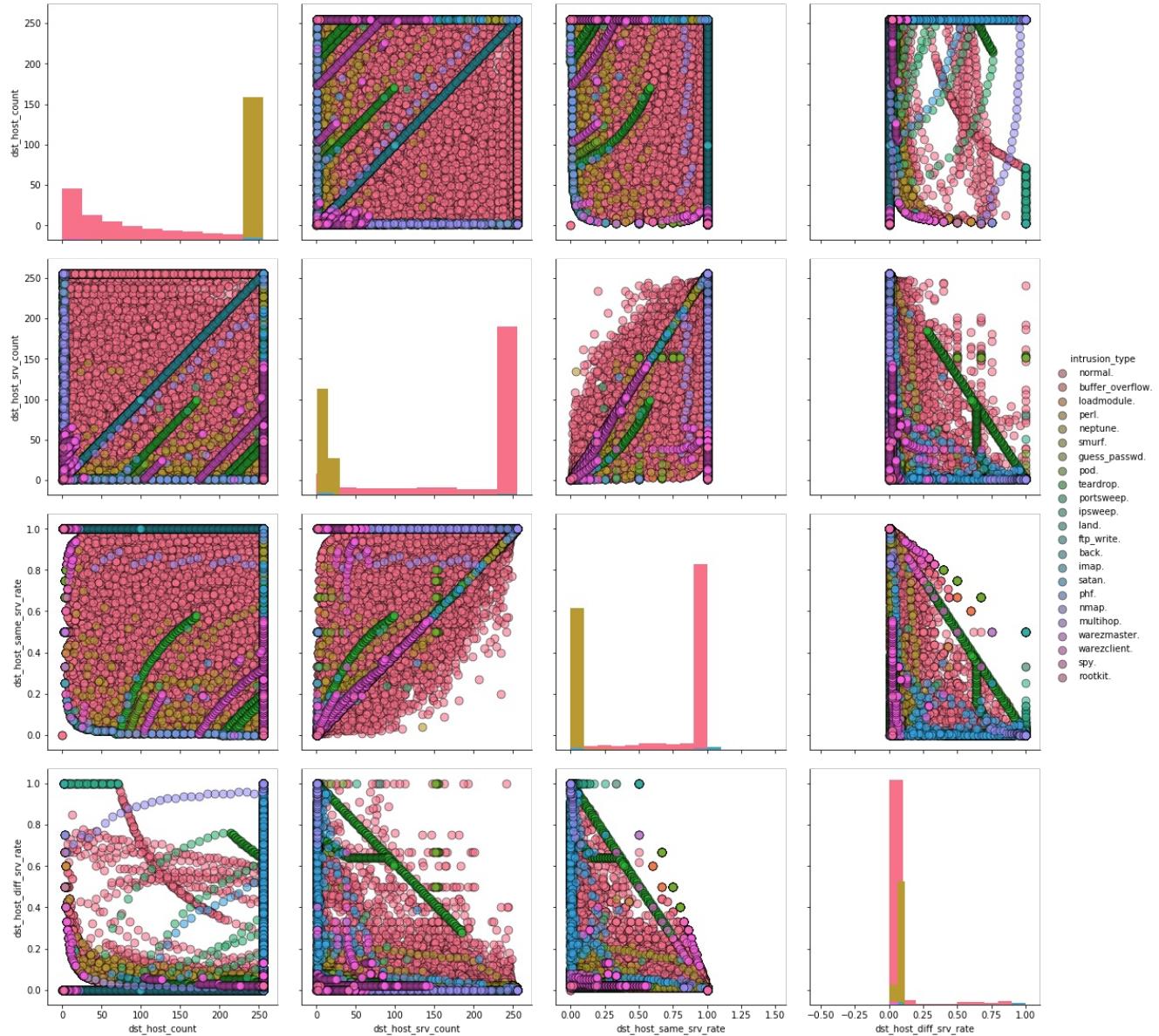
```
pairplot(data, 'intrusion_type',
features=['srv_error_rate', 'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate'])
```





In [21]:

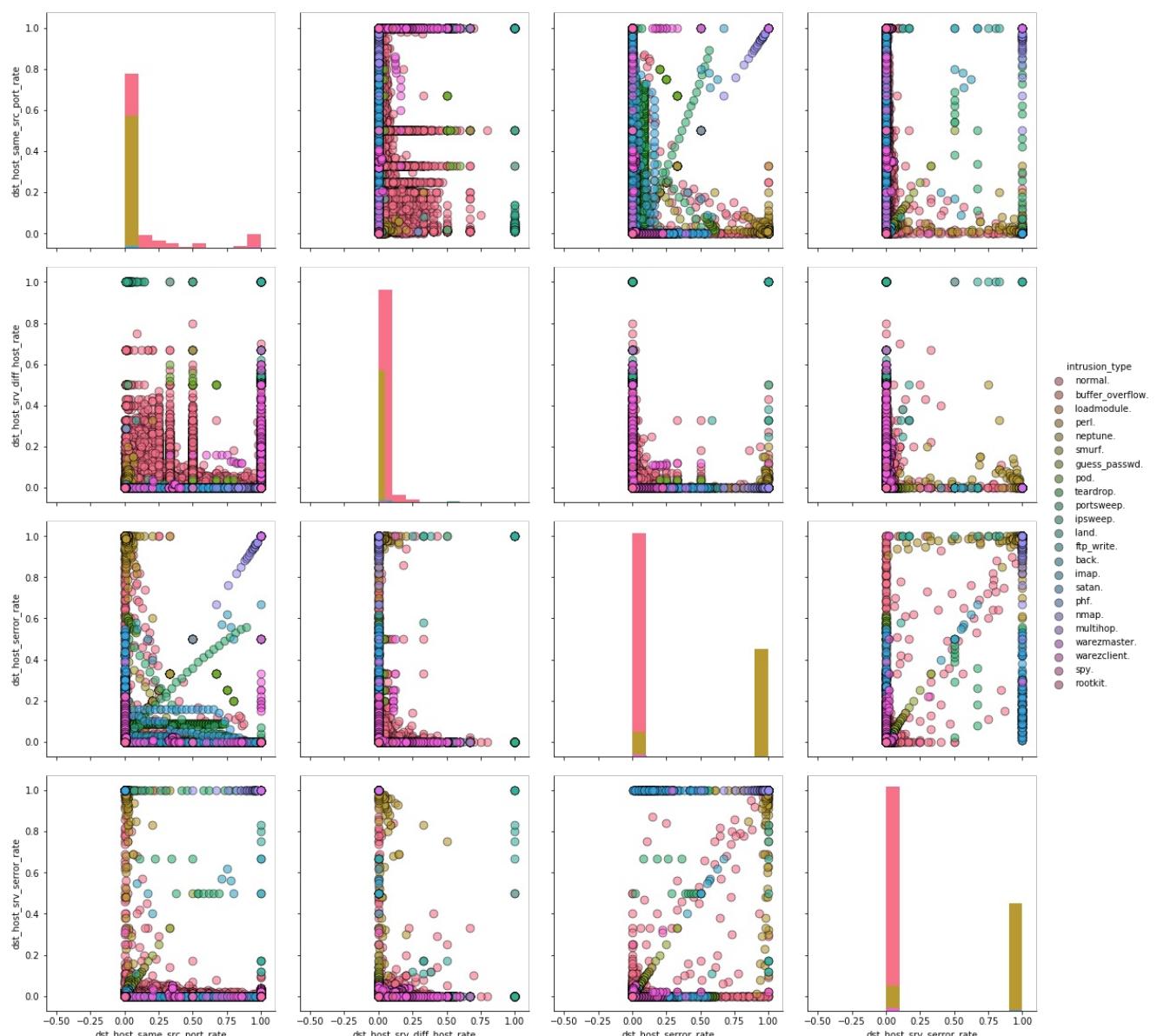
```
pairplot(data, 'intrusion_type',
features=['dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate'])
```



In [22]:

```
pairplot(data, 'intrusion_type',
```

```
features=['dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate'])
```



### **Observations:-**

- None of the pair plots are able to show linear separability/ almost linear separability between the different categories.

### **TSNE for Bivariate Analysis:-**

#### **Without categorical features:-**

In [30]:

```
from datetime import datetime

df = data.drop(['intrusion_type','protocol_type','service','flag'], axis=1)
Y = data['intrusion_type'].values
```

In [139]:

```
from sklearn.manifold import TSNE
import joblib
```

In [32]:

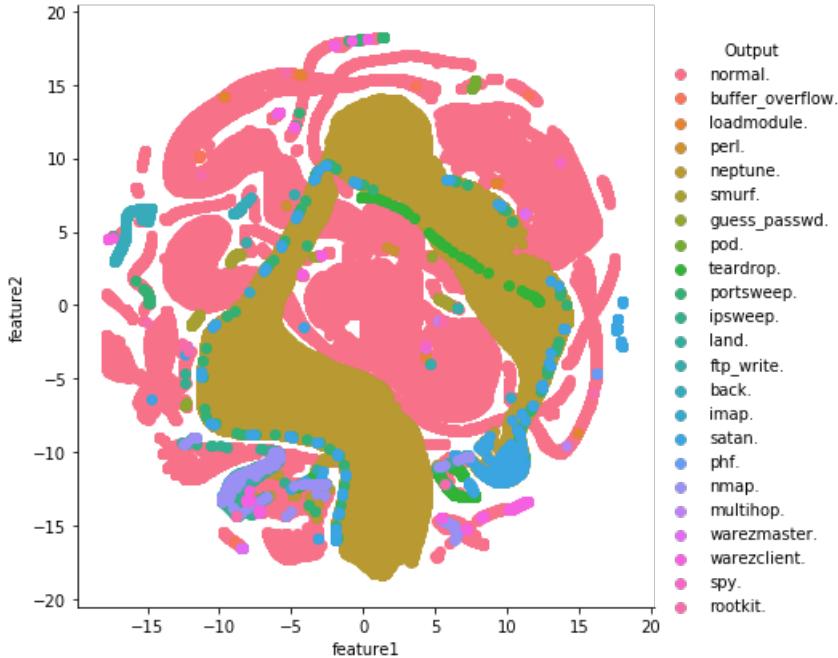
```
def tsne_func(data, label, no_components, perplexity_value, n_iter_value):
    """
        This function applies TSNE on the original dataset with no_components, perplexity_value, n_iter_value as the TSNE
        parameters and transforms the original dataset into TSNE transformed feature space with the tsne
        dataset containing
            number of features equal to the value specified for no_components and also plots the scatter p
        lot of the transformed
        data points along with their class label
    """
    start = datetime.now()
    print('TSNE with perplexity={} and no. of iterations={}'.format(perplexity_value, n_iter_value))
    tsne = TSNE(n_components=no_components, perplexity=perplexity_value, n_iter=n_iter_value)
    tsne_df1 = tsne.fit_transform(data)
    print(tsne_df1.shape)
    tsne_df1 = np.vstack((tsne_df1.T, Y)).T
    tsne_data1 = pd.DataFrame(data=tsne_df1, columns=['feature1', 'feature2', 'Output'])
    sns.FacetGrid(tsne_data1, hue='Output', size=6).map(plt.scatter, 'feature1', 'feature2').add_legen
    print('Total time taken:', datetime.now()-start)
    plt.show()
```

TSNE\_1:-

In [33]:

```
tsne_func(data=df, label=Y, no_components=2, perplexity_value=100, n_iter_value=500)
```

TSNE with perplexity=100 and no. of iterations=500  
(145586, 2)  
Total time taken: 0:27:15.022000

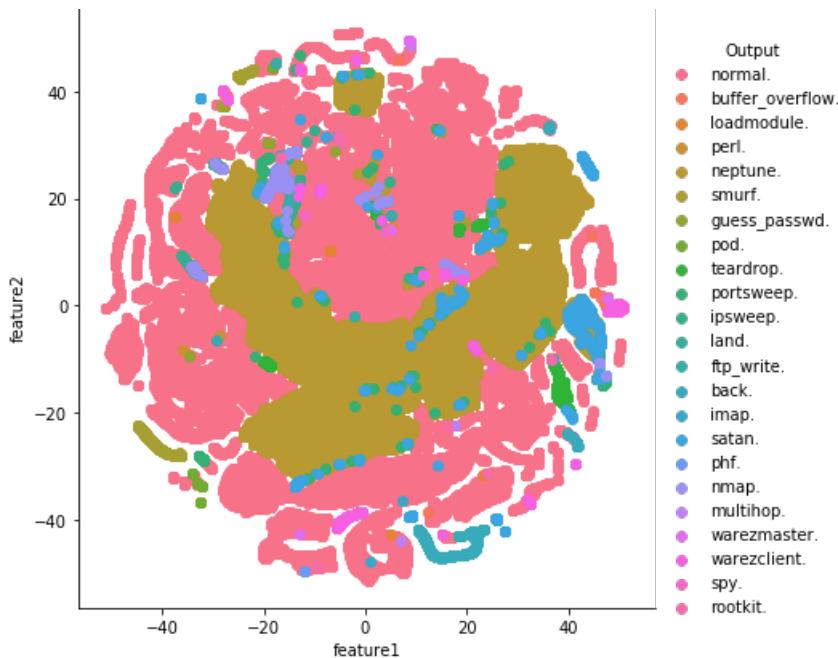


TSNE\_2:-

In [34]:

```
tsne_func(data=df, label=Y, no_components=2, perplexity_value=50, n_iter_value=1000)
```

TSNE with perplexity=50 and no. of iterations=1000  
(145586, 2)  
Total time taken: 0:43:27.225309



#### **Observations:-**

- From the above 2 graphs, it can be concluded that the data is not linearly separable/almost linearly separable in the TSNE\_transformed feature space.

#### **Train Test Split:-**

In [15]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(data.drop('intrusion_type', axis=1), data['intrusion_type'], stratify=data['intrusion_type'], test_size=0.25)
```

In [16]:

```
print('Train data')
print(X_train.shape)
print(Y_train.shape)
print('='*20)
print('Test data')
print(X_test.shape)
print(Y_test.shape)
```

```
Train data
(109189, 41)
(109189,)
=====
Test data
(36397, 41)
(36397,)
```

#### **Vectorizing Categorical features using one-hot encoding:-**

Categorical features in our dataset are:- 'protocol\_type', 'service', and 'flag'.

#### **Protocol\_type:-**

In [17]:

```
protocol = list(X_train['protocol_type'].values)
protocol = list(set(protocol))
```

```
print('Protocol types are:', protocol)
```

```
Protocol types are: ['udp', 'tcp', 'icmp']
```

In [18]:

```
from sklearn.feature_extraction.text import CountVectorizer
one_hot = CountVectorizer(vocabulary=protocol, binary=True)
train_protocol = one_hot.fit_transform(X_train['protocol_type'].values)
test_protocol = one_hot.transform(X_test['protocol_type'].values)
```

In [19]:

```
print(train_protocol[1].toarray())
train_protocol.shape
```

$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$

Out [19] :

(109189, 3)

### **Service:-**

In [20]:

```
service = list(X_train['service'].values)
service = list(set(service))
print('Service types are:\n', service)
```

Service types are:

```
['http_443', 'smtp', 'uucp_path', 'finger', 'pop_2', 'mtcp', 'nntp', 'auth', 'Z39_50', 'private',
'ntp_u', 'ssh', 'efs', 'eco_i', 'name', 'tftp_u', 'telnet', 'echo', 'whois', 'ftp', 'courier', 'rje',
'link', 'shell', 'netbios_ns', 'nnsp', 'hostnames', 'discard', 'daytime', 'gopher', 'http',
'imap4', 'red_i', 'time', 'ecr_i', 'ftp_data', 'csnet_ns', 'sunrpc', 'ldap', 'urh_i', 'ctf', 'netsat',
'kshell', 'netbios_dgm', 'urp_i', 'tim_i', 'uucp', 'netbios_ssn', 'klogin', 'other', 'pop_3',
'printer', 'domain', 'login', 'iso_tsap', 'IRC', 'bgp', 'sql_net', 'pm_dump', 'domain_u',
'systat', 'remote_job', 'X11', 'exec', 'vmnet', 'supdup']
```

In [21]:

```
from sklearn.feature_extraction.text import CountVectorizer
one_hot = CountVectorizer(vocabulary=service, binary=True)
train_service = one_hot.fit_transform(X_train['service'].values)
test_service = one_hot.transform(X_test['service'].values)
```

In [22]:

```
print(train_service[100].toarray())
```

In [23]:

```
train service.shape
```

Out [23] :

(109189, 66)

*Flag:-*

In [24]:

```
flag = list(X_train['flag'].values)
flag = list(set(flag))
print('flag types are:', flag)

flag types are: ['S1', 'S0', 'RSTOSO', 'S2', 'SH', 'S3', 'RSTO', 'SF', 'RSTR', 'OTH', 'REJ']
```

In [25]:

```
from sklearn.feature_extraction.text import CountVectorizer
one_hot = CountVectorizer(binary=True)
one_hot.fit(X_train['flag'].values)
train_flag = one_hot.transform(X_train['flag'].values)
test_flag = one_hot.transform(X_test['flag'].values)
```

In [26]:

```
print(test_flag[3000].toarray())
train_flag.shape
```

```
[[0 0 0 0 0 0 0 0 0 1 0]]
```

Out[26]:

```
(109189, 11)
```

In [27]:

```
X_train.drop(['protocol_type', 'service', 'flag'], axis=1, inplace=True)
X_test.drop(['protocol_type', 'service', 'flag'], axis=1, inplace=True)
```

## Applying Standardisation on the continuous features of our dataset:-

In [28]:

```
def feature_scaling(X_train, X_test, feature_name):

    """
    This function performs standardisation on the features
    """

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    scaler1 = scaler.fit_transform(X_train[feature_name].values.reshape(-1,1))
    scaler2 = scaler.transform(X_test[feature_name].values.reshape(-1,1))

    return scaler1, scaler2
```

### Duration:-

In [29]:

```
duration1, duration2 = feature_scaling(X_train, X_test, 'duration')
```

In [30]:

```
print(duration1[1])
```

```
[-0.10921678]
```

### src\_bytes :-

In [31]:

```
src_bytes1, src_bytes2 = feature_scaling(X_train, X_test, 'src_bytes')
```

In [32]:

```
print(src_bytes1[1])
```

```
[-0.00449211]
```

### ***dst\_bytes:-***

In [33]:

```
dst_bytes1, dst_bytes2 = feature_scaling(X_train, X_test, 'dst_bytes')
```

In [34]:

```
print(dst_bytes1[1])
```

```
[0.02427499]
```

### ***wrong\_fragment:-***

In [35]:

```
wrong_fragment1, wrong_fragment2 = feature_scaling(X_train, X_test, 'wrong_fragment')
```

In [36]:

```
print(wrong_fragment1[1])
```

```
[-0.08443666]
```

### ***urgent :-***

In [37]:

```
urgent1, urgent2 = feature_scaling(X_train, X_test, 'urgent')
```

In [38]:

```
print(urgent1[1])
```

```
[-0.00456235]
```

### ***hot :-***

In [39]:

```
hot1, hot2 = feature_scaling(X_train, X_test, 'hot')
```

In [40]:

```
print(hot1[1])
```

```
[-0.07099331]
```

### ***num\_failed\_logins***

```
In [41]:
```

```
num_failed_logins1, num_failed_logins2 = feature_scaling(X_train, X_test, 'num_failed_logins')
```

```
In [42]:
```

```
print(num_failed_logins1[1])
```

```
[-0.01917003]
```

### ***num\_compromised :-***

```
In [43]:
```

```
num_compromised1, num_compromised2 = feature_scaling(X_train, X_test, 'num_compromised')
```

```
In [44]:
```

```
num_compromised1[1]
```

```
Out[44]:
```

```
array([-0.00797605])
```

### ***root\_shell :-***

```
In [45]:
```

```
root_shell1, root_shell2 = feature_scaling(X_train, X_test, 'root_shell')
```

```
In [46]:
```

```
root_shell1[1]
```

```
Out[46]:
```

```
array([-0.01961637])
```

### ***su\_attempted :-***

```
In [47]:
```

```
su_attempted1, su_attempted2 = feature_scaling(X_train, X_test, 'su_attempted')
```

```
In [48]:
```

```
su_attempted1[1]
```

```
Out[48]:
```

```
array([-0.00915102])
```

### ***num\_root:-***

```
In [49]:
```

```
num_root1, num_root2 = feature_scaling(X_train, X_test, 'num_root')
```

```
In [50]:
```

```
num_root1[1]
```

```
Out[50]:  
array([-0.00997914])
```

#### ***num\_file\_creations:-***

```
In [51]:  
num_file_creations1, num_file_creations2 = feature_scaling(X_train, X_test, 'num_file_creations')  
  
In [52]:  
num_file_creations1[1]  
  
Out[52]:  
array([-0.02090036])
```

#### ***num\_shells:-***

```
In [53]:  
num_shells1, num_shells2 = feature_scaling(X_train, X_test, 'num_shells')  
  
In [54]:  
num_shells1[1]  
  
Out[54]:  
array([-0.01992092])
```

#### ***num\_access\_files:-***

```
In [55]:  
num_access_files1, num_access_files2 = feature_scaling(X_train, X_test, 'num_access_files')  
  
In [56]:  
num_access_files1[1]  
  
Out[56]:  
array([-0.0504473])
```

#### ***num\_outbound\_cmds:-***

```
In [57]:  
data['num_outbound_cmds'].value_counts()  
  
Out[57]:  
0    145586  
Name: num_outbound_cmds, dtype: int64
```

**- We will not use 'num\_outbound\_cmds' feature as it has all zero values.**

#### ***srv\_count:-***

```
In [58]:
```

```
srv_count1, srv_count2 = feature_scaling(X_train, X_test, 'srv_count')
```

```
In [59]:
```

```
srv_count1[1]
```

```
Out[59]:
```

```
array([-0.2265117])
```

***svr\_error\_rate:-***

```
In [60]:
```

```
serror_rate1, serror_rate2 = feature_scaling(X_train, X_test, 'serror_rate')
```

```
In [61]:
```

```
serror_rate1[1]
```

```
Out[61]:
```

```
array([-0.64306069])
```

***svr\_serror\_rate:-***

```
In [62]:
```

```
srv_serror_rate1, srv_serror_rate2 = feature_scaling(X_train, X_test, 'srv_serror_rate')
```

```
In [63]:
```

```
srv_serror_rate1[1]
```

```
Out[63]:
```

```
array([-0.64257258])
```

***rerror\_rate:-***

```
In [64]:
```

```
rerror_rate1, rerror_rate2 = feature_scaling(X_train, X_test, 'rerror_rate')
```

```
In [65]:
```

```
rerror_rate1[1]
```

```
Out[65]:
```

```
array([-0.35164423])
```

***svr\_rerror\_rate :-***

```
In [66]:
```

```
srv_rerror_rate1, srv_rerror_rate2 = feature_scaling(X_train, X_test, 'srv_rerror_rate')
```

```
In [67]:
```

```
srv_error_rate1[1]
```

Out[67]:

```
array([-0.35186104])
```

#### **same\_srv\_rate:-**

In [68]:

```
same_srv_rate1, same_srv_rate2 = feature_scaling(X_train, X_test, 'same_srv_rate')
```

In [69]:

```
same_srv_rate1[1]
```

Out[69]:

```
array([0.772007])
```

#### **diff\_srv\_rate:-**

In [70]:

```
diff_srv_rate1, diff_srv_rate2 = feature_scaling(X_train, X_test, 'diff_srv_rate')
```

In [71]:

```
diff_srv_rate1[1]
```

Out[71]:

```
array([-0.33700418])
```

#### **srv\_diff\_host\_rate:-**

In [72]:

```
srv_diff_host_rate1, srv_diff_host_rate2 = feature_scaling(X_train, X_test, 'srv_diff_host_rate')
```

In [73]:

```
srv_diff_host_rate1[1]
```

Out[73]:

```
array([-0.38512506])
```

#### **dst\_host\_count:-**

In [74]:

```
dst_host_count1, dst_host_count2 = feature_scaling(X_train, X_test, 'dst_host_count')
```

In [75]:

```
dst_host_count1[1]
```

Out[75]:

```
array([-1.76908291])
```

### ***dst\_host\_srv\_count:-***

In [76]:

```
dst_host_srv_count1, dst_host_srv_count2 = feature_scaling(X_train, X_test, 'dst_host_srv_count')
```

In [77]:

```
dst_host_srv_count1[1]
```

Out[77]:

```
array([1.09217033])
```

### ***dst\_host\_same\_srv\_rate:-***

In [78]:

```
dst_host_same_srv_rate1, dst_host_same_srv_rate2 = feature_scaling(X_train, X_test,  
'dst_host_same_srv_rate')
```

In [79]:

```
dst_host_same_srv_rate1[1]
```

Out[79]:

```
array([0.98122298])
```

### ***dst\_host\_diff\_srv\_rate:-***

In [80]:

```
dst_host_diff_srv_rate1, dst_host_diff_srv_rate2 = feature_scaling(X_train, X_test,  
'dst_host_diff_srv_rate')
```

In [81]:

```
dst_host_diff_srv_rate1[1]
```

Out[81]:

```
array([-0.41785189])
```

### ***dst\_host\_same\_src\_port\_rate:-***

In [82]:

```
dst_host_same_src_port_rate1, dst_host_same_src_port_rate2 = feature_scaling(X_train, X_test,  
'dst_host_same_src_port_rate')
```

In [83]:

```
dst_host_same_src_port_rate1[1]
```

Out[83]:

```
array([0.31960246])
```

### ***dst\_host\_srv\_diff\_host\_rate:-***

In [84]:

```
dst_host_srv_diff_host_rate1, dst_host_srv_diff_host_rate2 = feature_scaling(X_train, X_test,  
'dst_host_srv_diff_host_rate')
```

In [85]:

```
dst_host_srv_diff_host_rate1[1]
```

Out[85]:

```
array([0.51565566])
```

#### ***dst\_host\_serror\_rate:-***

In [86]:

```
dst_host_serror_rate1, dst_host_serror_rate2 = feature_scaling(X_train, X_test, 'dst_host_serror_rate')
```

In [87]:

```
dst_host_serror_rate1[1]
```

Out[87]:

```
array([-0.64406536])
```

#### ***dst\_host\_srv\_serror\_rate:-***

In [88]:

```
dst_host_srv_serror_rate1, dst_host_srv_serror_rate2 = feature_scaling(X_train, X_test,  
'dst_host_srv_serror_rate')
```

In [89]:

```
dst_host_srv_serror_rate1[1]
```

Out[89]:

```
array([-0.64136056])
```

#### ***dst\_host\_error\_rate:-***

In [90]:

```
dst_host_error_rate1, dst_host_error_rate2 = feature_scaling(X_train, X_test, 'dst_host_error_rate')
```

In [91]:

```
dst_host_error_rate1[1]
```

Out[91]:

```
array([-0.3616436])
```

#### ***dst\_host\_srv\_error\_rate:-***

In [92]:

```
dst_host_srv_error_rate1, dst_host_srv_error_rate2 = feature_scaling(X_train, X_test,  
'dst_host_srv_error_rate')
```

```
In [93]:  
dst_host_srv_rerror_rate[1]
```

```
Out[93]:  
array([-0.35467143])
```

### ***num\_failed\_logins :-***

```
In [94]:  
num_failed_logins1, num_failed_logins2 = feature_scaling(X_train, X_test, 'num_failed_logins')
```

```
In [95]:  
num_failed_logins1[1]  
  
Out[95]:  
array([-0.01917003])
```

### ***land:-***

```
In [96]:  
land1, land2 = np.array([X_train['land'].values]), np.array([X_test['land'].values])
```

```
In [97]:  
land1.shape  
  
Out[97]:  
(1, 109189)
```

### ***is\_host\_login :-***

```
In [98]:  
is_host_login1, is_host_login2 = np.array([X_train['is_host_login'].values]), np.array([X_test['is_host_login'].values])
```

```
In [99]:  
is_host_login1.shape  
  
Out[99]:  
(1, 109189)
```

### ***is\_guest\_login :-***

```
In [100]:  
is_guest_login1, is_guest_login2 = np.array([X_train['is_guest_login'].values]), np.array([X_test['is_guest_login'].values])
```

```
In [101]:  
is_guest_login1.shape
```

```
Out[101]:  
(1, 109189)
```

### **logged\_in :-**

```
In [102]:
```

```
logged_in1, logged_in2 = np.array([X_train['logged_in'].values]), np.array([X_test['logged_in'].values])
```

```
In [103]:
```

```
logged_in1.shape
```

```
Out[103]:
```

```
(1, 109189)
```

### **count:-**

```
In [104]:
```

```
count1, count2 = feature_scaling(X_train, X_test, 'count')
```

```
In [105]:
```

```
count1[1]
```

```
Out[105]:
```

```
array([-0.68085229])
```

### **dst\_host\_diff\_srv\_rate:-**

```
In [106]:
```

```
dst_host_diff_srv_rate1, dst_host_diff_srv_rate2 = feature_scaling(X_train, X_test,  
'dst_host_diff_srv_rate')
```

```
In [107]:
```

```
dst_host_diff_srv_rate1[1]
```

```
Out[107]:
```

```
array([-0.41785189])
```

## **Merging categorical and continuous features:-**

```
In [108]:
```

```
from scipy.sparse import hstack
```

```
In [109]:
```

```
X_train_1 = hstack((duration1, train_protocol, train_service, train_flag, src_bytes1,  
dst_bytes1, land1.T, wrong_fragment1, urgent1, hot1,  
num_failed_logins1, logged_in1.T, num_compromised1, root_shell1,  
su_attempted1, num_root1, num_file_creations1, num_shells1,  
num_access_files1, is_host_login1.T,  
is_guest_login1.T, count1, srv_count1, error_rate1,  
srv_error_rate1, rerror_rate1, srv_rerror_rate1, same_srv_rate1,
```

```
diff_srv_rate1, srv_diff_host_rate1, dst_host_count1,
dst_host_srv_count1, dst_host_same_srv_rate1,
dst_host_diff_srv_rate1, dst_host_same_src_port_rate1,
dst_host_srv_diff_host_rate1, dst_host_serror_rate1,
dst_host_srv_serror_rate1, dst_host_error_rate1,
dst_host_srv_error_rate1))
```

In [110]:

```
X_train_1.shape
```

Out[110]:

```
(109189, 117)
```

In [111]:

```
X_test_1 = hstack((duration2, test_protocol, test_service, test_flag, src_bytes2,
dst_bytes2, land2.T, wrong_fragment2, urgent2, hot2,
num_failed_logins2, logged_in2.T, num_compromised2, root_shell2,
su_attempted2, num_root2, num_file_creations2, num_shells2,
num_access_files2, is_host_login2.T,
is_guest_login2.T, count2, srv_count2, serror_rate2,
srv_serror_rate2, rerror_rate2, srv_error_rate2, same_srv_rate2,
diff_srv_rate2, srv_diff_host_rate2, dst_host_count2,
dst_host_srv_count2, dst_host_same_srv_rate2,
dst_host_diff_srv_rate2, dst_host_same_src_port_rate2,
dst_host_srv_diff_host_rate2, dst_host_serror_rate2,
dst_host_srv_serror_rate2, dst_host_error_rate2,
dst_host_srv_error_rate2))
```

In [112]:

```
X_test_1.shape
```

Out[112]:

```
(36397, 117)
```

In [113]:

```
import joblib
joblib.dump(X_train_1, 'X_train_1.pkl')
joblib.dump(X_test_1, 'X_test_1.pkl')
X_train_1 = joblib.load('X_train_1.pkl')
X_test_1 = joblib.load('X_test_1.pkl')
```

In [114]:

```
joblib.dump(Y_train, 'Y_train.pkl')
joblib.dump(Y_test, 'Y_test.pkl')
Y_train = joblib.load('Y_train.pkl')
Y_test = joblib.load('Y_test.pkl')
```

## Further Approach to our problem

- **We will first apply below classifiers on our dataset and evaluate their performance:**

1. Naive Bayes
2. Logistic Regression
3. SVM
4. Decision Tree
5. Random Forest
6. GBDT / XGBoost

- **Based on the performance metric scores we obtain from the above classifiers, we will apply below feature engineering techniques on our dataset to get additional features:**

1. Clustering features:- We will apply clustering on our dataset and add the clustering values as an additional feature to our dataset.
2. PCA transformed features:- We will apply PCA/TSNE/SVD on the dataset and will use the top 5 PCA features as additional features on our dataset.
3. Feature engineering using existing features:- We will apply feature engineering techniques like
  - (i) Adding 2 existing features, (e.g. new\_feature\_1 = src\_bytes + dst\_bytes)
  - (ii) Subtracting 2 existing features, (e.g. new\_feature\_2 = abs(src\_bytes - dst\_bytes)).

- **We will then apply the best performing classifiers from dataset 1 on dataset 2 and evaluate their performance.**

## Applying Machine Algorithms:-

### Utility Functions:-

In [116]:

```
import datetime as dt
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, precision_score, recall_score, f1_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.externals import joblib
```

In [12]:

```
def confusion_matrix_func(Y_test, y_test_pred):
    """
    This function plots the confusion matrix heatmap using the actual and predicted values.
    """
    C = confusion_matrix(Y_test, y_test_pred)
    cm_df = pd.DataFrame(C)
    labels = ['back', 'butter_overflow', 'loadmodule', 'guess_passwd', 'imap', 'ipsweep', 'warezmaster', 'rootkit',
              'multihop', 'neptune', 'nmap', 'normal', 'phf', 'perl', 'pod', 'portsweep',
              'ftp_write', 'satan', 'smurf',
              'teardrop', 'warezclient', 'land']
    plt.figure(figsize=(20,15))
    sns.set(font_scale=1.4)
    sns.heatmap(cm_df, annot=True, annot_kws={"size":12}, fmt='g', xticklabels=labels, yticklabels=labels)
    plt.ylabel('Actual Class')
    plt.xlabel('Predicted Class')

    plt.show()
```

In [13]:

```
def model(model_name, X_train, Y_train, X_test, Y_test):
    """
    This function computes the performance scores on the train and test data.
    """

    print('Fitting the model and prediction on train data:')
    start = dt.datetime.now()
    model_name.fit(X_train, Y_train)
    y_tr_pred = model_name.predict(X_train)
    print('Completed')
    print('Time taken:',dt.datetime.now()-start)
    print('='*50)
```

```

results_tr = dict()
y_tr_pred = model_name.predict(X_train)
results_tr['precision'] = precision_score(Y_train, y_tr_pred, average='weighted')
results_tr['recall'] = recall_score(Y_train, y_tr_pred, average='weighted')
results_tr['f1_score'] = f1_score(Y_train, y_tr_pred, average='weighted')

results_test = dict()
print('Prediction on test data:')
start = dt.datetime.now()
y_test_pred = model_name.predict(X_test)
print('Completed')
print('Time taken:', dt.datetime.now() - start)
print('='*50)

print('Performance metrics:')
print('='*50)
print('Confusion Matrix is:')
confusion_matrix_func(Y_test, y_test_pred)
print('='*50)
results_test['precision'] = precision_score(Y_test, y_test_pred, average='weighted')
print('Precision score is:')
print(precision_score(Y_test, y_test_pred, average='weighted'))
print('='*50)
results_test['recall'] = recall_score(Y_test, y_test_pred, average='weighted')
print('Recall score is:')
print(recall_score(Y_test, y_test_pred, average='weighted'))
print('='*50)
results_test['f1_score'] = f1_score(Y_test, y_test_pred, average='weighted')
print('F1-score is:')
print(f1_score(Y_test, y_test_pred, average='weighted'))
# add the trained model to the results
results_test['model'] = model

return results_tr, results_test

```

In [117]:

```

def print_grid_search_attributes(model):

    """
    This function prints all the grid search attributes
    """

    print('-----')
    print('|      Best Estimator      |')
    print('-----')
    print('\n\t{} \n'.format(model.best_estimator_))

    # parameters that gave best results while performing grid search
    print('-----')
    print('|      Best parameters      |')
    print('-----')
    print('\tParameters of best estimator : \n\n\t{} \n'.format(model.best_params_))

    # number of cross validation splits
    print('-----')
    print('|    No of CrossValidation sets    |')
    print('-----')
    print('\n\tTotal numbre of cross validation sets: {} \n'.format(model.n_splits_))

    # Average cross validated score of the best estimator, from the Grid Search
    print('-----')
    print('|      Best Score      |')
    print('-----')
    print('\n\tAverage Cross Validate scores of best estimator : \n\n\t{} \n'.format(model.best_score_))

```

In [11]:

```

def tpr_fpr_func(Y_tr, Y_pred):
    """
    This function computes the TPR and FPR scores using the actual and predicted values

```

```
'''  
results = dict()  
Y_tr = Y_tr.to_list()  
tp = 0; fp = 0; positives = 0; negatives = 0; length = len(Y_tr)  
for i in range(len(Y_tr)):  
    if Y_tr[i]=='normal.':  
        positives += 1  
    else:  
        negatives += 1  
  
for i in range(len(Y_pred)):  
    if Y_tr[i]=='normal.' and Y_pred[i]=='normal.':  
        tp += 1  
    elif Y_tr[i]!='normal.' and Y_pred[i]=='normal.':  
        fp += 1  
  
tpr = tp/positives  
fpr = fp/negatives  
  
results['tp'] = tp; results['tpr'] = tpr; results['fp'] = fp; results['fpr'] = fpr  
  
return results
```

## **Model\_1 :- Gaussian Naive Bayes**

In [119]:

```
hyperparameter = {'var_smoothing':[10**x for x in range(-9,3)]}

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb_grid = GridSearchCV(nb, param_grid=hyperparameter, cv=5, verbose=1, n_jobs=-1)
```

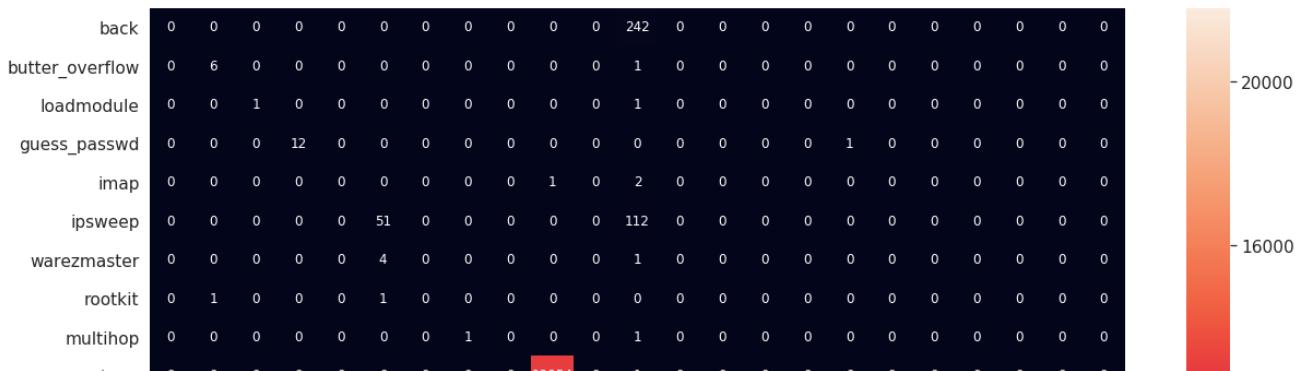
In [129]:

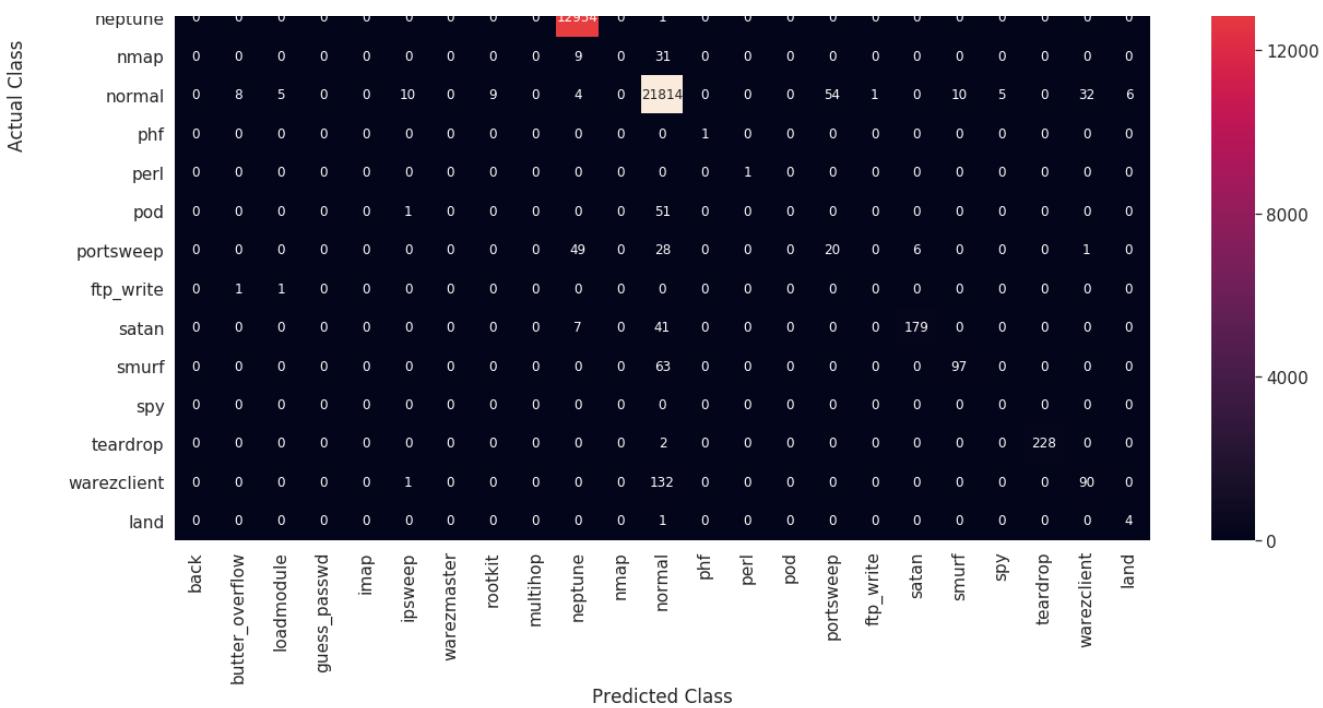
```
nb_grid_results_tr, nb_grid_results_test = model(nb_grid, X_train_1.toarray(), Y_train,  
X_test_1.toarray(), Y_test)
```

```
Fitting the model and prediction on train data:  
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   8.9s  
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:  13.1s finished
```

```
Completed
Time taken: 0:00:17.167590
=====
Prediction on test data:
Completed
Time taken: 0:00:00.712164
=====
Performance metrics:
=====
Confusion Matrix is:
```





```
=====
Precision score is:  
0.9637974665033534  
=====  
Recall score is:  
0.974201170426134  
=====  
F1-score is:  
0.9679678294214985
```

In [121]:

```
print_grid_search_attributes(nb_grid)
```

## Best Estimator

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

## Best parameters

Parameters of best estimator :

```
{'var smoothing': 10}
```

| No. of CrossValidation sets |

The following table summarizes the results of the simulation study.

| Best Score |

Average Cross Validate scores of best estimator :

0.9729276758647849

In [122]:

```
joblib.dump(nb_grid.best_estimator_, 'nb_gs.pkl')
```

```
Out[122]:
```

```
['nb_gs.pkl']
```

```
In [123]:
```

```
nb_gs = nb_grid.best_estimator_
y_tr_pred = nb_gs.predict(X_train_1.toarray())
y_test_pred = nb_gs.predict(X_test_1.toarray())
tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)
tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

```
In [124]:
```

```
tpr_fpr_train
```

```
Out[124]:
```

```
{'fp': 2225,
 'fpr': 0.051367886413482625,
 'tp': 65483,
 'tpr': 0.9940644260254425}
```

```
In [125]:
```

```
nb_grid_results_tr
```

```
Out[125]:
```

```
{'f1_score': 0.9671813437943309,
 'precision': 0.9632732426450655,
 'recall': 0.9738984696260612}
```

```
In [126]:
```

```
tpr_fpr_test
```

```
Out[126]:
```

```
{'fp': 710, 'fpr': 0.04917238035875061, 'tp': 21814, 'tpr': 0.9934420256853994}
```

```
In [127]:
```

```
nb_grid_results_test
```

```
Out[127]:
```

```
{'f1_score': 0.9679678294214985,
 'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,
 'precision': 0.9637974665033534,
 'recall': 0.974201170426134}
```

### **Observations from applying Naive Bayes Classifier:-**

- The test data has 36397 total no. of points. Out of these, 21958 are points belonging to Normal connections and remaining 14439 points belong to Bad connections.
- Out of the 21958 Normal connection points, 21814 (99.34%) were classified correctly by the Naive Bayes Classifier.
- Out of the 14439 points belonging to Bad connections, class Neptune has the highest no. of data points 12955, out of which 12954(99.99%) are classified correctly.
- Out of the classes having very less no. of data points, class guess\_passwd was classified with (12/13) 92.30% accuracy, class butter\_overflow with (6/7) 85.71% accuracy, class warezmaster with (4/5) 80% accuracy, class land with (4/5) 80% accuracy, class imap with (0/3) 0% accuracy, class loadmodule with (1/2) 50% accuracy, class rootkit with (0/2) 0% accuracy, class multihop with (0/2) 0% accuracy, class ftp\_write with (0/2) 0% accuracy, and classes phf and perl with both (1/1) 100% accuracy.
- Although the Naive bayes Classifier was able to classify points with a high f1 score of 0.9670, we can use more advanced Non-linear and linear classifiers ahead and we will try to classify the Normal and bad connections with a higher f1-score.

- false positives : 710
- false positive rate : 0.049
- true positive : 21814
- true positive rate : 0.9934
- As the train and test scores are almost similar and high, we can say that the model is "NEITHER OVERFITTING NOR UNDERFITTING".

## Model\_2:- Logistic Regression

In [190]:

```
hyperparameter = {'alpha':[0.001, 0.01, 0.1, 1, 10, 20, 30], 'penalty':['l1', 'l2']}
from sklearn.linear_model import SGDClassifier
lr = SGDClassifier(loss='log')
lr_grid = GridSearchCV(lr, param_grid=hyperparameter, cv=5, verbose=1, n_jobs=-1)
```

In [191]:

```
lr_grid_results_tr, lr_grid_results_test = model(lr_grid, X_train_1.toarray(), Y_train,
X_test_1.toarray(), Y_test)
```

Fitting the model and prediction on train data:  
Fitting 5 folds for each of 14 candidates, totalling 70 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   34 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done   70 out of  70 | elapsed:  4.2min finished
```

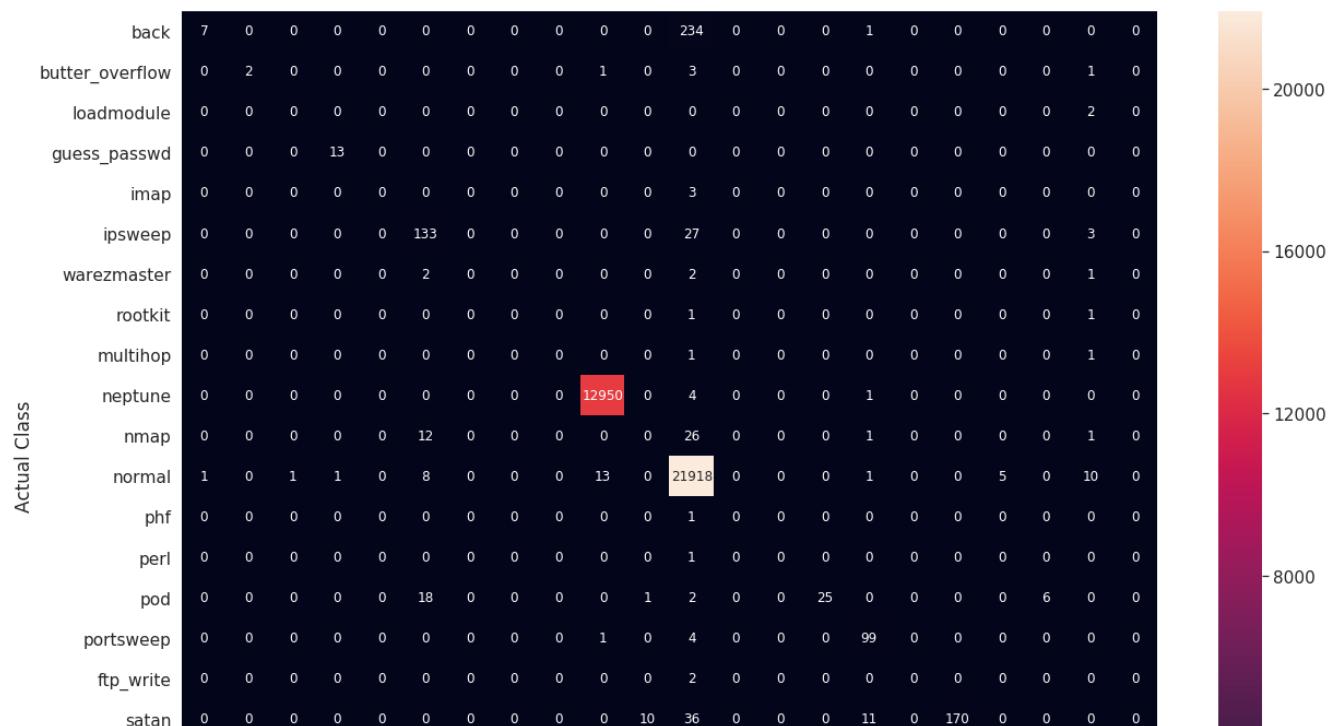
Completed  
Time taken: 0:04:29.892423

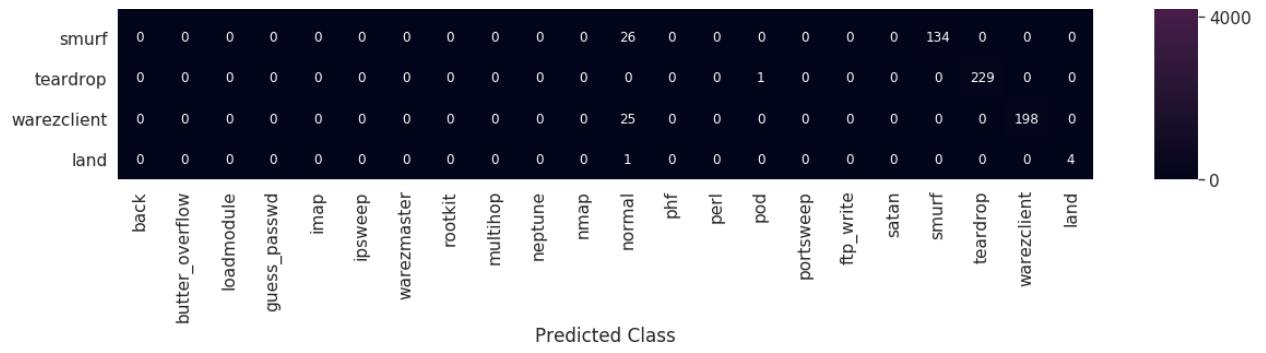
=====
Prediction on test data:

Completed  
Time taken: 0:00:00.014114

=====
Performance metrics:

=====
Confusion Matrix is:





```
=====
Precision score is:
0.9840039823310467
=====
Recall score is:
0.9858504821825975
=====
F1-score is:
0.9819585517619243
```

In [192]:

```
print_grid_search_attributes(lr_grid)
```

```
-----
|      Best Estimator      |
-----
SGDClassifier(alpha=0.001, average=False, class_weight=None,
               early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
               l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
               n_iter_no_change=5, n_jobs=None, penalty='l1', power_t=0.5,
               random_state=None, shuffle=True, tol=0.001,
               validation_fraction=0.1, verbose=0, warm_start=False)
```

```
-----
|      Best parameters      |
-----
```

Parameters of best estimator :

```
{'alpha': 0.001, 'penalty': 'l1'}
```

```
-----
|      No of CrossValidation sets      |
-----
```

Total numbre of cross validation sets: 5

```
-----
|      Best Score      |
-----
```

Average Cross Validate scores of best estimator :

```
0.9842841311853758
```

In [193]:

```
joblib.dump(lr_grid.best_estimator_, 'lr_gs.pkl')
```

Out[193]:

```
['lr_gs.pkl']
```

In [194]:

```
lr_gs = lr_grid.best_estimator_
y_tr pred = lr_gs.predict(X_train_1.toarray())
```

```
-- 
y_test_pred = lr_gs.predict(X_test_1.toarray())
tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)
tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [195]:

```
lr_grid_results_tr
```

Out[195]:

```
{'f1_score': 0.9813768763397716,
'precision': 0.9837322127432626,
'recall': 0.9852182912198115}
```

In [196]:

```
tpr_fpr_train
```

Out[196]:

```
{'fp': 1279,
'fpr': 0.02952787717880642,
'tp': 65754,
'tpr': 0.9981783404681664}
```

In [197]:

```
lr_grid_results_test
```

Out[197]:

```
{'f1_score': 0.9819585517619243,
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,
'precision': 0.9840039823310467,
'recall': 0.9858504821825975}
```

In [198]:

```
tpr_fpr_test
```

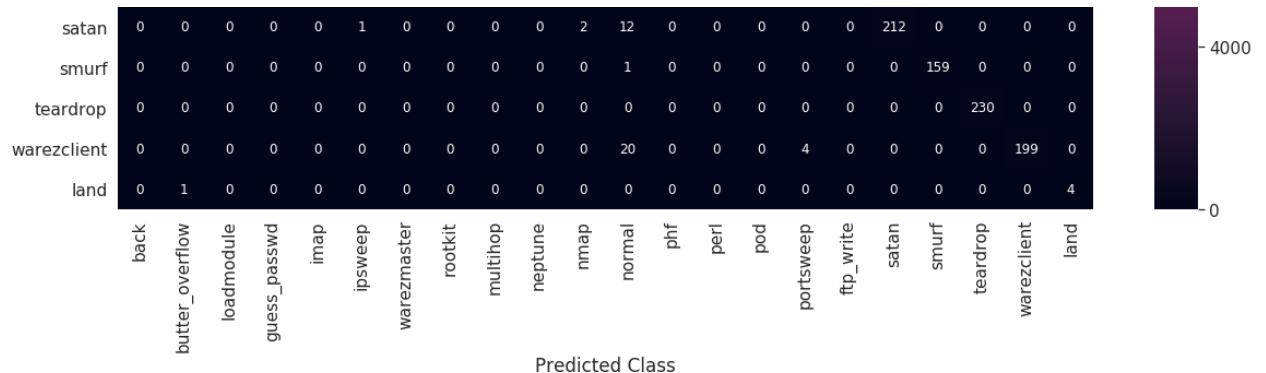
Out[198]:

```
{'fp': 399,
'fpr': 0.027633492624142945,
'tp': 21918,
'tpr': 0.9981783404681664}
```

#### **Observations from applying Logistic Regression Classifier:-**

- Out of the 21958 Normal connection points, 21918 (99.81%) were correctly classified by the LR Classifier.
- Out of the 14439 points belonging to Bad connections, class Neptune has the highest no. of data points 12955, out of which 12950(99.96%) were classified correctly.
- Out of the classes having very less no. of data points, class guess\_passwd was classified with (13/13) 100% accuracy, class butter\_overflow with (2/7) 28.15% accuracy, class warezmaster with(0/5) 0% accuracy, class land with(4/5) 80% accuracy, class imap with(0/3) 0% accuracy, class loadmodule with (0/2) 0% accuracy, class rootkit with(0/2) 0% accuracy, class multihop with (0/2) 0% accuracy, class ftp\_write with(0/2) 0% accuracy, and class phf with (0/1) 0% accuracy and class perl with (0/1) 0% accuracy.
- Classes with very low no. of data points like loadmodule, multihop, ftp\_write, phf were not classified correctly by this classifier.
- The Logistic Regression Classifier was able to classify points with a high f1 score of 0.9819 compared to the Naive Bayes Classifier.
- From the good performance of the Logistic Regression model, we can conclude that the data has some linearity in the higher dimensional space which was not visible in the lower dimensional 2D pair plots and Tsne transformed 2D space.
- True Positives = 21918





```
=====
Precision score is:
0.9966322870498145
=====
Recall score is:
0.9968129241420997
=====
F1-score is:
0.9966850328004752
```

In [201]:

```
print_grid_search_attributes(svm_grid)
```

```
-----
|      Best Estimator      |
-----
SGDClassifier(alpha=1e-06, average=False, class_weight=None,
               early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
               l1_ratio=0.15, learning_rate='optimal', loss='hinge',
               max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
               power_t=0.5, random_state=None, shuffle=True, tol=0.001,
               validation_fraction=0.1, verbose=0, warm_start=False)
```

```
-----
|      Best parameters      |
-----
```

Parameters of best estimator :

```
{'alpha': 1e-06, 'penalty': 'l1'}
```

```
-----
|      No of CrossValidation sets      |
-----
```

Total numbre of cross validation sets: 5

```
-----
|      Best Score      |
-----
```

Average Cross Validate scores of best estimator :

```
0.9966388555623735
```

In [202]:

```
joblib.dump(svm_grid.best_estimator_, 'svm_gs.pkl')
```

Out[202]:

```
['svm_gs.pkl']
```

In [204]:

```
svm_gs = svm_grid.best_estimator_
```

```
y_tr_pred = svm_gs.predict(X_train_1.toarray())
y_test_pred = svm_gs.predict(X_test_1.toarray())
svm_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)
svm_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [203]:

```
svm_grid_results_tr
```

Out [203]:

```
{'f1_score': 0.9967967182508746,
'precision': 0.996776105410674,
'recall': 0.9969502422405188}
```

In [205]:

```
svm_tpr_fpr_train
```

Out [205]:

```
{'fp': 210, 'fpr': 0.004848205009811844, 'tp': 65794, 'tpr': 0.998785560312111}
```

In [206]:

```
svm_grid_results_test
```

Out [206]:

```
{'f1_score': 0.9966850328004752,
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,
'precision': 0.9966322870498145,
'recall': 0.9968129241420997}
```

In [207]:

```
svm_tpr_fpr_test
```

Out [207]:

```
{'fp': 63, 'fpr': 0.004363183045917307, 'tp': 21931, 'tpr': 0.9987703798160124}
```

#### **Observations from applying SVM Classifier (SGD Classifier + "hinge" loss):-**

- Out of the 21958 Normal connection points, 21931 (99.85%) were correctly classified by the SVM Classifier.
- Out of the 17409 points belonging to Bad connections, class Neptune has the highest no. of data points 12955, out of which 12954(99.99%) were classified correctly.
- Out of the classes having very less no. of data points, class guess\_passwd was classified with (13/13) 100.0% accuracy, class buffer\_overflow with (1/7) 14.71% accuracy, class warezmaster with (3/5) 60% accuracy, class land with (4/5) 80% accuracy, class imap with (3/3) 100% accuracy, class loadmodule with (0/2) 0% accuracy, class rootkit with (0/2) 0% accuracy, class multihop with (0/2) 0% accuracy, class ftp\_write with (0/2) 0% accuracy, and class phf with (1/1) 100% accuracy and class perl with (0/1) 0% accuracy.
- The SVM Classifier was able to classify points with a high f1 score of 0.9966 compared to Naive Bayes and LR Classifiers.
- True Postives = 21931
- TPR = 0.9985
- False Positives = 63
- FPR = 0.0043
- The SVM Classifier has the lowest FPR and highest FPR than the NB and LR models.
- As the train and test scores are almost similar and high, we can say that the LR model is "NEITHER OVERFITTING NOR UNDERFITTING"

## **Model\_4 :- Decision Tree**

In [209]:

```

hyperparameter = {'max_depth':[5, 10, 20, 50, 100, 500], 'min_samples_split':[5, 10, 100, 500]}

from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(criterion='gini', splitter='best', class_weight='balanced')
decision_tree_grid = GridSearchCV(decision_tree, param_grid=hyperparameter, cv=3, verbose=1, n_jobs=-1)

```

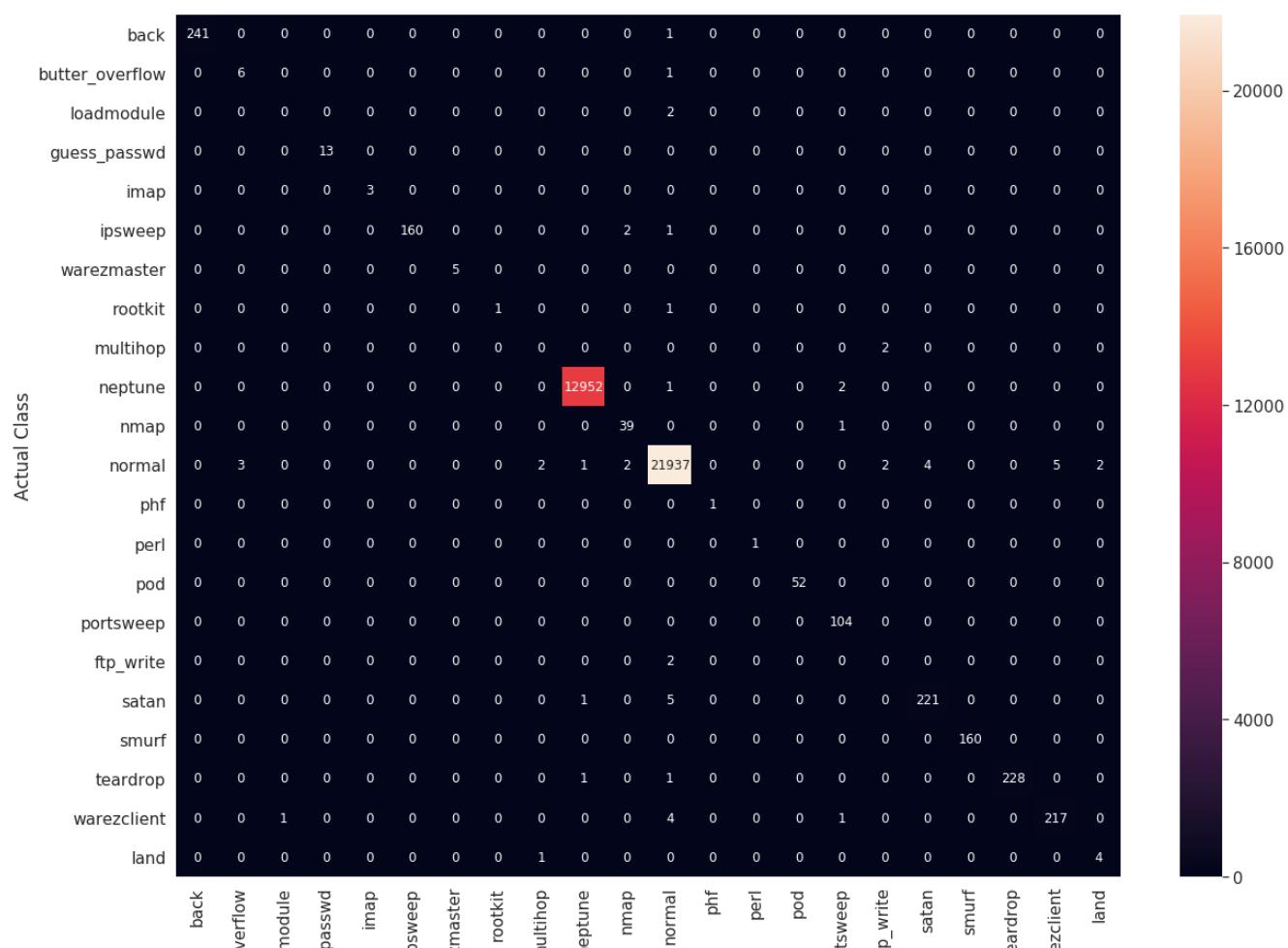
In [210]:

```
decision_tree_grid_results_tr, decision_tree_grid_results_test = model(decision_tree_grid,
X_train_1.toarray(), Y_train, X_test_1.toarray(), Y_test)
```

```
Fitting the model and prediction on train data:  
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  16.7s  
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:  32.8s finished
```

```
Completed
Time taken: 0:00:36.574308
=====
Prediction on test data:
Completed
Time taken: 0:00:00.018077
=====
Performance metrics:
=====
Confusion Matrix is:
```



```
butter_o    loadi    guess_    if      warez    rr      n      por     ft      tc      warf
Predicted Class

=====
Precision score is:
0.9986638296866037
=====
Recall score is:
0.9985713108223205
=====
F1-score is:
0.9986068375429693
```

In [211]:

```
print_grid_search_attributes(decision_tree_grid)
```

```
-----| Best Estimator |-----
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=100,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=5,
                      min_weight_fraction_leaf=0.0, presort=False,
                      random_state=None, splitter='best')
```

```
-----| Best parameters |-----
```

Parameters of best estimator :

```
{'max_depth': 100, 'min_samples_split': 5}
```

```
-----| No of CrossValidation sets |-----
```

Total number of cross validation sets: 3

```
-----| Best Score |-----
```

Average Cross Validate scores of best estimator :

```
0.9983881160190129
```

In [212]:

```
joblib.dump(decision_tree_grid.best_estimator_, 'decision_tree_gs.pkl')
```

Out[212]:

```
['decision_tree_gs.pkl']
```

In [195]:

```
dt_gs = decision_tree_grid.best_estimator_
y_tr_pred = dt_gs.predict(X_train_1.toarray())
y_test_pred = dt_gs.predict(X_test_1.toarray())
dt_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)
dt_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [196]:

```
decision_tree_grid_results_tr
```

Out [196]:

```
{'f1_score': 0.9997583211262271,  
 'precision': 0.9997729384543836,  
 'recall': 0.9997527223438258}
```

In [197]:

dt\_tpr\_fpr\_train

Out[197]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65853, 'tpr': 0.9996812095819292}
```

In [198]:

decision tree grid results test

Out [198] :

```
{'f1_score': 0.99860727686375,  
 'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,  
 'precision': 0.9986657857309603,  
 'recall': 0.9985713108223205}
```

In [199]:

dt tpr fpr test

Out[199]:

```
{'fp': 19, 'fpr': 0.001315880601149664, 'tp': 21937, 'tpr': 0.9990436287457874}
```

### ***Observations from applying Decision Tree Classifier:-***

- Out of the 21958 Normal connection points, 21934 (99.89%) were correctly classified by the Decision Tree Classifier.
  - Out of the 14439 points belonging to Bad connections, class Neptune has the highest no. of data points 12955, out of which 12953(99.98%) were classified correctly.
  - Out of the classes having very less no. of data points, class guess\_passwd was classified with (13/13) 100% accuracy, class butter\_overflow with (6/7) 85.71% accuracy, class warezmaster with(5/5) 100% accuracy, class land with(4/5) 80% accuracy, class imap with(3/3) 100% accuracy, class loadmodule with (0/2) 0% accuracy, class rootkit with(1/2) 50% accuracy, class multihop with (1/2) 50% accuracy, class ftp\_write with(0/2) 0% accuracy, class phf with (1/1) 100% accuracy and class perl with (1/1) 100% accuracy.
  - The Decision Tree Classifier was able to classify points with a higher f1 score of 0.9986 compared to all the previous Classifiers.
  - True Postives = 21937
  - TPR = 0.9990
  - False Positives = 19
  - FPR = 0.0013
  - The DT Classifier has the lowest FPR and highest TPR compared to all of the above models.
  - Thus, we can say that a non linear ML model like DT is able to learn the pattern from the data better compared to the linear classifiers.

## Model 5 :- Random Forest

In [128]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='gini', class_weight='balanced')
rf_grid = GridSearchCV(rf, param_grid=hyperparameter, cv=3, verbose=1, n_jobs=-1)
```

In [129]:

```
rf_grid_results_tr, rf_grid_results_test = model(rf_grid, X_train_1.toarray(), Y_train,
X_test_1.toarray(), Y_test)
```

Fitting the model and prediction on train data:  
Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:   51.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  7.8min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 14.6min finished
```

Completed

Time taken: 0:16:09.241562

=====

Prediction on test data:

Completed

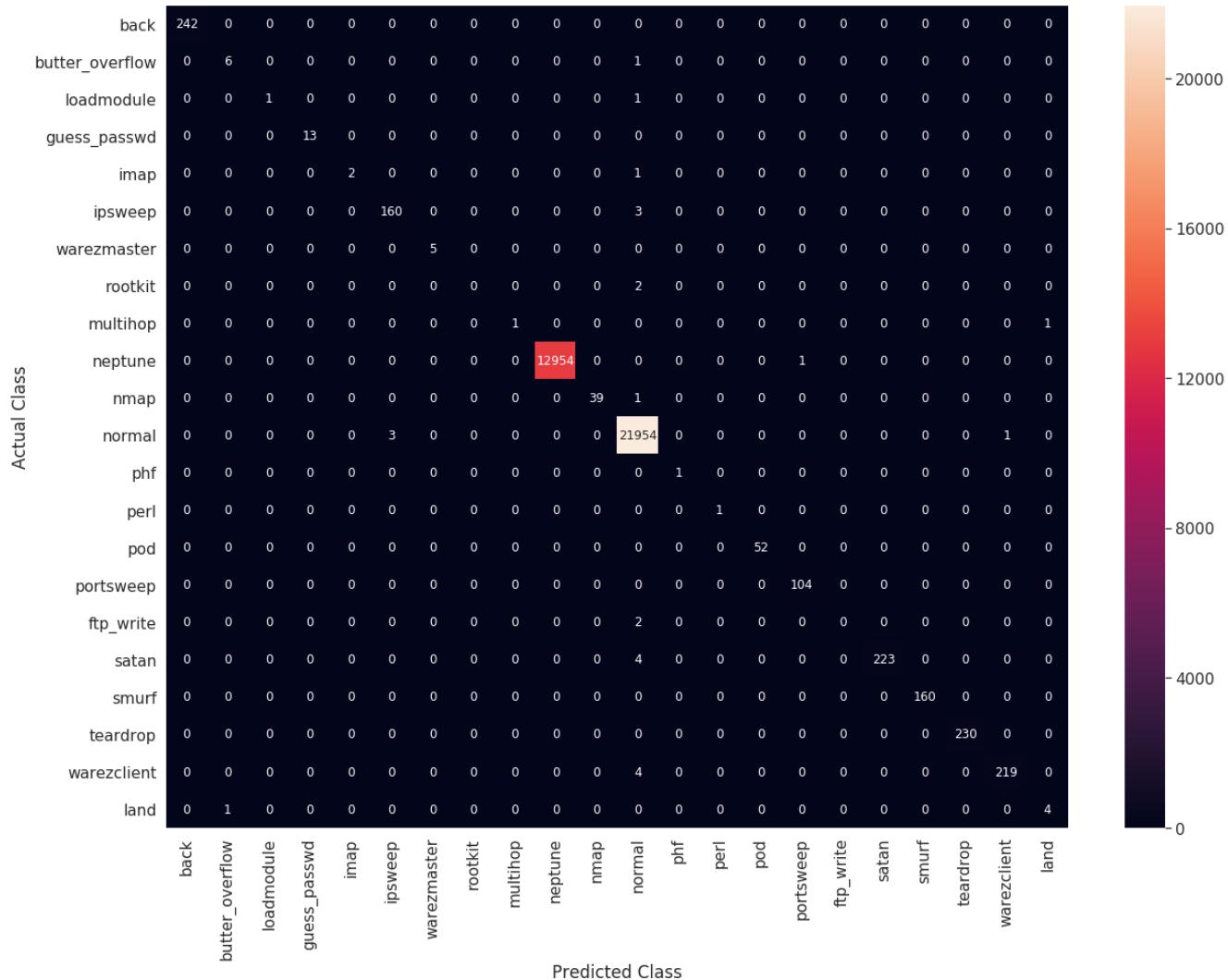
Time taken: 0:00:03.953175

=====

Performance metrics:

=====

Confusion Matrix is:



=====

Precision score is:

0.9991759996128703

=====

```
Recall score is:  
0.9992856554111602  
=====  
F1-score is:  
0.9992178456853797
```

In [136]:

```
print_grid_search_attributes(rf_grid)
```

```
-----  
|      Best Estimator      |  
-----  
  
RandomForestClassifier(bootstrap=True, class_weight='balanced',  
                      criterion='gini', max_depth=500, max_features='auto',  
                      max_leaf_nodes=None, min_impurity_decrease=0.0,  
                      min_impurity_split=None, min_samples_leaf=1,  
                      min_samples_split=10, min_weight_fraction_leaf=0.0,  
                      n_estimators=500, n_jobs=None, oob_score=False,  
                      random_state=None, verbose=0, warm_start=False)  
  
-----  
|      Best parameters      |  
-----  
Parameters of best estimator :  
  
{'max_depth': 500, 'min_samples_split': 10, 'n_estimators': 500}  
  
-----  
|      No of CrossValidation sets      |  
-----  
  
Total number of cross validation sets: 3  
  
-----  
|      Best Score      |  
-----  
  
Average Cross Validate scores of best estimator :  
  
0.9990292062387237
```

In [131]:

```
rf_gs = rf_grid.best_estimator_  
y_tr_pred = rf_gs.predict(X_train_1.toarray())  
y_test_pred = rf_gs.predict(X_test_1.toarray())  
rf_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)  
rf_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [132]:

```
rf_grid_results_tr
```

Out[132]:

```
{'f1_score': 0.9999179079366968,  
'precision': 0.9999189175382496,  
'recall': 0.9999175741146086}
```

In [133]:

```
rf_tpr_fpr_train
```

Out[133]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65866, 'tpr': 0.9998785560312111}
```

```
In [134]:  
rf_grid_results_test  
  
Out[134]:  
{'f1_score': 0.9992178456853797,  
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,  
'precision': 0.9991759996128703,  
'recall': 0.9992856554111602}
```

In [135]:

```
rf_tpr_fpr_test
```

Out[135]:

```
{'fp': 19, 'fpr': 0.001315880601149664, 'tp': 21954, 'tpr': 0.9998178340468167}
```

In [187]:

```
joblib.dump(rf_grid.best_estimator_, 'rf_gs.pkl')
```

Out[187]:

```
['rf_gs.pkl']
```

#### **Observations from applying Random Forest Classifier:-**

- Out of the 21958 Normal connection points, 21954 (99.98%) were correctly classified by the RF Classifier.
- Out of the 17409 points belonging to Bad connections, class Neptune has the highest no. of data points 12955, out of which 12954(99.99%) were classified correctly.
- Out of the classes having very less no. of data points, class guess\_passwd was classified with (13/13) 100% accuracy, class butter\_overflow with (6/7) 85.71% accuracy, class warezmaster with(5/5) 100% accuracy, class land with(4/5) 80% accuracy, class imap with(2/3) 66.67% accuracy, class loadmodule with (1/2) 50% accuracy, class rootkit with(0/2) 0% accuracy, class multihop with (1/2) 50% accuracy, class ftp\_write with(0/2) 0% accuracy, class phf with (1/1) 100% accuracy and class perl with (1/1) 100% accuracy.
- The RF Classifier was able to classify the Normal connection points with the highest f1\_score compared to all of the above classifiers.
- But this classifier was still unable to correctly classify bad connection classes like loadmodule, rootkit and ftp\_write.
- True Postives = 21954
- TPR = 0.9998
- False Positives = 19
- FPR = 0.0013
- The RF Classifier has a lower FPR compared to the DT model.
- As the train and test metrics like f1-score, tpr and fpr are almost similar, the model is NOT OVERFITTING.

## **Model\_6 :- XGBoost**

In [264]:

```
hyperparameter = {'max_depth':[2, 3, 5, 7, 10], 'n_estimators': [10, 50, 100, 200, 500]}  
  
from xgboost import XGBClassifier  
xgb = XGBClassifier(objective='multi:softprob')  
xgb_grid = RandomizedSearchCV(xgb, param_distributions=hyperparameter, cv=3, verbose=1, n_jobs=-1)
```

In [278]:

```
xgb_best = xgb_grid.best_estimator_
```

In [279]:

```
xgb_grid_results_tr, xgb_grid_results_test = model(xgb_best, X_train_1.toarray(), Y_train, X_test_1.toarray(), Y_test)
```

Fitting the model and prediction on train data:

Completed

Time taken: 0:19:06.398788

=====

Prediction on test data:

Completed

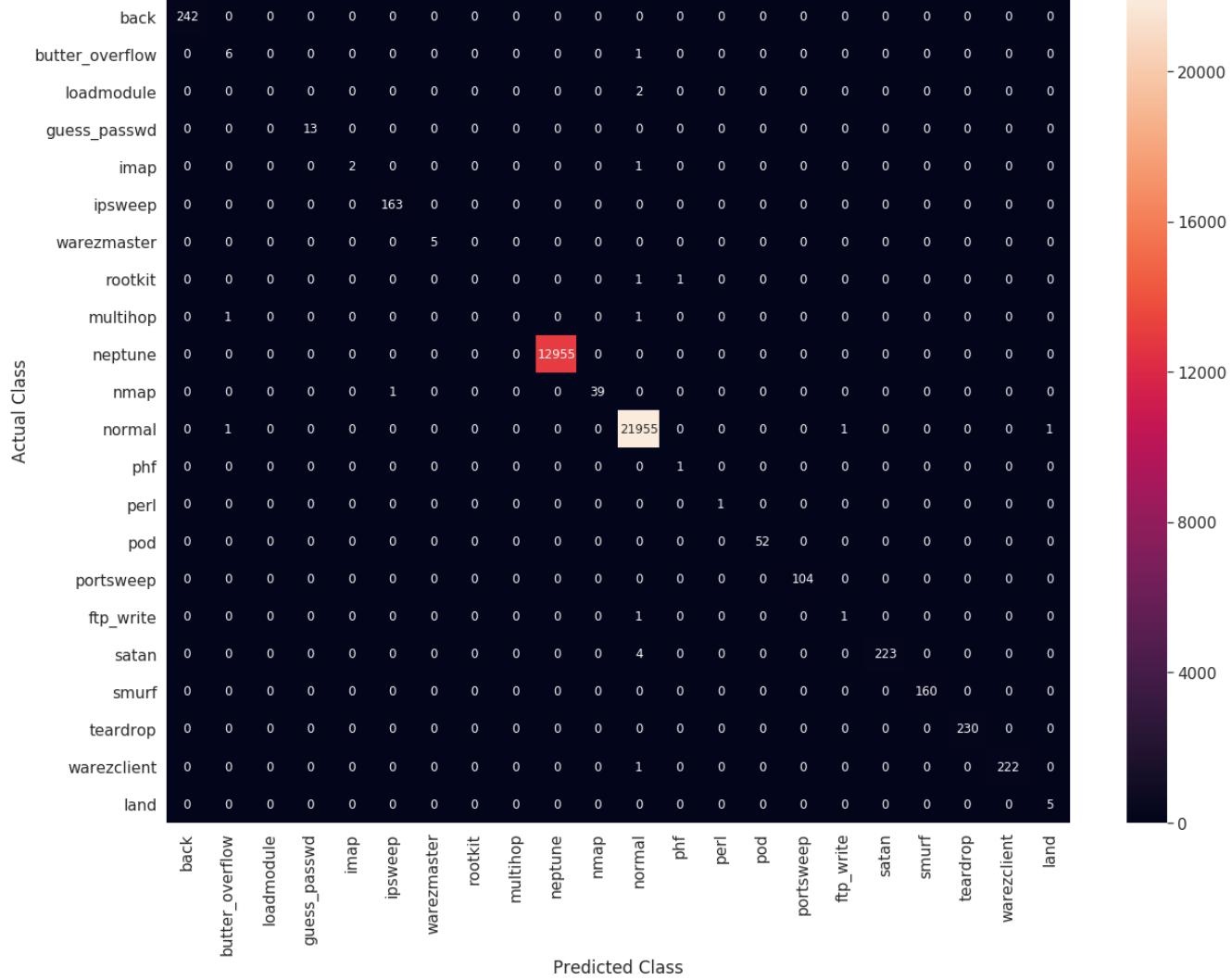
Time taken: 0:00:10.396326

=====

Performance metrics:

=====

Confusion Matrix is:



=====

Precision score is:  
0.9993660928288938

=====

Recall score is:  
0.9995054537461878

=====

F1-score is:  
0.9994282483855045

In [266]:

```
print_grid_search_attributes(xgb_grid)
```

```
|      Best Estimator      |
-----
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
|      Best parameters      |
-----
```

Parameters of best estimator :

```
{'n_estimators': 200, 'max_depth': 3}
```

```
|      No of CrossValidation sets      |
-----
```

Total number of cross validation sets: 3

```
|      Best Score      |
-----
```

Average Cross Validate scores of best estimator :

```
0.9991848995777963
```

In [267]:

```
xgb_gs = xgb_grid.best_estimator_
y_tr_pred = xgb_gs.predict(X_train_1.toarray())
y_test_pred = xgb_gs.predict(X_test_1.toarray())
xgb_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)
xgb_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [268]:

```
xgb_grid_results_tr
```

Out[268]:

```
{'f1_score': 0.9999816952580419,
 'precision': 0.9999817609521351,
 'recall': 0.9999816831365796}
```

In [269]:

```
xgb_tpr_fpr_train
```

Out[269]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65873, 'tpr': 0.9999848195039014}
```

In [270]:

```
xgb_grid_results_test
```

Out[270]:

```
{'f1_score': 0.9994282483855045,
 'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,
 'precision': 0.9993660928288938,
 'recall': 0.9995054537461878}
```

```
In [271]:
```

```
xgb_tpr_fpr_test
```

```
Out[271]:
```

```
{'fp': 12,  
 'fpr': 0.0008310824849366299,  
 'tp': 21955,  
 'tpr': 0.9998633755351125}
```

```
In [272]:
```

```
joblib.dump(xgb_grid.best_estimator_, 'xgb_gs.pkl')
```

```
Out[272]:
```

```
['xgb_gs.pkl']
```

#### ***Observations from applying XG Boost Classifier:-***

- Out of the 21958 Normal connection points, 21955 (99.98%) were correctly classified by the XGB Classifier.
- Out of the 17409 points belonging to Bad connections, class Neptune has the highest no. of data points 12955, out of which 12955(100.0%) were classified correctly.
- Out of the classes having very less no. of data points, class guess\_passwd was classified with (12/13) 92.30% accuracy, class butter\_overflow with (7/7) 100.0% accuracy, class warezmaster with(4/5) 80% accuracy, class land with(5/5) 100% accuracy, class imap with(3/3) 100% accuracy, class loadmodule with (0/2) 0% accuracy, class rootkit with(0/2) 0% accuracy, class multihop with (0/2) 0% accuracy, class ftp\_write with(1/2) 50% accuracy, class phf with (1/1) 100% accuracy and class perl with (1/1) 100% accuracy.
- The XGB Classifier was able to classify different classes with the highest f1-score(0.9994) compared to all of the above models.
- True Positives = 21955
- TPR = 0.9998
- False Positives = 12
- FPR = 0.00083
- The XGB Classifier has the highest TPR and lowest FPR compared to all of the previous models.
- As the train and test metrics like f1-score, tpr and fpr are almost similar, the model is NOT OVERFITTING.

#### ***Observation from ALL of the above classifiers:-***

- If we consider NORMAL connection points as 1 class and points belonging to all the other 22 BAD connection classes as the 2nd class, then XGB Classifier is the best classifier as it has a TPR and FPR of 0.9998 and 0.0011.
- Although XGB classifier had a better f1\_score than the RF classifier, if we go into details of the confusion matrix scores, we can observe that both classifiers have performed similarly on the different categories of attacks on our dataset.
- The RF Classifier has TPR and FPR of 0.9998 and 0.0013.
- The overall time taken for training + evaluation was less for the RF and DT classifiers compared to the XGB classifier.

***As the train and test metrics like f1-score, tpr and fpr have almost similar scores for train and test datasets, the models are NOT OVERFITTING.***

- A common pattern shown by all of the classifiers is that classes rootkit, ftp\_write and loadmodule were misclassified as class Normal by most of the classifiers.
- We will add more features in our dataset and try to improve the classifier performance.
- As DT, RF & XGB had the best performance, we will use these 3 classifiers ahead on the existing + feature engineered data.

## Adding new features:-

### 1. Clustering features (using MiniBatchKmeans):-

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html>

**KMeans with n=23 (for Multiclass Classification):-**

In [280]:

```
data = pd.read_pickle('data.pkl')
print('Shape of our dataset', data.shape)
```

Shape of our dataset (145586, 42)

In [281]:

```
from sklearn.cluster import MiniBatchKMeans
import numpy as np
```

In [282]:

```
kmeans_1 = MiniBatchKMeans(n_clusters=23, random_state=0, batch_size=128, max_iter=1000) #Multiclass
classification
kmeans_1.fit(X_train_1)
```

Out[282]:

```
MiniBatchKMeans(batch_size=128, compute_labels=True, init='k-means++',
init_size=None, max_iter=1000, max_no_improvement=10,
n_clusters=23, n_init=3, random_state=0,
reassignment_ratio=0.01, tol=0.0, verbose=0)
```

In [283]:

```
train_cluster_1 = kmeans_1.predict(X_train_1)
test_cluster_1 = kmeans_1.predict(X_test_1)
```

In [284]:

```
print('Length of train cluster is:', len(train_cluster_1))
train_cluster_1
```

Length of train cluster is: 109189

Out[284]:

```
array([14,  5,  1, ...,  1,  7,  2], dtype=int32)
```

In [285]:

```
print('Length of test cluster is', len(test_cluster_1))
test_cluster_1
```

Length of test cluster is 36397

Out[285]:

```
array([ 1,  3,  2, ...,  2, 14,  1], dtype=int32)
```

In [286]:

```
train_cluster_1 = np.array([train_cluster_1])
```

```
train_cluster_1.shape
```

Out[286]:

```
(1, 109189)
```

In [287]:

```
test_cluster_1 = np.array([test_cluster_1])
test_cluster_1.shape
```

Out[287]:

```
(1, 36397)
```

### **KMeans with n=2 (for solving Binary Classification problem to improve TPR and reduce FPR):-**

In [119]:

```
kmeans_2 = MiniBatchKMeans(n_clusters=2, random_state=0, batch_size=128, max_iter=1000) #Binary Classification
kmeans_2.fit(X_train_1)
```

Out[119]:

```
MiniBatchKMeans(batch_size=128, compute_labels=True, init='k-means++',
                 init_size=None, max_iter=1000, max_no_improvement=10,
                 n_clusters=2, n_init=3, random_state=0, reassignment_ratio=0.01,
                 tol=0.0, verbose=0)
```

In [120]:

```
train_cluster_2 = kmeans_2.predict(X_train_1)
test_cluster_2 = kmeans_2.predict(X_test_1)
```

In [121]:

```
print('Length of train cluster is:', len(train_cluster_2))
train_cluster_2
```

Length of train cluster is: 109189

Out[121]:

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [122]:

```
print('Length of test cluster is', len(test_cluster_2))
test_cluster_2
```

Length of test cluster is 36397

Out[122]:

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [123]:

```
train_cluster_2 = np.array([train_cluster_2])
train_cluster_2.shape
```

Out[123]:

```
(1, 109189)
```

```
In [124]:
```

```
test_cluster_2 = np.array([test_cluster_2])
test_cluster_2.shape
```

```
Out[124]:
```

```
(1, 36397)
```

## 2. PCA features :-

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

- We will add top 5 PCA features to our dataset. (We can add more or less and test if they improve the performance)

```
In [125]:
```

```
from sklearn.decomposition import PCA
```

```
In [126]:
```

```
pca = PCA(n_components=5)
pca.fit(X_train_1.toarray())
pca_train = pca.transform(X_train_1.toarray())
pca_test = pca.transform(X_test_1.toarray())
```

```
In [127]:
```

```
print(pca_train.shape)
print(pca_test.shape)
```

```
(109189, 5)
(36397, 5)
```

## 3. Additional feature engineering :-

### **src\_bytes + dst\_bytes**

```
In [128]:
```

```
feature_src_dst_1 = src_bytes1 + dst_bytes1
feature_src_dst_2 = src_bytes2 + dst_bytes2
```

```
In [129]:
```

```
feature_src_dst_1.shape
```

```
Out[129]:
```

```
(109189, 1)
```

### **src\_bytes - dst\_bytes**

```
In [130]:
```

```
feature_src_dst_3 = abs(src_bytes1 - dst_bytes1)
feature_src_dst_4 = abs(src_bytes2 - dst_bytes2)
```

```
In [131]:
```

```
feature_src_dst_3.shape
```

```
Out[131]:
```

```
(109189, 1)
```

**same\_srv\_rate + diff\_srv\_rate :-**

```
In [132]:
```

```
feature_5 = same_srv_rate1 + diff_srv_rate1  
feature_6 = same_srv_rate2 + diff_srv_rate2
```

```
In [133]:
```

```
feature_5.shape
```

```
Out[133]:
```

```
(109189, 1)
```

**dst\_host\_same\_srv\_rate + dst\_host\_diff\_srv\_rate :-**

```
In [134]:
```

```
feature_7 = dst_host_same_srv_rate1 + dst_host_diff_srv_rate1  
feature_8 = dst_host_same_srv_rate2 + dst_host_diff_srv_rate2
```

```
In [135]:
```

```
feature_7.shape
```

```
Out[135]:
```

```
(109189, 1)
```

## Adding clustering and PCA features to our dataset with the additional 4 features:-

```
In [ ]:
```

```
X_train_2 = hstack((X_train_1, pca_train, train_cluster_1.T, train_cluster_2.T, feature_src_dst_1,  
feature_src_dst_3, feature_5, feature_7))
```

```
In [ ]:
```

```
X_test_2 = hstack((X_test_1, pca_test, test_cluster_1.T, test_cluster_2.T, feature_src_dst_2, featu  
re_src_dst_4, feature_6, feature_8))
```

```
In [307]:
```

```
print('Train data:')
```

```
print(X_train_2.shape)
```

```
print('='*30)
```

```
print('Test data:')
```

```
print(X_test_2.shape)
```

```
Train data:  
(109189, 128)  
=====
```

```
Test data:  
(36397, 128)
```

```
In [308]:
```

```
joblib.dump(X_train_2, 'X_train_2.pkl')
```

```
joblib.dump(X_test_2, 'X_test_2.pkl')
```

Out[308]:

```
['X_test_2.pkl']
```

## Model\_1 :- Decision Tree

In [213]:

```
hyperparameter = {'max_depth':[5, 10, 20, 50, 100, 500], 'min_samples_split':[5, 10, 100, 500]}

from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(criterion='gini', splitter='best', class_weight='balanced')
decision_tree_grid = GridSearchCV(decision_tree, param_grid=hyperparameter, cv=5, verbose=1, n_jobs=-1)
```

In [214]:

```
decision_tree_results2_tr, decision_tree_results2_test = model(decision_tree_grid,
X_train_2.toarray(), Y_train, X_test_2.toarray(), Y_test)
```

Fitting the model and prediction on train data:

Fitting 5 folds for each of 24 candidates, totalling 120 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:   23.4s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:  1.5min finished
```

Completed

Time taken: 0:01:35.920685

=====

Prediction on test data:

Completed

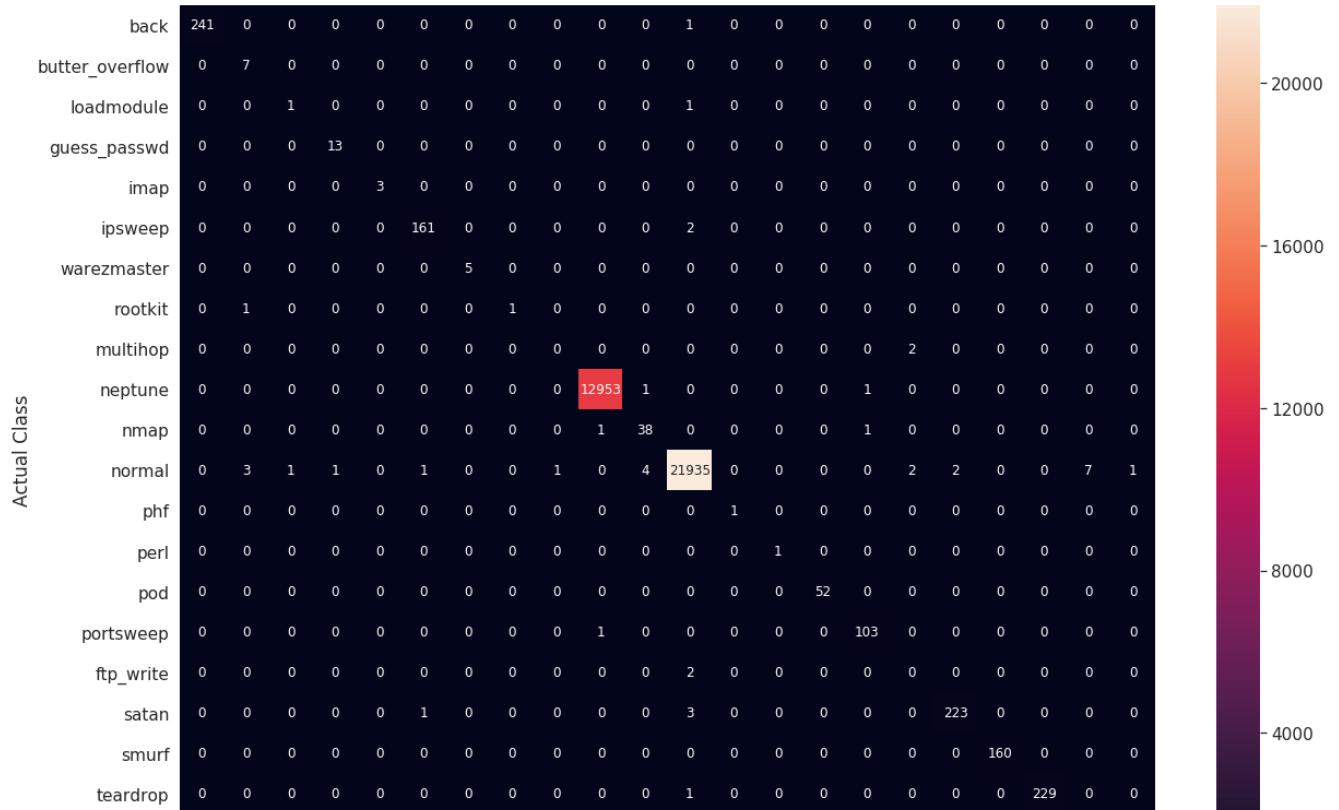
Time taken: 0:00:00.019800

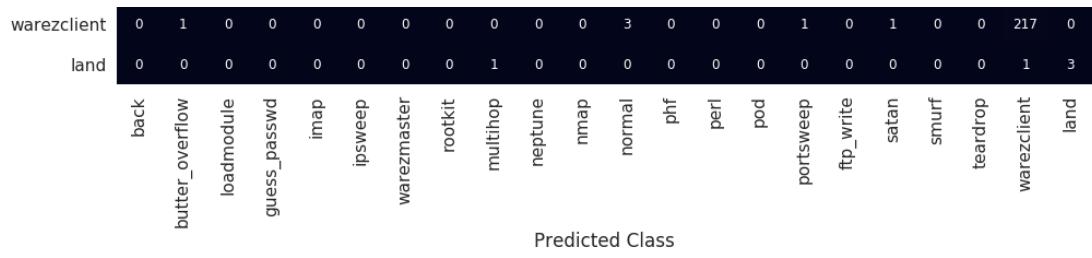
=====

Performance metrics:

=====

Confusion Matrix is:





0

```
=====
Precision score is:
0.9987461086510298
=====
Recall score is:
0.9986262604060774
=====
F1-score is:
0.9986676356047771
```

In [215]:

```
print_grid_search_attributes(decision_tree_grid)
```

```
-----
|      Best Estimator      |
-----
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=50,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

```
-----
|      Best parameters      |
-----
```

Parameters of best estimator :

```
{'max_depth': 50, 'min_samples_split': 5}
```

```
-----
|      No of CrossValidation sets      |
-----
```

Total number of cross validation sets: 5

```
-----
|      Best Score      |
-----
```

Average Cross Validate scores of best estimator :

```
0.9982507395433606
```

In [216]:

```
dt_gs = decision_tree_grid.best_estimator_
y_tr_pred = dt_gs.predict(X_train_2.toarray())
y_test_pred = dt_gs.predict(X_test_2.toarray())
dt_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)
dt_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [217]:

```
decision_tree_results2_tr
```

Out[217]:

```
{'f1_score': 0.9998475192034998,
```

```
'precision': 0.9998556316446825,  
'recall': 0.9998443066609274}
```

In [218]:

```
dt_tpr_fpr_train
```

Out[218]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65859, 'tpr': 0.9997722925585208}
```

In [219]:

```
decision_tree_results2_test
```

Out[219]:

```
{'f1_score': 0.9986676356047771,  
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,  
'precision': 0.9987461086510298,  
'recall': 0.9986262604060774}
```

In [220]:

```
dt_tpr_fpr_test
```

Out[220]:

```
{'fp': 13,  
'fpr': 0.0009003393586813492,  
'tp': 21935,  
'tpr': 0.9989525457691958}
```

In [221]:

```
joblib.dump(decision_tree_grid.best_estimator_, 'dt2.pkl')
```

Out[221]:

```
['dt2.pkl']
```

#### **Observation from DT-2 Classifier :-**

- The Decision Tree Classifier was able to classify points with a high f1 score of ~0.9986 which is similar to the performance of the 1st Decision Tree Classifier.
- True Postives = 21935
- TPR = 0.9989
- False Positives = 13
- FPR = 0.00090
- This DT Classifier has comparatively lower TPR and FPR than DT-1 model.

## **Model\_2 :- Random Forest**

In [152]:

```
hyperparameter = {'max_depth':[5, 10, 100, 500, 1000], 'n_estimators': [5, 10, 50, 100, 500],  
                  'min_samples_split':[5, 10, 100, 500]}  
  
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier(criterion='gini', class_weight='balanced')  
rf_grid = GridSearchCV(rf, param_grid=hyperparameter, cv=3, verbose=1, n_jobs=-1)
```

In [153]:

```
rf_grid_results_tr, rf_grid_results_test = model(rf_grid, X_train_2.toarray(), Y_train,
X_test_2.toarray(), Y_test)
```

Fitting the model and prediction on train data:  
Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 20.9min finished
```

Completed  
Time taken: 0:21:22.481366

=====

Prediction on test data:

Completed

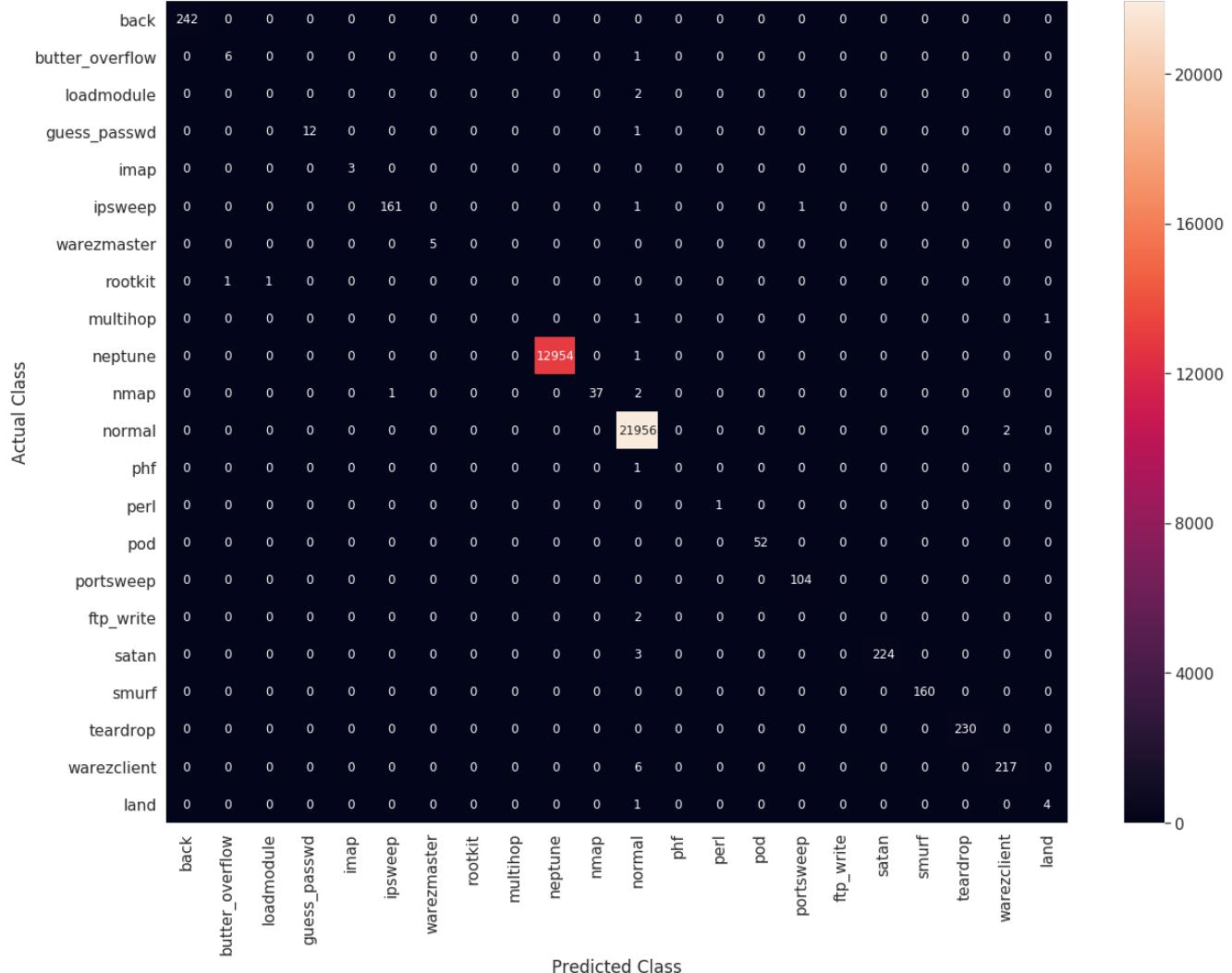
Time taken: 0:00:00.808180

=====

Performance metrics:

=====

Confusion Matrix is:



=====

Precision score is:

0.998983071168219

=====

Recall score is:

0.9992032310355249

=====

F1-score is:

0.9990899964742136

In [154]:

```
print_grid_search_attributes(rf_grid)
```

```
-----  
|      Best Estimator      |  
-----
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',  
                      criterion='gini', max_depth=1000, max_features='auto',  
                      max_leaf_nodes=None, min_impurity_decrease=0.0,  
                      min_impurity_split=None, min_samples_leaf=1,  
                      min_samples_split=10, min_weight_fraction_leaf=0.0,  
                      n_estimators=100, n_jobs=None, oob_score=False,  
                      random_state=None, verbose=0, warm_start=False)
```

```
-----  
|      Best parameters      |  
-----
```

Parameters of best estimator :

```
{'n_estimators': 100, 'max_depth': 1000, 'min_samples_split': 10}
```

```
-----  
|      No of CrossValidation sets      |  
-----
```

Total number of cross validation sets: 3

```
-----  
|      Best Score      |  
-----
```

Average Cross Validate scores of best estimator :

```
0.9991207905558253
```

In [155]:

```
rf_gs = rf_grid.best_estimator_  
y_tr_pred = rf_gs.predict(X_train_2.toarray())  
y_test_pred = rf_gs.predict(X_test_2.toarray())  
rf_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)  
rf_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [156]:

```
rf_grid_results_tr
```

Out[156]:

```
{'f1_score': 0.999945245227103,  
'precision': 0.9999458163014309,  
'recall': 0.999945049409739}
```

In [157]:

```
rf_tpr_fpr_train
```

Out[157]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65868, 'tpr': 0.9999089170234083}
```

In [158]:

```
rf_grid_results_test
```

Out[158]:

```
{'f1_score': 0.9990899964742136,
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,
'precision': 0.998983071168219,
'recall': 0.9992032310355249}
```

In [159]:

```
rf_tpr_fpr_test
```

Out[159]:

```
{'fp': 22,
'fpr': 0.0015236512223838216,
'tp': 21956,
'tpr': 0.9999089170234083}
```

In [ ]:

```
joblib.dump(rf_grid.best_estimator_, 'rf2.pkl')
```

#### **Observation from RF\_2 Classifier :-**

- This RF Classifier has a lower f1-score(0.9990) compared to the RF\_1 Classifier.
- True Postives = 21956
- TPR = 0.9999
- False Positives = 22
- FPR = 0.0015
- This RF Classifier has a higher TPR and higher FPR than the RF\_1 model, which indicates that adding new features have improved the classification of Good connections but has also increased the misclassification of Bad category connections by some amount.
- The no. of false positives are more than the DT\_2 classifier which is a Drawback of this model.

## **Model\_3 :- XG Boost**

- We will apply the XG Boost classifier on the same dataset and check if it shows any improvement.

In [309]:

```
hyperparameter = {'max_depth':[2, 3, 5, 7, 10], 'n_estimators': [10, 50, 100, 200, 500]}

from xgboost import XGBClassifier
xgb = XGBClassifier(objective='multi:softprob', n_jobs=-1)
xgb_grid = RandomizedSearchCV(xgb, param_distributions=hyperparameter, cv=3, verbose=1, n_jobs=-1)
```

In [310]:

```
xgb_grid_results_tr, xgb_grid_results_test = model(xgb_grid, X_train_2.toarray(), Y_train, X_test_2
.toarray(), Y_test)
```

Fitting the model and prediction on train data:

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 76.0min finished
```

Completed

Time taken: 2:00:09.666172

=====

Prediction on test data:

Completed

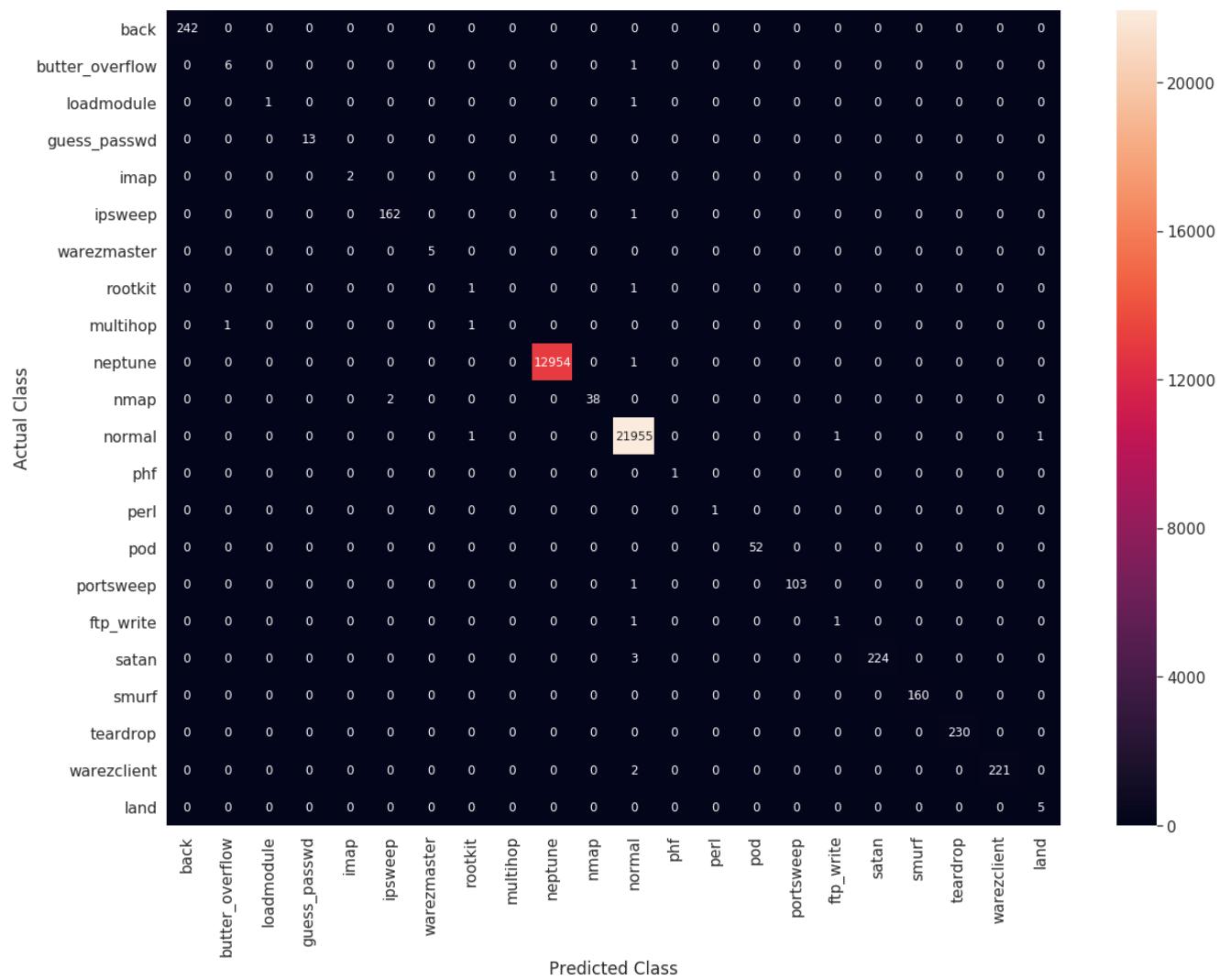
Time taken: 0:00:24.416710

=====

Performance metrics:

=====

Confusion Matrix is:



=====

Precision score is:

0.9994189203758796

=====

Recall score is:

0.999450504162431

=====

F1-score is:

0.9994241935579559

In [311]:

```
print_grid_search_attributes(xgb_grid)
```

```
-----| Best Estimator |-----
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
               colsample_bynode=1, colsample_bytree=1, gamma=0,
               learning_rate=0.1, max_delta_step=0, max_depth=3,
               min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
               nthread=None, objective='multi:softprob', random_state=0,
               reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
               silent=None, subsample=1, verbosity=1)
```

```
-----| Best parameters |-----
```

```
-----  
Parameters of best estimator :  
  
{'n_estimators': 500, 'max_depth': 3}  
-----  
| No of CrossValidation sets |  
-----  
Total numbre of cross validation sets: 3  
-----  
| Best Score |  
-----  
Average Cross Validate scores of best estimator :  
0.9991574242826659
```

In [312]:

```
xgb_gs = xgb_grid.best_estimator_  
y_tr_pred = xgb_gs.predict(X_train_2.toarray())  
y_test_pred = xgb_gs.predict(X_test_2.toarray())  
xgb_tpr_fpr_train = tpr_fpr_func(Y_train, y_tr_pred)  
xgb_tpr_fpr_test = tpr_fpr_func(Y_test, y_test_pred)
```

In [313]:

```
xgb_grid_results_tr
```

Out[313]:

```
{'f1_score': 0.9999816952580419,  
'precision': 0.9999817609521351,  
'recall': 0.9999816831365796}
```

In [314]:

```
xgb_tpr_fpr_train
```

Out[314]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65873, 'tpr': 0.9999848195039014}
```

In [315]:

```
xgb_grid_results_test
```

Out[315]:

```
{'f1_score': 0.9994241935579559,  
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,  
'precision': 0.9994189203758796,  
'recall': 0.999450504162431}
```

In [316]:

```
xgb_tpr_fpr_test
```

Out[316]:

```
{'fp': 12,  
'fpr': 0.0008310824849366299,  
'tp': 21955,  
'tpr': 0.9998633755351125}
```

In [ ]:

```
joblib.dump(xgb_grid.best_estimator_, 'xgb2_gs.pkl')
```

#### **Observation from XGBoost\_2 Classifier :-**

- This XG Boost Classifier was able to classify points with better accuracy of ~99.94 and high f1 score of ~0.9994 which is similar to the performance of the 1st XGB Classifier.
- True Positives = 21955
- TPR = 0.9998
- False Positives = 12
- FPR = 0.00083
- This XGB Classifier has the highest TPR of (99.98%), but the FPR(0.00083) is same as the XGB1 model(0.0008).

#### **Important Observation from the above 3 models:-**

- From the performance scores we have obtained from the above 3 models, we can conclude that adding new features has increased the TPR score as the no. correct classification of "Normal" class points has increased, but no significant improvement has been obtained on the FPR score for all the 3 models which is a drawback.

### **Summarizing Results of Multiclass Classification problem:-**

In [318]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ['Model', 'Train f1-score', 'Train TPR', 'Train FPR', 'Test f1-score', 'Test TPR', 'Test FPR']
x.add_row(['Naive Bayes', '0.9671', '99.40%', '5.13%', '0.9679', '99.34%', '4.91%'])
x.add_row(['Logistic Regression', '0.9813', '99.81%', '2.95%', '0.9819', '99.81%', '2.76%'])
x.add_row(['Support Vector Machine', '0.9967', '99.87%', '0.48%', '0.9966', '99.87%', '0.43%'])
x.add_row(['Decision Tree - 1', '0.9997', '99.96%', '0.0%', '0.9986', '99.90%', '0.13%'])
x.add_row(['Random Forest - 1', '0.9999', '99.98%', '0.0%', '0.9992', '99.98%', '0.13%'])
x.add_row(['XG Boost - 1', '0.9999', '100.0%', '0.0%', '0.9994', '99.98%', '0.083%'])
x.add_row(['Decision Tree - 2', '0.9998', '99.97%', '0.0%', '0.9986', '99.89%', '0.09%'])
x.add_row(['Random Forest - 2', '0.9999', '99.99%', '0.0%', '0.9990', '99.99%', '0.15%'])
x.add_row(['XG Boost - 2', '0.9999', '99.99%', '0.0%', '0.9994', '99.98%', '0.083%'])

print(x)

+-----+-----+-----+-----+-----+-----+
|      Model      | Train f1-score | Train TPR | Train FPR | Test f1-score | Test TPR | Test FPR |
+-----+-----+-----+-----+-----+-----+
|      Naive Bayes      | 0.9671 | 99.40% | 5.13% | 0.9679 | 99.34% | 4.91% |
| Logistic Regression | 0.9813 | 99.81% | 2.95% | 0.9819 | 99.81% | 2.76% |
| Support Vector Machine | 0.9967 | 99.87% | 0.48% | 0.9966 | 99.87% | 0.43% |
| Decision Tree - 1 | 0.9997 | 99.96% | 0.0% | 0.9986 | 99.90% | 0.13% |
| Random Forest - 1 | 0.9999 | 99.98% | 0.0% | 0.9992 | 99.98% | 0.13% |
| XG Boost - 1 | 0.9999 | 100.0% | 0.0% | 0.9994 | 99.98% | 0.083% |
| Decision Tree - 2 | 0.9998 | 99.97% | 0.0% | 0.9986 | 99.89% | 0.09% |
| Random Forest - 2 | 0.9999 | 99.99% | 0.0% | 0.9990 | 99.99% | 0.15% |
| XG Boost - 2 | 0.9999 | 99.99% | 0.0% | 0.9994 | 99.98% | 0.083% |
```

```
--+
```

- All the models have very close performance scores on train and test data, thus they are not OVERFITTING.

- The model XG Boost\_1 is our best model for intrusion detection as it has highest Test f1-score 0.9994 and TPR 99.98% as well as the least FPR of 0.08%.

## (II) Binary Classification :-

In this section, we will solve the Intrusion detection problem as a Binary Classification problem, where we will consider points belonging to class "Normal" as one class(Positive Class) and points belonging to the remaining 22 classes as the second class(Negative Class).

By formulating this problem as a binary classification problem, we will re-apply the models and try to improve our TPR and FPR.

The reason we are converting this problem to a binary classification problem is because organisations are more concerned about Normal and Bad connections getting classified correctly rather than each of the bad categories getting misclassified so that no Bad connections are allowed to gain access to the Internal network of the organisation by getting misclassified as a Normal connection which may otherwise result in a security threat.

In [138]:

```
data.head()
```

Out[138]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_s...
0	0	tcp	http	SF	181	5450	0	0	0	0	...	9	
1	0	tcp	http	SF	239	486	0	0	0	0	...	19	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	29	
3	0	tcp	http	SF	219	1337	0	0	0	0	...	39	
4	0	tcp	http	SF	217	2032	0	0	0	0	...	49	

5 rows × 42 columns

In [139]:

```
intrusion_binary = []
for i in data['intrusion_type'].values:
    if i=='normal.':
        intrusion_binary.append(1)
    else:
        intrusion_binary.append(0)
print(len(intrusion_binary))
print(intrusion_binary[:10])
```

145586

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

In [140]:

```
data['intrusion_binary'] = intrusion_binary
data.head()
```

Out[140]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_h...
0	0	tcp	http	SF	181	5450	0	0	0	0	...	1.0	
1	0	tcp	http	SF	239	486	0	0	0	0	...	1.0	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	1.0	

3	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_hc
4	0	tcp	http	SF	217	2032	0	0	0	0	...	1.0	

5 rows × 43 columns

In [141]:

```
data.drop('intrusion_type', axis=1, inplace=True)
print('Shape of the data is:')
print(data.shape)
print('*'*80)
print('Features of the dataset:')
print(data.columns)
```

Shape of the data is:

(145586, 42)

=====

Features of the dataset:

```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate', 'intrusion_binary'],
      dtype='object')
```

## 1. Exploratory Data Analysis :-

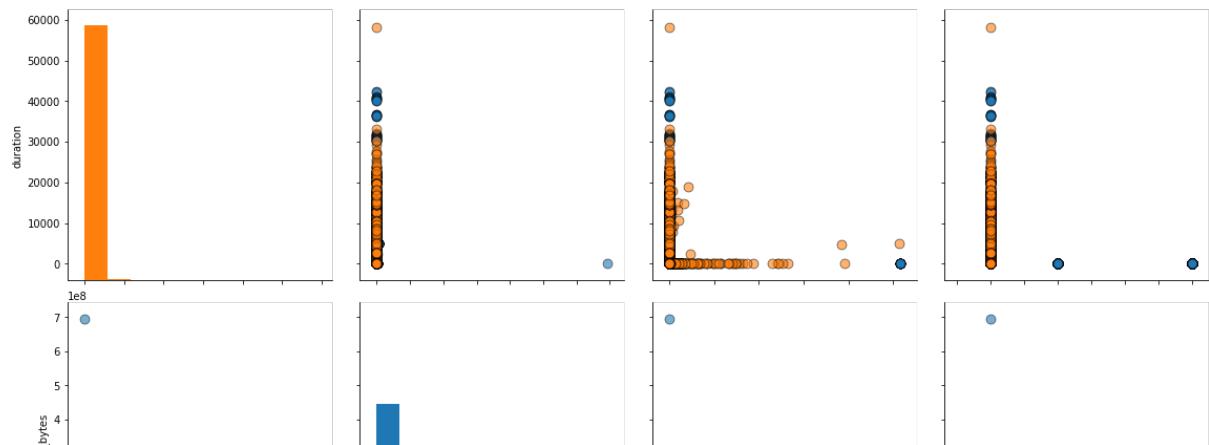
### Bivariate Analysis :-

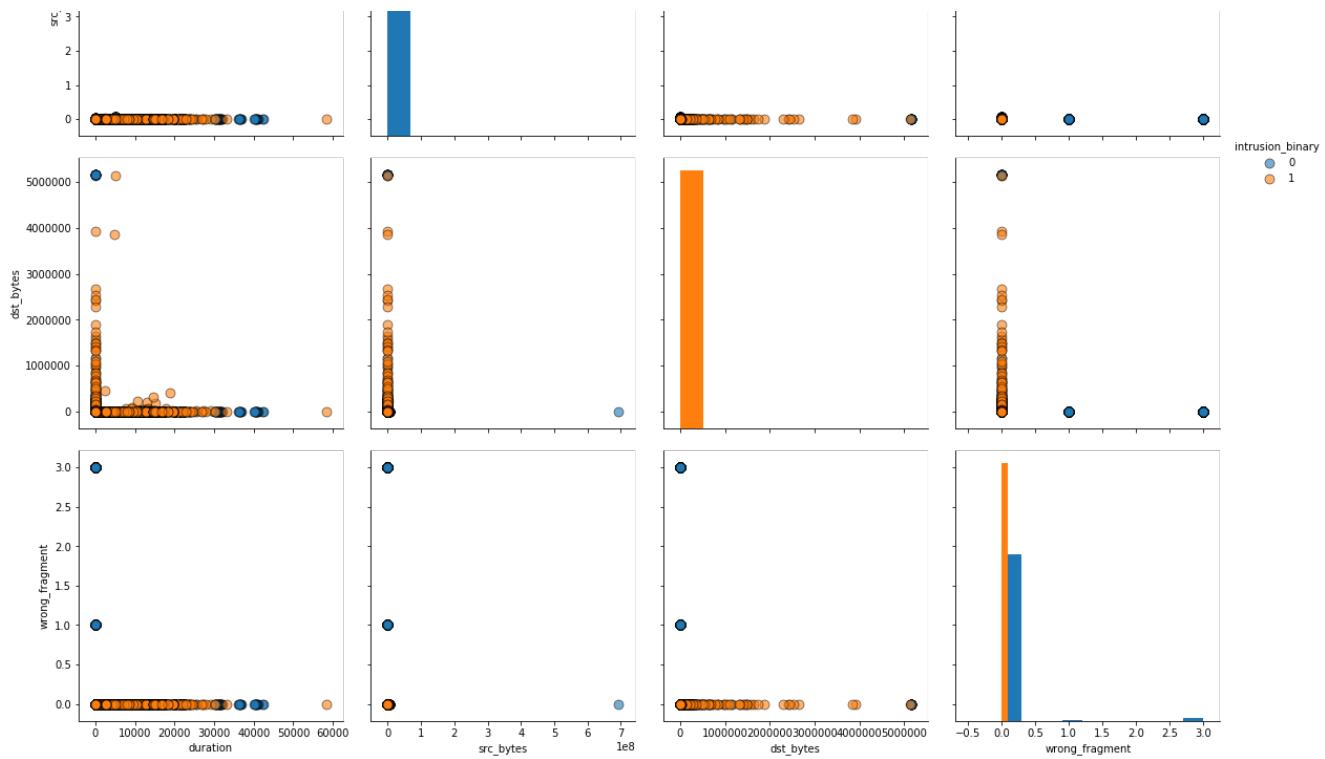
In [48]:

```
def pairplot(data, label, features=[]):
    """
    This function creates pairplot taking 4 features from our dataset as default parameters along
    with the output variable
    """
    sns.pairplot(data, hue=label, height=4, diag_kind='hist', vars=features,
                 plot_kws={'alpha':0.6, 's':80, 'edgecolor':'k'})
```

In [37]:

```
pairplot(data, 'intrusion_binary', features=['duration', 'src_bytes', 'dst_bytes', 'wrong_fragment'])
```





In [39]:

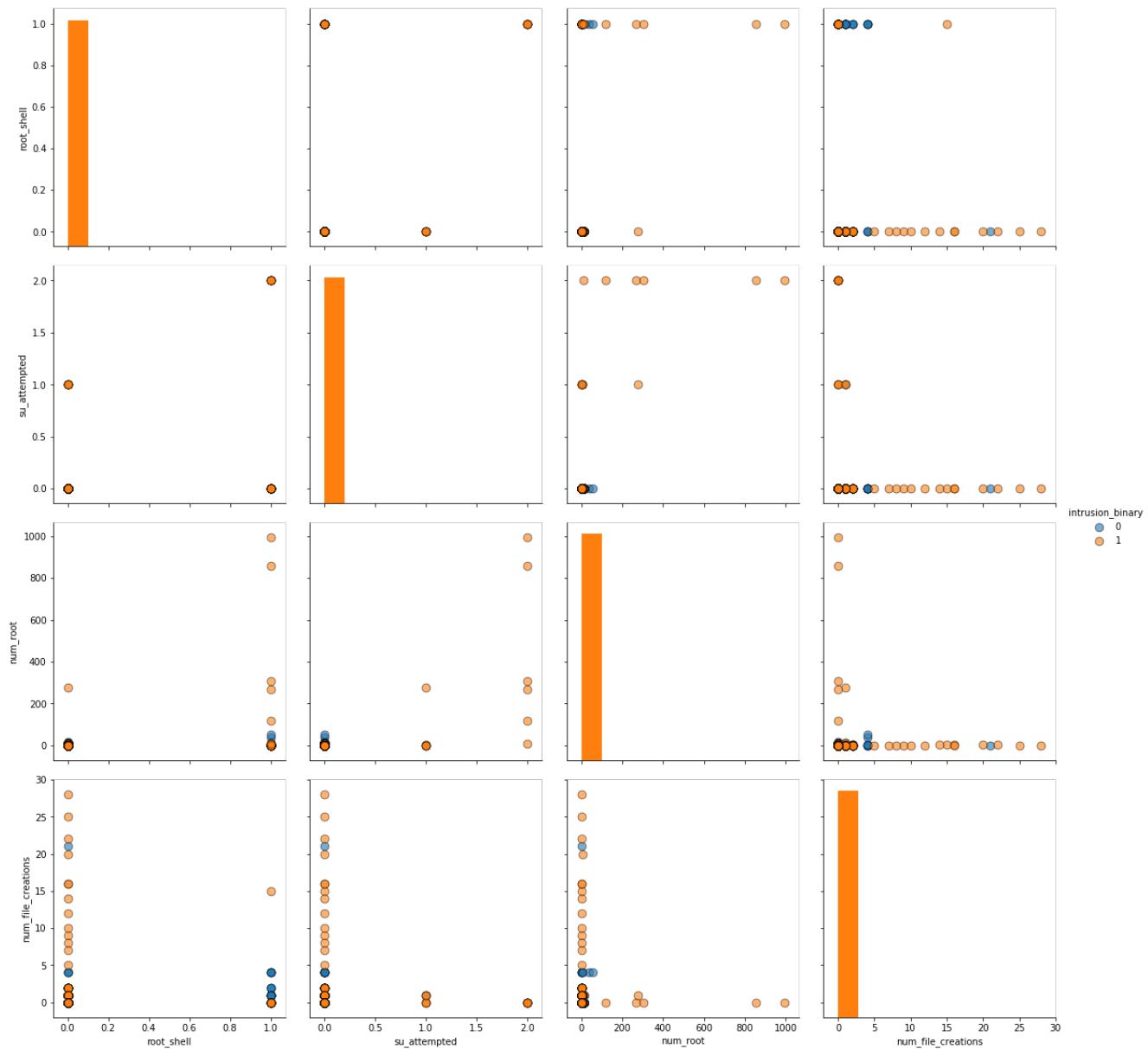
```
pairplot(data, 'intrusion_binary', features=['urgent', 'hot', 'num_failed_logins', 'num_compromised'])
```





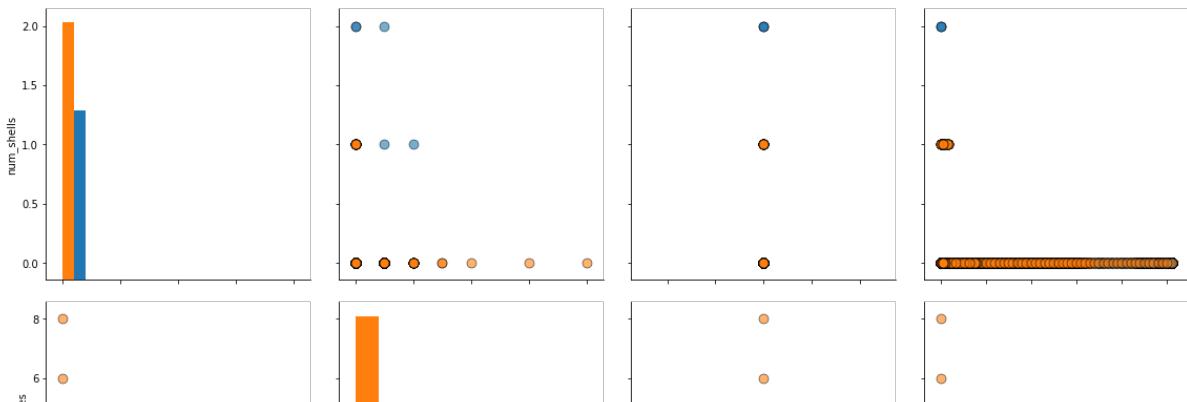
In [41]:

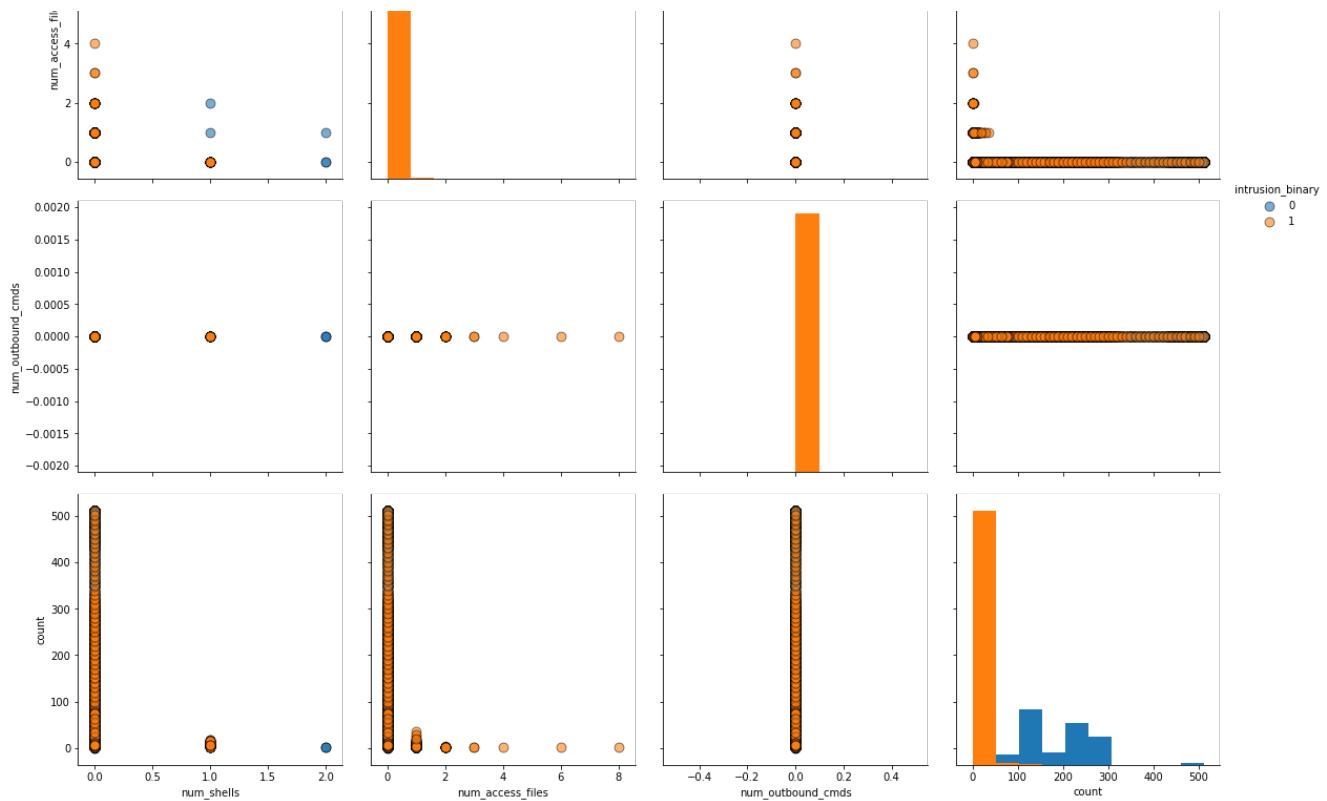
```
pairplot(data, 'intrusion_binary',
features=['root_shell','su_attempted','num_root','num_file_creations'])
```



In [42]:

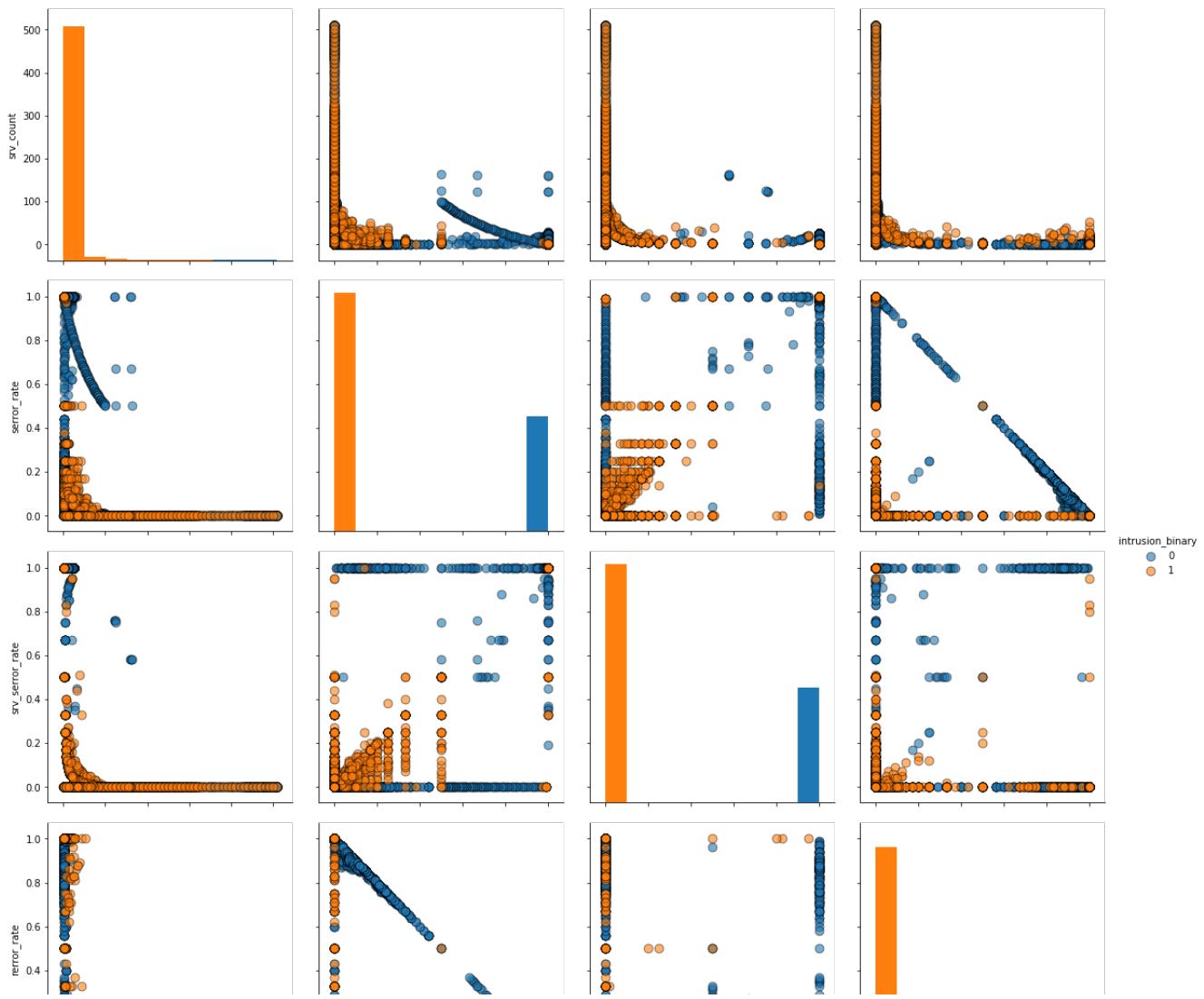
```
pairplot(data, 'intrusion_binary',
features=['num_shells','num_access_files','num_outbound_cmds','count'])
```

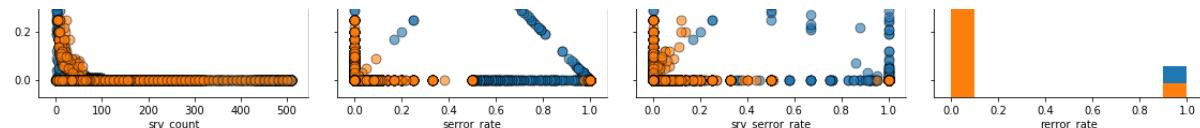




In [44]:

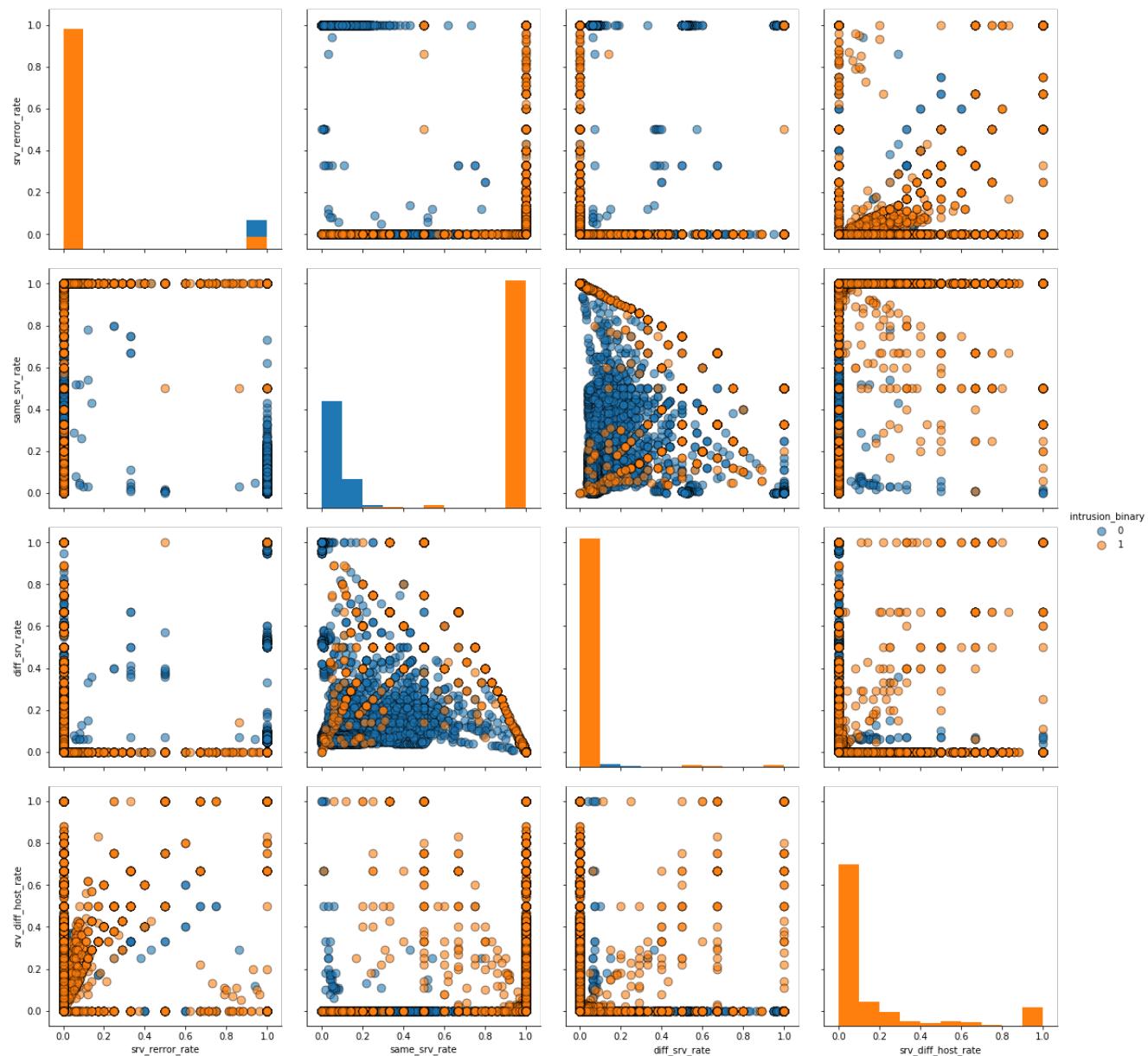
```
pairplot(data, 'intrusion_binary',
         features=['srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate'])
```





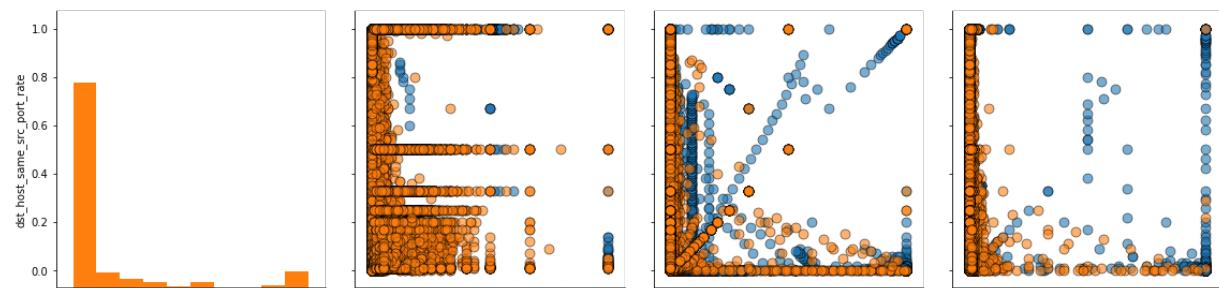
In [45]:

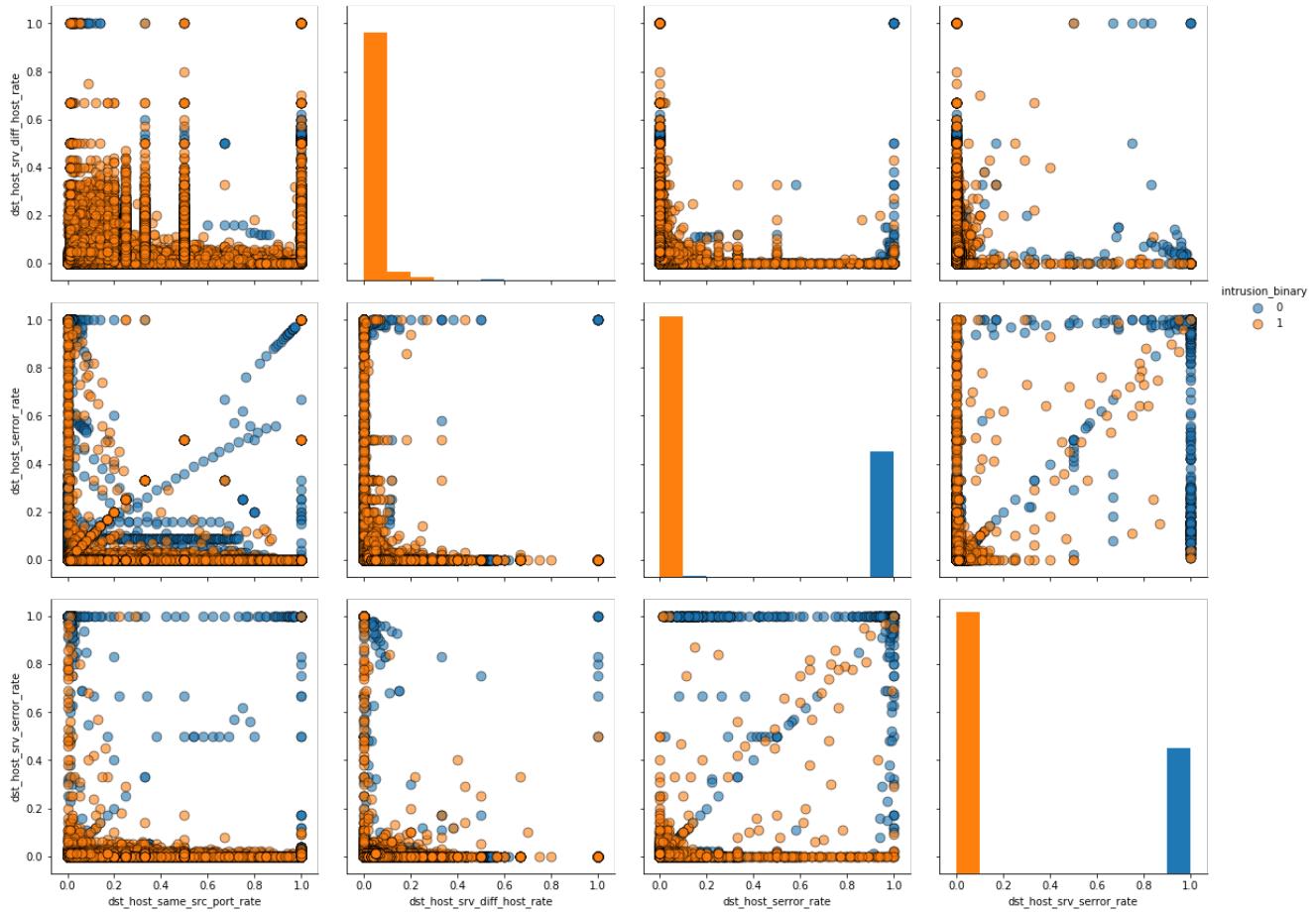
```
pairplot(data, 'intrusion_binary',
features=['srv_error_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate'])
```



In [46]:

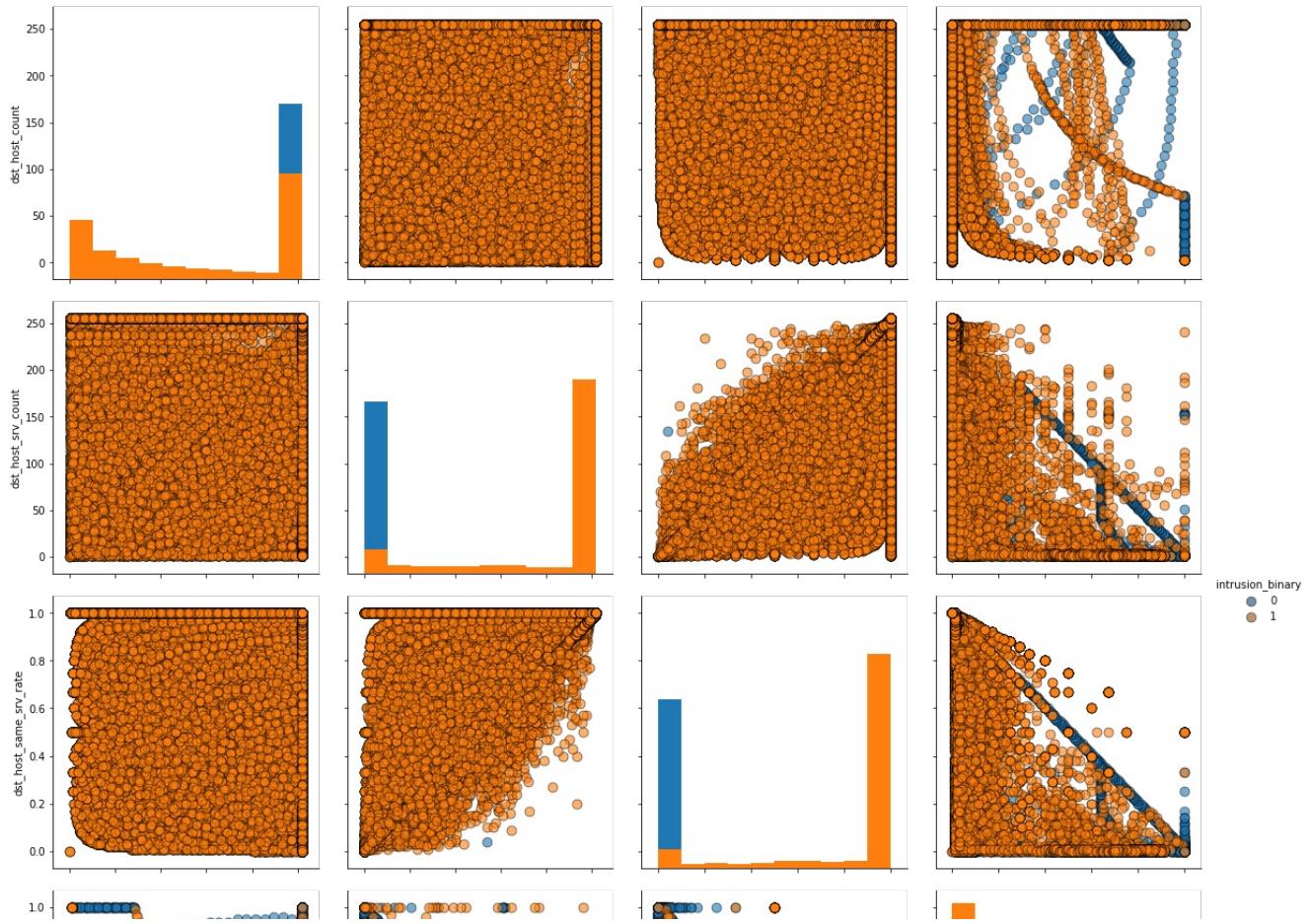
```
pairplot(data, 'intrusion_binary',
features=['dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate'])
```

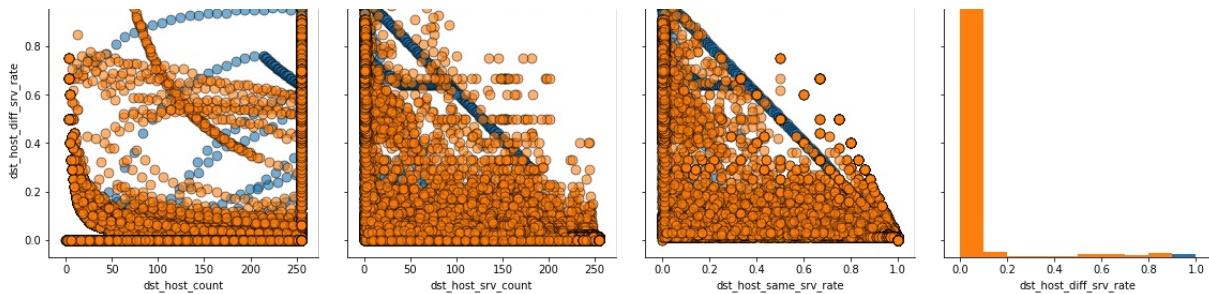




In [47]:

```
pairplot(data, 'intrusion_binary',
features=['dst_host_count','dst_host_srv_count','dst_host_same_srv_rate','dst_host_diff_srv_rate'])
```





#### **Observation from pairplots:-**

- Pairplots involving src\_bytes, dst\_bytes and duration show some separability between the 2 classes.
- Pairplots involving dst\_host\_count, dst\_host\_srv\_count, dst\_host\_same\_srv\_rate have high concentration of points belonging to class 1(Normal) and thus do not show any separability between the 2 categories.
- Remaining plots do not indicate any separability.

#### **TSNE for Bivariate Analysis:-**

In [49]:

```
def tsne_func(data, label, no_components, perplexity_value, n_iter_value):
    """
    This function applies TSNE on the original dataset with no_components, perplexity_value, n_iter_value as the TSNE
    parameters and transforms the original dataset into TSNE transformed feature space with the tsne
    dataset containing
    number of features equal to the value specified for no_components and also plots the scatter p
    lot of the transformed
    data points along with their class label
    """
    start = datetime.now()
    print('TSNE with perplexity={} and no. of iterations={}'.format(perplexity_value, n_iter_value))
    tsne = TSNE(n_components=no_components, perplexity=perplexity_value, n_iter=n_iter_value)
    tsne_df1 = tsne.fit_transform(data)
    print(tsne_df1.shape)
    tsne_df1 = np.vstack((tsne_df1.T, Y))
    tsne_data1 = pd.DataFrame(data=tsne_df1, columns=['feature1', 'feature2', 'Output'])
    sns.FacetGrid(tsne_data1, hue='Output', size=6).map(plt.scatter, 'feature1', 'feature2').add_le
    gend()
    print('Total time taken:', datetime.now()-start)
    plt.show()
```

In [57]:

```
df = data.drop(['intrusion_binary', 'service', 'flag', 'protocol_type'], axis=1)
Y = data['intrusion_binary'].values
```

In [58]:

```
from datetime import datetime
```

In [59]:

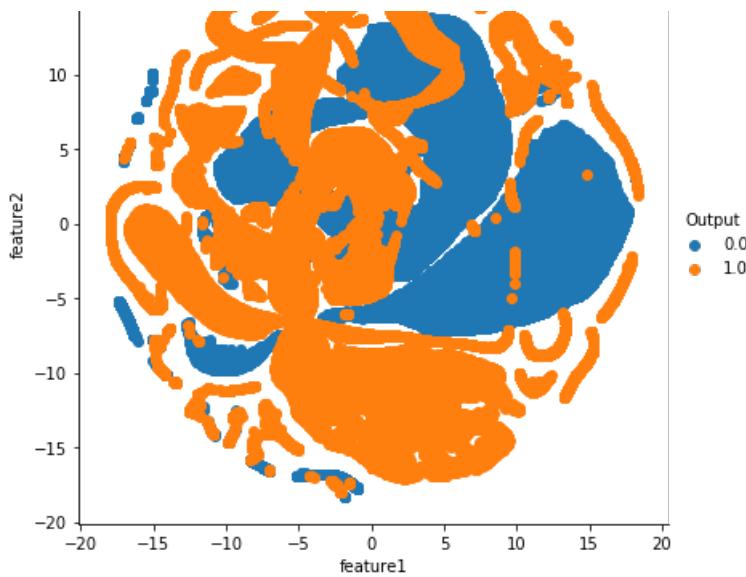
```
tsne_func(data=df, label=Y, no_components=2, perplexity_value=100, n_iter_value=500)
```

TSNE with perplexity=100 and no. of iterations=500

(145586, 2)

Total time taken: 0:28:12.846769

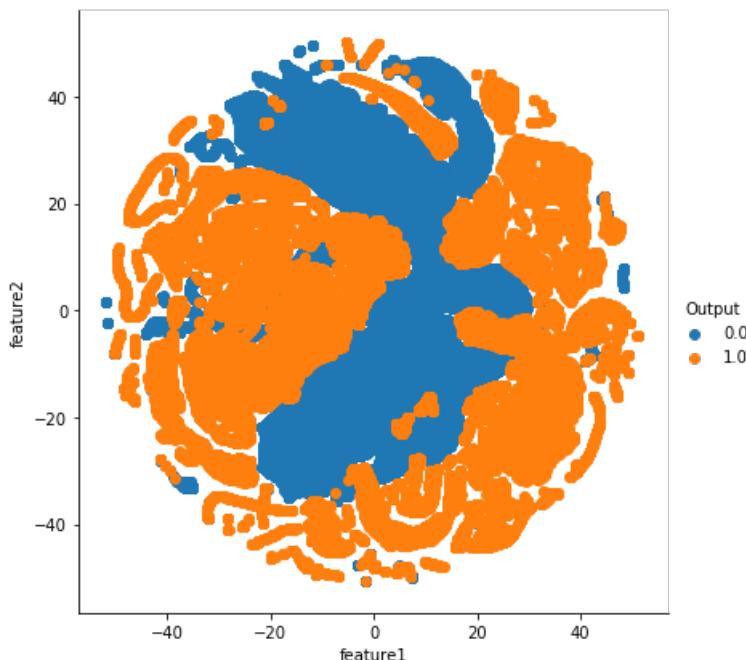




In [60]:

```
tsne_func(data=df, label=Y, no_components=2, perplexity_value=50, n_iter_value=1000)
```

TSNE with perplexity=50 and no. of iterations=1000  
 $(145586, 2)$   
 Total time taken: 0:44:24.317719



#### Observations:-

- None of the 2 plots show any linear separability/almost linear separability between the 2 classes in the TSNE transformed 2-D space.

## Merging features :-

In [142]:

```
X_train_3 = hstack((X_train_1, pca_train, train_cluster_2.T, feature_src_dst_1, feature_src_dst_3,
feature_5, feature_7))
```

```
In [143]:  
X_test_3 = hstack((X_test_1, pca_test, test_cluster_2.T, feature_src_dst_2, feature_src_dst_4, feature_6, feature_8))
```

In [144]:

```
print('Train data:')  
print(X_train_3.shape)  
print('='*30)  
print('Test data:')  
print(X_test_3.shape)
```

```
Train data:  
(109189, 127)  
=====
```

Test data:  
(36397, 127)

In [145]:

```
joblib.dump(X_train_3, 'X_train_3.pkl')  
joblib.dump(X_test_3, 'X_test_3.pkl')
```

Out[145]:

```
['X_test_3.pkl']
```

In [146]:

```
Y_train_new = []  
for i in Y_train:  
    if i=='normal.':  
        Y_train_new.append(1)  
    else:  
        Y_train_new.append(0)  
print(len(Y_train_new))
```

```
109189
```

In [147]:

```
Y_train_new[:5]
```

Out[147]:

```
[1, 1, 1, 1, 1]
```

In [148]:

```
Y_test_new = []  
for i in Y_test:  
    if i=='normal.':  
        Y_test_new.append(1)  
    else:  
        Y_test_new.append(0)  
print(len(Y_test_new))
```

```
36397
```

In [149]:

```
Y_test_new[:5]
```

Out[149]:

```
[1, 1, 1, 0, 1]
```

```
In [150]:
```

```
joblib.dump(Y_train_new, 'Y_train_new.pkl')
joblib.dump(Y_test_new, 'Y_test_new.pkl')
```

```
Out[150]:
```

```
['Y_test_new.pkl']
```

## Applying Machine learning models:-

```
In [151]:
```

```
from sklearn.metrics import roc_auc_score, roc_curve, auc
```

```
In [8]:
```

```
def confusion_matrix_func(Y_test, y_test_pred):
    """
    This function plots the confusion matrix for Binary classification problem
    """
    C = confusion_matrix(Y_test, y_test_pred)
    cm_df = pd.DataFrame(C)
    labels = ['BAD', 'NORMAL']
    plt.figure(figsize=(5,4))
    sns.set(font_scale=1.4)
    sns.heatmap(cm_df, annot=True, annot_kws={"size":12}, fmt='g', xticklabels=labels, yticklabels=labels)
    plt.ylabel('Actual Class')
    plt.xlabel('Predicted Class')

    plt.show()
```

```
In [9]:
```

```
def tpr_fpr_func_2(Y_tr, Y_pred):
    """
    This function computes the TPR and FPR using the actual and predicted values for each of the models.
    """
    results = dict()
    #Y_tr = Y_tr.to_list()
    tp = 0; fp = 0; positives = 0; negatives = 0; length = len(Y_tr)
    for i in range(len(Y_tr)):
        if Y_tr[i]==1:
            positives += 1
        elif Y_tr[i]==0:
            negatives += 1

    for i in range(len(Y_pred)):
        if Y_tr[i]==1 and Y_pred[i]==1:
            tp += 1
        elif Y_tr[i]==0 and Y_pred[i]==1:
            fp += 1

    tpr = tp/(positives)
    fpr = fp/(negatives)

    results['tp'] = tp; results['tpr'] = tpr; results['fp'] = fp; results['fpr'] = fpr

    return results
```

```
In [10]:
```

```
def model(model_name, X_train, Y_train, X_test, Y_test):
    """
    This function computes the performance metric scores on the train and test data.
    """

    print('Fitting the model and prediction on train data:')
```

```

start = dt.datetime.now()
model_name.fit(X_train, Y_train)
y_tr_pred = model_name.predict(X_train)
print('Completed')
print('Time taken:',dt.datetime.now()-start)
print('='*50)

results_tr = dict()
y_tr_pred = model_name.predict(X_train)
results_tr['precision'] = precision_score(Y_train, y_tr_pred, average='weighted')
results_tr['recall'] = recall_score(Y_train, y_tr_pred, average='weighted')
results_tr['f1_score'] = f1_score(Y_train, y_tr_pred, average='weighted')

results_test = dict()
print('Prediction on test data:')
start = dt.datetime.now()
y_test_pred = model_name.predict(X_test)
print('Completed')
print('Time taken:',dt.datetime.now()-start)
print('='*50)
print('Train Confusion Matrix is:')
confusion_matrix_func(Y_train, y_tr_pred)
print('='*50)
print('F1-score is:')
print(f1_score(Y_test, y_test_pred, average='weighted'))
print('='*50)

print('Test Performance metrics:')
print('='*50)
print('Confusion Matrix is:')
confusion_matrix_func(Y_test, y_test_pred)
print('='*50)
results_test['precision'] = precision_score(Y_test, y_test_pred, average='weighted')
print('Precision score is:')
print(precision_score(Y_test, y_test_pred, average='weighted'))
print('='*50)
results_test['recall'] = recall_score(Y_test, y_test_pred, average='weighted')
print('Recall score is:')
print(recall_score(Y_test, y_test_pred, average='weighted'))
print('='*50)
results_test['f1_score'] = f1_score(Y_test, y_test_pred, average='weighted')
print('F1-score is:')
print(f1_score(Y_test, y_test_pred, average='weighted'))
# add the trained model to the results
results_test['model'] = model

return results_tr, results_test

```

## Decision Tree:-

In [203]:

```

hyperparameter = {'max_depth':[5, 10, 20, 50, 100, 500], 'min_samples_split':[5, 10, 100, 500]}

from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(criterion='gini', splitter='best', class_weight='balanced')
decision_tree_grid = GridSearchCV(decision_tree, param_grid=hyperparameter, cv=5, verbose=1, n_jobs=-1)

```

In [204]:

```

decision_tree_grid_results_tr, decision_tree_grid_results_test = model(decision_tree_grid,
X_train_3.toarray(), Y_train_new, X_test_3.toarray(), Y_test_new)

```

Fitting the model and prediction on train data:  
Fitting 5 folds for each of 24 candidates, totalling 120 fits

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:   17.0s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   1.1min finished

```

```
Completed  
Time taken: 0:01:12.089501
```

```
=====
```

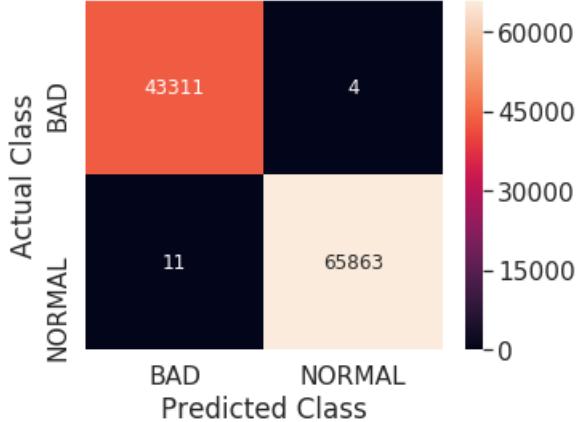
```
Prediction on test data:
```

```
Completed
```

```
Time taken: 0:00:00.014238
```

```
=====
```

```
Train Confusion Matrix is:
```



```
=====
```

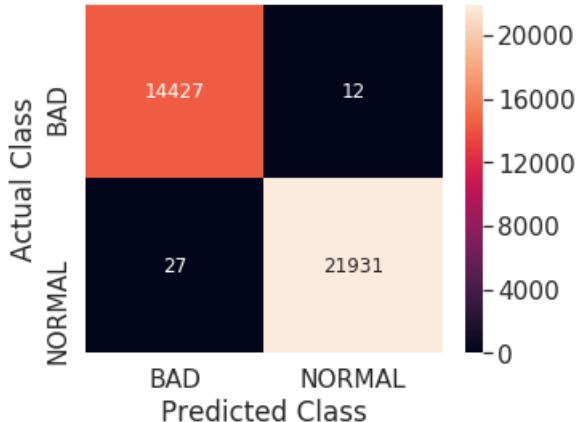
```
F1-score is:  
0.9989285782026252
```

```
=====
```

```
Test Performance metrics:
```

```
=====
```

```
Confusion Matrix is:
```



```
=====
```

```
Precision score is:  
0.9989290275806711
```

```
=====
```

```
Recall score is:
```

```
0.9989284831167404
```

```
=====
```

```
F1-score is:
```

```
0.9989285782026252
```

```
In [205]:
```

```
dt_gs = decision_tree_grid.best_estimator_
y_tr_pred = dt_gs.predict(X_train_3.toarray())
y_test_pred = dt_gs.predict(X_test_3.toarray())
dt_tpr_fpr_train = tpr_fpr_func_2(Y_train_new, y_tr_pred)
dt_tpr_fpr_test = tpr_fpr_func_2(Y_test_new, y_test_pred)
```

```
In [206]:
```

```
dt_tpr_fpr_train
```

```
auc_fpr_tpr_crdnn
```

Out[206]:

```
{'fp': 4, 'fpr': 9.234676209165416e-05, 'tp': 65863, 'tpr': 0.9998330145429153}
```

In [207]:

```
dt_tpr_fpr_test
```

Out[207]:

```
{'fp': 12,
 'fpr': 0.0008310824849366299,
 'tp': 21931,
 'tpr': 0.9987703798160124}
```

In [208]:

```
print_grid_search_attributes(decision_tree_grid)
```

```
-----| Best Estimator |-----
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=500,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=5,
                      min_weight_fraction_leaf=0.0, presort=False,
                      random_state=None, splitter='best')
```

```
-----| Best parameters |-----
```

Parameters of best estimator :

```
{'max_depth': 500, 'min_samples_split': 5}
```

```
-----| No of CrossValidation sets |-----
```

Total number of cross validation sets: 5

```
-----| Best Score |-----
```

Average Cross Validate scores of best estimator :

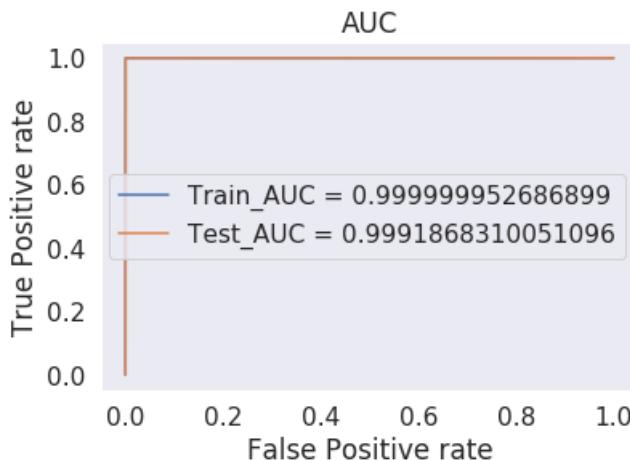
0.9989193050582018

In [209]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

from sklearn.metrics import roc_auc_score, roc_curve
clf=decision_tree_grid.best_estimator_
clf.fit(X_train_3, Y_train_new)
Y_1=clf.predict_proba(X_train_3)
Y1= Y_1[:,1]
train_fpr, train_tpr, tr_threshold = roc_curve(Y_train_new, Y1)
Y_2=clf.predict_proba(X_test_3)
Y2= Y_2[:,1]
test_fpr, test_tpr, te_threshold = roc_curve(Y_test_new, Y2)
plt.plot(train_fpr, train_tpr, label='Train_AUC = '+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label='Test_AUC = '+str(auc(test_fpr, test_tpr)))
plt.xlabel('False Positive rate')
plt.ylabel('True Positive rate')
plt.title('AUC')
plt.legend()
```

```
plt.grid()  
plt.show()
```



In [210]:

```
decision_tree_grid_results_tr
```

Out [210]:

```
{'f1_score': 0.9998626254244244,  
'precision': 0.999862635909193,  
'recall': 0.9998626235243477}
```

In [211]:

```
decision_tree_grid_results_test
```

Out [211]:

```
{'f1_score': 0.9989285782026252,  
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,  
'precision': 0.9989290275806711,  
'recall': 0.9989284831167404}
```

In [238]:

```
joblib.dump(decision_tree_grid.best_estimator_, 'dt_grid_3.pkl')
```

Out [238]:

```
['dt_grid_3.pkl']
```

#### **Observation from DT-3 Classifier :-**

- This DT Classifier was able to classify points with an AUC of 0.9991 and high f1 score of 0.9989.
- TPR = 0.9987
- FPR = 0.00083
- This DT Classifier has a TPR of (99.87%), but the FPR(0.00083) which is better than the previous DT classifiers.
- The train and test scores are very close, so the model is not overfitting.

## **Random Forests :-**

In [213]:

```
hyperparameter = {'max_depth':[5, 10, 100, 500, 1000], 'n_estimators': [5, 10, 50, 100, 500],  
                  'min_samples_leaf':[1, 5, 10, 50, 100]}
```

```

    min_samples_split=[5, 10, 100, 300]

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='gini', class_weight='balanced')
rf_grid = GridSearchCV(rf, param_grid=hyperparameter, cv=3, verbose=1, n_jobs=-1)

```

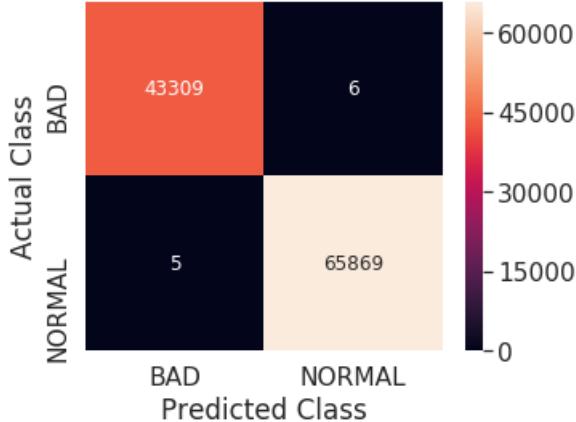
In [214]:

```
rf_grid_results_tr, rf_grid_results_test = model(rf_grid, X_train_3.toarray(), Y_train_new,
X_test_3.toarray(), Y_test_new)
```

Fitting the model and prediction on train data:  
Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:  55.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  8.2min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 15.2min finished
```

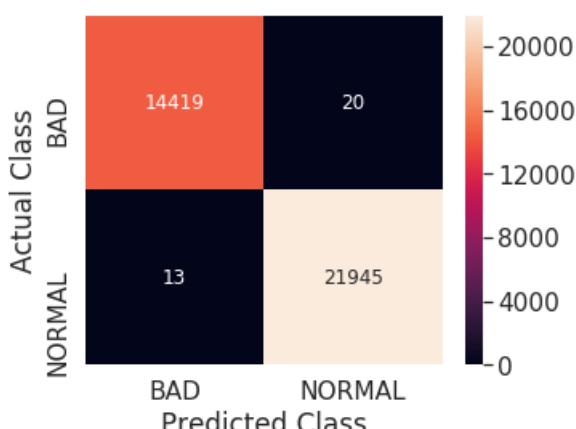
Completed  
Time taken: 0:15:35.049137  
=====  
Prediction on test data:  
Completed  
Time taken: 0:00:00.421488  
=====  
Train Confusion Matrix is:



=====  
F1-score is:  
0.9990932942013795

=====  
Test Performance metrics:  
=====

Confusion Matrix is:



```
Precision score is:  
0.9990933337457507  
=====  
Recall score is:  
0.9990933318680111  
=====  
F1-score is:  
0.9990932942013795
```

In [215]:

```
print_grid_search_attributes(rf_grid)
```

```
-----  
|      Best Estimator      |  
-----  
  
RandomForestClassifier(bootstrap=True, class_weight='balanced',  
                      criterion='gini', max_depth=500, max_features='auto',  
                      max_leaf_nodes=None, min_impurity_decrease=0.0,  
                      min_impurity_split=None, min_samples_leaf=1,  
                      min_samples_split=10, min_weight_fraction_leaf=0.0,  
                      n_estimators=100, n_jobs=None, oob_score=False,  
                      random_state=None, verbose=0, warm_start=False)
```

```
-----  
|      Best parameters      |  
-----
```

Parameters of best estimator :

```
{'max_depth': 500, 'min_samples_split': 10, 'n_estimators': 100}
```

```
-----  
|      No of CrossValidation sets      |  
-----
```

Total numbre of cross validation sets: 3

```
-----  
|      Best Score      |  
-----
```

Average Cross Validate scores of best estimator :

```
0.9992398501680573
```

In [218]:

```
rf_gs = rf_grid.best_estimator_  
y_tr_pred = rf_gs.predict(X_train_3.toarray())  
y_test_pred = rf_gs.predict(X_test_3.toarray())  
rf_tpr_fpr_train = tpr_fpr_func_2(Y_train_new, y_tr_pred)  
rf_tpr_fpr_test = tpr_fpr_func_2(Y_test_new, y_test_pred)
```

In [219]:

```
rf_tpr_fpr_train
```

Out[219]:

```
{'fp': 6, 'fpr': 0.00013852014313748125, 'tp': 65869, 'tpr': 0.999924097519507}
```

In [220]:

```
rf_tpr_fpr_test
```

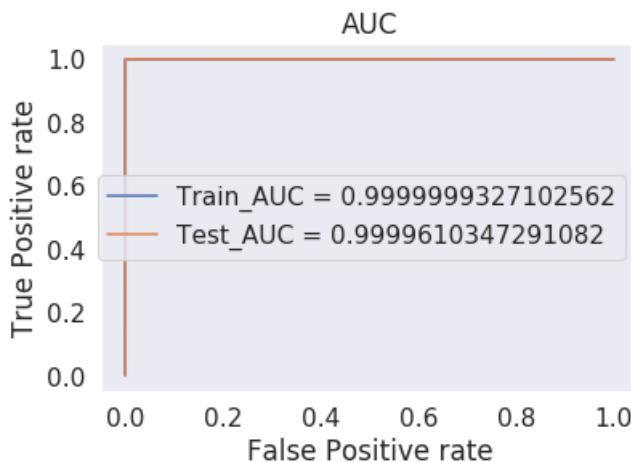
Out[220]:

```
{'fp': 20,  
'fpr': 0.0013851374748943832,  
'tn': 21915}
```

```
In [225]:  
'tpr': 0.9994079606521541}
```

In [226]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python  
  
from sklearn.metrics import roc_auc_score, roc_curve  
clf=rf_grid.best_estimator_  
clf.fit(X_train_3, Y_train_new)  
Y_1=clf.predict_proba(X_train_3)  
Y1= Y_1[:,1]  
train_fpr, train_tpr, tr_threshold = roc_curve(Y_train_new, Y1)  
Y_2=clf.predict_proba(X_test_3)  
Y2= Y_2[:,1]  
test_fpr, test_tpr, te_threshold = roc_curve(Y_test_new, Y2)  
plt.plot(train_fpr, train_tpr, label='Train_AUC = '+str(auc(train_fpr, train_tpr)))  
plt.plot(test_fpr, test_tpr, label='Test_AUC = '+str(auc(test_fpr, test_tpr)))  
plt.xlabel('False Positive rate')  
plt.ylabel('True Positive rate')  
plt.title('AUC')  
plt.legend()  
plt.grid()  
plt.show()
```



In [227]:

```
rf_grid_results_tr
```

Out[227]:

```
{'f1_score': 0.9998992570520558,  
'precision': 0.9998992570281391,  
'recall': 0.9998992572511883}
```

In [228]:

```
rf_grid_results_test
```

Out[228]:

```
{'f1_score': 0.9990932942013795,  
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,  
'precision': 0.9990933337457507,  
'recall': 0.9990933318680111}
```

In [229]:

```
joblib.dump(rf_grid.best_estimator_, 'rf_grid_3.pkl')
```

Out[229]:

```
['rf_grid_3.pkl']
```

#### **Observation from RF\_3 Classifier :-**

- This RF Classifier was able to classify points with an auc of 0.9999 and a high f1 score of ~0.9991.
- TPR = 0.9994
- False Positives = 15
- FPR = 0.0013
- This RF Classifier has a higher TPR of (99.94%), but a lower FPR(0.0010) than the DT\_3 model which is a drawback.

### **3. XGBoost :-**

In [230]:

```
hyperparameter = {'max_depth':[2, 3, 5, 7, 10], 'n_estimators': [10, 50, 100, 200, 500]}

from xgboost import XGBClassifier
xgb = XGBClassifier(objective='binary:logistic', n_jobs=-1)
xgb_grid = RandomizedSearchCV(xgb, param_distributions=hyperparameter, cv=3, verbose=1, n_jobs=-1)
```

In [232]:

```
xgb_grid_results_tr, xgb_grid_results_test = model(xgb_grid, X_train_3.toarray(), Y_train_new, X_test_3.toarray(), Y_test_new)
```

Fitting the model and prediction on train data:

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 4.1min finished
```

Completed

Time taken: 0:04:40.762277

```
=====
```

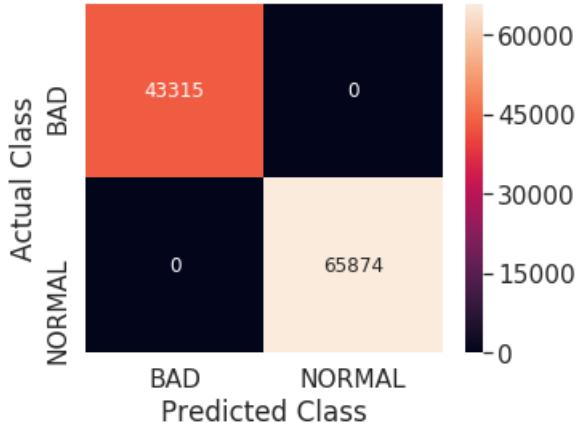
Prediction on test data:

Completed

Time taken: 0:00:00.111924

```
=====
```

Train Confusion Matrix is:



```
=====
```

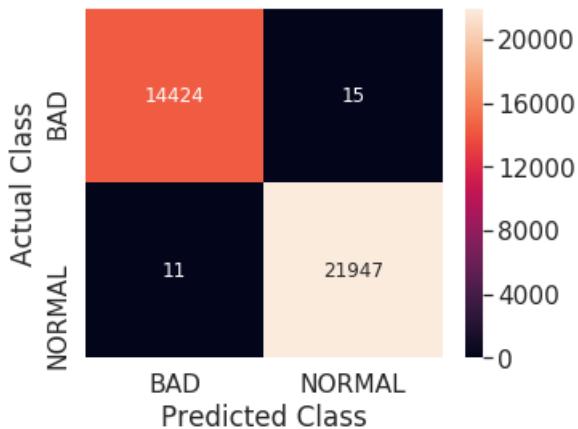
F1-score is:  
0.9992856384603864

```
=====
```

Test Performance metrics:

```
=====
```

Confusion Matrix is:



```
=====
Precision score is:
0.9992856467248608
=====
Recall score is:
0.9992856554111602
=====
F1-score is:
0.9992856384603864
```

In [233]:

```
print_grid_search_attributes(xgb_grid)
```

```
-----
|      Best Estimator      |
-----

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=200, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)

-----
|      Best parameters      |
-----

Parameters of best estimator :

{'max_depth': 10, 'n_estimators': 200}

-----
|      No of CrossValidation sets      |
-----

Total numbre of cross validation sets: 3

-----
|      Best Score      |
-----
```

Average Cross Validate scores of best estimator :

0.9993863850754197

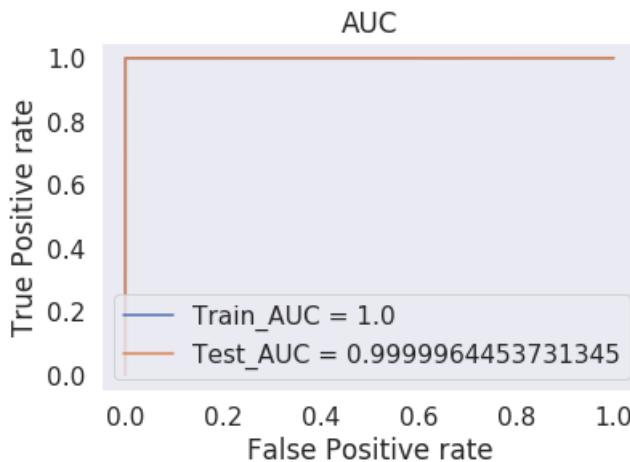
In [247]:

```
xgb_gs = xgb_grid.best_estimator_
y_tr_pred = xgb_gs.predict(X_train_3.toarray())
y_test_pred = xgb_gs.predict(X_test_3.toarray())
xgb_tpr_fpr_train = tpr_fpr_func_2(Y_train_new, y_tr_pred)
xgb_tpr_fpr_test = tpr_fpr_func_2(Y_test_new, y_test_pred)
```

In [234]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python

from sklearn.metrics import roc_auc_score, roc_curve
clf=xgb_grid.best_estimator_
clf.fit(X_train_3, Y_train_new)
Y_1=clf.predict_proba(X_train_3)
Y1= Y_1[:,1]
train_fpr, train_tpr, tr_threshold = roc_curve(Y_train_new, Y1)
Y_2=clf.predict_proba(X_test_3)
Y2= Y_2[:,1]
test_fpr, test_tpr, te_threshold = roc_curve(Y_test_new, Y2)
plt.plot(train_fpr, train_tpr, label='Train_AUC = '+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label='Test_AUC = '+str(auc(test_fpr, test_tpr)))
plt.xlabel('False Positive rate')
plt.ylabel('True Positive rate')
plt.title('AUC')
plt.legend()
plt.grid()
plt.show()
```



In [235]:

```
xgb_grid_results_tr
```

Out[235]:

```
{'f1_score': 1.0, 'precision': 1.0, 'recall': 1.0}
```

In [236]:

```
xgb_grid_results_test
```

Out[236]:

```
{'f1_score': 0.9992856384603864,
'model': <function __main__.model(model_name, X_train, Y_train, X_test, Y_test)>,
'precision': 0.9992856467248608,
'recall': 0.9992856554111602}
```

In [2]:

```
xgb_tpr_fpr_train
```

Out[2]:

```
{'fp': 0, 'fpr': 0.0, 'tp': 65874, 'tpr': 1.0}
```

In [4]:

```
... 100%|██████████| 1/1 [00:00<00:00, 1000.00it/s]
```

```
xgb_tpr_rpr_test
```

Out[4]:

```
{'fp': 15, 'fpr': 0.001038, 'tp': 21947, 'tpr': 0.99949904}
```

In [237]:

```
joblib.dump(xgb_grid.best_estimator_, 'xgb_grid_3.pkl')
```

Out[237]:

```
['xgb_grid_3.pkl']
```

#### **Observation from XGBoost\_3 Classifier :-**

- This XG Boost Classifier was able to classify points with a high test AUC of 0.9999 and a high f1-score of 0.9995 which is the highest of all of the previous models.
- True Postives = 21947
- TPR = 0.9994
- False Positives = 15
- FPR = 0.0010
- This XGB Classifier has a higher FPR than the DT\_3 model which is not desirable.

#### **Observation from all of the above models:-**

- The DT\_3 model has the best FPR of 0.0008 but it also has the least TPR of 0.9987 and a test AUC of 0.9991.
- The RF\_3 has a better TPR than the DT\_3 but it has a higher FPR which is not desirable.
- The XGB\_3 model has the best combination of FPR(0.001038) and TPR(0.9994) and an AUC of 0.9999 on the test data.
- As the train and test scores are almost same, the models are not Overfitting.

## **Summarizing results for Part 2 : Binary Classification Problem**

In [5]:

```
from prettytable import PrettyTable

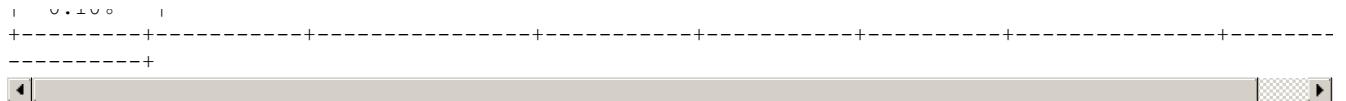
x = PrettyTable()

x.field_names = ['Model', 'Train_AUC', 'Train f1-score', 'Train TPR', 'Train FPR', 'Test AUC', 'Test f1-score', 'Test TPR', 'Test FPR']

x.add_row(['DT - 3', '0.9999', '0.9998', '99.98%', '0.00009%', '0.9991', '0.9989', '99.87%', '0.08%'])
x.add_row(['RF - 3', '0.9999', '0.9998', '99.99%', '0.013%', '0.9999', '0.9990', '99.94%', '0.13%'])
x.add_row(['XGB - 3', '1.0', '1.0', '100.0%', '0.0%', '0.9999', '0.9992', '99.94%', '0.10%'])

print(x)
```

Model	Train_AUC	Train f1-score	Train TPR	Train FPR	Test AUC	Test f1-score	Test TPR	Test FPR
DT - 3	0.9999	0.9998	99.98%	0.00009%	0.9991	0.9989	99.87%	0.08%
RF - 3	0.9999	0.9998	99.99%	0.013%	0.9999	0.9990	99.94%	0.13%
XGB - 3	1.0	1.0	100.0%	0.0%	0.9999	0.9992	99.94%	0.10%



## Conclusion :-

### ***For Multiclass Classification:-***

- The model XG Boost\_1 is our best model for intrusion detection as it has highest f1-score 0.9994 and TPR 99.98% as well as the least FPR of 0.08%.

### ***For Binary Classification:-***

- The XGB\_3 model has the best combination of FPR(0.01038%) and TPR(99.94%) and an AUC of 0.9999 on the test data, thus it is our best model for binary classification.