In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
pd.set_option('display.max_columns',50)
pd.set_option('display.max_rows',50000000)

from sklearn.naive_bayes import GaussianNB

from sklearn import metrics

from pandas import DataFrame

from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.model_selection import train_test_split

from keras.models import Sequential

from keras.layers import Dense, Conv2D

from keras.utils import np_utils
```

In [2]:

```
data = pd.read_csv("kddcup99_csv.csv")
```

In [3]:

```
data.shape
```

Out[3]:

```
(494020, 42)
```

In [4]:

```
(data.head(10))
```

Out[4]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent
0	0	tcp	http	SF	181	5450	0	0	0
1	0	tcp	http	SF	239	486	0	0	0
2	0	tcp	http	SF	235	1337	0	0	0
3	0	tcp	http	SF	219	1337	0	0	0
4	0	tcp	http	SF	217	2032	0	0	0
5	0	tcp	http	SF	217	2032	0	0	0
6	0	tcp	http	SF	212	1940	0	0	0
7	0	tcp	http	SF	159	4087	0	0	0
8	0	tcp	http	SF	210	151	0	0	0
9	0	tcp	http	SF	212	786	0	0	0

In [5]:

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 494020 entries, 0 to 494019
Data columns (total 42 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   duration                     494020 non-null  int64
 1   protocol_type                494020 non-null  object
 2   service                      494020 non-null  object
 3   flag                         494020 non-null  object
 4   src_bytes                    494020 non-null  int64
 5   dst_bytes                    494020 non-null  int64
 6   land                         494020 non-null  int64
 7   wrong_fragment               494020 non-null  int64
 8   urgent                       494020 non-null  int64
 9   hot                          494020 non-null  int64
 10  num_failed_logins            494020 non-null  int64
 11  logged_in                    494020 non-null  int64
 12  lnum_compromised             494020 non-null  int64
 13  lroot_shell                  494020 non-null  int64
 14  lsu_attempted                494020 non-null  int64
 15  lnum_root                    494020 non-null  int64
 16  lnum_file_creations          494020 non-null  int64
 17  lnum_shells                  494020 non-null  int64
 18  lnum_access_files            494020 non-null  int64
 19  lnum_outbound_cmds           494020 non-null  int64
 20  is_host_login                494020 non-null  int64
 21  is_guest_login               494020 non-null  int64
 22  count                        494020 non-null  int64
 23  srv_count                    494020 non-null  int64
 24  serror_rate                  494020 non-null  float64
 25  srv_serror_rate              494020 non-null  float64
 26  rerror_rate                  494020 non-null  float64
 27  srv_rerror_rate              494020 non-null  float64
 28  same_srv_rate                494020 non-null  float64
 29  diff_srv_rate                494020 non-null  float64
 30  srv_diff_host_rate           494020 non-null  float64
 31  dst_host_count               494020 non-null  int64
 32  dst_host_srv_count           494020 non-null  int64
 33  dst_host_same_srv_rate       494020 non-null  float64
 34  dst_host_diff_srv_rate       494020 non-null  float64
 35  dst_host_same_src_port_rate  494020 non-null  float64
 36  dst_host_srv_diff_host_rate  494020 non-null  float64
 37  dst_host_serror_rate         494020 non-null  float64
 38  dst_host_srv_serror_rate     494020 non-null  float64
 39  dst_host_rerror_rate         494020 non-null  float64
 40  dst_host_srv_rerror_rate     494020 non-null  float64
 41  label                        494020 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 158.3+ MB
None
```

In [54]:

```
protocol_types = []
services = []
flags = []


for i in range (494020):
    if(data['service'][i] not in services):
        services.append(data['service'][i])

    if(data['protocol_type'][i] not in protocol_types):
        protocol_types.append(data['protocol_type'][i])

    if(data['flag'][i] not in flags):
        flags.append(data['flag'][i])

print(protocol_types)
print(services)
print(flags)


protocol_type = []
service = []
flag = []

for i in range (494020):
    protocol_type.append(protocol_types.index(data['protocol_type'][i]))

    service.append(services.index(data['service'][i]))

    flag.append(flags.index(data['flag'][i]))

del data['protocol_type']
del data['service']
del data['flag']

data['protocol_type'] = protocol_type
data['service'] = service
data['flag'] = flag
```

```
[2, 0, 1]
[9, 11, 0, 61, 33, 1, 10, 3, 40, 25, 6, 7, 22, 13, 18, 8, 28, 48, 29, 17, 2
3, 31, 24, 44, 4, 46, 5, 55, 57, 39, 34, 14, 38, 21, 42, 58, 2, 20, 41, 37,
27, 36, 45, 19, 50, 51, 26, 12, 54, 43, 16, 30, 35, 56, 32, 52, 49, 15, 53,
47, 59, 60, 65, 63, 64, 62]
[0, 4, 2, 6, 7, 1, 10, 3, 9, 5, 8]
```

In [55]:

```
#label = []
labels = []
#for i in range (494020):
#    if(data['label'][i]=='normal'):
#        label.append(0)
#    else:
#        label.append(1)
#
for i in range (494020):
    if(data['label'][i] not in labels):
        labels.append(data['label'][i])

print(labels)

label=[]

for i in range (494020):
    label.append(labels.index(data['label'][i]))
#
#del data['label'] # deleting label column
#
del data['label']
#
#data['label']=label # adding label list as column
#
data['label'] = label
#
print(data.info())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\users\yaimg\appdata\local\programs\python\python39\lib\site-packages\pand
as\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3079             try:
-> 3080                 return self._engine.get_loc(casted_key)
   3081             except KeyError as err:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHa
shTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHa
shTable.get_item()

KeyError: 'label'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
<ipython-input-55-6b042759cf69> in <module>
      8 #
      9 for i in range (494020):
---> 10     if(data['label'][i] not in labels):
     11         labels.append(data['label'][i])
     12
```

```
c:\users\yaimg\appdata\local\programs\python\python39\lib\site-packages\pand
as\core\frame.py in __getitem__(self, key)
   3022             if self.columns.nlevels > 1:
   3023                 return self._getitem_multilevel(key)
-> 3024             indexer = self.columns.get_loc(key)
   3025             if is_integer(indexer):
   3026                 indexer = [indexer]

c:\users\yaimg\appdata\local\programs\python\python39\lib\site-packages\pand
as\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3080                 return self._engine.get_loc(casted_key)
   3081             except KeyError as err:
-> 3082                 raise KeyError(key) from err
   3083 
   3084         if tolerance is not None:

KeyError: 'label'
```

In [ ]:

```
print(data.columns)

#print(data['label'][1000:10000])

print(len(labels))
```

In [9]:

```
data = data.sample(frac=1).reset_index(drop=True)
```

In [ ]:

```
Y = data.label

del data['label']
```

In [11]:

```
#print(data.head)
```

In [12]:

```
X_train, X_test, Y_train, Y_test= train_test_split(data, Y, test_size=0.4, random_state

# applying standard scalar

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train = DataFrame(X_train)
X_test = DataFrame(X_test)


print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

X_train.head(10)

```

```
(296412, 41)
(197608, 41)
(296412,)
(197608,)
```

Out[12]:

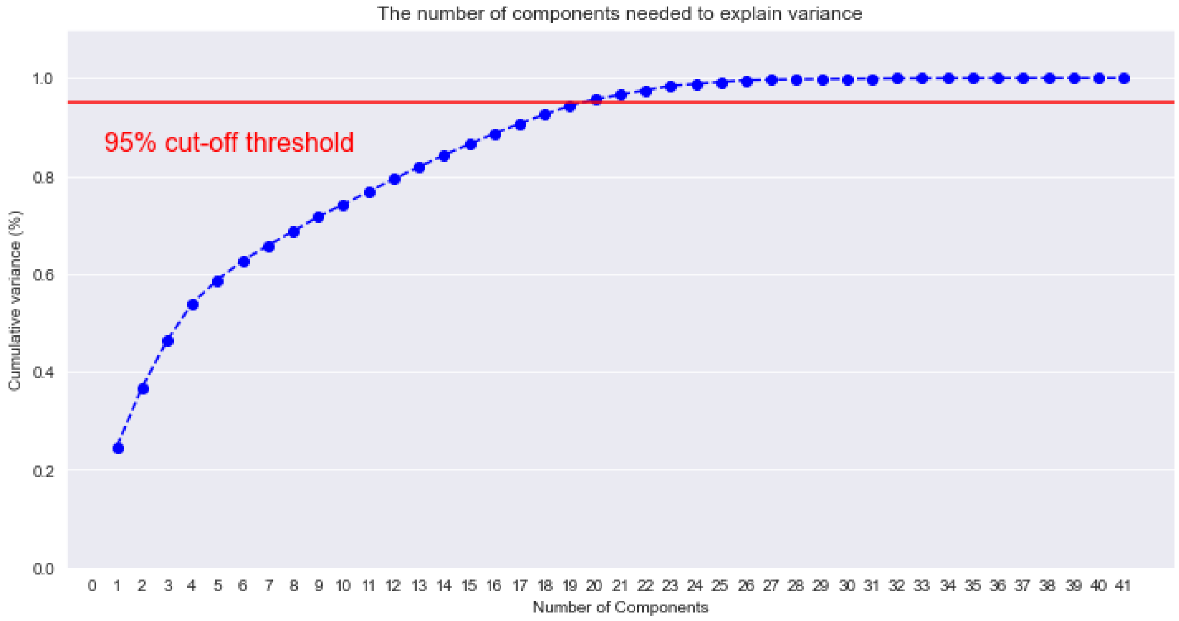
25	26	27	28	29	30	31	32	33	
3635	-0.255470	-0.203239	0.347301	0.626797	0.600611	-0.283137	0.827596	-0.156611	-0.465
3635	-0.255470	-0.203239	-3.191017	0.626797	0.600611	-0.283137	-1.166703	0.310130	-0.465
3635	-0.255470	-0.203239	0.347301	0.626797	0.600611	-0.283137	0.827596	-0.156611	-0.465
7398	0.469382	-0.203239	0.347301	-1.633856	-1.684793	0.263114	-1.249799	-0.156611	2.155
3635	-0.255470	-0.203239	0.347301	0.626797	0.600611	-0.283137	0.827596	-0.156611	-0.465
3635	-0.255470	-0.203239	0.347301	0.626797	0.600611	-0.283137	0.827596	-0.156611	-0.465
3635	-0.255470	-0.203239	0.347301	0.626797	0.600611	-0.283137	0.827596	-0.156611	-0.465
8811	0.469382	-0.203239	0.347301	-1.624436	-1.684793	0.354155	-1.249799	-0.156611	-0.465
4528	0.348573	-0.203239	0.347301	-1.633856	-1.684793	0.172072	-1.249799	-0.156611	2.155
3635	-0.255470	0.362446	-2.480263	0.626797	0.600611	-0.283137	-1.229025	0.310130	-0.465

In [13]:

```
#print(X_train.describe())
```

In [14]:

```
scaler = StandardScaler()

data_rescaled = scaler.fit_transform(data)

pca = PCA().fit(data_rescaled)

%matplotlib inline
plt.rcParams["figure.figsize"] = (12,6)

fig, ax = plt.subplots()
xi = np.arange(1, 42, step=1)
y = np.cumsum(pca.explained_variance_ratio_)

plt.ylim(0.0,1.1)
plt.plot(xi, y, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 42, step=1)) #change from 0-based array index to 1-based human-
plt.ylabel('Cumulative variance (%)')
plt.title('The number of components needed to explain variance')

plt.axhline(y=0.95, color='r', linestyle='-')
plt.text(0.5, 0.85, '95% cut-off threshold', color = 'red', fontsize=16)

ax.grid(axis='x')
plt.show()
```



In [15]:

```
#95% of variance

from sklearn.decomposition import PCA

pca = PCA(n_components = 21)

pca.fit(X_train)

reduced = pca.transform(X_train)
```

In [16]:

```
pca.explained_variance_ratio_
```

Out[16]:

```
array([0.24727502, 0.12087945, 0.09560429, 0.06572439, 0.04733285,
       0.03966175, 0.03235498, 0.0291513 , 0.02684011, 0.02605724,
       0.02570321, 0.02564157, 0.02558505, 0.02531757, 0.02451825,
       0.02271033, 0.02114997, 0.0205783 , 0.01875876, 0.01255146,
       0.01113734])
```

In [17]:

```
print(reduced[10])
```

```
[ 5.7235613  -2.1736085   0.43173231 -0.02693477 -0.12098627 -0.26466507
  0.02662796  0.07846841  0.03218412 -0.04082107  0.00866493 -0.010011
 -0.00992922 -0.02304034 -0.0820291   0.08165886 -0.04287394 -0.05863699
 -0.01630676  0.24691157 -0.01668    ]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [57]:

```
# gaussian naive bayes

gnb = GaussianNB()
y1_pred = gnb.fit(X_train, Y_train).predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, y1_pred))
```

```
Accuracy: 0.9041384964171 49
```

In [58]:

```
# decision tree

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(random_state=0)
y2_pred = dtc.fit(X_train, Y_train).predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, y2_pred))
```

```
Accuracy: 0.9995749159953038
```

In [*]:

```
# svm

from sklearn.svm import LinearSVC

svc= LinearSVC(max_iter=100000)
y3_pred = svc.fit(X_train, Y_train).predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, y3_pred))
```

In [ ]:

In [*]:

```
# k-means clustering

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=21)
y_kmeans = kmeans.fit_predict(data)
print(y_kmeans)

kmeans.cluster_centers_
```

In [*]:

```
Error =[]
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i).fit(data)
    kmeans.fit(data)
    Error.append(kmeans.inertia_)
import matplotlib.pyplot as plt
plt.plot(range(1, 11), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```

In [*]:

```
# k-means clustering

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2)
y_kmeans = kmeans.fit_predict(X_train)
print(y_kmeans)

kmeans.cluster_centers_
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```python

```

In [ ]:

```python
# svm once more
```

In [ ]:

```python
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

In [ ]:

```python
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, Y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, Y_train)
```

In [ ]:

```python
poly_pred = poly.predict(X_test)
rbf_pred = rbf.predict(X_test)
```

In [ ]:

```python
poly_accuracy = accuracy_score(Y_test, poly_pred)
poly_f1 = f1_score(Y_test, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
```

In [ ]:

```python
rbf_accuracy = accuracy_score(Y_test, rbf_pred)
rbf_f1 = f1_score(Y_test, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```

```

In [18]:

```
# fully connected neural network
```

In [19]:

```
import tensorflow as tf
from tensorflow.keras import backend as K

```

In [20]:

```
sess = tf.compat.v1.keras.backend.get_session()
```

In [21]:

```
from keras.layers import Dropout
```

In [27]:

```
model=Sequential()
model.add(Dense(184,input_dim=41,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(92,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(46,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(23,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy',
                                                                        tf.keras.metric
                                                                        tf.keras.metric
                                                                        tf.keras.metric
                                                                        tf.keras.metric
                                                                        tf.keras.metric
                                                                        tf.keras.metric
```

In [28]:

```
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 184)               7728      
_________________________________________________________________
dropout_3 (Dropout)          (None, 184)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 92)                17020     
_________________________________________________________________
dropout_4 (Dropout)          (None, 92)                0         
_________________________________________________________________
dense_6 (Dense)              (None, 46)                4278      
_________________________________________________________________
dropout_5 (Dropout)          (None, 46)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 23)                1081      
=================================================================
Total params: 30,107
Trainable params: 30,107
Non-trainable params: 0
_________________________________________________________________
None
```

In [29]:

```
y_train=np_utils.to_categorical(Y_train,num_classes=23)
y_test=np_utils.to_categorical(Y_test,num_classes=23)
print("Shape of y_train",Y_train.shape)
print("Shape of y_test",Y_test.shape)
```

```
Shape of y_train (296412,)
Shape of y_test (197608,)
```

In [30]:

```
tf.config.run_functions_eagerly(True)
tf.data.experimental.enable_debug_mode()
```

In [34]:

```
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=128,epochs=10,verb
```

```
Epoch 1/10
2316/2316 [==============================] - 346s 149ms/step - loss: 0.0352
- accuracy: 0.9920 - precision_1: 0.9953 - recall_1: 0.9820 - true_positives
_1: 5060961.0000 - true_negatives_1: 113356544.0000 - false_positives_1: 240
86.9629 - false_negatives_1: 92677.8203 - val_loss: 0.0124 - val_accuracy:
0.9970 - val_precision_1: 0.9953 - val_recall_1: 0.9825 - val_true_positives
_1: 5306118.5000 - val_true_negatives_1: 118790664.0000 - val_false_positive
s_1: 24776.6895 - val_false_negatives_1: 94527.1328
Epoch 2/10
2316/2316 [==============================] - 285s 123ms/step - loss: 0.0210
- accuracy: 0.9951 - precision_1: 0.9955 - recall_1: 0.9830 - true_positives
_1: 5551591.5000 - true_negatives_1: 124225000.0000 - false_positives_1: 254
05.3965 - false_negatives_1: 96070.0000 - val_loss: 0.0098 - val_accuracy:
0.9977 - val_precision_1: 0.9956 - val_recall_1: 0.9835 - val_true_positives
_1: 5797394.5000 - val_true_negatives_1: 129659136.0000 - val_false_positive
s_1: 25990.3398 - val_false_negatives_1: 97266.5156
Epoch 3/10
2316/2316 [==============================] - 288s 124ms/step - loss: 0.0164
- accuracy: 0.9963 - precision_1: 0.9956 - recall_1: 0.9840 - true_positives
_1: 6043355.5000 - true_negatives_1: 135093520.0000 - false_positives_1: 265
33.5605 - false_negatives_1: 98324.5078 - val_loss: 0.0082 - val_accuracy:
0.9980 - val_precision_1: 0.9957 - val_recall_1: 0.9845 - val_true_positives
_1: 6289404.0000 - val_true_negatives_1: 140527632.0000 - val_false_positive
s_1: 27032.5938 - val_false_negatives_1: 99285.3594
Epoch 4/10
2316/2316 [==============================] - 289s 125ms/step - loss: 0.0145
- accuracy: 0.9969 - precision_1: 0.9958 - recall_1: 0.9849 - true_positives
_1: 6535573.0000 - true_negatives_1: 145962000.0000 - false_positives_1: 274
99.2188 - false_negatives_1: 100126.9375 - val_loss: 0.0069 - val_accuracy:
0.9983 - val_precision_1: 0.9959 - val_recall_1: 0.9853 - val_true_positives
_1: 6781784.0000 - val_true_negatives_1: 151396096.0000 - val_false_positive
s_1: 27923.9062 - val_false_negatives_1: 100913.5312
Epoch 5/10
2316/2316 [==============================] - 289s 125ms/step - loss: 0.0133
- accuracy: 0.9973 - precision_1: 0.9960 - recall_1: 0.9857 - true_positives
_1: 7028074.5000 - true_negatives_1: 156830432.0000 - false_positives_1: 283
23.5293 - false_negatives_1: 101644.5156 - val_loss: 0.0071 - val_accuracy:
0.9986 - val_precision_1: 0.9961 - val_recall_1: 0.9861 - val_true_positives
_1: 7274420.5000 - val_true_negatives_1: 162264560.0000 - val_false_positive
s_1: 28689.8652 - val_false_negatives_1: 102303.7734
Epoch 6/10
2316/2316 [==============================] - 286s 123ms/step - loss: 0.0132
- accuracy: 0.9974 - precision_1: 0.9962 - recall_1: 0.9865 - true_positives
_1: 7520797.0000 - true_negatives_1: 167698688.0000 - false_positives_1: 290
58.0801 - false_negatives_1: 102946.3750 - val_loss: 0.0063 - val_accuracy:
0.9984 - val_precision_1: 0.9962 - val_recall_1: 0.9868 - val_true_positives
_1: 7767120.0000 - val_true_negatives_1: 173132800.0000 - val_false_positive
s_1: 29425.8340 - val_false_negatives_1: 103618.4844
Epoch 7/10
2316/2316 [==============================] - 308s 133ms/step - loss: 0.0108
- accuracy: 0.9977 - precision_1: 0.9963 - recall_1: 0.9872 - true_positives
_1: 8013526.0000 - true_negatives_1: 178567168.0000 - false_positives_1: 297
55.5156 - false_negatives_1: 104238.2969 - val_loss: 0.0056 - val_accuracy:
0.9987 - val_precision_1: 0.9964 - val_recall_1: 0.9875 - val_true_positives
_1: 8259940.0000 - val_true_negatives_1: 184001280.0000 - val_false_positive
s_1: 30066.7676 - val_false_negatives_1: 104821.4922
Epoch 8/10
```

```
2316/2316 [==============================] - 297s 128ms/step - loss: 0.0113
- accuracy: 0.9978 - precision_1: 0.9965 - recall_1: 0.9878 - true_positives
_1: 8506372.0000 - true_negatives_1: 189435648.0000 - false_positives_1: 303
81.3125 - false_negatives_1: 105402.8906 - val_loss: 0.0061 - val_accuracy:
0.9985 - val_precision_1: 0.9965 - val_recall_1: 0.9880 - val_true_positives
_1: 8752818.0000 - val_true_negatives_1: 194869744.0000 - val_false_positive
s_1: 30691.0137 - val_false_negatives_1: 105962.2812
Epoch 9/10
2316/2316 [==============================] - 290s 125ms/step - loss: 0.0108
- accuracy: 0.9980 - precision_1: 0.9966 - recall_1: 0.9883 - true_positives
_1: 8999292.0000 - true_negatives_1: 200304128.0000 - false_positives_1: 310
18.9570 - false_negatives_1: 106508.9219 - val_loss: 0.0058 - val_accuracy:
0.9988 - val_precision_1: 0.9966 - val_recall_1: 0.9886 - val_true_positives
_1: 9245788.0000 - val_true_negatives_1: 205738256.0000 - val_false_positive
s_1: 31299.4199 - val_false_negatives_1: 107012.1484
Epoch 10/10
2316/2316 [==============================] - 285s 123ms/step - loss: 0.0102
- accuracy: 0.9980 - precision_1: 0.9967 - recall_1: 0.9888 - true_positives
_1: 9492279.0000 - true_negatives_1: 211172624.0000 - false_positives_1: 315
78.8887 - false_negatives_1: 107539.2266 - val_loss: 0.0052 - val_accuracy:
0.9988 - val_precision_1: 0.9967 - val_recall_1: 0.9890 - val_true_positives
_1: 9738754.0000 - val_true_negatives_1: 216606704.0000 - val_false_positive
s_1: 31877.6191 - val_false_negatives_1: 108067.0078
```

Out[34]:

```
<keras.callbacks.History at 0x23eae64c7c0>
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [37]:

```
# k means
from sklearn.cluster import KMeans
```

In [51]:

```
km = KMeans(n_clusters=23)
y_predicted = km.fit_predict(data[['duration', 'src_bytes', 'dst_bytes', 'land', 'wrong
        'urgent', 'hot', 'num_failed_logins', 'logged_in', 'lnum_compromised',
        'lroot_shell', 'lsu_attempted', 'lnum_root', 'lnum_file_creations',
        'lnum_shells', 'lnum_access_files', 'lnum_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', 'protocol_type', 'service', 'flag']])
y_predicted
```

Out[51]:

```
array([16,  0,  0, ...,  0,  0, 16])
```

In [56]:

```
print("Accuracy = " ,metrics.accuracy_score(Y, y_predicted))
```

```
Accuracy =  0.13589530788227197
```

In [45]:

```
len(y_predicted)
```

Out[45]:

```
494020
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
# GAUSSIAN NAIVE BAYES

print("Confusion Matrix for Naive Bayes")

cm = confusion_matrix(Y_test, y1_pred)

print(cm)

# DECISION TREE

print("Confusion Matrix for Decision Tree")

cm = confusion_matrix(Y_test, y2_pred)

print(cm)

# SVM

print("Confusion Matrix for SVM")

cm = confusion_matrix(Y_test, y3_pred)

print(cm)

# K-MEANS CLUSTERING

print("Confusion Matrix for K-Means Clustering")

cm = confusion_matrix(Y_test, y1_pred)

print(cm)

# FULLY CONNECTED NEURAL NETWORK

print("Confusion Matrix for Neural Network")

cm = confusion_matrix(Y_test, y1_pred)

print(cm)
```

In [ ]:

```
# GAUSSIAN NAIVE BAYES

# Accuracy

print("Accuracy = " ,metrics.accuracy_score(Y_test, y1_pred)) # definition too

# Precision

print("Precision = " ,metrics.precision_score(Y_test, y1_pred))

# Recall

print("Recall = " ,metrics.recall_score(Y_test, y1_pred))

# F1 score

print("F1 score = " ,metrics.f1_score(Y_test, y1_pred))
```

In [ ]:

```
# DECISION TREE

# Accuracy

print("Accuracy = " ,metrics.accuracy_score(Y_test, y2_pred))

# Precision

print("Precision = " ,metrics.precision_score(Y_test, y2_pred))

# Recall

print("Recall = " ,metrics.recall_score(Y_test, y2_pred))

# F1 score

print("F1 score = " ,metrics.f1_score(Y_test, y2_pred))

```

In [ ]:

```
# SVM

# Accuracy

print("Accuracy = " ,metrics.accuracy_score(Y_test, y3_pred))

# Precision

print("Precision = " ,metrics.precision_score(Y_test, y3_pred))

# Recall

print("Recall = " ,metrics.recall_score(Y_test, y3_pred))

# F1 score

print("F1 score = " ,metrics.f1_score(Y_test, y3_pred))
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```