Importing Libraries

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder,OneHotEncoder from sklearn import preprocessing

from sklearn.feature\_selection import RFE

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split # Import train\_test\_split function from sklearn.preprocessing import Normalizer

import warnings warnings.filterwarnings("ignore")

# dataset doesn't have column names, so we have to provide it col\_names = ["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","label","difficulty\_level"]

# loading the dataset

traind = pd.read\_csv('KDDTrain+.txt',header=None, names=col\_names) testd = pd.read\_csv('KDDTest+.txt',header=None, names=col\_names)

# data = pd.concat([traind, testd], axis=0) # fill all the NaN field with 0

# traindata.update(traindata(col\_names).fillna(0))

# traindata[col\_names]=traindata[col\_names].fillna(0) # traindata.fillna(value=0, inplace=True)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # print dataset | | | | | | | | | | | | |
| traind |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **duration** | **protocol\_type** | **service** | **flag** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **...** | **dst\_host\_** |
| **0** | 0 | tcp | ftp\_data | SF | 491 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **1** | 0 | udp | other | SF | 146 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **2** | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **3** | 0 | tcp | http | SF | 232 | 8153 | 0 | 0 | 0 | 0 | ... |  |
| **4** | 0 | tcp | http | SF | 199 | 420 | 0 | 0 | 0 | 0 | ... |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **34555** | 0 | tcp | http | SF | 246 | 350 | 0 | 0 | 0 | 0 | ... |  |
| **34556** | 0 | udp | other | SF | 516 | 4 | 0 | 0 | 0 | 0 | ... |  |
| **34557** | 1 | tcp | smtp | SF | 2265 | 278 | 0 | 0 | 0 | 0 | ... |  |
| **34558** | 0 | tcp | http | SF | 217 | 280 | 0 | 0 | 0 | 0 | ... |  |
| **34559** | 0 | tcp | kshell | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |  |

34560 rows × 43 columns



# remove attribute 'difficulty\_level'

traind.drop(['difficulty\_level'],axis=1,inplace=True) traind.shape

(34560, 42)

# descriptive statistic of dataset traind.describe()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **duration** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **num\_failed\_logins** |
| **count** | 34560.000000 | 3.456000e+04 | 3.456000e+04 | 34560.000000 | 34560.000000 | 34560.000000 | 34560.000000 | 34560.000000 |
| **mean** | 298.852517 | 1.966618e+04 | 3.417070e+03 | 0.000087 | 0.022917 | 0.000116 | 0.205498 | 0.001100 |
| **std** | 2659.593706 | 2.060165e+06 | 8.593824e+04 | 0.009317 | 0.255830 | 0.017010 | 2.188124 | 0.042342 |
| **min** | 0.000000 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **50%** | 0.000000 | 4.400000e+01 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 0.000000 | 2.780000e+02 | 5.350000e+02 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **max** | 42862.000000 | 3.817091e+08 | 5.153771e+06 | 1.000000 | 3.000000 | 3.000000 | 77.000000 | 4.000000 |

8 rows × 38 columns



# number of attack labels traind['label'].value\_counts()

|  |  |
| --- | --- |
| normal | 18415 |
| neptune | 11393 |
| ipsweep | 981 |
| satan | 968 |

portsweep 809

smurf 712

nmap 407

back 267

warezclient 257

teardrop 249

pod 49

guess\_passwd 13

buffer\_overflow 10

warezmaster 9

imap 5

multihop 4

rootkit 4

phf 2

land 2

ftp\_write 1

loadmodule 1

spy 1

Name: label, dtype: int64

# changing attack labels to their respective attack class def change\_attack\_labels(df):

df.label.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm df.label.replace(['ftp\_write','guess\_passwd','httptunnel','imap','multihop','named','phf','sendmail',

'snmpgetattack','snmpguess','spy','warezclient','warezmaster','xlock','xsnoop'],'R2L',inplace=True) df.label.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inplace=True) df.label.replace(['buffer\_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'],'U2R',inplace=True)

# calling the function change\_attack\_labels(traind)

traind

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **duration** | **protocol\_type** | **service** | **flag** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **...** | **dst\_host\_** |
| **0** | 0 | tcp | ftp\_data | SF | 491 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **1** | 0 | udp | other | SF | 146 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **2** | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **3** | 0 | tcp | http | SF | 232 | 8153 | 0 | 0 | 0 | 0 | ... |  |
| **4** | 0 | tcp | http | SF | 199 | 420 | 0 | 0 | 0 | 0 | ... |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **34555** | 0 | tcp | http | SF | 246 | 350 | 0 | 0 | 0 | 0 | ... |  |
| **34556** | 0 | udp | other | SF | 516 | 4 | 0 | 0 | 0 | 0 | ... |  |
| **34557** | 1 | tcp | smtp | SF | 2265 | 278 | 0 | 0 | 0 | 0 | ... |  |
| **34558** | 0 | tcp | http | SF | 217 | 280 | 0 | 0 | 0 | 0 | ... |  |
| **34559** | 0 | tcp | kshell | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |  |

34560 rows × 42 columns



# distribution of attack classes traind.label.value\_counts()

normal 18415

Dos 12672

Probe 3165

R2L 292

U2R 15

Name: label, dtype: int64

# importing required libraries for normalizing data from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

# selecting numeric attributes columns from data numeric\_col = traind.select\_dtypes(include='number').columns

# using standard scaler for normalizing std\_scaler = StandardScaler()

def normalize(df,col): for i in col:

arr = df[i]

arr = np.array(arr)

df[i] = std\_scaler.fit\_transform(arr.reshape(len(arr),1)) return df

# data before normalization traind.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **duration** | **protocol\_type** | **service** | **flag** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **...** | **dst\_host\_srv\_** |
| **0** 0 | tcp | ftp\_data | SF | 491 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **1** 0 | udp | other | SF | 146 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **2** 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |  |
| **3** 0 | tcp | http | SF | 232 | 8153 | 0 | 0 | 0 | 0 | ... |  |
| **4** 0 | tcp | http | SF | 199 | 420 | 0 | 0 | 0 | 0 | ... |  |

5 rows × 42 columns



# calling the function

# traindata = normalize(traindata.copy(),numeric\_col) traindata = normalize(traind.copy(),numeric\_col)

# data after normalization traindata.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **duration** | **protocol\_type** | **service** | **flag** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **...** | **dst\_** |
| **0** 0.103271 | tcp | ftp\_data | SF | 0.009308 | 0.039763 | -0.009317 | 0.089579 | 0.006804 | 0.093916 | ... |  |
| **1** 0.103271 | udp | other | SF | 0.009475 | 0.039763 | -0.009317 | 0.089579 | 0.006804 | 0.093916 | ... |  |
| **2** 0. 103271 | tcp | private | S0 | 0.009546 | 0.039763 | -0.009317 | 0.089579 | 0.006804 | 0.093916 | ... |  |
| **3** 0.103271 | tcp | http | SF | 0.009433 | 0.055109 | -0.009317 | 0.089579 | 0.006804 | 0.093916 | ... |  |
| **4** 0.103271 | tcp | http | SF | 0.009449 | 0.034875 | -0.009317 | 0.089579 | 0.006804 | 0.093916 | ... |  |

5 rows × 42 columns



# One-hot-encoding

# selecting categorical data attributes cat\_col = ['protocol\_type','service','flag']

# creating a dataframe with only categorical attributes categorical = traindata[cat\_col]

categorical.head()

**4** tcp http SF

## protocol\_type service flag

**1** udp other SF

**2** tcp private S0

**3** tcp http SF

**0** tcp ftp\_data SF

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **protocol\_type\_icmp** | **protocol\_type\_tcp** | **protocol\_type\_udp** | **service\_IRC** | **service\_X11** | **service\_Z39\_50** | **service\_aol** | **s** |
| **0** 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
| **1** 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |
| **2** 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
| **3** 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
| **4** 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |

# Binary Classification



5 rows × 81 columns

# one-hot-encoding categorical attributes

categorical = pd.get\_dummies(categorical,columns=cat\_col) categorical.head()

# changing attack labels into two categories 'normal' and 'abnormal'

bin\_label = pd.DataFrame(traindata.label.map(lambda x:'normal' if x=='normal' else 'abnormal')) # creating a dataframe with binary labels (normal,abnormal)

bin\_data = traindata.copy() bin\_data['label'] = bin\_label

# label encoding (0,1) binary labels (abnormal,normal) lec = preprocessing.LabelEncoder()

enc\_label = bin\_label.apply(lec.fit\_transform) bin\_data['intrusion'] = enc\_label

lec.classes\_

array(['abnormal', 'normal'], dtype=object)

np.save("lec\_classes.npy",lec.classes\_,allow\_pickle=True)

# dataset with binary labels and label encoded column bin\_data.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **duration** | **protocol\_type** | **service** | **flag** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **...** | **dst** |
| **0** -0.112369 | tcp | ftp\_data | SF | -0.009308 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |  |
| **1** -0.112369 | udp | other | SF | -0.009475 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |  |
| **2** -0.112369 | tcp | private | S0 | -0.009546 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |  |
| **3** -0.112369 | tcp | http | SF | -0.009433 | 0.055109 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |  |
| **4** -0.112369 | tcp | http | SF | -0.009449 | -0.034875 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |  |

5 rows × 43 columns



# one-hot-encoding attack label

bin\_data = pd.get\_dummies(bin\_data,columns=['label'],prefix="",prefix\_sep="") bin\_data['label'] = bin\_label

bin\_data

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **duration** | **protocol\_type** | **service** | **flag** | **src\_bytes** | **dst\_bytes** | **land** | **wrong\_fragment** | **urgent** | **hot** | **...** |
| **0** | -0.112369 | tcp | ftp\_data | SF | -0.009308 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **1** | -0.112369 | udp | other | SF | -0.009475 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **2** | -0.112369 | tcp | private | S0 | -0.009546 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **3** | -0.112369 | tcp | http | SF | -0.009433 | 0.055109 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **4** | -0.112369 | tcp | http | SF | -0.009449 | -0.034875 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **34555** | -0.112369 | tcp | http | SF | -0.009427 | -0.035690 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **34556** | -0.112369 | udp | other | SF | -0.009296 | -0.039716 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **34557** | -0.111993 | tcp | smtp | SF | -0.008447 | -0.036528 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **34558** | -0.112369 | tcp | http | SF | -0.009441 | -0.036504 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |
| **34559** | -0.112369 | tcp | kshell | S0 | -0.009546 | -0.039763 | -0.009317 | -0.089579 | -0.006804 | -0.093916 | ... |

34560 rows × 45 columns



# creating a dataframe with only numeric attributes of binary class dataset and encoded label attribute numeric\_bin = bin\_data[numeric\_col]

numeric\_bin['intrusion'] = bin\_data['intrusion']

# finding the attributes which have more than 0.5 correlation with encoded attack label attribute corr= numeric\_bin.corr()

corr\_y = abs(corr['intrusion']) highest\_corr = corr\_y[corr\_y >0.5] highest\_corr.sort\_values(ascending=True)

|  |  |
| --- | --- |
| count | 0.475103 |
| srv\_serror\_rate | 0.547395 |
| serror\_rate | 0.549021 |
| dst\_host\_serror\_rate | 0.501261 |

|  |  |
| --- | --- |
| dst\_host\_srv\_serror\_rate | 0.654846 |
| logged\_in | 0.383108 |
| dst\_host\_same\_srv\_rate | 0.192170 |
| dst\_host\_srv\_count | 0.220509 |
| same\_srv\_rate | 0.254936 |
| intrusion | 1.000000 |
| Name: intrusion, dtype: | float64 |

# selecting attributes found by using pearson correlation coefficient

numeric\_bin = bin\_data[['count','srv\_serror\_rate','serror\_rate','dst\_host\_serror\_rate','dst\_host\_srv\_serror\_rate', 'logged\_in','dst\_host\_same\_srv\_rate','dst\_host\_srv\_count','same\_srv\_rate']]

# joining the selected attribute with the one-hot-encoded categorical dataframe numeric\_bin = numeric\_bin.join(categorical)

# then joining encoded, one-hot-encoded, and original attack label attribute bin\_data = numeric\_bin.join(bin\_data[['intrusion','abnormal','normal','label']])

bin\_data.drop('label', axis=1,inplace=True)

bin\_data

34560 rows × 93 columns

**34555** -0.669719 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**34556** 0.422194 -0.634850 -0.640722 -0.642689 -0.627953 -0.807763

**34557** -0.730867 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**34558** -0.652249 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**34559** 0.151399 1.597756 1.593371 NaN NaN -0.807763

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **count** | **srv\_serror\_rate** | **serror\_rate** | **dst\_host\_serror\_rate** | **dst\_host\_srv\_serror\_rate** | **logged\_in** | **dst\_host\_sam** |
| **0** | -0.722131 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | -0.807763 |  |
| **1** | -0.626043 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | -0.807763 |  |
| **2** | 0.334841 | 1.597756 | 1.593371 | 1.601676 | 1.612222 | -0.807763 |  |
| **3** | -0.695925 | -0.188329 | -0.193903 | -0.575358 | -0.605551 | 1.237986 |  |
| **4** | -0.477543 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | 1.237986 |  |
| **...** | ... | ... | ... | ... | ... | ... |  |



# saving final dataset to disk bin\_data.to\_csv("bin\_data.csv")

# bin\_data.drop('label', axis=1,inplace=True) # final dataset for binary classification bin\_data

34560 rows × 93 columns

## count srv\_serror\_rate serror\_rate dst\_host\_serror\_rate dst\_host\_srv\_serror\_rate logged\_in dst\_host\_sam

**0** -0.722131 -0.634850 -0.640722 -0.642689 -0.627953 -0.704763

**1** -0.626043 -0.634850 -0.640722 -0.642689 -0.627953 -0.603460

**2** 0.334841 1.597756 1.593371 1.601676 1.612222 -0.737164

**3** -0.695925 -0.188329 -0.193903 -0.575358 -0.605551 0.952012

**4** -0.477543 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**...** ... ... ... ... ... ...

**34555** -0.669719 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**34556** 0.422194 -0.634850 -0.640722 -0.642689 -0.627953 -0.807763

**34557** -0.730867 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**34558** -0.652249 -0.634850 -0.640722 -0.642689 -0.627953 1.237986

**34559** 0.151399 1.597756 1.593371 NaN NaN -0.807763

traindata.fillna(value=0, inplace=True)

traindata

for i in traindata.select\_dtypes('object').columns: le = LabelEncoder().fit(traindata[i]) traindata[i] = le.transform(traindata[i])

# for i in testdata.select\_dtypes('object').columns: # le = LabelEncoder().fit(testdata[i])

# testdata[i] = le.transform(testdata[i])

**34558** -0.112369 1 23 9 -0.009441 -0.036504 -0.009317 -0.089579 -0.006804 -0.093916 ...

**34559** -0.112369 1 29 5 -0.009546 -0.039763 -0.009317 -0.089579 -0.006804 -0.093916 ...

34560 rows × 42 columns

## duration protocol\_type service flag src\_bytes dst\_bytes land wrong\_fragment urgent hot ...

**0** -0.112369 1 20 9 -0.009308 -0.039763 -0.009317 -0.089579 -0.006804 -0.093916 ...

**1** -0.112369 2 42 9 -0.009475 -0.039763 -0.009317 -0.089579 -0.006804 -0.093916 ...

**2** -0.112369 1 47 5 -0.009546 -0.039763 -0.009317 -0.089579 -0.006804 -0.093916 ...

**3** -0.112369 1 23 9 -0.009433 0.055109 -0.009317 -0.089579 -0.006804 -0.093916 ...

**4** -0.112369 1 23 9 -0.009449 -0.034875 -0.009317 -0.089579 -0.006804 -0.093916 ...

**...** ... ... ... ... ... ... ... ... ... ... ...

**34555** -0.112369 1 23 9 -0.009427 -0.035690 -0.009317 -0.089579 -0.006804 -0.093916 ...

**34557** -0.111993 1 52 9 -0.008447 -0.036528 -0.009317 -0.089579 -0.006804 -0.093916 ...

Defining and Training LSTM Model

**34556** -0.112369

2

42

9

-0.009296

-0.039716 -0.009317

-0.089579 -0.006804 -0.093916

...

traindata=traindata.astype(float)



from sklearn.metrics import accuracy\_score # for calculating accuracy of model

from sklearn.model\_selection import train\_test\_split # for splitting the dataset for training and testing from sklearn.metrics import classification\_report # for generating a classification report of model import pickle # saving and loading trained model

from os import path import numpy as np import pandas as pd

import matplotlib.pyplot as plt

bin\_data = pd.read\_csv('bin\_data.csv') bin\_data.drop(bin\_data.columns[0],axis=1,inplace=True)

# importing library for LSTM layers

from keras.layers import LSTM, Dense, Activation, Dropout from keras.models import Sequential

X = bin\_data.iloc[:,0:93] # dataset excluding target attribute (encoded, one-hot-encoded,original)

Y = bin\_data[['intrusion']].values # target attribute

# splitting the dataset 70% for training and 30% testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,Y, test\_size=0.30, random\_state=46) X\_train = X\_train.to\_numpy()

X\_train = np.reshape(X\_train, (X\_train.shape[0],X\_train.shape[1],1)) lst = Sequential() # initializing model

# input layer and LSTM layer with 50 neurons

lst.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1)))

# outpute layer with sigmoid activation lst.add(Dense(1, activation='sigmoid'))

# defining loss function, optimizer, metrics and then compiling model lst.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

X\_train = X\_train.astype('float32')

# defining loss function, optimizer, metrics and then compiling model lst.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

# X\_train = X\_train.astype('float32')

lst.summary()

Model: "sequential\_16"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |
| --- | --- | --- |
| lstm\_1 (LSTM) | (None, 93, 50) | 10400 |
| dense\_1 (Dense) | (None, 93, 1) | 51 |

=================================================================

Total params: 10,451

Trainable params: 10,451

Non-trainable params: 0

# training the model on training dataset

history = lst.fit(X\_train, y\_train, epochs=20, batch\_size=1000,validation\_split=0.3)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch | 11/20 |  | | | | | | | | |
| 26/26  Epoch | [==============================] 12/20 | - 15s | 368ms/step | - loss: | 0.6920 | - accuracy: | 0.8487 | - val\_loss: | 0.6877 | - val\_a |
| 26/26  Epoch 26/26 | [==============================] 13/20 [==============================] | - 13s  - 10s | 403ms/step  442ms/step | * loss: * loss: | 0.5901  0.3305 | * accuracy: * accuracy: | 0.8607  0.8617 | * val\_loss: * val\_loss: | 0.4484  0.4262 | * val\_a * val\_a |
| Epoch 26/26 | 14/20 [==============================] | - 11s | 412ms/step | - loss: | 0.3284 | - accuracy: | 0.8512 | - val\_loss: | 0.4139 | - val\_a |
| Epoch | 15/20 |  |  |  |  |  |  |  |  |  |
| 26/26  Epoch 26/26 | [==============================] 16/20 [==============================] | - 11s  - 12s | 471ms/step  425ms/step | * loss: * loss: | 0.4216  0.4058 | * accuracy: * accuracy: | 0.8922  0.9110 | * val\_loss: * val\_loss: | 0.4070  0.2021 | * val\_a * val\_a |
| Epoch 26/26 | 17/20 [==============================] | - 12s | 433ms/step | - loss: | 0.4748 | - accuracy: | 0.9223 | - val\_loss: | 0.3175 | - val\_a |
| Epoch | 18/20 |  |  |  |  |  |  |  |  |  |
| 26/26  Epoch | [==============================] 19/20 | - 12s | 446ms/step | - loss: | 0.2080 | - accuracy: | 0.9166 | - val\_loss: | 0.3010 | - val\_a |
| 26/26  Epoch 26/26 | [==============================] 20/20 [==============================] | - 11s  - 12s | 439ms/step  454ms/step | * loss: * loss: | 0.2318  0.1069 | * accuracy: * accuracy: | 0.9235  0.9240 | * val\_loss: * val\_loss: | 0.2075  0.2470 | * val\_a * val\_a |

lst.evaluate(X\_test, y\_test, batch\_size=100,verbose=1)

87/87 [==============================] - 3s 46ms/step - loss: 0.3975 - accuracy: 0.9350 [0.4061363935470581, 0.9350323534011841]

p = lst.predict(X\_test)

from sklearn import metrics

from sklearn.metrics import accuracy\_score, confusion\_matrix from sklearn.metrics import recall\_score

from sklearn.metrics import precision\_score from sklearn.metrics import f1\_score

from sklearn.metrics import log\_loss

from sklearn.metrics import PrecisionRecallDisplay

y\_eval = np.argmax(p, axis=1)

f1 = f1\_score(y\_test, y\_eval, average=None)

print("F1",f1[0])

F1 0.8339777640518839

pscore = precision\_score(y\_test, y\_eval, average=None)

print("PC",pscore[0])

PC 0.9971026490066225

rscore = recall\_score(y\_test, y\_eval, average=None)

print("RC",rscore[0])

RC 0.8933497536945813

# bin\_data.drop(bin\_data.columns[96],axis=1,inplace=True) bin\_data.fillna(value=0)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **count** | **srv\_serror\_rate** | **serror\_rate** | **dst\_host\_serror\_rate** | **dst\_host\_srv\_serror\_rate** | **logged\_in** | **dst\_host\_sam** |
| **0** | -0.722131 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | -0.807763 |  |
| **1** | -0.626043 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | -0.807763 |  |
| **2** | 0.334841 | 1.597756 | 1.593371 | 1.601676 | 1.612222 | -0.807763 |  |
| **3** | -0.695925 | -0.188329 | -0.193903 | -0.575358 | -0.605551 | 1.237986 |  |
| **4** | -0.477543 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | 1.237986 |  |
| **...** | ... | ... | ... | ... | ... | ... |  |
| **34555** | -0.669719 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | 1.237986 |  |
| **34556** | 0.422194 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | -0.807763 |  |
| **34557** | -0.730867 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | 1.237986 |  |
| **34558** | -0.652249 | -0.634850 | -0.640722 | -0.642689 | -0.627953 | 1.237986 |  |
| **34559** | 0.151399 | 1.597756 | 1.593371 | 0.000000 | 0.000000 | -0.807763 |  |

34560 rows × 93 columns



bin\_data.replace(np.nan, 0, inplace=True)

**Importing the Model and Generating Synthetic Data Using CTGAN**

# importing ctgan

from ctgan import CTGANSynthesizer

ctgan = CTGANSynthesizer(epochs=10)

ctgan.fit(bin\_data)

sampleTr = ctgan.sample(500)

np.save("generateddatafromctgan.csv", sampleTr)

X = sampleTr.iloc[:,0:93] # dataset excluding target attribute (encoded, one-hot-encoded,original)

Y = sampleTr[['intrusion']].values # target attribute

# splitting the dataset 70% for training and 30% testing

X\_train, X\_test, y\_train, Y\_test = train\_test\_split(X,Y, test\_size=0.3, random\_state=50)

l, a =lst.evaluate(X\_train, y\_train, batch\_size=100,verbose=1) print("Loss: %f\t Accuracy: %f"% (l,a))

4/4 [==============================] - 0s 25ms/step - loss: 1.0716 - accuracy: 0.9010

Loss: 1.071646 Accuracy: 0.910983

# p = model.predict(X\_test) p = lst.predict(X\_test)

y\_eval = np.argmax(p, axis=1)

from sklearn.metrics import roc\_curve, auc

fpr, tpr, threshold = roc\_curve(Y\_test, y\_eval) roc\_auc = auc(fpr, tpr)

roc\_auc

0.7323649691358025

rscore = recall\_score(Y\_test, y\_eval, average=None)

rscore[0]

0.17525455315305752

pscore = precision\_score(Y\_test, y\_eval, average=None)

# pscore[0]

f1score = f1\_score(Y\_test, y\_eval, average=None)

f1score[0]

0.49375000000000001

logLoss = log\_loss(Y\_test, y\_eval)

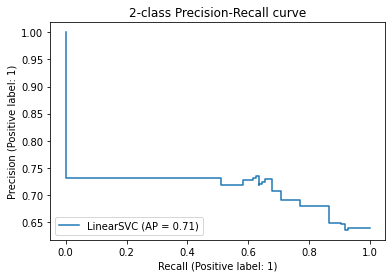
logLoss

13.35526540249173

from sklearn.metrics import PrecisionRecallDisplay

display = PrecisionRecallDisplay.from\_predictions(Y\_test, y\_eval, name="LinearSVC")

\_ = display.ax\_.set\_title("2-class Precision-Recall curve")



 0s completed at 13:08