Created on Thu Jul 26 22:52:20 2018

@author: bipra

"""

# -\*- coding: utf-8 -\*-

"""

Created on Tue May 15 00:11:41 2018

@author: bipra

"""

# -\*- coding: utf-8 -\*-

"""

Created on Sat Feb 10 13:01:52 2018

@author: bipra

"""

# Part 1 - Data Preprocessing

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TRAINING SET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Importing the training set

dataset\_train = pd.read\_csv('UNSW\_NB15\_training-set\_5000.csv')

#training\_set = dataset\_train.iloc[:, 5:6].values

training\_set = dataset\_train.iloc[:, [3,7,10,27,35]].values

#Features: sbytes, sttl, smean, ct\_dst\_sport\_ltm

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder\_X = LabelEncoder()

training\_set[:, 0] = labelencoder\_X.fit\_transform(training\_set[:, 0])

# Feature Scaling | Normalizing ---- TRAINING SET

from sklearn.preprocessing import MinMaxScaler

sclr = MinMaxScaler(feature\_range = (0, 1))

training\_set\_scaled = sclr.fit\_transform(training\_set)

# Creating a data structure with 80 timesteps and 1 output

X\_train = []

y\_train = []

for i in range(1, 5451):

X\_train.append(training\_set\_scaled[i-1:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshaping

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# Part 2 - Building the RNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

from keras.layers import TimeDistributed

from keras.layers import Bidirectional

from numpy import array

from random import random

from numpy import cumsum

from keras import optimizers

import math

# Initialising the BLSTM

regressor = Sequential()

# Adding the first BLSTM layer and some Dropout regularisation

regressor.add(Bidirectional(LSTM(units = 5, return\_sequences=True), input\_shape=(X\_train.shape[1], 1), merge\_mode='concat'))

regressor.add(Dropout(0.2))

# Adding hidden layer

regressor.add(Bidirectional(LSTM(units = 220, return\_sequences=True)))

regressor.add(Dropout(0.2))

regressor.add(Bidirectional(LSTM(units = 240, return\_sequences=True)))

regressor.add(Dropout(0.2))

regressor.add(Bidirectional(LSTM(units = 260)))

regressor.add(Dropout(0.2))

# Adding the output layer

regressor.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the RNN

regressor.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics=['accuracy','mse'])

import time

start=time.time()

# Fitting the BLSTM to the Training set | \*\*\*\* Splitting Training set into TRAIN & VALIDATION set \*\*\*\*\*\*

history = regressor.fit(X\_train, y\_train, epochs = 100, validation\_split=0.33, batch\_size = 132)

end=time.time()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import time

start = time.time()

x\_pred = regressor.predict(X\_train) #predict

end = time.time()

x\_pred = (x\_pred>0.162244).astype(int) #continious to integer casting

y\_train = (y\_train>0.5).astype(int) ##continious to integer casting

#confusion matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_train, x\_pred)

print (cm)

#plotting Confusion Matrix

plt.clf()

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)

classNames = ['Attack','Normal']

plt.title('Attack or Normal Confusion Matrix - Test Data')

plt.ylabel('True label')

plt.xlabel('Predicted label')

tick\_marks = np.arange(len(classNames))

plt.xticks(tick\_marks, classNames, rotation=45)

plt.yticks(tick\_marks, classNames)

s = [['TP', 'FP'], ['FN','TN']]

for i in range(2):

for j in range(2):

plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))

plt.show()

#accuracy score

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred, normalize=False)

#classification report

from sklearn.metrics import classification\_report

report = classification\_report(y\_train, x\_pred)

print(report)

print(history.history['acc'])

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

plt.show()

plt.plot(history.history['acc'])

plt.title('Model Accuracy')

plt.ylabel('percentage')

plt.xlabel('epoch')

plt.legend(['accuracy'], loc='upper left')

plt.show()

##\*\*\*\*\*\*\*\*\*\*\*\* TESTING MSE of the Network\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

##\*\*\*\*\*\*\*\*\*\*\*MSE becoming zero means your expected neuron outputs are exactly matched by

#actual neuron outputs. That means network is ideally trained and goal is 100%.

#Practically speaking if you are dealing with certain classification task,

#this means your features are enough strong to result in 100% classification rate.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TEST SET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Importing Test set

#dataset\_test = pd.read\_csv('testset\_1\_nor2393\_att1288.csv')

#dataset\_test = pd.read\_csv('UNSW\_NB15\_testing-set\_10000.csv')

#dataset\_test = pd.read\_csv('testset\_conf\_4092a\_112n.csv')

dataset\_test = pd.read\_csv('UNSW\_NB15\_training-set.csv')

test\_set = dataset\_test.iloc[:, [3,7,10,27,35]].values

#Features: service, sbytes, sttl, smean, ct\_dst\_sport\_ltm

labelencoder\_X\_test = LabelEncoder()

test\_set[:, 0] = labelencoder\_X\_test.fit\_transform(test\_set[:, 0])

# Feature Scaling | Normalizing ---- TEST SET

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0, 1))

test\_set\_scaled = sclr.fit\_transform(test\_set)

# Creating a data structure with 60 timesteps and 1 output

X\_test = []

y\_test = []

for i in range(1, 82332):

X\_test.append(test\_set\_scaled[i-1:i, 0])

y\_test.append(test\_set\_scaled[i, 0])

X\_test, y\_test = np.array(X\_test), np.array(y\_test)

# Reshaping

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Performance Analysis \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Performance Analysis \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import time

start = time.time()

y\_pred = regressor.predict(X\_test) #predict

end = time.time()

y\_pred = (y\_pred>0.173008).astype(int) #continious to integer casting

y\_test = (y\_test>0.8).astype(int) ##continious to integer casting

#confusion matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print (cm)

#plotting Confusion Matrix

plt.clf()

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)

classNames = ['Attack','Normal']

plt.title('Attack or Normal Confusion Matrix - Test Data')

plt.ylabel('True label')

plt.xlabel('Predicted label')

tick\_marks = np.arange(len(classNames))

plt.xticks(tick\_marks, classNames, rotation=45)

plt.yticks(tick\_marks, classNames)

s = [['TP', 'FP'], ['FN','TN']]

for i in range(2):

for j in range(2):

plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))

plt.show()

#accuracy score

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred, normalize=False)

#classification report

from sklearn.metrics import classification\_report

report = classification\_report(y\_test, y\_pred)

print(report)