# 1. Libraries and settings

import pandas as pd

import numpy as np

import math

import sklearn

import sklearn.preprocessing

import datetime

import os

import matplotlib.pyplot as plt

import tensorflow as tf

import matplotlib.pyplot as plt

import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Reshape, GlobalAveragePooling1D

from keras.layers import Conv2D, MaxPooling2D, Conv1D, MaxPooling1D

from keras.utils import np\_utils

#display parent directory and working directory

print(os.path.dirname(os.getcwd())+':', os.listdir(os.path.dirname(os.getcwd())));

print(os.getcwd()+':', os.listdir(os.getcwd()));

# 2. Analyze data

df = pd.read\_csv("../../prices-split-adjusted.csv", index\_col = 0)

print(df.info())

print(df.head())

print(df.values.shape)

# number of different stocks

print('\nnumber of different stocks: ', len(list(set(df.symbol))))

print(list(set(df.symbol))[:10])

df.tail()

df.describe()

#3.plot data

plt.figure(figsize=(15, 5));

plt.subplot(1,2,1);

plt.plot(df[df.symbol == 'EQIX'].open.values, color='red', label='open')

plt.plot(df[df.symbol == 'EQIX'].close.values, color='green', label='close')

plt.plot(df[df.symbol == 'EQIX'].low.values, color='blue', label='low')

plt.plot(df[df.symbol == 'EQIX'].high.values, color='black', label='high')

plt.title('stock price')

plt.xlabel('time [days]')

plt.ylabel('price')

plt.legend(loc='best')

#plt.show()

plt.subplot(1,2,2);

plt.plot(df[df.symbol == 'EQIX'].volume.values, color='black', label='volume')

plt.title('stock volume')

plt.xlabel('time [days]')

plt.ylabel('volume')

plt.legend(loc='best');

# 3. Manipulate data

#- choose a specific stock

#- drop feature: volume

#- normalize stock data

#- create train and test data sets

def feature\_normalize(train):

train\_norm = train.apply(lambda x: (x - np.min(x)) / (np.max(x) - np.min(x))) #標準化(介於0~1之間)

return train\_norm

## 很重要 切割視窗

def create\_segments\_and\_labels(df, time\_steps, step):#, label\_name):

"""

This function receives a dataframe and returns the reshaped segments

of x,y,z acceleration as well as the corresponding labels

Args:

df: Dataframe in the expected format

time\_steps: Integer value of the length of a segment that is created

Returns:

reshaped\_segments

labels:

"""

#圖畫中的overlap越高，代表資料中的相關性越強

#圖中80筆資料一次跳40筆，代表其並非相關性高

#feature 有四個

N\_FEATURES = 4

#選擇測試切出20%

test\_set\_size\_percentage = 20

segments = []

labels = []

data\_raw = df.as\_matrix()

#創造時間窗，將所有選擇特徵一起切割視窗

for i in range(0, len(data\_raw) - time\_steps, step):#

segments.append(data\_raw[i: i + time\_steps])

segments = np.array(segments);

test\_set\_size = int(np.round(test\_set\_size\_percentage/100\*segments.shape[0]));

train\_set\_size = segments.shape[0] - (test\_set\_size);

#以訓練資料占比分割訓練測試集，並以視窗最後一筆資料當作預測值

x\_train = segments[:train\_set\_size,:-1,:]

y\_train = segments[:train\_set\_size,-1,:]

# x\_valid = data[train\_set\_size:train\_set\_size+valid\_set\_size,:-1,:]

# y\_valid = data[train\_set\_size:train\_set\_size+valid\_set\_size,-1,:]

x\_test = segments[train\_set\_size:,:-1,:]

y\_test = segments[train\_set\_size:,-1,:]

return [x\_train, y\_train, x\_test, y\_test]

# return [x\_train, y\_train, x\_valid, y\_valid, x\_test, y\_test]

# choose one stock & drop volume

df\_stock = df[df.symbol == 'EQIX'].copy()

df\_stock.drop(['symbol'],1,inplace=True)

df\_stock.drop(['volume'],1,inplace=True)

cols = list(df\_stock.columns.values)

print('df\_stock.columns.values = ', cols)

# normalize stock

df\_stock\_norm = df\_stock.copy()

df\_stock\_norm = feature\_normalize(df\_stock\_norm)

# create train, test data

time\_steps = 20# choose sequence length

step = 5

x\_train, y\_train, x\_test, y\_test = create\_segments\_and\_labels(df\_stock\_norm, time\_steps, step)

print('x\_train.shape = ',x\_train.shape)

print('y\_train.shape = ', y\_train.shape)

# print('x\_valid.shape = ',x\_valid.shape)

# print('y\_valid.shape = ', y\_valid.shape)

print('x\_test.shape = ', x\_test.shape)

print('y\_test.shape = ',y\_test.shape)

df\_stock\_norm.values.shape

plt.plot(df\_stock\_norm.open.values, color='red', label='open')

plt.plot(df\_stock\_norm.close.values, color='green', label='close')

plt.plot(df\_stock\_norm.low.values, color='blue', label='low')

plt.plot(df\_stock\_norm.high.values, color='black', label='high')

#plt.plot(df\_stock\_norm.volume.values, color='gray', label='volume')

plt.title('stock')

plt.xlabel('time [days]')

plt.ylabel('normalized price/volume')

plt.legend(loc='best')

plt.show()

#reshape資料

num\_time\_periods, num\_sensors = x\_train.shape[1], x\_train.shape[2]

input\_shape = (num\_time\_periods\*num\_sensors) ## 80\*3 每一筆資料 80(時間窗) 3個變數( xyz)

x\_train\_reshape = x\_train.reshape(x\_train.shape[0], input\_shape).astype('float32')

print(f"x\_train\_reshape.shape:{x\_train\_reshape.shape}")

x\_test\_reshape = x\_test.reshape(x\_test.shape[0], input\_shape).astype('float32')

print(f"x\_test\_reshape.shape:{x\_test\_reshape.shape}")

#建立模型

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten, LSTM, TimeDistributed, RepeatVector, GRU

from keras.layers import SimpleRNN, Activation, Dense, RNN

from keras.layers.normalization import BatchNormalization

from keras.optimizers import Adam

from keras.callbacks import EarlyStopping, ModelCheckpoint

import matplotlib.pyplot as plt

from keras.callbacks import ReduceLROnPlateau

%matplotlib inline

## build one To One Model(shape):

model\_gru = Sequential()

model\_gru.add(GRU(units=128, input\_shape=( x\_train.shape[1], x\_train.shape[2])))

#units隱藏層神經元個數

model\_gru.add(Dropout(0.1))

model\_gru.add(Dense(64, activation = 'relu'))

model\_gru.add(Dropout(0.1))

model\_gru.add(Dense(16, activation = 'relu'))#三層隱藏層

model\_gru.add(Dropout(0.1))

model\_gru.add(Dense(num\_classes, activation='softmax'))

#num\_classes不能改

#開始訓練

model\_gru.compile(loss='MSE',

optimizer='adam', metrics=['mse'])

model\_gru.summary()

learning\_rate\_function = ReduceLROnPlateau(monitor='mean\_squared\_error',

patience=3, #準確率重複3次就要減少

verbose=1,

factor=0.5, #準確率乘上factor設成下一個learning\_rate

min\_lr=0.00001)

print("\n--- Fit the model ---\n")

train\_history = model\_gru.fit(x=x\_train, y= y\_train, validation\_split=0.18, epochs=200, batch\_size=10,callbacks=[learning\_rate\_function],verbose=2)#callbacks=[learning\_rate\_function], verbose=2)

print("\n--- Learning curve of model training ---\n")

get\_ipython().magic('matplotlib inline')

## IPython有一組預先定義好的所謂的魔法函數（Magic Functions），你可以通過命令列的語法形式來訪問它們。

#繪圖

import matplotlib.pyplot as plt

def show\_train\_history(train\_history, train, validation):

plt.plot(train\_history.history[train])

plt.plot(train\_history.history[validation])

plt.title("Train History")

plt.ylabel(train)

plt.xlabel('Epoch')

plt.show()

show\_train\_history(train\_history, "mean\_squared\_error", "val\_loss") ## 訓練正確率圖

scorelstm = model\_gru.evaluate(x\_test, y\_test)

print(f"MSE:{scores[0]}")