# 1. Libraries and settings

import pandas as pd

import numpy as np

import math

import sklearn

import sklearn.preprocessing

from sklearn import metrics

from sklearn.metrics import classification\_report

import seaborn as sns

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()

#看資料

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:

"""

#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(df) - time\_steps, step):#

segments.append(df.values[i: i + time\_steps])

#以當期四種特徵預測下一期收盤價

rate = (df.open.values[i + time\_steps]-df.open.values[i + time\_steps-1])/df.open.values[i + time\_steps-1]

temp = rate

if temp < 0:

if temp <= -0.2:

label =0

elif temp <= -0.1:

label =1

elif temp < 0:

label =2

else:

if temp == 0:

label =3

elif temp <= 0.1:

label =4

elif temp <= 0.2:

label =5

elif temp >0.2:

label =6

labels.append([label])

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

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

print(train\_set\_size)

# segments = np.array(segments);

reshaped\_segments\_train = np.asarray(segments[:train\_set\_size], dtype= np.float32).reshape(-1, time\_steps, N\_FEATURES)

reshaped\_segments\_test = np.asarray(segments[train\_set\_size:], dtype= np.float32).reshape(-1, time\_steps, N\_FEATURES)

labels\_train = np.asarray(labels[:train\_set\_size])

labels\_test = np.asarray(labels[train\_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,:,:]#(1394, 19, 4)

# y\_train = lables[:train\_set\_size,-1,:]#(1394, 4)

# 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 reshaped\_segments\_train, labels\_train, reshaped\_segments\_test,labels\_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\_test.shape = ',x\_test.shape)

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

num\_classes = 7

y\_train\_oneshot = np\_utils.to\_categorical(y\_train, num\_classes)

print(f"y\_train\_oneshot:{y\_train\_oneshot.shape}")

y\_test\_oneshot = np\_utils.to\_categorical(y\_test, num\_classes)

print(f"y\_test\_oneshot:{y\_test\_oneshot.shape}")

#繪刪除特徵後圖形

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

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

model\_cnn1d = Sequential()

num\_time\_periods\*num\_sensors

model\_cnn1d.add(Reshape((num\_time\_periods, num\_sensors), input\_shape=(input\_shape,)))

model\_cnn1d.add(Conv1D(100, 2, activation='relu', input\_shape=(num\_time\_periods, num\_sensors)))

model\_cnn1d.add(Conv1D(100, 2, activation='relu'))

model\_cnn1d.add(MaxPooling1D(3))

model\_cnn1d.add(Conv1D(160, 2, activation='relu'))

model\_cnn1d.add(Conv1D(160, 2, activation='relu'))

model\_cnn1d.add(GlobalAveragePooling1D())

model\_cnn1d.add(Dropout(0.5))

model\_cnn1d.add(Dense(num\_classes, activation='softmax'))

print(model\_cnn1d.summary())

#開始訓練

BATCH\_SIZE = 400

EPOCHS = 50

from keras.callbacks import ReduceLROnPlateau

learning\_rate\_function = ReduceLROnPlateau(monitor='val\_acc',

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) #降

# training stops early

callbacks\_list = [

keras.callbacks.ModelCheckpoint(

filepath='best\_model.{epoch:02d}-{val\_loss:.2f}.h5',

monitor='val\_loss', save\_best\_only=True),

keras.callbacks.EarlyStopping(monitor='acc', patience=1)

]

#...

model\_cnn1d.compile(loss='categorical\_crossentropy',

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

history = model\_cnn1d.fit(x\_train\_reshape, y\_train\_oneshot, batch\_size=BATCH\_SIZE, epochs=EPOCHS, callbacks=[learning\_rate\_function], validation\_split=0.2, verbose=1)

# %%

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

#繪圖

#訓練驗證圖

# summarize history for accuracy and loss

plt.figure(figsize=(6, 4))

plt.plot(history.history['acc'], "g--", label="Accuracy of training data")

plt.plot(history.history['val\_acc'], "g", label="Accuracy of validation data")

plt.plot(history.history['loss'], "r--", label="Loss of training data")

plt.plot(history.history['val\_loss'], "r", label="Loss of validation data")

plt.title('Model Accuracy and Loss')

plt.ylabel('Accuracy and Loss')

plt.xlabel('Training Epoch')

plt.ylim(0)

plt.legend()

plt.show()

#正確誤差圖

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, "acc", "val\_acc") ## 訓練正確率圖

show\_train\_history(train\_history, "loss", "val\_loss") ## 訓練誤差圖

#評估測試準確度

score = model\_cnn1d.evaluate(x\_test\_reshape, y\_test\_oneshot, verbose=1)

print("\nAccuracy on test data: %0.2f" % score[1])

print("\nLoss on test data: %0.2f" % score[0])

#混沌矩陣

# %%

print("\n--- Confusion matrix for test data ---\n")

y\_pred\_test = model\_cnn1d.predict(x\_test\_reshape)

# Take the class with the highest probability from the test predictions

max\_y\_pred\_test = np.argmax(y\_pred\_test, axis=1)

max\_y\_test = np.argmax(y\_test\_oneshot, axis=1)

show\_confusion\_matrix(max\_y\_test, max\_y\_pred\_test)

# %%

print("\n--- Classification report for test data ---\n")

print(classification\_report(max\_y\_test, max\_y\_pred\_test))