```
# 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
# %%

print("\n--- Create neural network model ---\n")

input_shape = (x_train.shape[1], x_train.shape[2], 1)    ##   定義   CNN 的輸入維度!! (10*10)
```

```
# 建立簡單的線性執行的模型
# model.add(Convolution2D(32, (3, 3), activation='relu', input_shape=(1,28,28), data_format='channels_first'))
model_cnn2d = Sequential()
# 建立卷積層，filter=32,即 output space 的深度, Kernal Size: 3x3, activation function 採用 relu
model_cnn2d.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))   ## 注意喔!!  這一邊放的是 (10,10,1) 的型態
# 建立卷積層，filter=64,即 output size, Kernal Size: 3x3, activation function 採用 relu
# model.add(Conv2D(64, (3, 3), activation='relu'))
# 建立池化層，池化大小=2x2，取最大值
model_cnn2d.add(MaxPooling2D(pool_size=(2, 2)))
# Dropout 層隨機斷開輸入神經元，用於防止過度擬合，斷開比例:0.25
model_cnn2d.add(Dropout(0.25))
# Flatten 層把多維的輸入一維化，常用在從卷積層到全連接層的過渡。
model_cnn2d.add(Flatten())
# 全連接層: 128 個 output
model_cnn2d.add(Dense(128, activation='relu'))
# Dropout 層隨機斷開輸入神經元，用於防止過度擬合，斷開比例:0.5
model_cnn2d.add(Dropout(0.5))
# 使用全連接層 softmax activation function，將結果分類
model_cnn2d.add(Dense(num_classes, activation='softmax'))

# 編譯: 選擇損失函數、優化方法及成效衡量方式
model_cnn2d.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

model_cnn2d.summary()
#開始訓練
# x_train.reshape(x_train.shape[0], input_shape).astype('float32')
x_train_2D = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.shape[2],1).astype('float32')
x_test_2D = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape[2],1).astype('float32')
```

```
# 進行訓練，訓練過程會存在 train_history 變數中
# 定義梯度下降批量
# batch_size = 10
# 定義分類數量 (y 的數量)
num_classes = 7
# 定義訓練週期 (epochs 的值不能太大)
# epochs = 25
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


#train_history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
verbose=1, validation_split=0.2)
# train_history = model.fit(x_train_2D, y_train_oneshot, batch_size=batch_size,
epochs=epochs, verbose=1, validation_data=(x_val,y_val))
train_history = model_cnn2d.fit(x_train_2D, y_train_oneshot, validation_split=0.2,
epochs=300, batch_size=50,callbacks = [learning_rate_function], 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_cnn2d.evaluate(x_test_2D, 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_cnn2d.predict(x_test_2D)
# 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))
```