金融股市預測

第四組

統計四:105304036 陳慧霜

統計四:105304032 杜明軒

統計四:105304025 張順益

目錄

研究動機

身為商學院學生的我們，一直以來都知道金融股市的漲跌很難透過人實際進行預測與操作，然而近幾年大數據的觀念出現，讓我們對於預測有更多想法。是否能透過電腦，亦即機器學習的概念來嘗試預測出股市的漲跌趨勢等等，而這堂大數據分析的課程恰好給予我們機會進行實作，能透過機器學習來實際對股市進行預測與判讀，因此決定主題為股市分析與預測，來嘗試透過大數據的機器學習來解決預測問題。

資料前處理

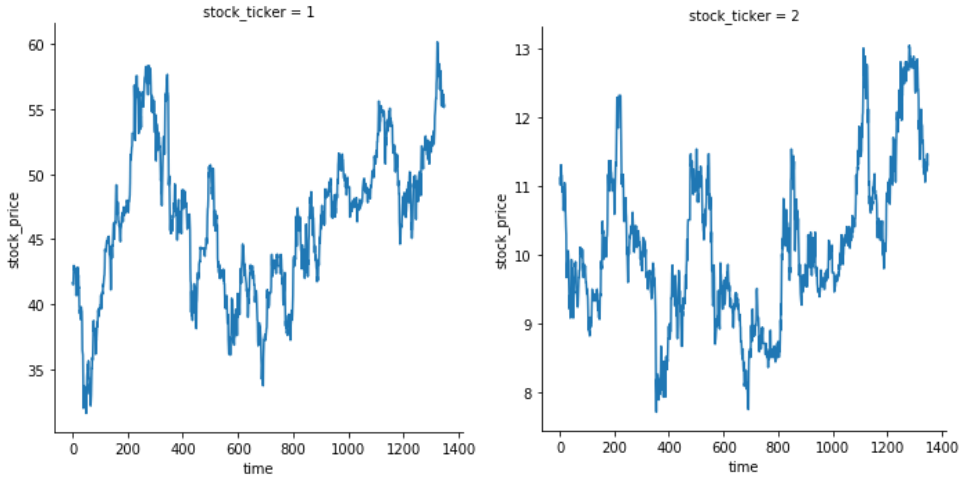
在經過我們內部討論以及參考網路上的各種做法後，我們決定不要選擇過多變數以免造成模型複雜與混亂，因此在變數選擇上我們挑選幾個最重要的變數來進行分析，依序選定了時間、股價、殖利率、買價、賣價、當日最高點、當日最低點，而我們也發現了其中有數個缺值，我們推斷其可能是當日未開市，因此我們決定以上一日的資料來填補缺值，此外我們還創了股票類型這一個變數來分辨此五種類別為何，再將此五種資料個別切分為訓練集有 1362 筆，驗證集有 350 筆，測試集為 50 筆來進行分析。

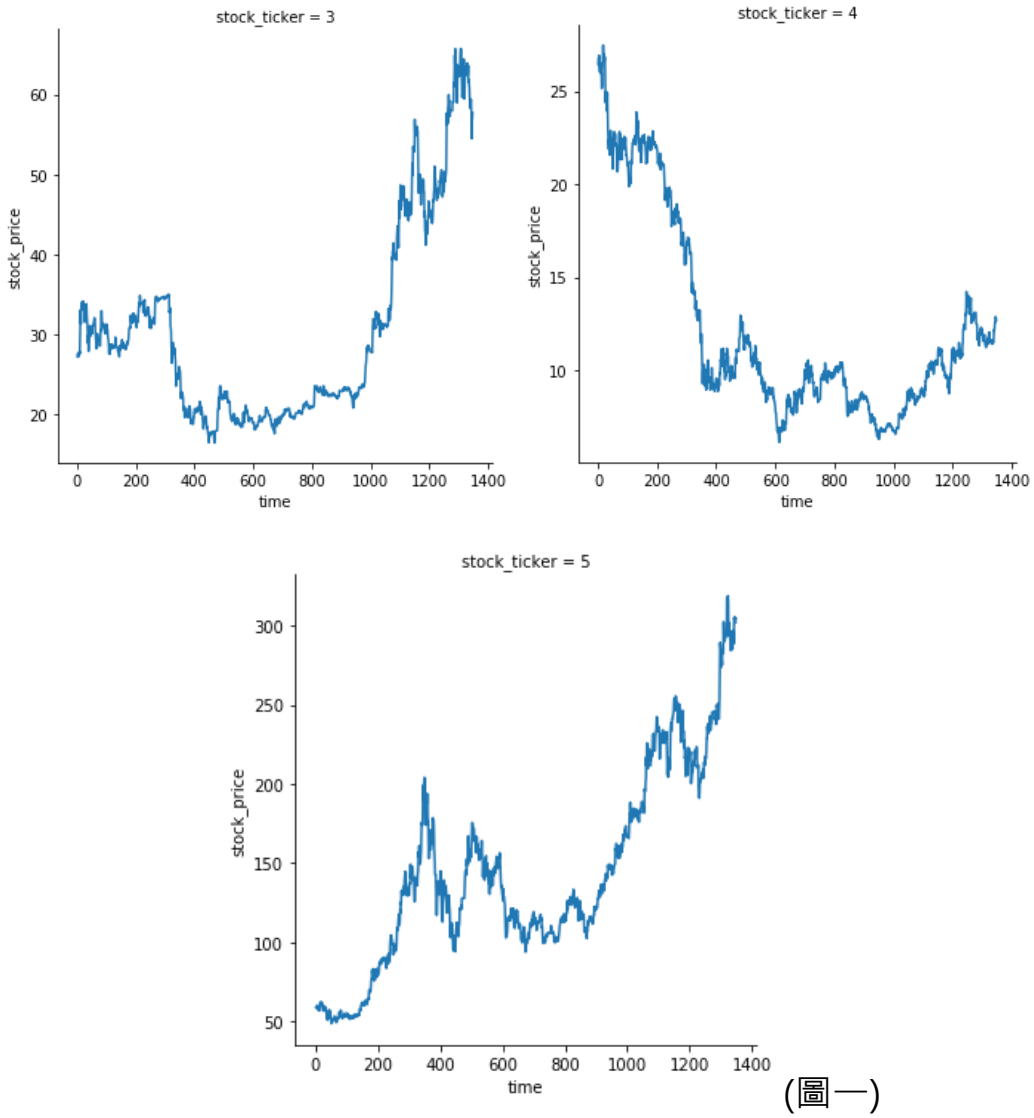
模型選擇

因為此次所做的金融股市預測與時間序列相關，因此我們認為運用 CNN 的模型並不合適，之後我們找到了 RNN 模型中的 LSTM 是屬於長短期記憶模型，他相較於一般的 RNN 模型多了輸入控制，輸出控制以及忘記控制，此時股價預測就不單單只是受到時間前後而導致比重差距還會有這一些資料的重要性以及在往後的資料中也會不斷地藉由忘記控制將資料不停地進行更新，因此我們決定選擇 LSTM 來做為最後的模型預測。

資料建模

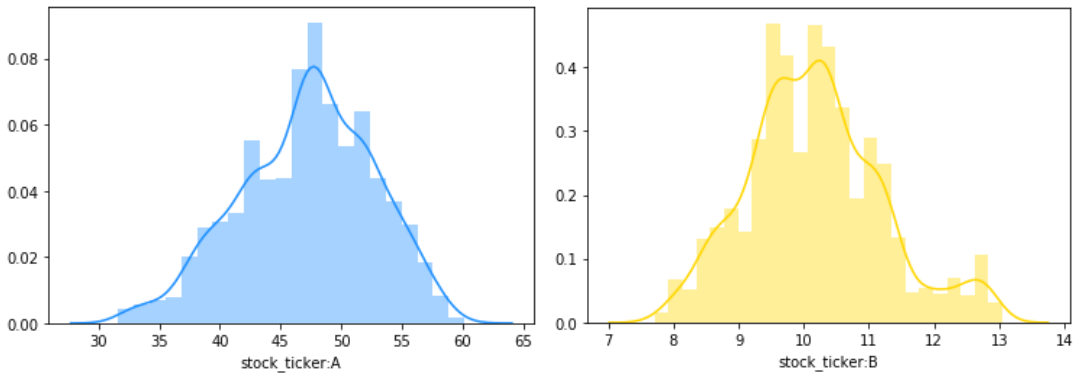
我們先進行了敘述統計的部分，看看此五種資料的分布情況

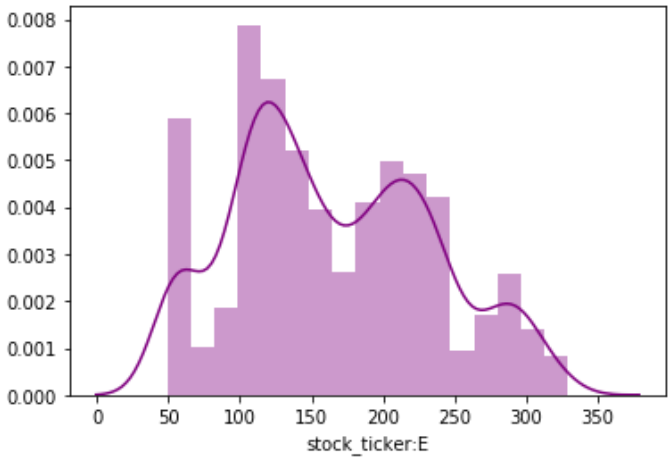




(圖一)

由上述圖一折線圖中，我們發現 Stock_ticker 2 以及 Stock_ticker 4 的數值最為平穩並維持在 50 以下，Stock_ticker 1 以及 Stock_ticker 3 則在 50 上下做不穩定的移動，Stock_ticker 5 則是最為不穩定的，數值分布從 50 至 300，趨勢則大致為爬升





(圖二)

由圖二機率密度可知，

Stock_ticker 1 分佈落於 30~60 之間並成常態分布，股價浮動大

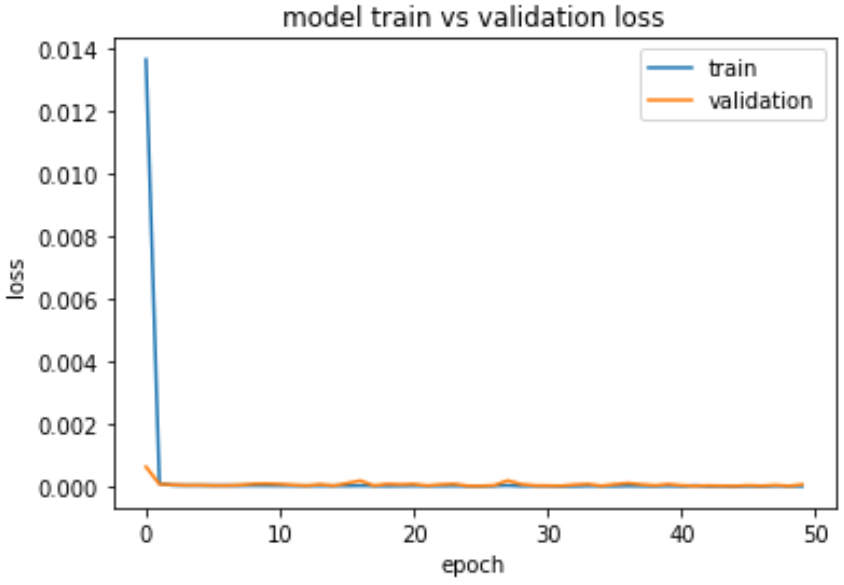
Stock_ticker 2 分佈落於 7~14 之間並成常態分布，股價浮動大

Stock_ticker 3 分佈落於 10~80 之間並呈雙峰(20、60)分佈，股價大致為漲幅

Stock_ticker 4 分佈落於 5~30 之間並呈右偏分佈，股價大致為跌幅

Stock_ticker 5 分佈落於 50~300 之間並呈雙峰(120、210)分佈，股價大致為漲幅

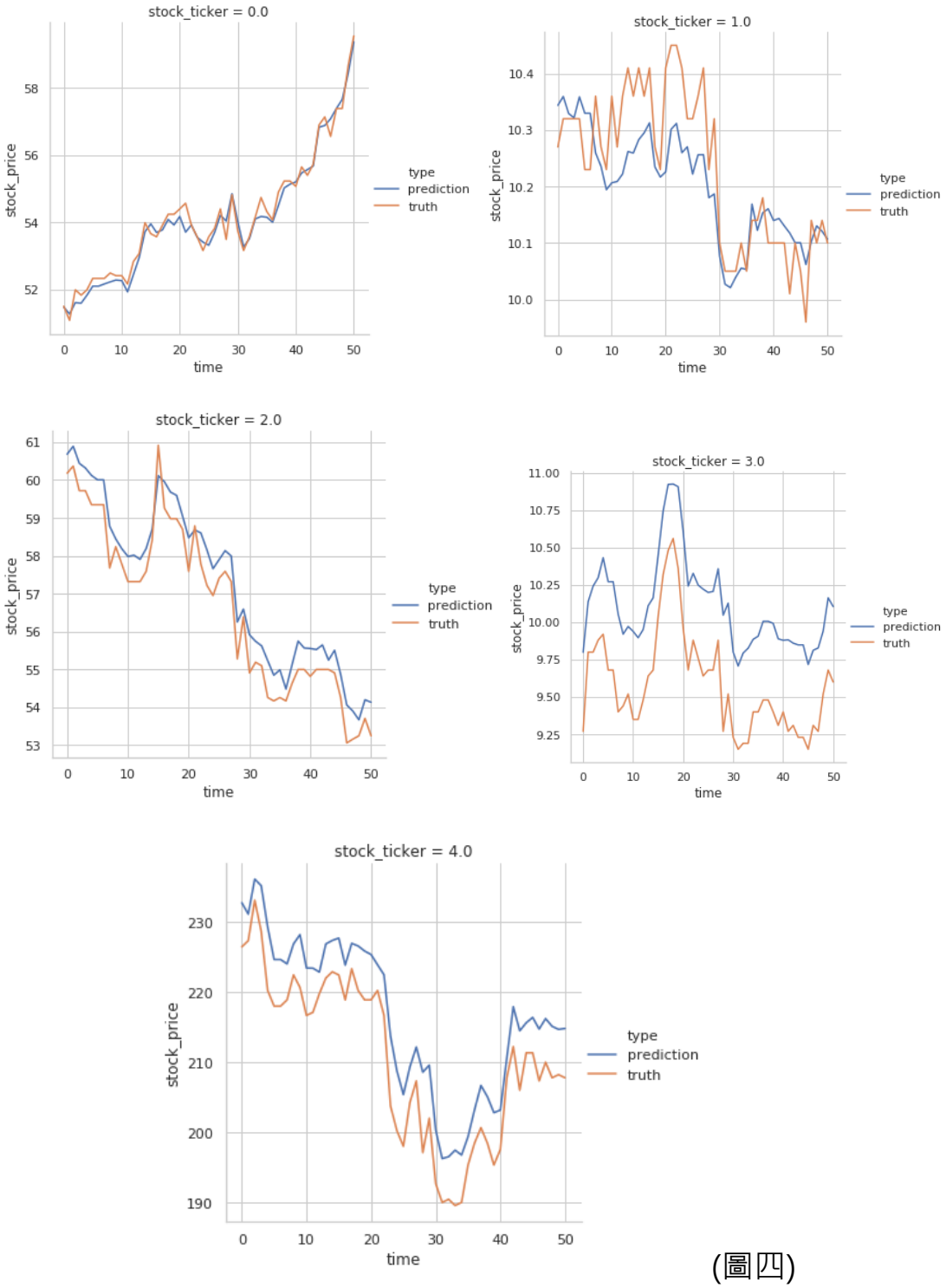
進行完敘述統計了解資料的大致分布後，開始進行建模，我們先將資料轉換為符合 LSTM 模型的格式，之後開始進行建模，以 mse 為損失函數，找出最佳的模型之後由下圖三可以看出我們的模型已經收斂了，甚至已經提早收斂，



(圖三)

因此我們使用了這一個模型來進行預測。

之後我們先由圖四來觀察預測以及實際值的差距，我們可以看出其預測線的趨勢與實際值相當接近。



(圖四)

再來我們計算出測試資料集的總體 RMSE 為 2.897，

而其各自的測試資料集的 RMSE:

Stock A:0.265

StockB:0.084

StockC:0.664

StockD:0.533

StockE:6.376

從中可以看出 E 股的 RMSE 最大，我們認為是因為這一支股票相較於其他股票的全距以及標準差都大很多，所以它的波動比較大，而導致其 RMSE 特別高，而其他四種股票的 RMSE 都已經很小了，已經符合我們的預期了。

之後我們將預測的值根據之後漲跌平盤的標準將其轉為三元類別資料後再看各股的預測是否準確。

A 股 accuracy=0.78 precision=0.789 recall=0.78

F1-score=0.748

預測 \ 真實	跌	平	漲
跌	2	6	0
平	1	33	0
漲	0	4	4

B 股 accuracy=0.7 precision=0.794 recall=0.7

F1-score=0.609

預測 \ 真實	跌	平	漲
跌	1	8	0
平	0	33	0
漲	0	7	1

C 股 accuracy=0.8 precision=0.808 recall=0.8

F1-score=0.772

預測 \ 真實	跌	平	漲
跌	2	5	0
平	0	37	2
漲	0	3	1

D 股 accuracy=0.9 precision=0.896 recall=0.9

F1-score=0.893

預測 \ 真實	跌	平	漲
跌	5	1	0
平	0	37	1
漲	0	3	3

E 股 accuracy=0.84 precision=0.855 recall=0.84

F1-score=0.82

預測 \ 真實	跌	平	漲
跌	3	2	0
平	1	36	0
漲	0	5	3

由上述的資料中可以看出其預測的準確率大概落於 0.8 左右，而其他指標也都大概落於 0.8 左右，因此我們認為預測結果是還不錯的。

結論

透過這次實作分析，我們發現其實透過機器學習對股市進行預測與判讀是可行的，但真正要能派上用場則更加困難。以這次分析為例，我們以模型的簡化為原則，因此不論在變數選擇上，或是參數調整，都無法跟實際預測一樣複雜，因此做出來的結果不論是準確率或是其他數據都僅在 0.8 左右，故我們找出以下幾項可以改進的地方，第一，由於此模型的測試集的 RMSE 明顯比 training 高上許多，因此不排除有過度配適(Overfitting)的問題，故我們可以考慮減少模型複雜度；第二，目前只有針對五家公司四年左右之資料進行操作，其涵蓋範圍並不大，若能增加公司數與拉長時間的話，我們可以配適更完整的模型，如加入目標變數正相關或負相關的公司收盤價或者加入該產業過去的平均股價資訊等；第三，在特徵值(X)

的部分，我們認為若可以收集到如大盤指數、消息面、技術面、政策面等等的資料的話，可以讓模型學習不同方向的資訊，使其更加完整；第四，LSTM 耗費時間長，如果可以使用平行運算的方法將運算量降低，將有效提高其效率。綜上所述，雖然目前有越來越多金融應用機器學習、深度學習的案例，然而股市結果的變化難以完全經由模型得出，就如同牛頓所說:「我算得出星球運動軌跡，卻算不出人性的瘋狂。」表示股市本身就難以透過公式計算出來，模型只能學習過去的事物，當有其餘新事物加入時，模型很難去測量出其影響的走向， 因此以目前的技術來說，機器學習等相關應用只能輔助人類快速判斷出可能的結果，實際的操作方法、決斷仍需要我們自己決定，期望在未來技術越來越純熟時，有一天可以解決這個難題。更何況這只是以 5 支股票所做出的結果，若是將筆數拉大則複雜與困難度可想而知。也因此我們學習到實務上進行預測與分析還有許多要考量的因素，也透過這次實作了解了機器學習是如何應用在最常見的股市上。

將之後發的測試資料，進行預測後發現 E 的漲跌是最少大多為平盤，而 C 的漲跌是最多平盤最少。

附錄

Code:

https://github.com/sandy1990418/Big_data_analysis/blob/master/%E5%A4%A7%E6%95%B8%E6%93%9A%E6%9C%9F%E6%9C%AB%E5%A0%B1%E5%91%8A_20191229.ipynb

```
import numpy as np
import pandas as pd
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Lambda
import matplotlib.pyplot as plt
import seaborn as sns
from keras import models
from keras.models import load_model
#import tensorflow as tf
#import os
#from os import listdir
#from os.path import isfile, isdir, join
#from sklearn.model_selection import  train_test_split
#from keras.layers.core import RepeatVector
#from keras.layers import TimeDistributed
#from keras.layers import Flatten
#from sklearn.preprocessing import StandardScaler
letterList = ["A","B","C","D","E"]
def data_clean(List):
```

```
  data_all =pd.DataFrame()
  training=pd.DataFrame()
  testing=pd.DataFrame()
  validation=pd.DataFrame()
  for f in List:
    #data= open(path+"/"+f)
    names = ''.join([f+".xlsx"])
    data = pd.read_excel(names)
    ##修改資料名稱
    data.rename(columns={'Unnamed: 0':'time'}, inplace=True)
    data.rename(columns={'Unnamed: 1':'stock_price'}, inplace=Tru
e)
    data.columns = [col.replace('-', '') for col in data.columns]
    ##遠端
    #data.reset_index(inplace=True)
    #data.rename(columns={'level_0':'time'}, inplace=True)
    #data.rename(columns={'level_1':'stock_price'}, inplace=True)
    ##增加股票代碼
    data['stock_ticker']=pd.DataFrame(np.repeat(ord(f.lower()) - 9
6,data.shape[0],axis=0))
    columns = data.columns
    data = data.reindex(columns=columns)
    ##lag1 期
    #data["lag_stock_price"]=data["stock_price"].shift(1)
    ##挑出要使用的變數
    data=data[['time', 'stock_price', 'stock_ticker',' DIVIDEND YI
ELD',' ASK PRICE', ' BID PRICE', ' PRICE HIGH', ' PRICE LOW']]
    ##imputation using previous value
    data=data.fillna(method='pad')##filling the missing data with p
revious one
    ##拆分資料
    train=len(data)-400
    test_len=len(data)-51
    ##train
    tx_data=data.iloc[:train,:]
    ##validation
    validation_data=data.iloc[train:test_len,:]
    ##test
```

```
    tex_data=data.iloc[test_len:,:]
    ##train 15ompan
    training = training.append(tx_data)
    ##validation append
    validation= validation.append(validation_data)
    ##test append
    testing = testing.append(tex_data)
    ##all data set append
    data_all = data_all.append(data)
  return data_all ,testing,training,validation
def splite_data(data):
  A=data[data[1]==0.0]
  B=data[data[1]==1.0]
  C=data[data[1]==2.0]
  D=data[data[1]==3.0]
  E=data[data[1]==4.0]
  return A, B, C, D, E
data_all,testing,training,validations=data_clean(letterList)
data_all.isnull().sum()
sns.relplot(x="time", y="stock_price",kind="line",
            hue='stock_ticker',
            data=training)
x_1=data_all.loc[data_all['stock_ticker']==1, ['stock_price']]
x_2=data_all.loc[data_all['stock_ticker']==2, ['stock_price']]
x_3=data_all.loc[data_all['stock_ticker']==3, ['stock_price']]
x_4=data_all.loc[data_all['stock_ticker']==4, ['stock_price']]
x_5=data_all.loc[data_all['stock_ticker']==5, ['stock_price']]
y=data_all.loc[data_all['stock_ticker']==1, ['time']]

# plot density
fig, axes = plt.subplots(2, 3, figsize=(10, 6), sharey=True, dpi=100)
sns.distplot(x_1 , color="dodgerblue", ax=axes[0][0], axlabel='stock_ticker:A')
sns.distplot(x_2 , color="gold", ax=axes[0][1], axlabel='stock_ticker:B')
```

```
sns.distplot(x_3 , color="red", ax=axes[0][2], axlabel='stock_ticker:C')
sns.distplot(x_4 , color="green", ax=axes[1][0], axlabel='stock_ticker:D')
sns.distplot(x_5 , color="purple", ax=axes[1][1], axlabel='stock_ticker:E')
training.index =training[ 'time']
#training.fillna(0, inplace=True) #fill na using 0
training=training.drop("time",1)
values = training.values
# integer encode direction
encoder = LabelEncoder()
values[:,1] = encoder.fit_transform(values[:,1])


testing.index=testing['time']
#testing.fillna(0, inplace=True) #fill na using 0
testing=testing.drop("time",1)
values_testing = testing.values
# integer encode direction
encoder = LabelEncoder()
values_testing[:,1] = encoder.fit_transform(values_testing[:,1])



validations.index =validations['time']
#validations.fillna(0, inplace=True) #fill na using 0
validations=validations.drop("time",1)
values_validation = validations.values
# integer encode direction
encoder = LabelEncoder()
values_validation[:,1] = encoder.fit_transform(values_validation[:,1])
values = values.astype('float32')
# testing
values_testing = values_testing.astype('float32')
# validation
values_validation = values_validation.astype('float32')
```

```
# normalize features
# training
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)


# validation
scaler_validation = MinMaxScaler(feature_range=(0, 1))
scaled_validation = scaler_validation.fit_transform(values_validat
ion)

# testing
scaler_testing = MinMaxScaler(feature_range=(0, 1))
scaled_testing = scaler_testing.fit_transform(values_testing)
train_X ,train_y=scaled[:,1:7], scaled[:,0]
validation_X,validation_y= scaled_validation[:,1:7], scaled_valida
tion[:,0]
test_X, test_y = scaled_testing[:,1:7], scaled_testing[:,0]
print(test_X.shape, len(test_X), test_y.shape)

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0],1,6))
test_X = test_X.reshape((test_X.shape[0],1,6))
validation_X = validation_X.reshape((validation_X.shape[0], 1,6))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape,val
idation_X.shape,validation_y.shape)
model = Sequential()
model.add(LSTM(50, input_shape=(1,6), return_sequences=True))
model.add(Lambda(lambda x: x[:, -50:, :]))
model.add(LSTM(50, input_shape=(50, 6), return_sequences=True))
model.add(Dense(50))
model.add(LSTM(50, input_shape=(50, 6), return_sequences=True))
model.add(Dense(50))
model.add(LSTM(50, input_shape=(50,6), return_sequences=False))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
history = model.fit(train_X,train_y,epochs=50,batch_size=64,validation_data=(validation_X,validation_y))
model.save("LSTM_MODEL.h5")

##load model(another way using sklearn2pmml)
#https://stackoverflow.com/questions/33221331/export-python-scikit-learn-models-into-pmml
model = load_model('LSTM_MODEL.h5')
print(history.history.keys())
#summarize history for loss plt.plot(history.history['loss']) plt.plot(history.history['val_loss']) plt.title('model loss')
##經 loss 圖看，兩組數據性能應為相似，若兩條線分開則有可能應該提前停止訓練
pyplot.plot(history.history['loss'])
pyplot.plot(history.history['val_loss'])
pyplot.title('model train vs validation loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'validation'], loc='upper right')
pyplot.show()
# make a prediction
yhat = model.predict(test_X)

# invert scaling for forecast
test_X = test_X.reshape((test_X.shape[0], 6))
inv_yhat = concatenate((yhat, test_X[:,:]), axis=1)
inv_yhat= scaler_testing.inverse_transform(inv_yhat)
test_X = test_X.reshape((test_X.shape[0], 6))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:,:]), axis=1)
inv_yhat = scaler_testing.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:,:]), axis=1)
inv_y = scaler_testing.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE(for total)
```

```
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
inv_yhat = concatenate((yhat, test_X[:,:]), axis=1)
inv_yhat = scaler_testing.inverse_transform(inv_yhat)
inv_y = concatenate((test_y, test_X[:,:]), axis=1)
inv_y = scaler_testing.inverse_transform(inv_y)
df = pd.DataFrame(inv_yhat)
df_true = pd.DataFrame(inv_y)
A,B,C,D,E=splite_data(df)
Atrue,Btrue,Ctrue,Dtrue,Etrue=splite_data(df_true)



# calculate RMSE(for individual)
#A company
rmse = sqrt(mean_squared_error(A[0], Atrue[0]))
print('Test A company RMSE: %.3f' % rmse)
#B company
rmse = sqrt(mean_squared_error(B[0], Btrue[0]))
print('Test B company RMSE: %.3f' % rmse)
#C company
rmse = sqrt(mean_squared_error(C[0], Ctrue[0]))
print('Test C company RMSE: %.3f' % rmse)
#D company
rmse = sqrt(mean_squared_error(D[0], Dtrue[0]))
print('Test D company RMSE: %.3f' % rmse)
#E company
rmse = sqrt(mean_squared_error(E[0], Etrue[0]))
print('Test E company RMSE: %.3f' % rmse)
def  gen_percent_change_stock_price(data,data2):
  data["lag_stock_price"]=data[0].shift(1)
  ##算出平均、sd、ercent_change_stock_price
  data['percent_change_stock_price']= (data[0]-data["lag_stock_price"])/data["lag_stock_price"]
  data["mean_stock_price"]=data['percent_change_stock_price'].mean()
  data["sd_stock_price"]=data['percent_change_stock_price'].std()
```

```
  data2["lag_stock_price"]=data2[0].shift(1)
  data2['percent_change_stock_price']= (data2[0]-data2["la
g_stock_price"])/data2["lag_stock_price"]
  data2['mean_stock_price']=data["mean_stock_price"]
  data2['sd_stock_price']=data["sd_stock_price"]
  return data2


def conditions_to_determine_classification(data2):
    if  (data2['percent_change_stock_price'] >(data2["mean
_stock_price"]+data2["sd_stock_price"])):
        return  1
    elif (data2['percent_change_stock_price'] >=(data2["me
an_stock_price"]-data2["sd_stock_price"])):
        return  0
    else:
        return -1

##generate the  percent change  stock price
class_A=gen_percent_change_stock_price(Atrue,A)
class_B=gen_percent_change_stock_price(Btrue,B)
class_C=gen_percent_change_stock_price(Ctrue,C)
class_D=gen_percent_change_stock_price(Dtrue,D)
class_E=gen_percent_change_stock_price(Etrue,E)

##gen the classification
class_A['Class'] = class_A.apply(conditions_to_determine_c
lassification, axis=1)
class_B['Class'] = class_B.apply(conditions_to_determine_c
lassification, axis=1)
class_C['Class'] = class_C.apply(conditions_to_determine_c
lassification, axis=1)
class_D['Class'] = class_D.apply(conditions_to_determine_c
lassification, axis=1)
class_E['Class'] = class_E.apply(conditions_to_determine_c
lassification, axis=1)
```

```
##export data using excel
class_A.to_excel('class_A.xlsx')
class_B.to_excel('class_B.xlsx')
class_C.to_excel('class_C.xlsx')
class_D.to_excel('class_D.xlsx')
class_E.to_excel('class_E.xlsx')
print(class_A,class_B,class_C,class_D,class_E)

##A 21ompany
# Print the confusion matrix
print('A_COMPANY \n\n Confusion Matrix\n',metrics.confusion_matri
x(A_TRUE.iloc[1:,11], class_A.iloc[1:,11]))
# Print the precision and recall, among other metrics
print(metrics.classification_report(A_TRUE.iloc[1:,11], class_A.i
loc[1:,11], digits=3))

##B 21ompany
# Print the confusion matrix
print('\n\n B_COMPANY \n\n Confusion Matrix\n',metrics.confusion_
matrix(B_TRUE.iloc[1:,11], class_B.iloc[1:,11]))
# Print the precision and recall, among other metrics
print(metrics.classification_report(B_TRUE.iloc[1:,11], class_B.i
loc[1:,11], digits=3))

##C 21ompany
# Print the confusion matrix
print('\n\n C_COMPANY \n\n Confusion Matrix\n',metrics.confusion_
matrix(C_TRUE.iloc[1:,11], class_C.iloc[1:,11]))
# Print the precision and recall, among other metrics
print(metrics.classification_report(C_TRUE.iloc[1:,11], class_C.i
loc[1:,11], digits=3))

##D 21ompany
# Print the confusion matrix
```

```
print('\n\n D_COMPANY \n\n Confusion Matrix\n',metrics.confusion_
matrix(D_TRUE.iloc[1:,11], class_D.iloc[1:,11]))
# Print the precision and recall, among other metrics
print(metrics.classification_report(D_TRUE.iloc[1:,11], class_D.i
loc[1:,11], digits=3))

##E 22ompany
# Print the confusion matrix
print('\n\n E_COMPANY \n\n Confusion Matrix\n',metrics.confusion_
matrix(E_TRUE.iloc[1:,11], class_E.iloc[1:,11]))
# Print the precision and recall, among other metrics
print(metrics.classification_report(E_TRUE.iloc[1:,11], class_E.i
loc[1:,11], digits=3))
```

參考資料

https://brohrer.mcknote.com/zh-Hant/how_machine_learning_works/how_rnns_lstm_work.html?source=post_page-----cd72af64413a----------

https://github.com/birdomi/MI-LSTM

https://www.finlab.tw/%E5%88%A9%E7%94%A8MI-LSTM%E9%A0%90%E6%B8%AC%E8%82%A1%E5%83%B9/

http://proceedings.mlr.press/v95/li18c/li18c.pdf

https://github.com/thundercomb/pytorch-stock-predictor-rnn/blob/9044879b000763c926a90e19525dd3bf2b5b2170/keras-stock-predictor-lstm.py

https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

https://www.kaggle.com/humamfauzi/multiple-stock-prediction-using-single-nn

https://zhuanlan.zhihu.com/p/59862381

https://zhuanlan.zhihu.com/p/51812293

https://medium.com/%E5%AD%B8%E4%BB%A5%E5%BB%A3%E6%89%8D/%E5%84%AA%E5%8C%96%E6%BC%94%E7%AE%97-5a4213d08943

https://stackoverflow.com/questions/43034960/many-to-one-and-many-to-many-lstm-examples-in-keras

https://www.kaggle.com/raoulma/ny-stock-price-prediction-rnn-lstm-gru

https://zhuanlan.zhihu.com/p/31783805

https://medium.com/neuronio/predicting-stock-prices-with-lstm-349f5a0974d4

https://towardsdatascience.com/aifortrading-2edd6fac689d?#f8df

https://github.com/umbertogriffo/Predictive-Maintenance-using-LSTM/blob/master/src/lstm/regression.py

https://towardsdatascience.com/forecasting-with-technical-

indicators-and-gru-lstm-rnn-multivariate-time-series-a3244dcbc38b

https://www.datacamp.com/community/tutorials/lstm-python-stock-market

https://zhuanlan.zhihu.com/p/67318909

https://leemeng.tw/seaborn-cheat-sheet.html

https://www.itread01.com/content/1545185286.html