國立成功大學
109 學年度第 2 學期
期末報告

課程名稱:資料科學導論

授課老師:李政德

報告主題:見微知著-讓 python 成為你的股票理專

組別:　教父

組員:　統計 112 H24081333 林家同

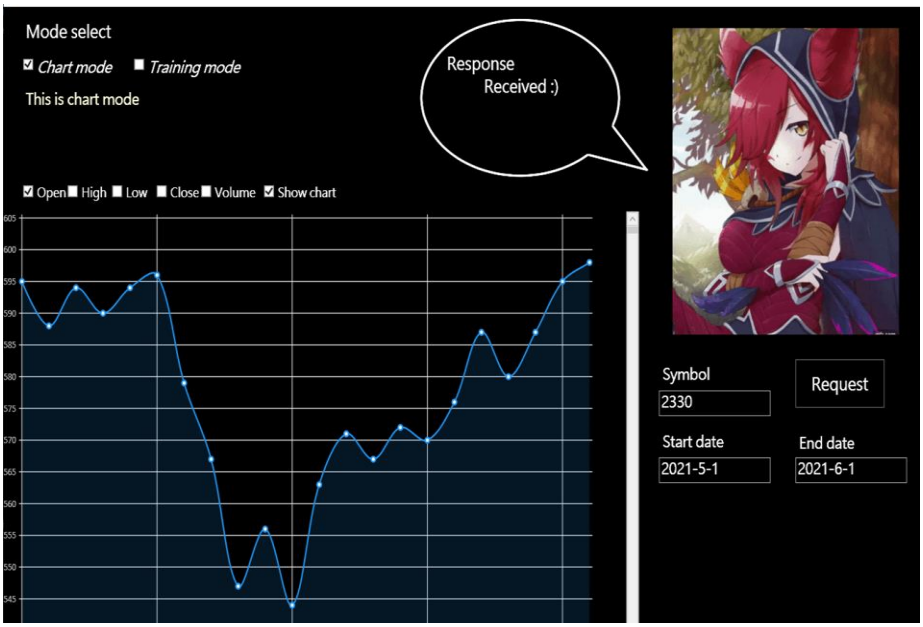
資訊 112 F74086250 莊上緣

資訊 112 F74082272 李培綸

Table of Contents

0. Brief introduction of the problem

股票交易市場的運作方式，取決於買賣雙方協商出都能接受的交換價格，在股市中這些交易商品指的是公司股票，然而大環境政策、經濟週期等變動會讓股票的供需產生變化，隨著時間推移使股價產生不小的波動。因此我們想藉由這次的 project，更精確地找出哪些數據才是影響股價的核心指標，掌握投資的重要資訊，我們除了會以市場每日的交易資料要做為主要的資訊，還會參考公司的財報，取出像 EPS 等重要的指標，拿去跟股價變動算相關係數，再把那些高度相關的資料作為特徵值，給予不同的權重，再吃進去我們的模型。我們希望利用幾檔較具代表性的股票來做出股價預測模型，並搭配 UI 介面設計成一個可以與使用者互動的程式。下圖是我們在上課示範過的 UI 主畫面，使用者可以輸入股票代碼，想拿來預測的日期，就可以印出股價折線圖，並且左上角還可以依照使用者偏好來做圖表外觀上的調整。



1. Data description and preprocessing

我們的 dataset 來自 Finmind,Finmind 是一個由台灣創作者開發的金融大數據資源包，這個資源包提供以台股為主的金融數據，使用者只要先在終端機安裝 finmind，並申辦帳號，就可以取得台股技術面、基本面、籌碼面的 data，像是個股的每日成交價、融資融卷數、本益比、股價淨值比等資訊，另外他們也有自己的 Github，裡面有原始爬蟲的.py 檔。接著，在 jupyter notebook 輸入帳密後，會獲得一組 token，我們可以把它想像成一個金鑰，拿著他便可連上 finmind。設定完想要取得的個股和時間區段，就可以送出請求了，我們拿到的第一手資料都是一個完整的 dataframe。分次取得技術面、基本面、籌碼面以及大盤的資料我們再將得到的多個 dataframe 做整合。整合後的 dataframe 包含了同一個股的多個金融數據，而這些特徵值就是 column names。


```
In [273]: url = "https://api.finmindtrade.com/api/v4/login"
          account = {
              "user_id": "user_id",
              "password": "password",
          }
          token = requests.post(url, data=account).json()['token']

          url = "https://api.finmindtrade.com/api/v4/data"
          parameter = {
              "dataset": "TaiwanStockPrice",
              "data_id": "2330",
              "start_date": "2016-06-04",
              "end_date": "2021-06-03",
              "token":token
          }
          data = pd.DataFrame(requests.get(url, params=parameter).json()["data"])
          data['average'] = data["Trading_money"]/data["Trading_Volume"]
          data
```

Out[273]:

	date	stock_id	Trading_Volume	Trading_money	open	max	min	close	spread	Trading_turnover	average
0	2016-06-04	2330	2631269	420392883	160.0	160.0	159.5	160.0	-0.5	1428	159.768113
1	2016-06-06	2330	28898951	4651663660	161.0	162.0	160.0	161.0	1.0	8667	160.963063
2	2016-06-07	2330	47573770	7726094806	161.5	163.5	161.0	162.0	1.0	12269	162.402408
3	2016-06-08	2330	45587946	7522713617	164.0	166.0	163.5	165.5	3.5	15173	165.015410

	日期	股票編號	當日成交量	當日成交金額	開盤價	最高價	最低價	收盤價	資價差	周轉率	...	昨日融卷	融資變化	融券變化	殖利率	本益比	價淨值比	大盤融資餘額(百萬)	大盤融卷量	大盤指數	大盤
0	2016-06-04	2330	2631269	420392883	160.0	160.0	159.5	160.0	-0.5	1428	...	3812	-26	-73	3.75	14.20	3.24	12994.9943	360288	8591.57	4
1	2016-06-06	2330	28898951	4651663660	161.0	162.0	160.0	161.0	1.0	8667	...	3739	-260	35	3.73	14.29	3.26	12967.5393	368731	8597.11	5
2	2016-06-07	2330	47573770	7726094806	161.5	163.5	161.0	162.0	1.0	12269	...	3774	-173	487	3.70	14.37	3.28	12935.2080	379136	8679.90	82
3	2016-06-08	2330	45587946	7522713617	164.0	166.0	163.5	165.5	3.5	15173	...	4261	-436	62	3.63	14.69	3.35	12945.7301	378743	8715.48	35
4	2016-06-13	2330	54689228	8843806836	161.0	162.5	161.0	162.0	-3.5	14466	...	4323	44	-1046	3.70	14.37	3.28	12925.4617	355334	8536.22	-179
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1217	2021-05-28	2330	30720737	18082265480	587.0	592.0	582.0	590.0	8.0	31581	...	425	118	2	1.69	28.30	7.89	23336.5365	556327	16870.86	269
1218	2021-05-31	2330	31557426	18751716595	595.0	597.0	590.0	597.0	7.0	29798	...	427	-967	-43	1.68	28.63	7.98	23404.7805	551036	17068.43	197
1219	2021-06-01	2330	18405285	10985893229	598.0	599.0	595.0	598.0	1.0	20318	...	384	-377	-7	1.67	28.68	8.00	24012.6099	563095	17162.38	93
1220	2021-06-02	2330	22416789	13362065937	600.0	600.0	593.0	595.0	-3.0	25170	...	377	34	-12	1.68	28.54	7.96	24049.8196	550856	17165.04	2
1221	2021-06-03	2330	31703679	18939839664	600.0	600.0	596.0	596.0	1.0	20749	...	365	401	-40	1.68	28.59	7.97	24595.8470	559529	17246.16	81

1222 rows × 24 columns

2. Insights discovered from the data

將資料整合後，下一步就可以開始處理多變量分析的問題，市場裡充斥著影響股價的各種因子，所以我們的目標是將先前取得的特徵作為自變數，探討多項變數對收盤價 y 的影響。我們篩選特徵值的依據是用各個特徵的移動平均和收盤價算出相關係數的平均，利用 python 內建的函式 rolling，再 drop 掉其中低度相關與負相關，去蕪存菁後留下可用資料表格，經篩選後我們留下來 13 個特徵值，並建立新的 dataframe。

3. Methodology details

獲取一個要真正拿來訓練模型的 dataframe 後，我們要先對資料進行預處理，這邊運用的是將所有的資料進行(0,1)的標準化，消除特徵間單位和尺度，可以有助於演算法的優化。

In [40]:
```
from sklearn.preprocessing import MinMaxScaler
values = df_new.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
print(scaled)
scaled.shape[0]
```

```
[[0.00869565 0.0107949  0.00764818 ... 0.17531891 0.01452332 0.47088767]
 [0.01062802 0.01275761 0.01147228 ... 0.18363432 0.01512964 0.47178086]
 [0.01256039 0.01373896 0.01434034 ... 0.19388207 0.02419057 0.52365922]
 ...
 [0.84927536 0.87438665 0.84894837 ... 0.36300702 0.95284463 0.46984675]
 [0.85120773 0.87438665 0.84894837 ... 0.37154895 0.96172279 0.52253771]
 [0.84927536 0.85672228 0.83938815 ...        nan 0.95091512 0.40174338]]
```

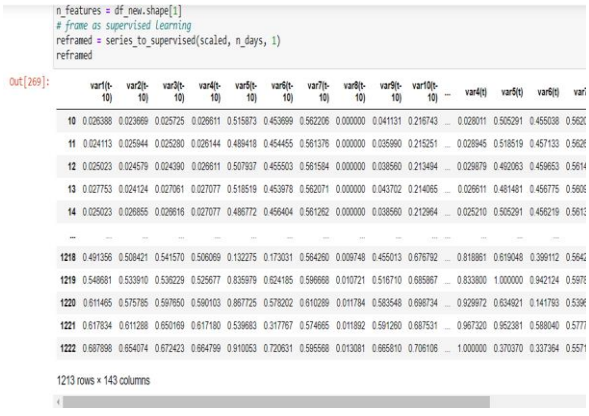
Out[40]: 1223

In [41]:
```
print(scaled)
```

```
[[0.00869565 0.0107949  0.00764818 ... 0.17531891 0.01452332 0.47088767]
 [0.01062802 0.01275761 0.01147228 ... 0.18363432 0.01512964 0.47178086]
 [0.01256039 0.01373896 0.01434034 ... 0.19388207 0.02419057 0.52365922]
 ...
 [0.84927536 0.87438665 0.84894837 ... 0.36300702 0.95284463 0.46984675]
 [0.85120773 0.87438665 0.84894837 ... 0.37154895 0.96172279 0.52253771]
 [0.84927536 0.85672228 0.83938815 ...        nan 0.95091512 0.40174338]]
```

因為我們拿到的資料是一個連續性的數量變化、隨時間先後排列的時間序列資料，所以在機器學習方法可以被使用之前，我們必須將時間序列重新建構成監督式學習問題，讓一個特徵值的序列變為一對序列的輸入和輸出。我們運用 pandas 中的 shift()函示便可將序列進行一步兩步甚至是多步的時間平移，我們考慮市場十天前到當天的交易資訊會對當天的交易價做影響，所以透過數據平移的方式將我們的目標便量往前平移十個時間單位，建立 13*10+13 的的新 dataframe，有了標籤後它就變成了監督式學習的數據，這樣我們就可以訓練 LSTM 模型了。

```
將時間序列轉換為監督式學習
om pandas import DataFrame
om pandas import concat

f series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

specify the number of lag hours
days = 10 #可再調整
features = df_new.shape[1]
frame as supervised learning
framed = series_to_supervised(scaled, n_days, 1)
```



```
n_features = df_new.shape[1]
# frame as supervised learning
reframed = series_to_supervised(scaled, n_days, 1)
reframed
```

Out[269]:

	var1(t-10)	var2(t-10)	var3(t-10)	var4(t-10)	var5(t-10)	var6(t-10)	var7(t-10)	var8(t-10)	var9(t-10)	var10(t-10)	...	var4(t)	var5(t)	var6(t)	var
10	0.026388	0.023669	0.025725	0.026611	0.515873	0.453699	0.562206	0.000000	0.041131	0.216743	...	0.028011	0.505291	0.455038	0.562
11	0.024113	0.025944	0.025280	0.026144	0.489418	0.454455	0.561376	0.000000	0.035990	0.215251	...	0.028945	0.518519	0.457133	0.562
12	0.025023	0.024579	0.024390	0.026611	0.507937	0.455503	0.561584	0.000000	0.038560	0.213494	...	0.029879	0.492063	0.459653	0.561
13	0.027753	0.024124	0.027061	0.027077	0.518519	0.453978	0.562071	0.000000	0.043702	0.214065	...	0.029611	0.481481	0.456775	0.560
14	0.025023	0.026855	0.029616	0.027077	0.486772	0.456404	0.561262	0.000000	0.038560	0.212964	...	0.025210	0.505291	0.456219	0.561
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1218	0.491356	0.508421	0.541570	0.506089	0.132275	0.173031	0.564260	0.009748	0.455013	0.676792	...	0.818861	0.619048	0.399112	0.564
1219	0.548681	0.533910	0.536229	0.525677	0.835979	0.624185	0.596668	0.010721	0.516710	0.685867	...	0.833800	1.000000	0.942124	0.597
1220	0.611465	0.575785	0.597650	0.590103	0.867725	0.578202	0.610289	0.011784	0.583548	0.698734	...	0.929972	0.634921	0.141793	0.539
1221	0.617834	0.611288	0.650169	0.617180	0.539683	0.317767	0.574665	0.011892	0.591260	0.687531	...	0.967320	0.952381	0.588040	0.577
1222	0.687898	0.654074	0.672423	0.664799	0.910053	0.720631	0.595568	0.013081	0.665810	0.706106	...	1.000000	0.370370	0.337364	0.557

1213 rows × 143 columns

由於股票的許多數值波動都是跟時間相關的，所以我們選擇了 LSTM 這個可以支援時間序列模型相關預測的機器學習演算法來幫助我們。我們會透過 import keras 這個 package 來幫助我們搭建一個 LSTM 神經網路，keras 是一個專門用來訓練深度學習模型的 API，在這次專案裡面我們使用的是 sequential 模型，是一種透過堆疊很多隱藏層來建構出的深度神經網路，在神經網路中，每個神經元皆會配有一個權重，這個權重代表著該神經元之重要性，而這個權重會在神經網路模型每次學習時，根據學習結果與大量輸入的資料而進行不斷地更新，我們必須建立輸入輸出控制與隱藏層，選擇適當的優化器與損失函數。而程式碼中的 input_shape 其實是一個三維資料，代表 batch size,time step,input data, batch 代表一次要輸入的資料筆數，time step 代表資料的時間維度，股票如果要藉由前 10 天的資料預測，則 time step 為 10。Input data 代表單獨一個時間的資料，而 units 為每一個 step 輸出的數量，接著，我們挑選 ReLU(Rectified Linear Unit)作為激勵函數，它可以幫助我們解決反向傳播訓練時產生的梯度消失問題，並使部分神經元輸出為 0，讓神經網路變得稀疏，我們也一邊使用 dropout 隨機消除神經元，緩解過度擬和的問題。另外，損失函數(loss function)可以幫助我們做現在 ML 模型好壞的評估，並以數值呈現。資料完成了一次完整的訓練，這個過程稱為一個 epoch，epoch 的數量增加，會使神經網路中權重的不斷更新，一般來說，當訓練次數增加，準確度會較高。另外，這次專案中有使用到 stack LSTM，相對於單層的 LSTM 來說，這個堆疊 LSTM 的每個神經元具有更多的層數。為了將一個深度學習模型優化，原理是減少總 LSTM 神經元數，相較於較多的神經元與深度較淺的神經網路模型，深度較深，較多層的神經元組成的模型，更能近似地代表更多數的函數，也就是說，一個深層且分層的神經網路模型會比一個淺層且大的神經網路模型更能代表某些功能。

```
#Design LSTM model
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout,BatchNormalization

model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.4))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
print(model.summary())

# fit network
history = model.fit(train_X, train_y, epochs=150, batch_size=256,validation_data=(test_X, test_y), verbose=0, shuffle=False)
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_days*n_features))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -12:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
```

```
###stacked LSTM

model=Sequential()
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2]),return_sequences=True))
model.add(Dropout(0.4))
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.4))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

# fit network
history = model.fit(train_X, train_y, epochs=150, validation_data=(test_X, test_y), verbose=0, shuffle=False)
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_days*n_features))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -12:]), axis=1)
inv_yhat = scaler.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[:, -12:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
```

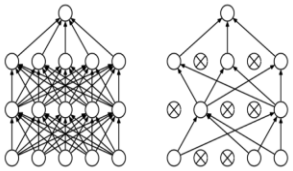
4. Evaluation and Results

(1)資料切分:Train/test (訓練與驗證)、X/y (變量與 target):

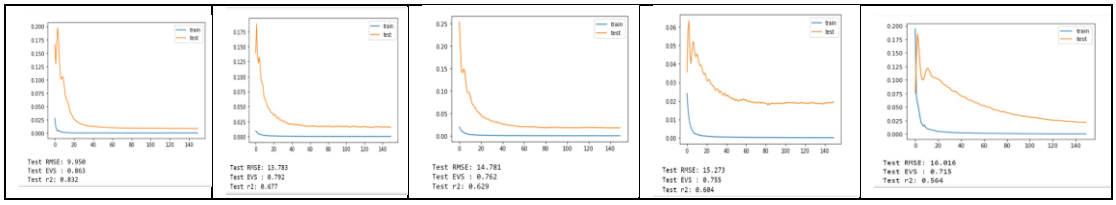
我們以長榮公司這檔股票做為我們的 dataset 來源，我們一開始的 raw data 的 time interval 是取 5 年(ex:2016/6/4~2021/6/4)，因此我們決定嘗試看看使用 4:1 的方式去分配資料中用來 train 的比例與用來驗證的比例。

(2) dropout size:

我們將其中幾個神經元 drop 掉，可以看出，在每層 drop 掉百分之四十的神經元數量，他們之間複雜的關係問題得到了大大的改善，神經元之間的連線變得較為單純。我們這邊也有做試驗，並搭配三個迴歸評價指標，分別是 RMSE(均方根誤差)、EVS(可解釋變異)、R^2(判定係數)。發現當 dropout size 為 0.4 時，損失函數的收斂情形最好。



(左圖:dropout=0，右圖:dropout=0.4)



(由左至右分別是 dropout size=0.0,0.2,0.4,0.6,0.8)

Dropout / Test	0.0	0.2	0.4	0.6	0.8
RMSE	9.950	13.783	14.781	15.273	16.016
EVS	0.863	0.792	0.762	0.755	0.715
R^2	0.832	0.677	0.629	0.604	0.564

(3) plot& evaluation:

除了迴歸指標，我們也透過上面那些步驟預測出的模型做出各項精準度評比。為了方便製作 confusion_matrix，我們決定將股票的漲跌趨勢做數字代號分類，製作了一個副程式，用來計算時間段中的真實股價漲跌趨勢與預測股價漲跌趨勢，比較每一天與前一天的股價差別，漲價為 1，下跌為 0，而若維持不變則為 2，以下是我們的真實股價與預測股價的摺線圖，黑線為真實股價，綠線為預測股價，我們將剛剛副程式算出的真實與預測股價之趨勢 list 匯入 sklearn 提供之 confusion_matrix 並得出結果。可以看出這個模型的 accuracy 為接近 6 成。

```
def confusion_matrix_list(real,predict):
    real_tendency=[]
    predicted_tendency=[]

    for i in range(1,len(real)):
        if(real[i]-real[i-1]>0):
            real_tendency.append(1)
        if(real[i]-real[i-1]==0):
            real_tendency.append(2)
        if(real[i]-real[i-1]<0):
            real_tendency.append(0)
    for i in range(1,len(predict)):
        if(predict[i]-predict[i-1]>0):
            predicted_tendency.append(1)
        if(predict[i]-predict[i-1]==0):
            pedicted_tendency.append(2)
        if(predict[i]-predict[i-1]<0):
            predicted_tendency.append(0)

    return real_tendency,predicted_tendency
```

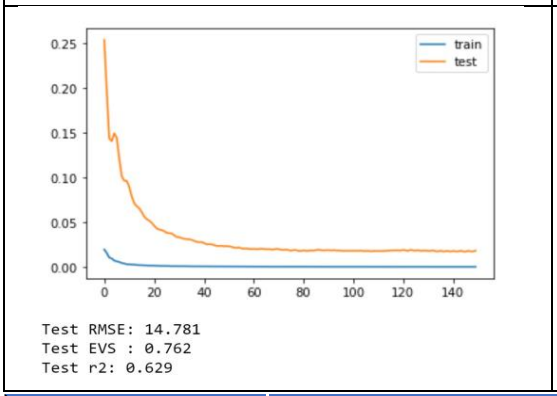

```
real,predict=confusion_matrix_list(inv_y,inv_yhat)

from sklearn.metrics import confusion_matrix
matrix=confusion_matrix(real,predict,labels=[1,0,2])
print(matrix)

from sklearn.metrics import classification_report
report=classification_report(real,predict,labels=[1,0,2])
print(report)
```

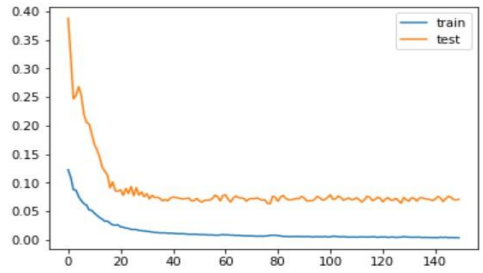
```
[[93 39  0]
 [42 28  0]
 [ 5  5  0]]
              precision    recall  f1-score   support

           1       0.66      0.70      0.68       132
           0       0.39      0.40      0.39        70
           2       0.00      0.00      0.00        10

    accuracy                           0.57       212
   macro avg       0.35      0.37      0.36       212
weighted avg       0.54      0.57      0.56       212
```

（將趨勢 list 匯入 sklearn confusion_matrix 並得出其各項預測精準度評比）

(4) Loss Function:

對於一個連續性的數值，評估其最佳化模型有一個準則，利用回歸問題的損失函數，以下的兩張圖，x 軸代表訓練的時間，y 軸代表計算的損失。這裡將提供兩個常用的方法，MSE 和 MAE。MSE 就是計算實際值與預測值的平方差，再平均，而因為有平方項，確保 Loss Function≥0，因此模型一定會向最小誤差的方向去收斂，然而 outlier 在 MSE 會被指數性的放大。而相對於 MSE，MAE 對 outlier 較為寬容，但它有一個令人詬病的缺點就是在接近最佳解時，MAE 的部分梯度仍保持一個定值時。從下面那張圖可以看出 MAE 無法在低損失時收斂，這樣的表現不利於模型的學習。另外，我們也可以適度引用判定係數，不管在任何模型，R^2都會被計算成 0~1 之間的分數，越接近 1 代表表現越好。skleran 還有另一個函式 explained variance score, 其計算方式與判定係數稍有不同，如果 error 的平均值為 0，結果才會一樣。

<table>
<tr>
<td>MSE (Mean-Square Error)
$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}_{\text{The square of the difference between actual and predicted}}^2$</td>
<td>MAE (Mean-Absolute Error)
$MAE = \underbrace{\frac{1}{n}}_{\text{Divide by the total number of data points}}\underbrace{\Sigma}_{\text{Sum of}}\underbrace{\left|\overbrace{y}^{\text{Actual output value}} - \overbrace{\hat{y}}^{\text{Predicted output value}}\right|}_{\text{The absolute value of the residual}}$</td>
</tr>
<tr>
<td> 

Test RMSE: 14.781
Test EVS : 0.762
Test r2: 0.629</td>
<td> 

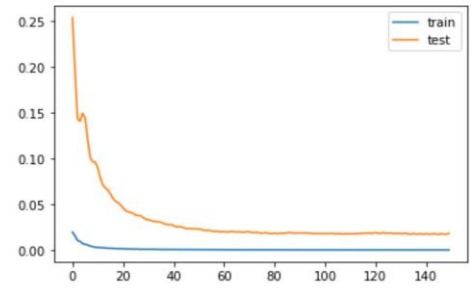
Test MAE: 7.837
Test EVS : 0.803
Test r2: 0.701</td>
</tr>
</table>

L-func / Test	MSE	MAE
RMSE/MAE	14.781	7.837

EVS	0.762	0.803
r2	0.629	0.701

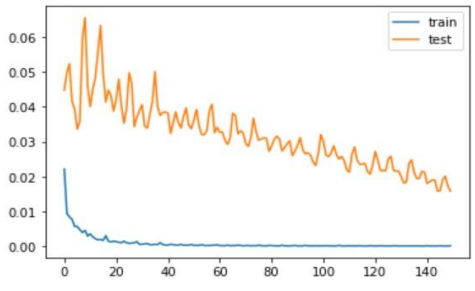
(5) optimizer:

前面有提過 LSTM 的每個神經元都有自己的權重，而損失函數，其實是利用權重 w 和 bias 計算出的預測值和真實值的函數，而優化器的任務，就是對權重和 bias 的調節。以下三個優化器由 Tensorflow 官方提供的，其中以 AdamOptimizer 最常被使用，因為它對學習率有個約束，使得每一次學習率都有一個確定範圍，這個應算法能快速找到參數更新並找到梯度正確下降的方向，而另外兩個優化器在收斂表現就不如 Adam.



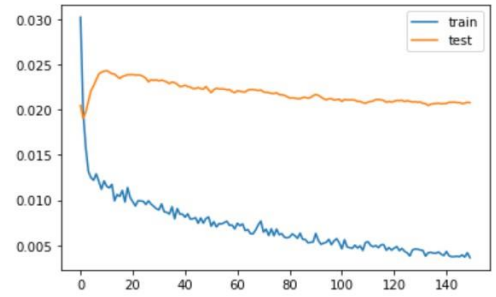

```
Test RMSE: 14.781
Test EVS : 0.762
Test r2: 0.629
```

(adam)




```
Test RMSE: 13.796
Test EVS : 0.785
Test r2: 0.677
```

(RMSprop)




```
Test RMSE: 15.848
Test EVS : 0.695
Test r2: 0.573
```

(SGD)

Optimizer / Test	adam	RMSprop	SGD
RMSE	14.781	13.796	15.848
EVS	0.762	0.785	0.695
R^2	0.629	0.677	0.573

(6) stacked LSTM:

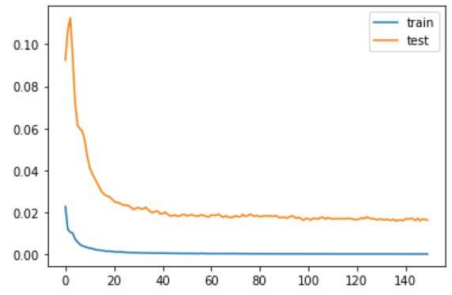
```
###stacked LSTM

model=Sequential()
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2]),return_sequences=True))
model.add(Dropout(0.4))
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.4))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()
```

左下圖的輸出是建立在只有一層 LSTM 下，因此最後的 output shape 是一個二維數據；右下圖的數據是建立在多層的 LSTM，我們會在第一層 LSTM 加上 returnsequence=True 讓輸出為三維，以便進行 LSTM 的堆疊，這邊要注意的是最後的 dense 輸出一樣是二維的數據。因為有深層且分層的神經網路模型，所以不管在 loss 值及判定係數上表現都比單層還要好，但是在收斂速度上好像顯得過快，我們懷疑隱藏層的增加是否還要搭配更多的參數調整，不然會過度擬和的問題而導致梯度下降過快。但基本上, 堆疊 lstm 在深度學習上能使模型更深入，幫我們預測更有挑戰性複雜的問題。

One stack

Layer (type)	Output Shape	Param #
lstm_133 (LSTM)	(None, 50)	12800
dropout_76 (Dropout)	(None, 50)	0
dense_103 (Dense)	(None, 1)	51

Total params: 12,851
Trainable params: 12,851
Non-trainable params: 0

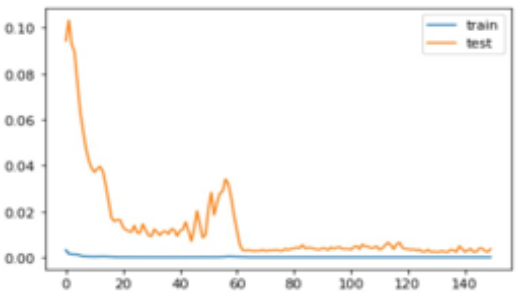


Test RMSE: 14.033
Test EVS : 0.792
Test r2: 0.665

Two stack

Layer (type)	Output Shape	Param #
lstm_136 (LSTM)	(None, 10, 50)	12800
dropout_79 (Dropout)	(None, 10, 50)	0
lstm_137 (LSTM)	(None, 50)	20200
dropout_80 (Dropout)	(None, 50)	0
dense_105 (Dense)	(None, 1)	51

Total params: 33,051
Trainable params: 33,051
Non-trainable params: 0

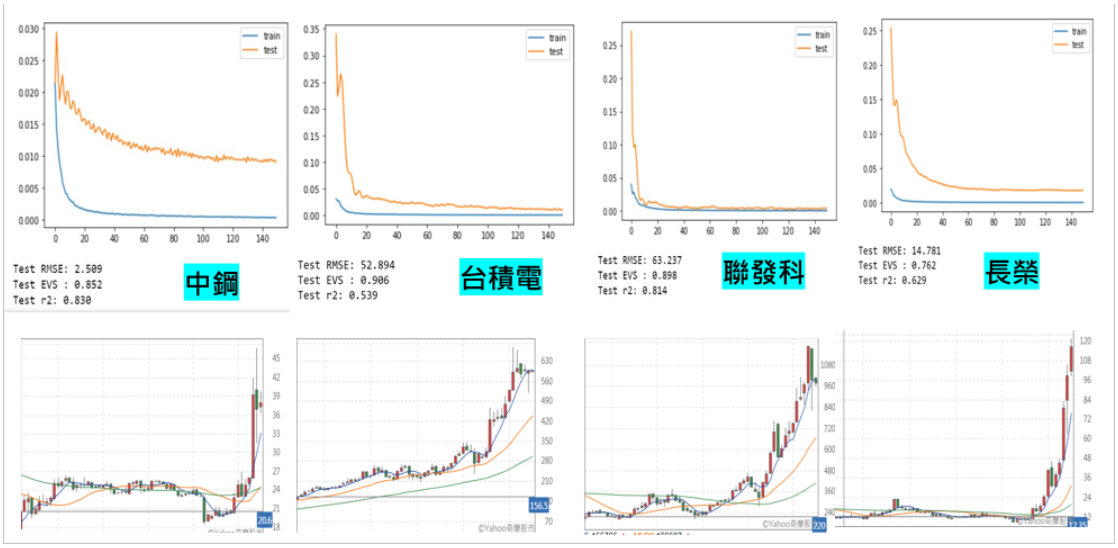


Test RMSE: 6.613
Test EVS : 0.936
Test r2: 0.926

5. Conclusions and novelty

(1) 不同股票的模型參數要適當調整(如:監督式學習的標籤):

我們先是選定長榮公司的資料進行模型訓練，而得到相關的實驗數據。因此我們好奇，同樣 lstm 的 baseline 是否適用於其他公司。我們利用四家上市公司的資料做測試，並分別繪出過去五年的歷史股價，以及損失函數收斂的情形，我們認為，台積電和聯發科的損失函數之所以能平穩的收斂，是因為他們的公司有較穩定的獲利能力，而使股價平穩的成長。因此在每輪 lstm 訓練進行後，繪製出的擬和曲線之趨勢也較為一致，在 loss function 中較少出現收斂反升的情形。我們當初選擇以十天做為一個 step size，每十天就為其放上一個標籤，是因為以我們目前對於股票的理解，若只有三五天好像太短，但是若將時間拉到一次兩周或是一個月則似乎又太長，因此我們取 10 天，但是後來我們發現這個 step size 似乎也跟每檔股票的個性有不小的關係，因此我們可以再多做嘗試。



(2) 特徵提取、多重共線性:

這裡是我們推測，為何使用了多變量時間序列模型，但是 accuracy 卻還是不慎理想的一個可能的原因，我們認為可能發生了多重共線性，因為在現實生活中，很難找到一組互不相關，又對因變數 y 產生重要影響的變數，我們在訓練時丟入的許多特徵值可能也是因彼此不完全是不相關，而產生了多重共線性的問題，我們在網路上找到了一個看起來可行的解決方案，就是 drop 掉其中幾個變量，我們發現他的原理，有可能跟剛剛的 dropout 很類似，都是將一群互相之間關係錯縱複雜的預測因子 drop 掉幾個，讓其之間的相互影響得到有效的減低。

6. The contribution of each team member

組員名稱	貢獻
統計系112林家同	1.使用網路爬蟲自FinMind獲取Dataset並找出較佳的特徵值 2.測試不同的LSTM函式參數並嘗試找出最佳組合 3.測試數據&圖表之統整
資訊系112莊上緣	1.Final project 投影片製作 2.建構自定義副程式並將部分參數微調測試模塊化 3.深度學習之概念查詢並彙整
資訊系112李培綸	1.建構UI圖像化使用者介面 2.LSTM 框架想法提供與觀念釋疑 3.將最終程式碼進行優化並封包

Reference:
https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/
https://cinnamonaitaiwan.medium.com/cnn%E6%A8%A1%E5%9E%8B-%E6%90%8D%E5%A4%B1%E5%87%BD%E6%95%B8-loss-function-647e13956c50
https://finmindtrade.com/

報告完畢