Midterm Exam of Quantitative Finance

07355003 Pei-Hsuan Hsu

題目：

以 2010-01-01 年至 2018-12-31 之 S&P 500 指數的日資料(daily data)為研究對象。以當日及過去 30 日之每日(Open，High，Low，Close，Volume)為特徵(features)，預測 10 日後之 Close。訓練樣本 (train set) 與測試樣本 (test set) 比為 8:2，以訓練樣本建立預測模型。特徵預處理 (preprocessing) 方法考慮 sklearn 之 MinMaxScaler 和 StandardScaler。以下列模型為例，配合 sklearn 之 Pipeline，GridSearchCV 作法，決定模型最佳超參數 (hyperparameter(s))，亦即做下列模型之參數調整(tune parameters)。交叉驗證 (cross-validation) 方法取為 sklearn 之 KFold，scoring 取為 neg mean absolute error。決定最佳超參數準則為:盡可能避免 overfitting 及希望交叉驗證 (cross-validation) 時可考慮 mean test score 和 std test score (這是我遇到業界問的問題，沒有標準答案)，其中 score 為 neg mean absolute error。

Step1：蒐集並整理資料

引入所需套件：

```
import numpy as np
import pandas as pd
import pandas_datareader as web
from sklearn.preprocessing import MinMaxScaler,
StandardScaler
from sklearn.model_selection import KFold
# from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPRegressor
import plotly
import plotly.graph_objs as go
```

使用 pandas_datareader 抓取股票資料：

```
ticker = '^GSPC'
start = '2010-01-01'
end = '2018-12-31'
df = web.data.DataReader(ticker, 'yahoo', start, end)
df = df.drop(['Adj Close'], axis=1)from sklearn.pipeline
```

抓取目標並建立特徵：

```
shift = -10
target = 'Close'
features = ['Open', 'High', 'Low', 'Close', 'Volume']

target_next = target+'_next'+str(-shift)
target_next_pred = target_next+'_pred'
df[target_next] = df[target].shift(shift)
df = df.dropna()



# Features
lag = 30
for i in range(1,lag+1):
    for feature in features:
        df[feature+"_lag"+str(i)] = df[feature].shift(i)

df = df.dropna()
```

將目標及特徵抓出：

```
X = df.drop([target_next], axis=1)
y = df[target_next]
```

*為簡化模型不做多項式特徵的生成

將資料分成訓練及測試樣本：

```
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X , y ,
test_size = test_size , random_state = 37)
```

(a)以 sklearn 之 Ridge (ridge regression)為例，超參數 alpha 考慮 0.1, 0.01, 0.001, 共 3 種情況(簡化問題)。

Step1：使用 `Pipeline` 建立流程，並用 `GridSearchCV` 找最適超參數

```
pipeline1 = Pipeline([('scaler', MinMaxScaler()),
                  ('regression', Ridge(random_state = 123457))
                  ])


parameters1 = {'scaler' : [StandardScaler(), MinMaxScaler()],
            'regression__alpha': np.logspace(-3,-1,3),
            }

scoring = 'neg_mean_absolute_error'

n_splits = 5
cv = KFold(n_splits=n_splits, shuffle = True, random_state = 123457)

(model_name, model) = pipeline1.steps[1]
SearchCV_a = GridSearchCV(estimator = pipeline1,
                     param_grid = parameters1,
                     scoring = scoring,
                     cv = cv,
                     return_train_score = True,
                     verbose = 1,
                     n_jobs = -1)

SearchCV_a.fit(X_train, y_train)
```

Step2：利用 `cv_results_` 並轉成 dataframe 的形式來查看結果

```
results_a = pd.DataFrame.from_dict(SearchCV_a.cv_results_)
```

評分公式：

score =(越小越好)

$$\left|\frac{\text{mean\_test\_score}}{\text{mean(mean\_test\_score)}} - \frac{\text{mean\_train\_score}}{\text{mean(mean\_train\_score)}}\right| + \text{std\_test\_score} + \text{std\_train\_score}$$

由於直接用 mean test score – mean train score 放進評分會造成無法評估兩種 mean 是否都很大，因此先除平均數再做相減。

```
results_a['score'] =
   abs((results_a['mean_test_score']/results_a['mean_test_score'].mean() - \
   results_a['mean_train_score']/results_a['mean_train_score'].mean())) + \
       (results_a['std_test_score'] + results_a['std_train_score'])
```

Step3：接著將 score 最小的挑出來

```
best = results_a[results_a['score'] == min(results_a['score'])]
```

結果：使用 MinMaxScale 做正規化且 alpha 為 0.01 之 Ridge 模型為最佳

Step4：將最佳的結果丟入 pipeline 中做預測，並存入 y_test_pred 及 y_train_pred 中

```
pipeline1_2 = Pipeline([('scaler' , best['param_scaler'][0]),
                        ('regression' , Ridge(
                         alpha = best['param_regression__alpha'][0]))
                         ])

best_model1_2 = pipeline1_2.fit(X_train, y_train)
y_test_pred1_2 = best_model1.predict(X_test)
y_train_pred1_2 = best_model1.predict(X_train)

y_test_pred = pd.concat([y_test , pd.Series(y_test_pred1_2 , index
   = y_test.index , name = 'y_test_pred1')] , axis = 1)
y_train_pred = pd.concat([y_train , pd.Series(y_train_pred1_2 ,
   index = y_train.index , name = 'y_train_pred1')] , axis = 1)
```

(b) 以 sklearn 之 MLPRegressor (類神經網絡之多層感知機 (MultiLayer Perceptron, MLP)) 為例, 超參數 alpha 考慮 0.1, 0.01, 0.001, 共 3 種情況 (簡化問題)。hidden layer sizes 考慮: 一個 hidden layer 的 sizes 為 32, 64, 128, 二個 hidden layer 的 sizes 為 64, 64, 共 4 種情況 (簡化問題)。

Step1：使用 `Pipeline` 建立流程，並用 `GridSearchCV` 找最適超參數

```
pipeline2 = Pipeline([('scaler', MinMaxScaler()),
                  ('MLP' , MLPRegressor(solver = 'lbfgs', random_state =
   123457))
                  ])

parameters2 = {'scaler' : [StandardScaler(), MinMaxScaler()],
             'MLP__alpha': np.logspace(-3,-1,3),
             'MLP__hidden_layer_sizes':[(32) , (64) , (128) , (64,64)],
             }

SearchCV_b = GridSearchCV(estimator = pipeline2,
                      param_grid = parameters2,
                      scoring = scoring,
                      cv = cv,
                      return_train_score = True,
                      verbose = 1,
                      n_jobs = -1)

SearchCV_b.fit(X_train, y_train)
```

Step2：利用 `cv_results_` 並轉成 dataframe 的形式來查看結果

```
results_b = pd.DataFrame.from_dict(SearchCV_b.cv_results_)
```

Step3：評分並將 score 最小的挑出來

```
results_b['score'] =
   abs((results_b['mean_test_score']/results_b['mean_test_score'].mean() - \
        results_b['mean_train_score']/results_b['mean_train_score'].mean())) + \
       (results_b['std_test_score'] + results_b['std_train_score']) #*
   (results_a['rank_test_score']*0.01)

bestb = results_b[results_b['score'] == min(results_b['score'])]
best = best.append(bestb, ignore_index=True)
```

結果：使用 StandardScaler 做正規化且 alpha 為 0.01、hidden layers sizes 為單層 64 個 node 之 MLPRegressor 為最佳

Step4：將最佳的結果丟入 `pipeline` 中做預測，並存入 `y_test_pred` 及 `y_train_pred` 中

```
pipeline2_2 = Pipeline([('scaler', best['param_scaler'][1]),
                  ('MLP' , MLPRegressor(alpha = best['param_MLP__alpha'][1] ,
   hidden_layer_sizes = best['param_MLP__hidden_layer_sizes'][1], solver =
   'lbfgs', random_state = 123457))
                  ])

best_model2_2 = pipeline2_2.fit(X_train, y_train)
y_test_pred2_2 = best_model2_2.predict(X_test)
y_train_pred2_2 = best_model2_2.predict(X_train)

y_test_pred['y_test_pred2'] = y_test_pred2_2
y_train_pred['y_train_pred2'] = y_train_pred2_2
```

(c) 同時考慮 (a), (b)。

Stpe1：將(a)、(b)小題之結果合併成 dataframe

```
all_results = results_a.append(results_b, ignore_index = True)
```

Stpe2：挑出 score 最小的

```
bestc = all_results[all_results['score'] == min(all_results['score'])]
best = best.append(bestc, ignore_index = True)
```

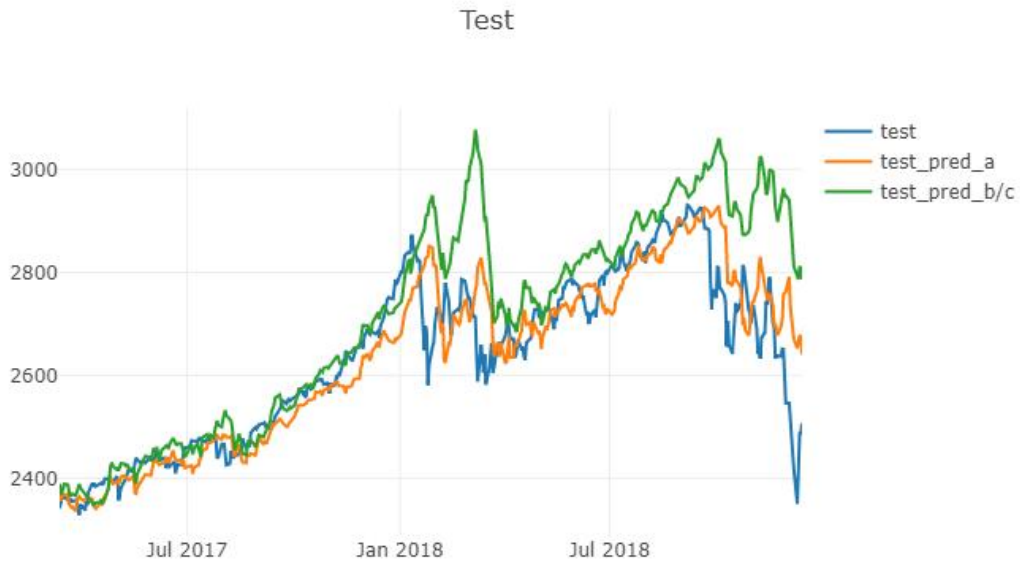
結果：和(b)小題選出之模型一樣

(d) 分別畫出 (a), (b), (c) 之 best model 於 train set, test set 的 10 日後 Close 及其 Close 預測值圖形，並分別計算其 mean absolute error。

1. Train Set

圖中顯示在訓練樣本中(a)、(b)、(c)選出之模型的預測幾乎是準的

2. Test Set



圖中顯示在訓練樣本中(a)、(b)、(c)選出之模型的預測在 2018 年 1 月以前誤差較小，但隨時間越近誤差越大

計算 MSE：

```
from sklearn.metrics import mean_squared_error

test_MSE_a = mean_squared_error(y_test_pred['Close_next10'], y_test_pred['y_test_pred1'])
test_MSE_bc = mean_squared_error(y_test_pred['Close_next10'], y_test_pred['y_test_pred2'])
```

結果顯示(a)小題之 MSE 為 4828.10，(b)、(c)小題之 MSE 為 16854.17。將所有結果做比較後，使用自訂的評分標準有可能會挑出 MSE 大的模型，這表示評分公式還須調整。

(e) 解釋 mean test score 和 std test score 於此問題之意義與重要性。

在看分數之前要先知道分數的標準是什麼，這次使用的評分標準為 negative mean squared error，因此分數要越小越好。mean test score 表示進行交叉驗證中所有結果的平均分數，std test score 表交叉驗證中所有結果分數之標準差。若 mean test score 小，則平均而言此種模型下，資料預測平均誤差小；若 std test score 小，則表示資料間交互預測的誤差小，較無 overfitting 的問題。因此兩者皆是在挑選模型時的重要指標。

(f) 此題目為何考慮 Ridge 而不用 Lasso？

由於本次所使用的資料集特徵為共線性非常高，若使用 Lasso 來建模的話，參數估計會有偏誤，這時候必須在高度相關的變數間做挑選，而這時候有可能我們會把重要的特徵給去除；但是若使用 Ridge 來建模就不會產生這樣的問題，此外 Ridge 所使用的正則化方法為 L2 norm，可以防止 overfitting 的發生，且實務上也認為 Ridge 的預測效果較 Lasso 好。