資料科學

Data Science

作業三 HW3

電機所 R11921038 江讀晉

2022/11/18

Synopsis

- Problem 1：`python hw3-p1.py`
- Problem 2：`python hw3-p2.py`
- Problem 3：`python hw3-p3.py`
- Package：numpy、pandas、matplotlib、sklearn、pmdarima

Problem 1. One-by-One Feature Selection

a. Feature selection method

One-by-one selection（naïve approach）的作法為，評估每個 feature 後挑選出最有用的前幾名來使用，因此需要適當的評估機制。課堂投影片中，提到了 similarity、statistic 和 information theory 為本的三種方向，由於本題的資料有給 label，可以得知分為正值和負值兩類，因此我選擇使用 Fisher score 此種 supervised feature selection 的方式，並將計算出的分數結果排序。Fisher score 的公式：

$$fisher\ score(f_i) = \frac{\sum_{j=1}^{c} n_j(\mu_{ij} - \mu_i)^2}{\sum_{j=1}^{c} n_j \sigma_{ij}^2}$$

b. Result

Fig.1 和 Fig.2 呈現出兩個 classifier（decision tree、SVM）所執行出的數據和圖表。由 Fig.1 可以看出，decision tree 的 accuracy 較高、feature 數目較少，整體表現較好。

	Decision tree	SVM
Accuracy	0.8705	0.8692
Number of features	35	75
Features	['Hsa.8147' 'Hsa.692' 'Hsa.37937' 'Hsa.1832' 'Hsa.692' 'Hsa.692' 'Hsa.36689' 'Hsa.1131' 'Hsa.2456' 'Hsa.6814' 'Hsa.8125' 'Hsa.36952' 'Hsa.601' 'Hsa.2928' 'Hsa.773' 'Hsa.957' 'Hsa.2344' 'Hsa.3306' 'Hsa.2097' 'Hsa.831' 'Hsa.4689' 'Hsa.8068' 'Hsa.6472' 'Hsa.1221' 'Hsa.2291' 'Hsa.462' 'Hsa.3016' 'Hsa.3305' 'Hsa.1047' 'Hsa.10755' 'Hsa.11616' 'Hsa.821' 'Hsa.5392' 'Hsa.1130' 'Hsa.1660']	['Hsa.8147' 'Hsa.692' 'Hsa.37937' 'Hsa.1832' 'Hsa.692' 'Hsa.692' 'Hsa.36689' 'Hsa.1131' 'Hsa.2456' 'Hsa.6814' 'Hsa.8125' 'Hsa.36952' 'Hsa.601' 'Hsa.2928' 'Hsa.773' 'Hsa.957' 'Hsa.2344' 'Hsa.3306' 'Hsa.2097' 'Hsa.831' 'Hsa.4689' 'Hsa.8068' 'Hsa.6472' 'Hsa.1221' 'Hsa.2291' 'Hsa.462' 'Hsa.3016' 'Hsa.3305' 'Hsa.1047' 'Hsa.10755' 'Hsa.11616' 'Hsa.821' 'Hsa.5392' 'Hsa.1130' 'Hsa.1660' 'Hsa.11673' 'Hsa.2800' 'Hsa.5398' 'Hsa.1205' 'Hsa.4252' 'Hsa.14069' 'Hsa.1073' 'Hsa.43279' 'Hsa.2863' 'Hsa.33' 'Hsa.5444' 'Hsa.678' 'Hsa.2588' 'Hsa.3331' 'Hsa.1588' 'Hsa.5971' 'Hsa.951' 'Hsa.466' 'Hsa.878' 'Hsa.56' 'Hsa.41323' 'Hsa.549' 'Hsa.832' 'Hsa.1435' 'Hsa.490' 'Hsa.1902' 'Hsa.8010' 'Hsa.10664' 'Hsa.7395' 'Hsa.41260' 'Hsa.2645' 'Hsa.3152' 'Hsa.2705' 'Hsa.33965' 'Hsa.2451' 'Hsa.5346' 'Hsa.853' 'Hsa.702' 'Hsa.25322' 'Hsa.1617']

Fig.1 兩 classifier 執行後的數據

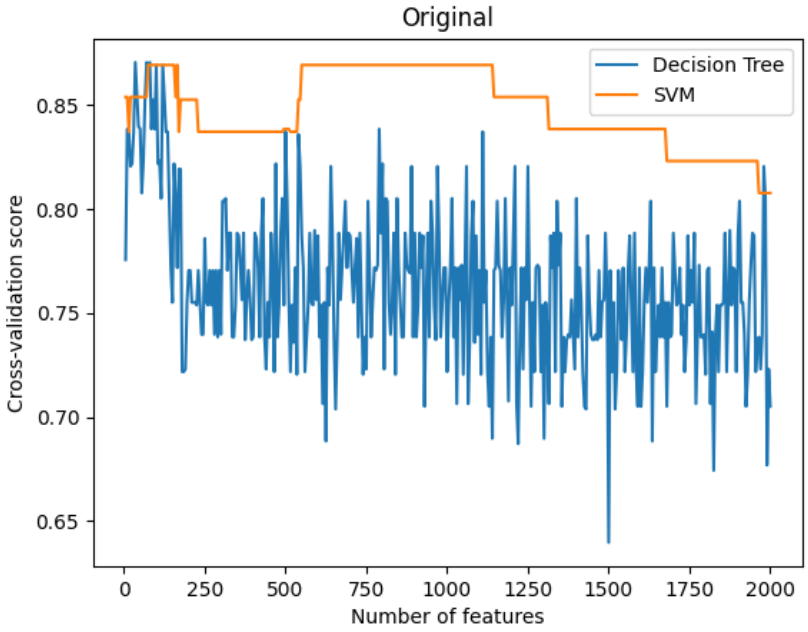

Fig.2 兩 classifier 執行後的圖表

c. Code description

Load data、feature evaluation、plot graph 的方式和 sample code 類似，便不特別描述。以下為我自己主要的實作部分。

1. Fisher score：計算分數後排序各個 feature 的 index。

```
def fisher_score(x, y):
    mean = np.mean(x, axis=0)

    # Sector
    c0 = y == 0
    c1 = y == 1

    # Compute fisher score
    num = len(x[c0]) * (np.mean(x[c0], axis=0) - mean) ** 2 + len(x[c1]) * (np.mean(x[c1], axis=0) - mean) ** 2
    den = len(x[c0]) * np.var(x[c0], axis=0) + len(x[c1]) * np.var(x[c1], axis=0)
    fisher = np.divide(num, den, out=np.zeros_like(num), where=den!=0)

    return fisher

# 3. Fisher score without sklearn
ranking_idx = fisher_score(x, y)
ranking_idx = np.argsort(ranking_idx)[::-1]
```

d. Discussion

根據 Fig.1 所示，可以發現原先資料集中的 feature 其實是有重複的，因此，所選出的 feature 個數其實遠比實際需要的多。如果要提升 feature selection 表現，應該要將重複的 feature 刪除。

Problem 2. Subset-Based Feature Selection

a. Algorithm

本題我所使用的演算法為 genetic algorithm（GA），其中的 operator 描述如 Fig.3 所示。此外，由於 2000 個 feature 轉換為 2000-bit 的 chromosome 的執行時間較久，收斂時間也較長，因此，我先將資料使用 fisher score 進行前處理，僅使用分數前 100 名的 feature 進行 chromosome coding。

超參數的部分，generation 設為 500 代，population size 設為 100，crossover 的機率為 0.8，mutation 機率則是設為 0.03。

Operator/Object	Description
Individual	將 feature index 使用 one-hot coding 的方式轉為 bitstring，即第 i 個 feature 就對應至第 i 個 bit（Big-endian）。如果有選擇該 feature，則對應的 bit 為 1，反之為 0。
Fitness function	使用 decision tree 分群，並計算 accuracy，以 accuracy value 作為 fitness。
Selection	使用 Russian roulette、tournament、truncation 三種方式實驗。其 selection pool 僅包含本代的 individual。
Crossover	使用 one-point crossover，crossover 的機率為事先給定。
Mutation	挑選其中一個 bit 進行變異，mutation 的機率為事先給定，其值大約為 1/(chromosome length)。
Population replacement	將上一代 population 中表現最好的 individual 保留至下一代，其餘則是由上一代 population 中進行 crossover 和 mutation 中得到。

Fig.3 GA 內容描述

b. Result

結果如 Fig.4 和 Fig.5 所述。如同 hw3-1，由於 feature 沒有刪除重複的項目，因此挑選出的 feature 仍有重複，此為可以改進的地方。此外，根據我自己的測試結果，無論使用何種 selection 方式，以及調整 population size、generation、crossover 和 mutation 的機率，其最佳解通常不是出現在後面幾代。因此，我認為 selection 的方式可能還可以再做更改，除了改變 selection operator，或許也可以朝 model GA 改進，如使用 DSMGA-II、GOMEA 等演算法，來改善 accuracy 以及收斂效果。

	GA best individual
Accuracy	0.9192
Best result	In 79-th generation
Number of features	49

Fig.4 GA 執行後的數據

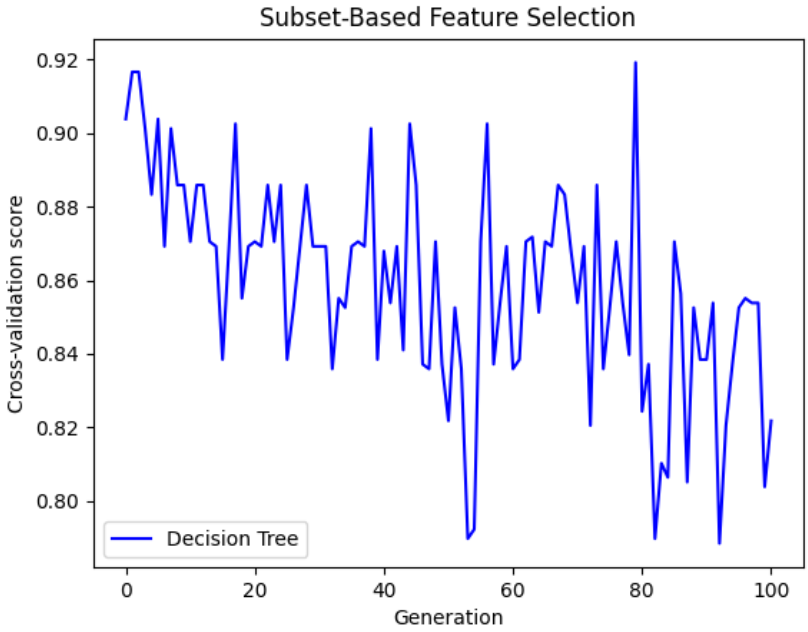

Fig.5 每一代最佳解之折線圖

c. Code description

Load data、feature evaluation、plot graph 的方式和 sample code 類似，便不特別描述。Fisher score 則沿用 hw3-1。以下為我自己主要的實作部分。

1. Genetic algorithm

```
def genetic_algorithm(data, bit_length, generation, population_size, p_cross=0.8, p_mut=0.1):
    '''
    population: (population_size, bit_length)
    fitness: (population_size, )
    parents: (population_size, bit_length)
    children: (population_size, bit_length)

    best individual: (generation, bit_length)
    best fitness: (generation, )
    '''

    # Initialize population
    population = np.random.randint(2, size=(population_size, bit_length))
    population = population.tolist()

    # Track the best
    best_individual = []
    best_fitness = []

    # Start evolution
    for gen in range(generation + 1):
        if gen % 10 == 0:
            print(f'Generation: {gen}')
        # Evaluate the fitness of each individual
        fitness = []
        for individual in population:
            fitness.append(fitness_function(data, individual))

        # Save the best individual
        best_individual.append(population[np.argmax(fitness)])
```

```
        best_fitness.append(np.max(fitness))

        # Select the parents based on roulette wheel selection
        parents = selection(population, fitness, population_size, type='roulette')

        # Crossover
        children = parents
        for i in range(0, population_size, 2):
            if random.random() < p_cross:
                crossover_point = random.randint(1, bit_length - 1)
                children[i][crossover_point:], children[i + 1][crossover_point:] = parents[i + 1][crossover_point:],
parents[i][crossover_point:]

        # Mutation
        for i in range(population_size):
            if random.random() < p_mut:
                mutation_point = random.randint(0, bit_length - 1)
                children[i][mutation_point] = 1 - children[i][mutation_point]

        # Replace the old population with the new one
        population = children
        population[0] = best_individual[-1]

    return best_individual, best_fitness
```

2. GA 中的 fitness function 和 selection

```
def fitness_function(data, individual):
    # Select Top m feature
    feature_idx = np.where(np.array(individual) == 1)[0]
    data_subset = data[:, feature_idx]

    # Build random forest
    clf = DecisionTreeClassifier(random_state=0)
    # clf = SVC(kernel='rbf', random_state=0) #build SVM

    # Calculate validation score
    scores = cross_val_score(clf, data_subset, y, cv=5)

    # Save the score calculated with m feature
    return scores.mean()

def selection(population, fitness, population_size, type='roulette'):
    if type == 'roulette':
        # Select the parents based on roulette wheel selection
        fitness = np.array(fitness)
        fitness = fitness / fitness.sum()
        parent_idx = np.random.choice(population_size, size=population_size, p=fitness)
        parents = [population[i] for i in parent_idx.tolist()]
    elif type == 'tournament':
        # Select the parents based on tournament selection
        parents = []
        for i in range(population_size):
            idx = np.random.choice(population_size, size=2, replace=False)
            if fitness[idx[0]] > fitness[idx[1]]:
                parents.append(population[idx[0]])
            else:
                parents.append(population[idx[1]])
    else:
        fitness_idx = np.argsort(fitness)
        parents = [population[idx] for idx in fitness_idx[:population_size // 2]]
        parents += [population[idx] for idx in fitness_idx[:population_size // 2]]

    return parents
```

3. Main function 中執行部分

```
### Subset-Based Feature Selection ###
# Use genetic algorithm to select the best subset of features
x_truncated = x[:, ranking_idx[:100]]
bit_length = x_truncated.shape[1]
generation = 100
population_size = 100
feature_history, score_history = genetic_algorithm(x_truncated, bit_length, generation, population_size, 0.8, 0.03)
num_features = np.sum(feature_history, axis=1)
```

Problem 3. ARIMA Forecast

a. Parameter choice

ARIMA(p, d, q)(P, D, Q)(s)總共有七個參數，全部用 grid search 或是 random search 可能時間成本較高，因此先做幾個 pmdarima 提供的方法以及投影片提及的步驟篩選出 p、d、q 和 Q，再嘗試不同的 P、Q、s。

1. 觀察 time series，經過一階差分後，資料已經變得較為穩定，如 Fig.6 所示。
2. 使用 pmdarima 提供的 method 計算本題的 d 和 D。
3. 畫出 autocorrelation function（ACF）和 partial autocorrelation function（PACF），如 Fig.7 所示，可觀察出 lag 數值和 function 變化，因而決定 p 和 q。
4. 固定 p、d、q 和 Q，嘗試不同的 P、Q、s。此部分使用 auto_arima()輔助。

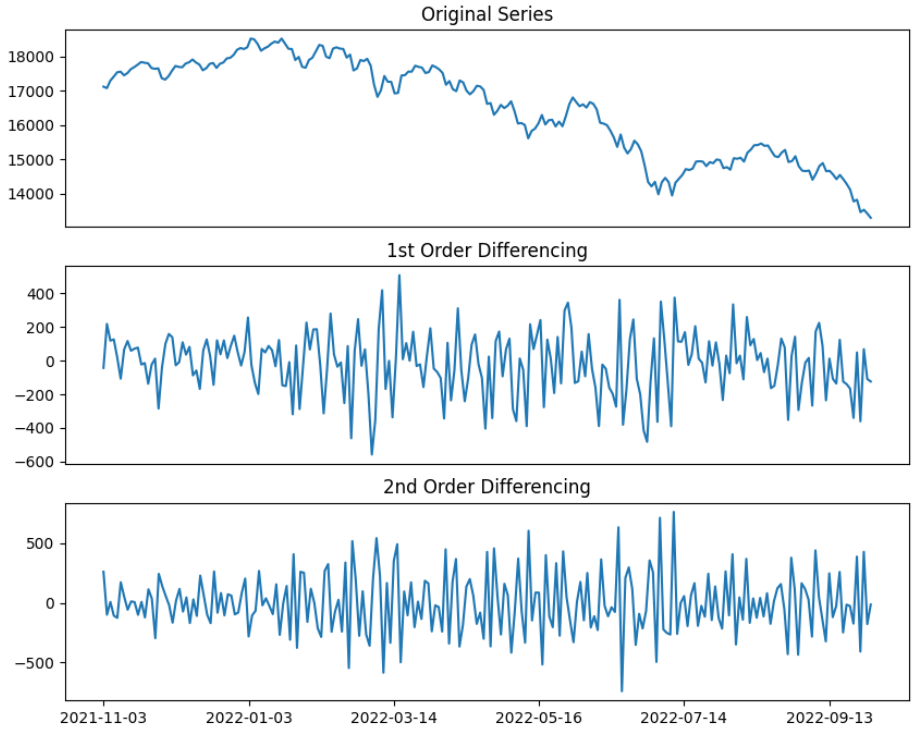

Fig.6 一階差分和二階差分之結果

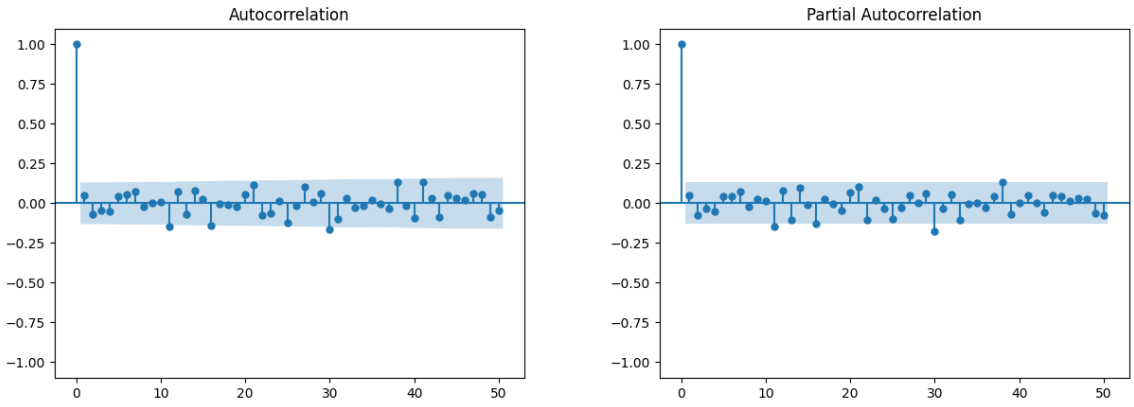

Fig.7 ACF 和 PACF，可看出在 lag=1 之後，和當期資料已無明顯相關性

最後，我使用了兩組參數，分別執行 ARIMA，其結果如 Fig.8、Fig.9 所示。藍線為 training data，橘線為 test data，綠線為預測結果，淺紫色區域為 confidence interval。

(p, d, q)(P, D, Q)(s)	(0, 1, 0)(1, 0, 1, 30)
MSE	88733.210
描述	預測結果為一緩降的曲線

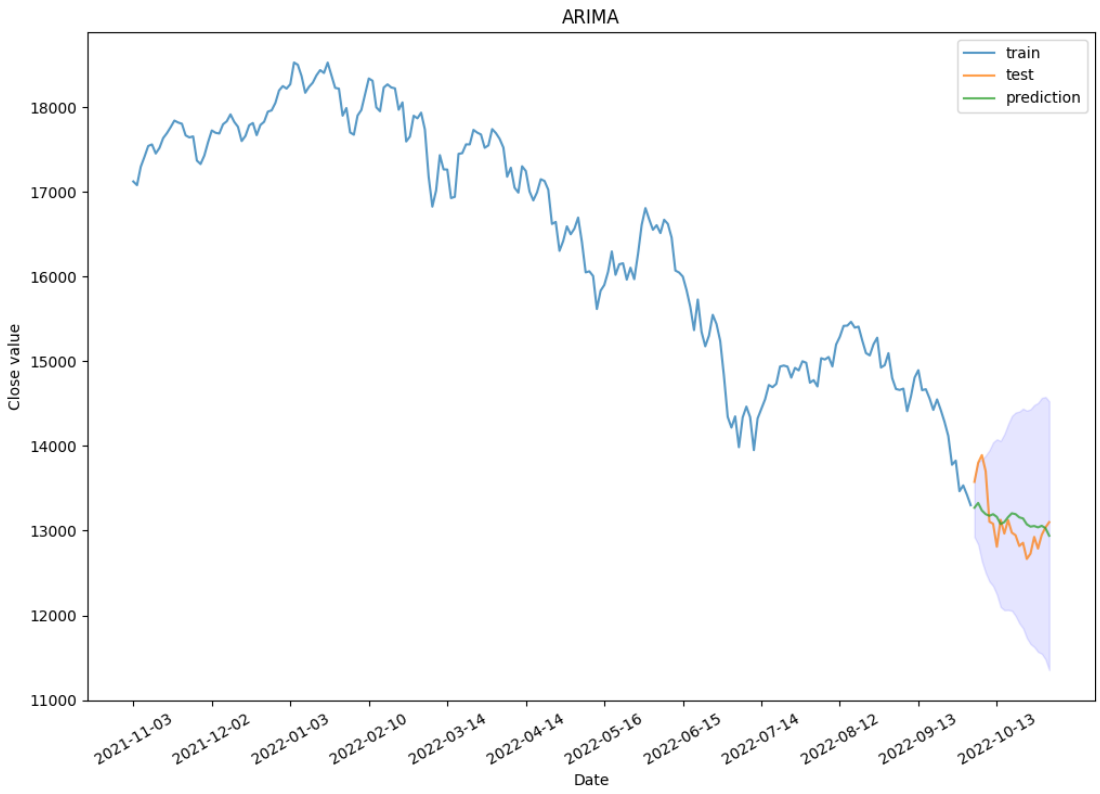

Fig.8 第一組參數和結果

(p, d, q)(P, D, Q)(s)	(0, 1, 0)(2, 1, 0, 30)
MSE	100712.783
描述	預測結果的波動傾向較接近 ground truth，但其信賴區間範圍也更廣

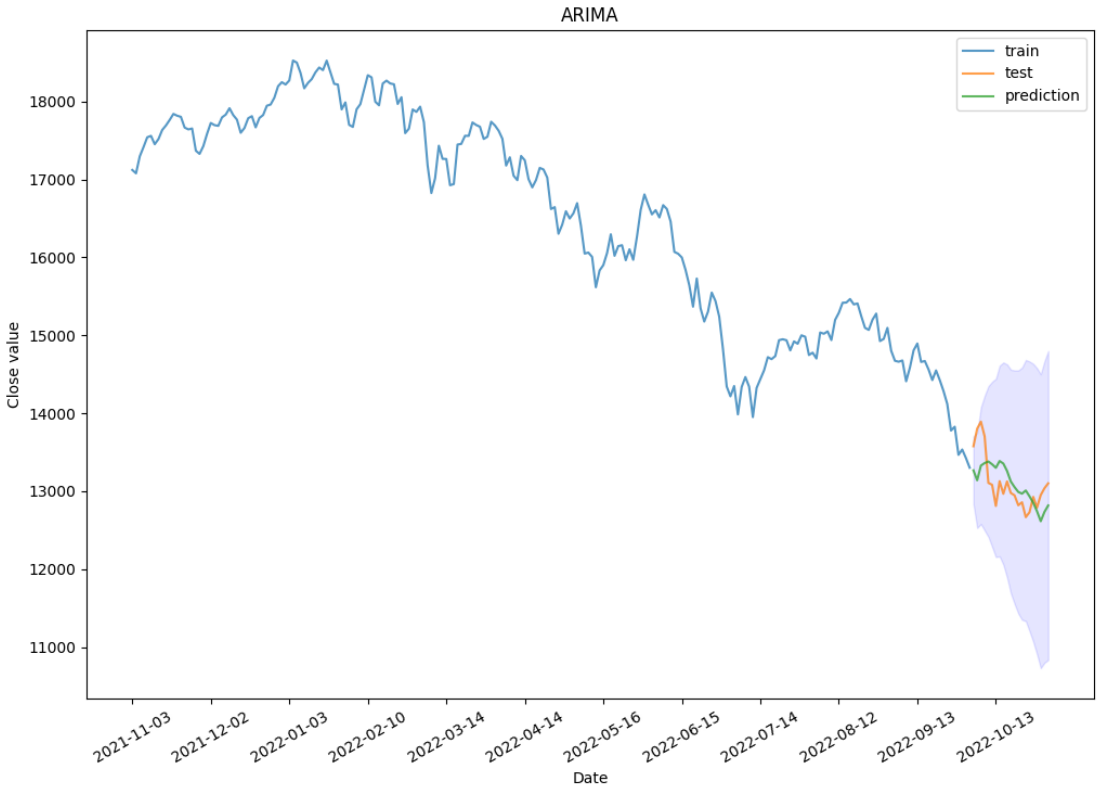

Fig.9 第二組參數和結果

b. Discussion

除了如題目要求，用 training data 一次預測 test data，pmdarima 也提供每預測一筆 data 就使用 ground truth 去更新 trained model，此種方法會呈現出較為精準的預測。結果如 Fig.10 所示。

(p, d, q)(P, D, Q)(s)	(0, 1, 0)(1, 0, 1, 30)
MSE	45775.794
描述	每預測一筆資料即更新 model 的預測結果，其預測曲線和信賴區間皆有不錯的 fitting 效果

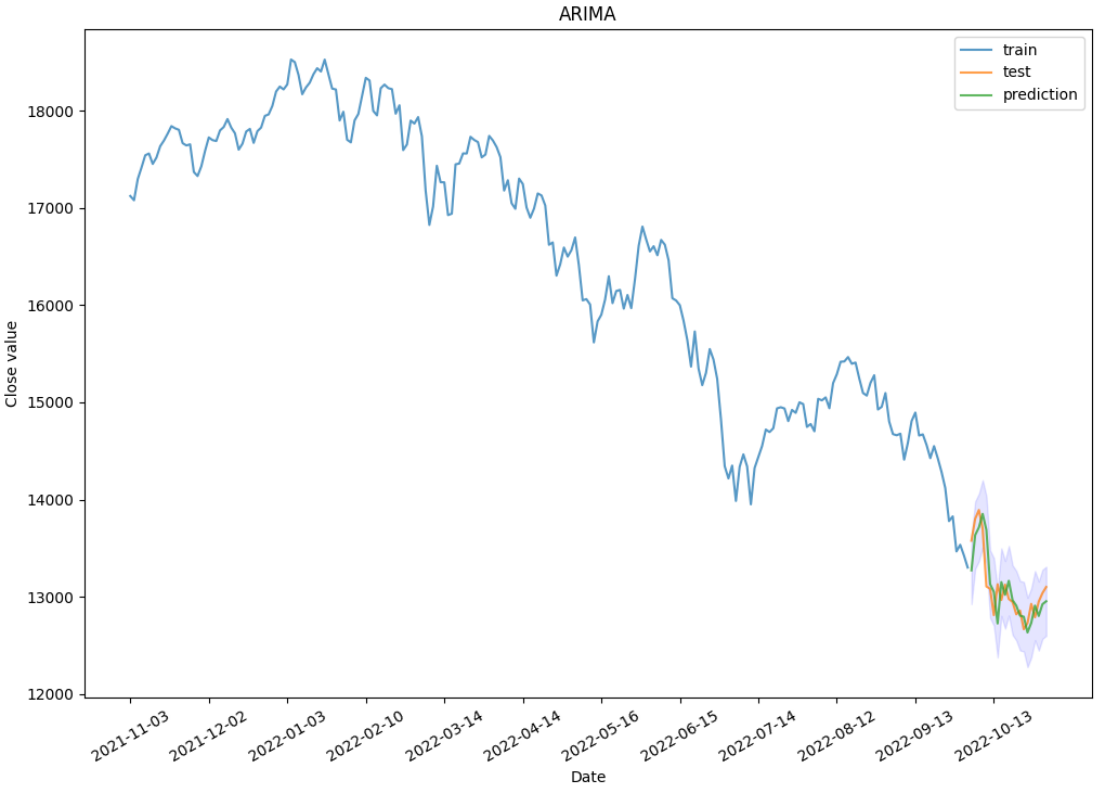

Fig.10 第一組參數每預測一筆資料即更新 model 的預測結果

Reference

[1] Jason Brownlee. Simple Genetic Algorithm From Scratch in Python. 2021.

https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/

[2] Taylor G Smith. pmdarima: ARIMA estimators for Python. 2022.

https://alkaline-ml.com/pmdarima/index.html

[3] Yogesh Verma. Quick way to find p, d and q values for ARIMA. 2022

https://analyticsindiamag.com/quick-way-to-find-p-d-and-q-values-for-arima/

[4] 農二代 Webwr。時間序列模型 python 應用-銅價格預測。2021。

https://adaptable-haze-butterfly-551.medium.com/arima 時間序列模型 python 應用-銅價格預測-一-4f91693e3ec6

[5] Cindy Li。時間序列探索(二)：ARIMA 家族簡介。2021。

https://medium.com/@cindy050244_52136/時間序列探索-二-arima 家族簡介-8d533f0b18d6

[6] 黃業忠。Python 中做时间序列分析。2016。

https://www.ichdata.com/use-python-to-do-time-series.html