Intro to FinTech – Final Project report

資工二 B07902046 高長聖

1. Introduction:

首先，我將詳述在本次作業，我使用的策略及流程:先建立策略模型，然後分析 2011/1/3 ~ 2019/12/16 之間的台股開盤價，藉此定義模型中參數範圍。最後，利用 GA(遺傳演算法)來尋找最佳參數。

而在文末，我將探討為何我使用的策略無法達到更佳的獲利，並找出未來能夠繼續讓模型更加優化的方法。

2. Implementation:

(1) 模型設計與資料分析:

我使用的模型，是混和傳統數據，包括 MA、RSI、KD 值，以及較直覺的股票漲跌策略，包括已經連續漲、跌幾天。並在買與賣分別設定界線，若全部符合，即執行買賣，反之則不動作。

MA: 設定短期天數、長期天數，及短期 MA-長期 MA 所要超過界線。經由分析，可知隔日開盤價的漲跌幅度大致落在[-300, 300]之間。因此，短期天數的範圍為[1, 20]，長期為[21, 50]，差異程度範圍[-300, 300]。

RSI: 設定短期天數、長期天數，即短期 RSI-長期 RSI 所要超過的界線。而 RSI 的範圍為[0, 1]，因此，設定短期天數範圍[1, 20], 長期天數範圍[1, 50]，差異程度範圍[0, 1]。

KD: 設定 K 值小於 D 值時，D 值的上界；以及 K 值大於 D 值時，D 值的下界。由於 K 和 D 值都位在[0, 100]，因此上界和下界範圍皆為[0, 100]。

連續漲跌天數: 設定連續漲跌天數的界線，以及目前是漲或是跌的的狀態。經由分析，所有連續漲、連續跌的天數，都是在[0, 8]。因此，設定連續漲跌天數界線範圍[0, 8]。

(2) GA(遺傳演算法)設計:

定義染色體: 模型本身，而基因即為模型參數，共 14 個參數。

Fitness function: 採用多點之間的比較。由於有可能有時間長短所造成的資料差異，因此不採用單純獲利率，而是將每一點的最佳可能獲利算出，並檢查當前模型在那天的獲利，算出兩者之間的比例。

Encoding: 利用 Python 中的 list，將所有參數，以數值的方式，按照固定的順序存於其中。

初始族群: 完全由隨機產生。族群大小依照所設定的值進行調整。

Selection: 依照 fitness function 的大小，挑選出所需族群規模的模型個體。並採用菁英主義(elitism)，將母代的模型也加入挑選名單中。

Crossover: 全體互相交配，也就是交配率 100%。子代從父代和母代那邊，隨機繼承各個模型參數。這樣的邏輯，是假設個體會有所突出的原因，在於某幾個圖的的基因(參數)互相搭配。因此採用隨機繼承的方式，尋找其他可能的參數搭配方式。

Mutation: 每個參數的突變機率皆為 1/10。突變的方式，即是從新再從參數的定義域中，隨機選取一個值。

終止條件: 依照一開始設定的回合數，迭代此回合後即可終止。

以下，為最終寫出來的程式:

daily_data: 2011/1/3~2019/12/16 的股市 ohlcv_daily 資料。

generation_scale: 設定族群大小。

test_round: 設定迭代次數。

start_day: 訓練資料的起始位置。藉此控制訓練資料大小。

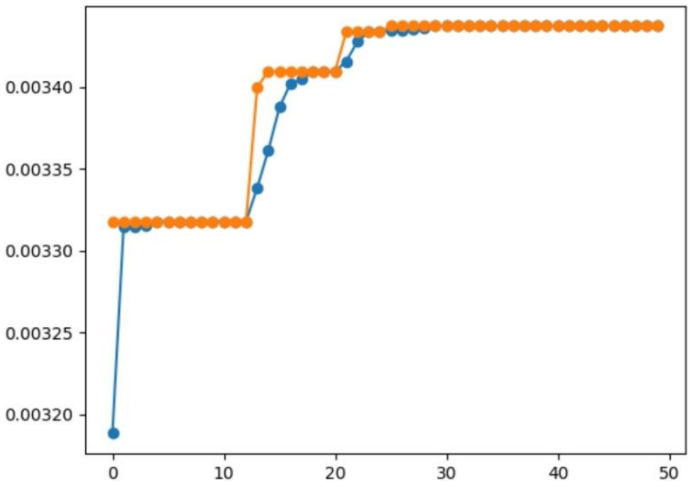
const_range: 各個參數的定義域。

回傳值: 在每一回合的平均 fitness 以及最大 fitness

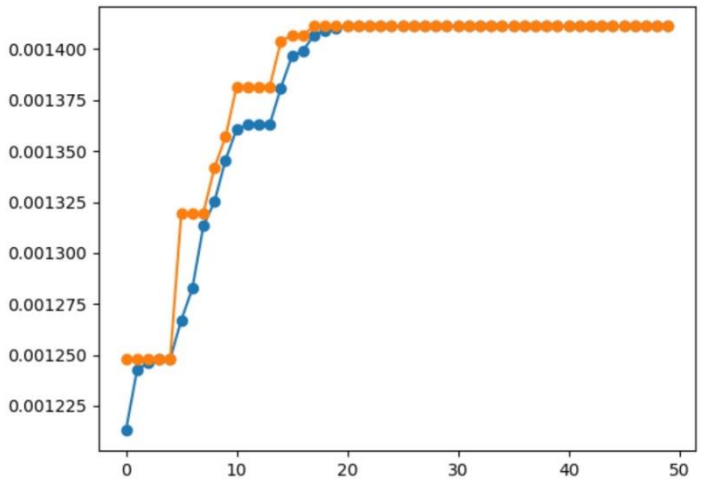
```
50
51  def GA_Training(daily_data, generation_scale, test_round, start_day, const_range):
52      max_pft = 0
53      max_const = None
54      const_num = len(const_range)
55      survive_set = []#survived gene
56      avg_pft_set = []#average profit of each round
57      max_pft_set = []#max profit of each round
58      #create the first generation
59      for i in range(generation_scale):
60          new_gene = []
61          for j in range(const_num):
62              new_gene.append(rdi(const_range[j]))
63          new_pft = pft_Esitmate(daily_data, start_day, new_gene)
64          survive_set.append([new_pft, new_gene])
65      #start to traing, test_round rounds
66      for rd in range(test_round):
67          print('Round ', rd)
68          for creature in survive_set:
69              print('survived creature:', creature)
70          new_set = survive_set.copy() #elite classic, preserve all creatures in parent geneartion
71          #crossover
72          for i in range(generation_scale):
73              for j in range(i + 1, generation_scale):
74                  ng = []
75                  for k in range(const_num): #inherit both parent's gene in possibilities of 0.5
76                      ng.append(survive_set[i][1][k]) if rdi([1, 10]) <= 5  \
77                      else ng.append(survive_set[j][1][k])
78                  new_pft = pft_Esitmate(daily_data, start_day, ng) #calculate the fitness function
79                  print('new creature:', new_pft, ng)
80                  new_set.append([new_pft, ng])
81          #selection, find the top {generation_scale} creatures
82          new_set.sort(key = sort_gene, reverse = True)
83          pft_value = [a[0] for a in new_set[0:generation_scale]]
84          #find the average profit and the max profit
85          avg_pft_set.append([rd, sum(pft_value) / len(pft_value)])
86          max_pft_set.append([rd, max(pft_value)])
87          if new_set[0][0] > max_pft:
88              max_pft = new_set[0][0]
89              max_const = new_set[0][1]
90          #new survive set
91          survive_set = new_set[0:generation_scale]
92          print('now max pft = ', max_pft, max_const)
93          #mutation
94          for i in range(1, generation_scale):
95              for j in range(const_num): #mutation in the possibilities of 1/10
96                  if rdi([1, 100]) <= 10: survive_set[i][1][j] = rdi(const_range[j])
97
98      return avg_pft_set, max_pft_set
```

(3) 結果:

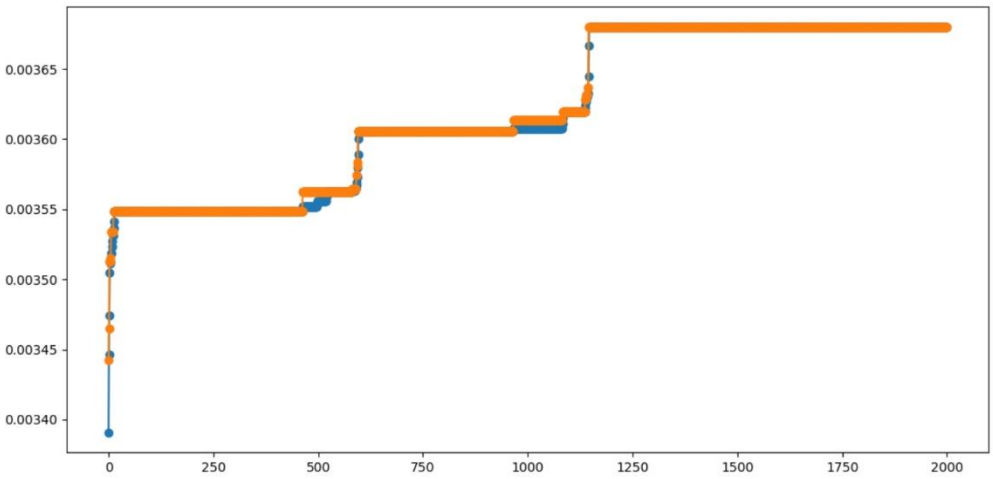
橫軸:迭代次數；縱軸:fitness；橘線:最大 fitness；藍線:平均 fitness



(a) Data size: 14, Test round: 50



(b) Data size: 140, Test round: 50



(c) Data size: 14, Test round: 2000

(4) 分析:

首先，上面三張圖都顯示，當我的 GA 迭代夠多次之後，最大 fitness 會達到某個值而穩定下來。這表示，我所設計的遺傳演算法，具有尋找區域最佳解的能力。

再來，上面三張圖也都顯示，當最大 fitness 達到平穩狀態後，平均 fitness 也會跟進，最後趨近於最大 fitness。這表示，在菁英主義(elitism)的搭配下，族群可以迅速往好的方向演進。然而，這也增加演算法跳脫出區域最佳解的難度。

最後，比較圖(a)和圖(c)，同樣的 Testing data size 都為 14，而一個迭代 50 次，另一個迭代 2000 次。可以發現，當不理會平穩狀態而繼續迭代下去的話，每隔一段時間，都有可能讓最大值 fitness 繼續進步。推測，這是因為 mutation 的關係，創造出一個或者是多個特別強大的個體，使得整體族群跳脫出區域最佳解之中，往更好的地方邁進。這也顯示出，GA 中的 mutation，實際上有利於族群的發展。

(5) 結論:

最終，我使用了由 Testing data size 為 14，Test round 為 50 的測試資料，選擇了:

短期 MA: 4，長期 MA: 48，買進長期 MA-短期 MA: 203，賣出短期 MA-長期 MA: 213

短期 RSI: 18，長期 RSI: 46，買進長期 RSI-短期 RSI: 0.81，賣出短期 RSI-長期 RSI:0.77
買進 D 值上界: 71，賣出 D 值下界: 14
買進時的趨勢: 較前一天價格低，連續漲跌界線: 跌 8 天
賣出時的趨勢: 較前一天價格高，連續漲跌界線: 漲 6 天
作為最終買賣策略模型。

(6) 缺失:

首先，在資料分析方面，我僅以每日的開盤價格進行分析，錯失許多微小細節，模型的設計方面也較不全面。再來，由於我 GA 採用 elitism 的關係，會加深受限於區域最佳解的影響，導致跨區域的演化速度較慢。最後，我所使用的 crossover 方式、mutation rate、generation scale，以及族群初始化的方式，都僅僅是證明能找到區域最佳，無法說明擁有更好，找到全域最佳解的能力。

3. Improvement:

(1) 從模型下手:

可以增加模型的強度，例如，將分析資料從僅僅是每日的開盤價，到對 OHLC 都進行分析，或者是納入每分鐘的交易趨勢。

(2) 從 GA 下手:

GA 最大的缺點，就是容易墮入區域最佳解之中，而不易找到全域最佳解。因此，或許在 crossover 的方式，或者 mutation rate，都可以進行調整。也可以搭配災變(catastrophe)的運用，當最大 fitness 和平均 fitness 趨於平穩狀態時便摧毀掉當前優秀染色體，以加速區域最佳位置的轉變。

(3) 配合其他演算法:

配合蟻群演算法、模擬退火法、梯度遞增法等演算法的運用，彼此互相彌補對方的缺點，也可以找到更好的參數解。