[Dacon] 블럭 장난감 제조 공정 최적화 경진대회

_ (팀명)

2020년 월 일 (제출날짜)

1. 본 코드는 대회 참가를 돕고자 단순 예시를 작성한 것으로 참고용으로 사용바랍니다.
2. 본 코드는 자유롭게 수정하여 사용 할 수 있습니다.
3. 추가 모듈 보러가기: https://bit.ly/36MNs76 (https://bit.ly/36MNs76)

1. 라이브러리 및 데이터

Library & Data

In [1]:

```
import pandas as pd
import numpy as np
import multiprocessing
import warnings
from copy import deepcopy
from module.genome import Genome, genome_score
import datetime
warnings.filterwarnings(action='ignore')
np.random.seed(777)
```

In [2]:

```
!python --version
print('Pandas : %s'%(pd.__version__))
print('Numpy : %s'%(np.__version__))
```

```
Python 3.7.6
Pandas : 1.0.1
Numpy : 1.18.1
```

2. 데이터 전처리

Data Cleansing & Pre-Processing

In [3]:

```
# 입력하세요.
```

3. 탐색적 자료분석

Exploratory Data Analysis

In [4]:

```
# 입력하세요.
```

4. 변수 선택 및 모델 구축

Feature Engineering & Initial Modeling

In [5]:

```
CPU_CORE = multiprocessing.cpu_count() # 멀티프로세싱 CPU 사용 수
N_POPULATION = 200                      # 세대당 생성수
N_BEST = 10                             # 베스트 수
N_CHILDREN = 5                         # 자손 유전자 수
PROB_MUTATION = 0.4                    # 돌연변이
REVERSE = True                         # 배열 순서 (False: ascending order, True: descending order)

score_ini = 10                         # 초기 점수
input_length = 125                     # 입력 데이터 길이
output_length_1 = 5 * 2                # Event (CHECK_1~4, PROCESS)
output_length_2 = 12 * 2                   # MOL(0~5.5, step:0.5)
h1 = 50                                # 히든레이어1 노드 수
h2 = 50                                # 히든레이어2 노드 수
h3 = 50                                # 히든레이어3 노드 수
EPOCHS = 2000                            # 반복 횟수

genomes = []
for _ in range(N_POPULATION):
    genome = Genome(score_ini, input_length, output_length_1, output_length_2, h1, h2, h3)
    genomes.append(genome)
try:
    for i in range(N_BEST):
        genomes[i] = best_genomes[i]
except:
    best_genomes = []
    for _ in range(N_BEST):
        genome = Genome(score_ini, input_length, output_length_1, output_length_2, h1, h2, h3)
        best_genomes.append(genome)
```

In [6]:

```
best_genomes[0].forward(np.zeros((1, 125)))
```

Out[6]:

```
('CHECK_1', 'CHECK_1', 0.0, 0.0)
```

5. 모델 학습 및 검증

Model Tuning & Evaluation

1. PRT는 고정값 사용
2. Event A, Event B (MOL_A, MOL_B) 를 같은 값으로 제한
3. Event는 CHECK와 PROCESS 만 사용함

4. 목적 함수로 수요 부족분만 고려함
5. Event와 MOL에 대해 인공신경망 모델을 만들어 유전 알고리즘으로 학습

In [7]:

```
n_gen = 1
score_history = []
high_score_history = []
mean_score_history = []
best_gen = None
best_score_ever = 0
while n_gen <= EPOCHS:
    print('EPOCH', n_gen, datetime.datetime.now())
    genomes = np.array(genomes)
    while len(genomes)%CPU_CORE != 0:
        genomes = np.append(genomes, Genome(score_ini, input_length, output_length_1, output_length_
    genomes = genomes.reshape((len(genomes)//CPU_CORE, CPU_CORE))

    for idx, _genomes in enumerate(genomes):
        if __name__ == '__main__':
            pool = multiprocessing.Pool(processes=CPU_CORE)
            genomes[idx] = pool.map(genome_score, _genomes)
            pool.close()
            pool.join()
    genomes = list(genomes.reshape(genomes.shape[0]*genomes.shape[1]))

     # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    # 평균 점수
    s = 0
    for i in range(N_BEST):
        s += genomes[i].score
    s /= N_BEST

    # Best Score
    bs = genomes[0].score

    # Best Model 추가
    if best_genomes is not None:
        genomes.extend(best_genomes)

    # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    score_history.append([n_gen, genomes[0].score])
    high_score_history.append([n_gen, bs])
    mean_score_history.append([n_gen, s])

    if genomes[0].score > best_score_ever:
        best_score_ever = genomes[0].score
        best_gen = genomes[0]

    # 결과 출력
    print('EPOCH #%s₩tHistory Best Score: %s₩tBest Score: %s₩tMean Score: %s' % (n_gen, genomes[0].s

    # 모델 업데이트
    best_genomes = deepcopy(genomes[:N_BEST])

    # CHILDREN 생성
    for i in range(N_CHILDREN):
        new_genome = deepcopy(best_genomes[0])
        a_genome = np.random.choice(best_genomes)
        b_genome = np.random.choice(best_genomes)
        for j in range(input_length):
            cut = np.random.randint(new_genome.w1.shape[1])
            new_genome.w1[j, :cut] = a_genome.w1[j, :cut]
            new_genome.w1[j, cut:] = b_genome.w1[j, cut:]

        for j in range(h1):
            cut = np.random.randint(new_genome.w2.shape[1])
            new_genome.w2[j, :cut] = a_genome.w2[j, :cut]
            new_genome.w2[j, cut:] = b_genome.w2[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w3.shape[1])
            new_genome.w3[j, :cut] = a_genome.w3[j, :cut]
            new_genome.w3[j, cut:] = b_genome.w3[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w4.shape[1])
            new_genome.w4[j, :cut] = a_genome.w4[j, :cut]
            new_genome.w4[j, cut:] = b_genome.w4[j, cut:]

        for j in range(input_length):
            cut = np.random.randint(new_genome.w5.shape[1])
            new_genome.w5[j, :cut] = a_genome.w5[j, :cut]
            new_genome.w5[j, cut:] = b_genome.w5[j, cut:]

        for j in range(h1):
            cut = np.random.randint(new_genome.w6.shape[1])
            new_genome.w6[j, :cut] = a_genome.w6[j, :cut]
            new_genome.w6[j, cut:] = b_genome.w6[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w7.shape[1])
            new_genome.w7[j, :cut] = a_genome.w7[j, :cut]
            new_genome.w7[j, cut:] = b_genome.w7[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w8.shape[1])
            new_genome.w8[j, :cut] = a_genome.w8[j, :cut]
            new_genome.w8[j, cut:] = b_genome.w8[j, cut:]

        best_genomes.append(new_genome)

    # 모델 초기화
    genomes = []
    for i in range(int(N_POPULATION / len(best_genomes))):
        for bg in best_genomes:
            new_genome = deepcopy(bg)
            mean = 0
            stddev = 0.2
            # 50% 확률로 모델 변형
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w1 += new_genome.w1 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w2 += new_genome.w2 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w3 += new_genome.w3 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w4 += new_genome.w4 * np.random.normal(mean, stddev, size=(h3, output_le
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w5 += new_genome.w5 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w6 += new_genome.w6 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w7 += new_genome.w7 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w8 += new_genome.w8 * np.random.normal(mean, stddev, size=(h3, output_le
            genomes.append(new_genome)

    if REVERSE:
        if bs < score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_le
    else:
        if bs > score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_le

    n_gen += 1
```

```
EPOCH 1 2020-07-01 01:56:25.236502
EPOCH #1        History Best Score: 82.6869582127399    Best Score: 82.6869582127
399     Mean Score: 80.03549654494219
EPOCH 2 2020-07-01 02:05:16.042431
EPOCH #2        History Best Score: 84.06589294343647   Best Score: 84.0658929434
3647    Mean Score: 82.99712560324883
EPOCH 3 2020-07-01 02:14:32.055062
EPOCH #3        History Best Score: 84.18727295125808   Best Score: 84.1872729512
5808    Mean Score: 83.62307704596165
EPOCH 4 2020-07-01 02:23:15.714755
EPOCH #4        History Best Score: 84.51718189674426   Best Score: 84.5171818967
4426    Mean Score: 83.91956847183313
EPOCH 5 2020-07-01 02:32:10.387098
EPOCH #5        History Best Score: 85.15434045767805   Best Score: 85.1543404576
7805    Mean Score: 84.49390682203803
EPOCH 6 2020-07-01 02:39:54.300298
EPOCH #6        History Best Score: 85.15434045767805   Best Score: 85.1364653374
4301    Mean Score: 84.64969968507715
EPOCH 7 2020-07-01 02:47:57.868190
EPOCH #7        History Best Score: 85.15434045767805   Best Score: 85.1364653374
4301    Mean Score: 84.76641348315849
EPOCH 8 2020-07-01 02:55:46.258508
EPOCH #8        History Best Score: 85.15434045767805   Best Score: 84.9999800312
3115    Mean Score: 82.59590753907044
EPOCH 9 2020-07-01 03:03:51.709855
EPOCH #9        History Best Score: 85.15434045767805   Best Score: 85.1071084823
3363    Mean Score: 83.71253206139446
EPOCH 10 2020-07-01 03:12:36.788216
EPOCH #10       History Best Score: 85.46829005589105   Best Score: 85.4682900558
9105    Mean Score: 83.22359684837531
EPOCH 11 2020-07-01 03:21:22.061404
EPOCH #11       History Best Score: 85.94138920574375   Best Score: 85.9413892057
4375    Mean Score: 84.66916693061282
EPOCH 12 2020-07-01 03:30:45.238220
EPOCH #12       History Best Score: 86.07340058609873   Best Score: 86.0734005860
9873    Mean Score: 85.68320021332791
EPOCH 13 2020-07-01 03:39:30.548308
EPOCH #13       History Best Score: 86.2678159051551    Best Score: 86.2678159051
551     Mean Score: 85.66990657381086
EPOCH 14 2020-07-01 03:48:43.075605
EPOCH #14       History Best Score: 86.2678159051551    Best Score: 85.9413892057
4375    Mean Score: 85.44814205966485
EPOCH 15 2020-07-01 03:57:43.559113
EPOCH #15       History Best Score: 86.2678159051551    Best Score: 86.2678159051
551     Mean Score: 86.00845342499716
EPOCH 16 2020-07-01 04:06:42.960515
EPOCH #16       History Best Score: 86.43476157136394   Best Score: 86.4347615713
6394    Mean Score: 85.86660112814329
EPOCH 17 2020-07-01 04:15:38.040474
EPOCH #17       History Best Score: 86.43476157136394   Best Score: 86.3848282824
1246    Mean Score: 86.09556223297531
EPOCH 18 2020-07-01 04:24:50.126950
EPOCH #18       History Best Score: 86.43476157136394   Best Score: 86.4159691166
0932    Mean Score: 86.23780985000995
EPOCH 19 2020-07-01 04:34:11.979308
EPOCH #19       History Best Score: 86.56454658759853   Best Score: 86.5645465875
9853    Mean Score: 85.874300225217
EPOCH 20 2020-07-01 04:42:56.650106
EPOCH #20       History Best Score: 86.56454658759853   Best Score: 86.5506016282
8414    Mean Score: 86.20631489273853
EPOCH 21 2020-07-01 04:52:08.593964
EPOCH #21       History Best Score: 86.69781016336603   Best Score: 86.6978101633
6603    Mean Score: 86.3200737124061
EPOCH 22 2020-07-01 05:00:59.651681
EPOCH #22       History Best Score: 86.75252520442643   Best Score: 86.7525252044
2643    Mean Score: 86.28045877785605
EPOCH 23 2020-07-01 05:09:43.953465
EPOCH #23       History Best Score: 86.75252520442643   Best Score: 86.6862095375
141     Mean Score: 86.10239576072256
EPOCH 24 2020-07-01 05:18:30.000583
EPOCH #24       History Best Score: 87.3277268198691l   Best Score: 87.3277268198
6911    Mean Score: 86.25749079209348
EPOCH 25 2020-07-01 05:27:16.231209
EPOCH #25       History Best Score: 87.32772681986911   Best Score: 86.7951763441
9767    Mean Score: 86.48694451109233
EPOCH 26 2020-07-01 05:36:01.294956
EPOCH #26       History Best Score: 87.32772681986911   Best Score: 87.0694673530
5982    Mean Score: 86.71206961542916
EPOCH 27 2020-07-01 05:45:30.292206
EPOCH #27       History Best Score: 87.75692171290257   Best Score: 87.7569217129
0257    Mean Score: 86.79725776577739
EPOCH 28 2020-07-01 05:54:34.931597
EPOCH #28       History Best Score: 87.78228542866648   Best Score: 87.7822854286
6648    Mean Score: 86.60812879425808
EPOCH 29 2020-07-01 06:03:37.778784
EPOCH #29       History Best Score: 88.00783390162401   Best Score: 88.0078339016
2401    Mean Score: 87.4846896684355
EPOCH 30 2020-07-01 06:12:20.226528
EPOCH #30       History Best Score: 88.18221839504554   Best Score: 88.1822183950
4554    Mean Score: 87.69841070104818
EPOCH 31 2020-07-01 06:21:13.476382
EPOCH #31       History Best Score: 88.18221839504554   Best Score: 88.1678442608
0687    Mean Score: 87.78052111113604
EPOCH 32 2020-07-01 06:30:01.978932
EPOCH #32       History Best Score: 88.30395077483432   Best Score: 88.3039507748
3432    Mean Score: 88.08075966112078
EPOCH 33 2020-07-01 06:38:58.365526
EPOCH #33       History Best Score: 88.30395077483432   Best Score: 88.3039507748
3432    Mean Score: 88.13770976039626
EPOCH 34 2020-07-01 06:47:59.995968
EPOCH #34       History Best Score: 88.412475196485     Best Score: 88.4124751964
85      Mean Score: 88.16952344993966
EPOCH 35 2020-07-01 06:57:05.062217
EPOCH #35       History Best Score: 88.412475196485     Best Score: 88.3831512130
```

```
4126    Mean Score: 88.0585822235562
EPOCH 36 2020-07-01 07:05:57.801437

---------------------------------------------------------------------------
-
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-7-31ece2cd64e8> in <module>
     15     if __name__ == '__main__':
     16         pool = multiprocessing.Pool(processes=CPU_CORE)
---> 17         genomes[idx] = pool.map(genome_score, _genomes)
     18         pool.close()
     19         pool.join()

C:\ProgramData\Anaconda3\lib\multiprocessing\pool.py in map(self, func, it
erable, chunksize)
    266         in a list that is returned.
    267         '''
--> 268         return self._map_async(func, iterable, mapstar, chunksize).get()
    269 
    270     def starmap(self, func, iterable, chunksize=None):

C:\ProgramData\Anaconda3\lib\multiprocessing\pool.py in get(self, timeout)
    649 
    650     def get(self, timeout=None):
--> 651         self.wait(timeout)
    652         if not self.ready():
    653             raise TimeoutError

C:\ProgramData\Anaconda3\lib\multiprocessing\pool.py in wait(self, timeout)
    646 
    647     def wait(self, timeout=None):
--> 648         self._event.wait(timeout)
    649 
    650     def get(self, timeout=None):

C:\ProgramData\Anaconda3\lib\threading.py in wait(self, timeout)
    550             signaled = self._flag
    551             if not signaled:
--> 552                 signaled = self._cond.wait(timeout)
    553             return signaled
    554 

C:\ProgramData\Anaconda3\lib\threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterr
upt)
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:

KeyboardInterrupt: 
```

In [8]:

```
# 재고 계산
from module.simulator import Simulator
simulator = Simulator()
order = pd.read_csv('module/order.csv')
submission = best_gen.predict(order)
_, df_stock = simulator.get_score(submission)

# PRT 개수 계산
PRTs = df_stock[['PRT_1', 'PRT_2', 'PRT_3', 'PRT_4']].values
PRTs = (PRTs[:-1] - PRTs[1:])[24*23:]
PRTs = np.ceil(PRTs * 1.1)
PAD = np.zeros((24*23+1, 4))
PRTs = np.append(PRTs, PAD, axis=0).astype(int)

# Submission 파일에 PRT 입력
submission.loc[:, 'PRT_1':'PRT_4'] = PRTs
submission.to_csv('Dacon_baseline2.csv', index=False)
```

In [ ]:

```
while n_gen <= 2000:
    print('EPOCH', n_gen, datetime.datetime.now())
    genomes = np.array(genomes)
    while len(genomes)%CPU_CORE != 0:
        genomes = np.append(genomes, Genome(score_ini, input_length, output_length_1, output_length_
    genomes = genomes.reshape((len(genomes)//CPU_CORE, CPU_CORE))

    for idx, _genomes in enumerate(genomes):
        if __name__ == '__main__':
            pool = multiprocessing.Pool(processes=CPU_CORE)
            genomes[idx] = pool.map(genome_score, _genomes)
            pool.close()
            pool.join()
    genomes = list(genomes.reshape(genomes.shape[0]*genomes.shape[1]))

     # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    # 평균 점수
    s = 0
    for i in range(N_BEST):
        s += genomes[i].score
    s /= N_BEST

    # Best Score
    bs = genomes[0].score

    # Best Model 추가
    if best_genomes is not None:
        genomes.extend(best_genomes)

    # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    score_history.append([n_gen, genomes[0].score])
    high_score_history.append([n_gen, bs])
    mean_score_history.append([n_gen, s])

    if genomes[0].score > best_score_ever:
        best_score_ever = genomes[0].score
        best_gen = genomes[0]

    # 결과 출력
    print('EPOCH #%s\tHistory Best Score: %s\tBest Score: %s\tMean Score: %s' % (n_gen, genomes[0].s

    # 모델 업데이트
    best_genomes = deepcopy(genomes[:N_BEST])

    # CHILDREN 생성
    for i in range(N_CHILDREN):
        new_genome = deepcopy(best_genomes[0])
        a_genome = np.random.choice(best_genomes)
        b_genome = np.random.choice(best_genomes)

        for j in range(input_length):
            cut = np.random.randint(new_genome.w1.shape[1])
            new_genome.w1[j, :cut] = a_genome.w1[j, :cut]
            new_genome.w1[j, cut:] = b_genome.w1[j, cut:]
        for j in range(h1):
            cut = np.random.randint(new_genome.w2.shape[1])
            new_genome.w2[j, :cut] = a_genome.w2[j, :cut]
            new_genome.w2[j, cut:] = b_genome.w2[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w3.shape[1])
            new_genome.w3[j, :cut] = a_genome.w3[j, :cut]
            new_genome.w3[j, cut:] = b_genome.w3[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w4.shape[1])
            new_genome.w4[j, :cut] = a_genome.w4[j, :cut]
            new_genome.w4[j, cut:] = b_genome.w4[j, cut:]

        for j in range(input_length):
            cut = np.random.randint(new_genome.w5.shape[1])
            new_genome.w5[j, :cut] = a_genome.w5[j, :cut]
            new_genome.w5[j, cut:] = b_genome.w5[j, cut:]

        for j in range(h1):
            cut = np.random.randint(new_genome.w6.shape[1])
            new_genome.w6[j, :cut] = a_genome.w6[j, :cut]
            new_genome.w6[j, cut:] = b_genome.w6[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w7.shape[1])
            new_genome.w7[j, :cut] = a_genome.w7[j, :cut]
            new_genome.w7[j, cut:] = b_genome.w7[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w8.shape[1])
            new_genome.w8[j, :cut] = a_genome.w8[j, :cut]
            new_genome.w8[j, cut:] = b_genome.w8[j, cut:]

        best_genomes.append(new_genome)

    # 모델 초기화
    genomes = []
    for i in range(int(N_POPULATION / len(best_genomes))):
        for bg in best_genomes:
            new_genome = deepcopy(bg)
            mean = 0
            stddev = 0.2
            # 50% 확률로 모델 변형
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w1 += new_genome.w1 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w2 += new_genome.w2 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w3 += new_genome.w3 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w4 += new_genome.w4 * np.random.normal(mean, stddev, size=(h3, output_le
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w5 += new_genome.w5 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w6 += new_genome.w6 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w7 += new_genome.w7 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w8 += new_genome.w8 * np.random.normal(mean, stddev, size=(h3, output_le
            genomes.append(new_genome)

    if REVERSE:
        if bs < score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_ler
    else:
        if bs > score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_ler

    n_gen += 1
```

6. 결과 및 결언

Conclusion & Discussion

결과 그래프

In [ ]:

```
import matplotlib.pyplot as plt

# Score Graph
score_history = np.array(score_history)
high_score_history = np.array(high_score_history)
mean_score_history = np.array(mean_score_history)

plt.plot(score_history[:,0], score_history[:,1], '-o', label='BEST')
plt.plot(high_score_history[:,0], high_score_history[:,1], '-o', label='High')
plt.plot(mean_score_history[:,0], mean_score_history[:,1], '-o', label='Mean')
plt.legend()
plt.xlim(0, EPOCHS)
plt.ylim(bottom=0)
plt.xlabel('Epochs')
plt.ylabel('Score')
plt.show()
```

Submission 파일 만들기

In [ ]:

```
# 재고 계산
from module.simulator import Simulator
simulator = Simulator()
order = pd.read_csv('module/order.csv')
submission = best_gen.predict(order)
_, df_stock = simulator.get_score(submission)

# PRT 개수 계산
PRTs = df_stock[['PRT_1', 'PRT_2', 'PRT_3', 'PRT_4']].values
PRTs = (PRTs[:-1] - PRTs[1:])[24*23:]
PRTs = np.ceil(PRTs * 1.1)
PAD = np.zeros((24*23+1, 4))
PRTs = np.append(PRTs, PAD, axis=0).astype(int)

# Submission 파일에 PRT 입력
submission.loc[:, 'PRT_1':'PRT_4'] = PRTs
submission.to_csv('Dacon_baseline2.csv', index=False)
```

점수 향상 팁

해당 코드는 단순한 모델로 다음 방법으로 점수 향상을 꾀할 수 있습니다.

1. 성형 공정 2개 라인을 따로 모델링
2. CHANGE, STOP 이벤트 활용
3. 수요 초과분 외 다양한 양상을 반영하는 목적함수
4. 유전 알고리즘 외 효율적인 학습 기법