[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
warnings.filterwarnings(action='ignore')
np.random.seed(777)
```

In [2]:

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

```
Pandas : 1.0.4
Numpy : 1.18.5

Python 3.6.10 :: Anaconda, Inc.
```

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 [3]:

```
CPU_CORE = multiprocessing.cpu_count() # 멀티프로세싱 CPU 사용 수
N_POPULATION = 100                     # 세대당 생성수
N_BEST = 10                            # 베스트 수
N_CHILDREN = 10                        # 자손 유전자 수
PROB_MUTATION = 0.4                    # 돌연변이
REVERSE = False                        # 배열 순서 (False: ascending order, True: descending order)

score_ini = 1e8                        # 초기 점수
input_length = 125                     # 입력 데이터 길이
output_length_1 = 5                    # Event (CHECK_1~4, PROCESS)
output_length_2 = 12                   # MOL(0~5.5, step:0.5)
h1 = 50                                # 히든레이어1 노드 수
h2 = 50                                # 히든레이어2 노드 수
h3 = 50                                # 히든레이어3 노드 수
EPOCHS = 50                            # 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(5):
        genome = Genome(score_ini, input_length, output_length_1, output_length_2, h1, h2, h3)
        best_genomes.append(genome)
```

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 [4]:

```
n_gen = 1
score_history = []
high_score_history = []
mean_score_history = []
while n_gen <= EPOCHS:
    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])

    # 결과 출력
    print('EPOCH #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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
            # 40% 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
```

```
EPOCH #1        History Best Score: 18348048.0  Best Score: 18348048.0  Mean Score:
22550315.0
EPOCH #2        History Best Score: 18348048.0  Best Score: 18770788.0  Mean Score:
20155474.0
EPOCH #3        History Best Score: 18348048.0  Best Score: 18348048.0  Mean Score:
20740431.8
EPOCH #4        History Best Score: 17193009.0  Best Score: 17193009.0  Mean Score:
20307857.0
EPOCH #5        History Best Score: 17193009.0  Best Score: 17858066.0  Mean Score:
18920058.2
EPOCH #6        History Best Score: 17193009.0  Best Score: 17918297.0  Mean Score:
19030970.3
EPOCH #7        History Best Score: 17164404.0  Best Score: 17164404.0  Mean Score:
19007307.4
EPOCH #8        History Best Score: 16675184.0  Best Score: 16675184.0  Mean Score:
19171448.2
EPOCH #9        History Best Score: 16320747.0  Best Score: 16320747.0  Mean Score:
17184760.3
EPOCH #10       History Best Score: 16320747.0  Best Score: 16423436.0  Mean Score:
17314959.9
EPOCH #11       History Best Score: 16320747.0  Best Score: 16347001.0  Mean Score:
17319843.4
EPOCH #12       History Best Score: 16320747.0  Best Score: 16360330.0  Mean Score:
17133919.7
EPOCH #13       History Best Score: 16311787.0  Best Score: 16311787.0  Mean Score:
17430764.6
EPOCH #14       History Best Score: 16311787.0  Best Score: 16320747.0  Mean Score:
16841586.7
EPOCH #15       History Best Score: 16311787.0  Best Score: 16703683.0  Mean Score:
17644549.7
EPOCH #16       History Best Score: 16311787.0  Best Score: 16607859.0  Mean Score:
17640134.7
EPOCH #17       History Best Score: 16311787.0  Best Score: 16887762.0  Mean Score:
18041121.9
EPOCH #18       History Best Score: 16311787.0  Best Score: 16311787.0  Mean Score:
17142910.3
EPOCH #19       History Best Score: 15788756.0  Best Score: 15788756.0  Mean Score:
17168003.3
EPOCH #20       History Best Score: 15788756.0  Best Score: 16179745.0  Mean Score:
17126660.9
EPOCH #21       History Best Score: 15788756.0  Best Score: 15817042.0  Mean Score:
17467492.4
EPOCH #22       History Best Score: 15788756.0  Best Score: 16216225.0  Mean Score:
17410340.4
EPOCH #23       History Best Score: 15788756.0  Best Score: 16580718.0  Mean Score:
17686202.0
EPOCH #24       History Best Score: 15788756.0  Best Score: 16059050.0  Mean Score:
17159110.0
EPOCH #25       History Best Score: 15788756.0  Best Score: 16563710.0  Mean Score:
```

```
17172376.4
EPOCH #26       History Best Score: 15788756.0  Best Score: 17079010.0  Mean Score:
17956957.6
EPOCH #27       History Best Score: 15788756.0  Best Score: 16270476.0  Mean Score:
17453260.6
EPOCH #28       History Best Score: 15545866.0  Best Score: 15545866.0  Mean Score:
17392707.1
EPOCH #29       History Best Score: 15545866.0  Best Score: 16509061.0  Mean Score:
16979788.6
EPOCH #30       History Best Score: 15545866.0  Best Score: 16434526.0  Mean Score:
18030475.1
EPOCH #31       History Best Score: 15545866.0  Best Score: 16786005.0  Mean Score:
18375700.3
EPOCH #32       History Best Score: 15545866.0  Best Score: 15545866.0  Mean Score:
16773249.2
EPOCH #33       History Best Score: 15545866.0  Best Score: 15587961.0  Mean Score:
16937171.4
EPOCH #34       History Best Score: 15545866.0  Best Score: 15545866.0  Mean Score:
16744946.3
EPOCH #35       History Best Score: 15545866.0  Best Score: 15629521.0  Mean Score:
17377796.5
EPOCH #36       History Best Score: 15545866.0  Best Score: 15735999.0  Mean Score:
16455824.0
EPOCH #37       History Best Score: 15545866.0  Best Score: 16379453.0  Mean Score:
17088621.0
EPOCH #38       History Best Score: 15545866.0  Best Score: 15545866.0  Mean Score:
17271248.2
EPOCH #39       History Best Score: 15508522.0  Best Score: 15508522.0  Mean Score:
16250665.3
EPOCH #40       History Best Score: 14893161.0  Best Score: 14893161.0  Mean Score:
15849511.6
EPOCH #41       History Best Score: 14893161.0  Best Score: 14893161.0  Mean Score:
15742069.9
EPOCH #42       History Best Score: 14893161.0  Best Score: 14995430.0  Mean Score:
16281590.9
EPOCH #43       History Best Score: 14847234.0  Best Score: 14847234.0  Mean Score:
15847288.2
EPOCH #44       History Best Score: 14847234.0  Best Score: 14847234.0  Mean Score:
15966725.0
EPOCH #45       History Best Score: 14847234.0  Best Score: 15509869.0  Mean Score:
16104815.6
EPOCH #46       History Best Score: 14847234.0  Best Score: 15560712.0  Mean Score:
16163948.9
EPOCH #47       History Best Score: 14527759.0  Best Score: 14527759.0  Mean Score:
15693807.9
EPOCH #48       History Best Score: 14527759.0  Best Score: 14996424.0  Mean Score:
16209651.5
EPOCH #49       History Best Score: 14527759.0  Best Score: 14527759.0  Mean Score:
15865203.3
EPOCH #50       History Best Score: 14527759.0  Best Score: 14630090.0  Mean Score:
15196706.0
```

6. 결과 및 결언

Conclusion & Discussion

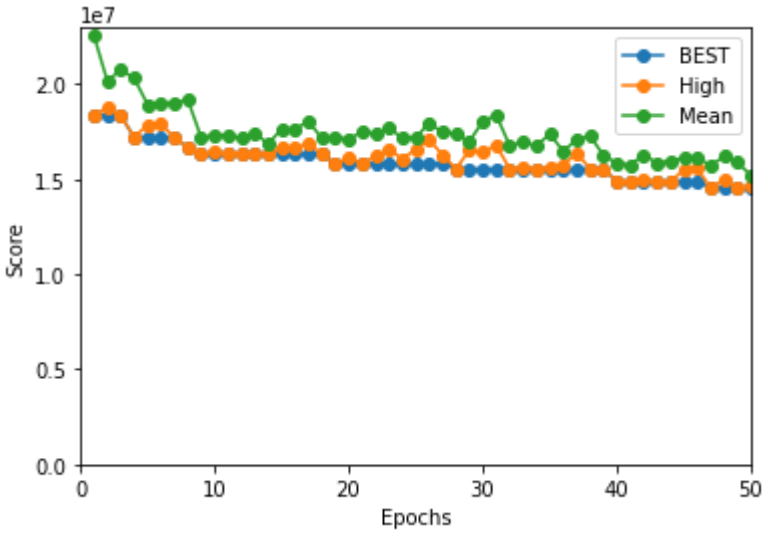
결과 그래프

In [5]:

```
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 [6]:

```
# 재고 계산
from module.simulator import Simulator
simulator = Simulator()
order = pd.read_csv('module/order.csv')
submission = best_genomes[0].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_baseline.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

In [ ]: