Course Work 2

201208888 조민기

이 프로젝트는 fluccs_data 라는 폴더 내에 제공된 데이터가 모두 들어있는 환경을 가정하고 있습니다.

1. 프로그램 실행 및 설명

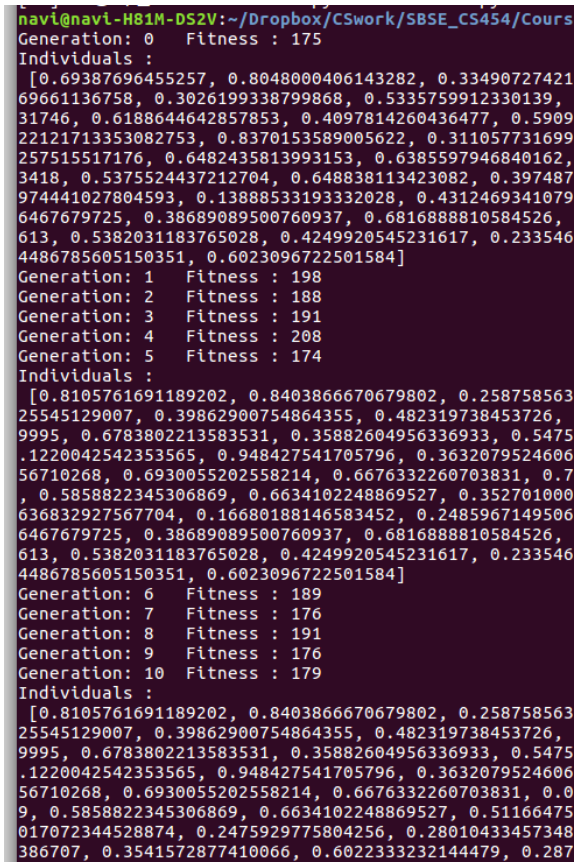
python train.py 를 실행하시면 'fluccs_data' 폴더 안의 모든 데이터를 자동으로 사용하여 트레이닝 프로세스를 실행하게 됩니다. 다만 이는 아래에 서술한 마지막 과정만을 수행하므로 어떤 SEED 가 선택되느냐에 따라 성능이 달라질 수 있다.

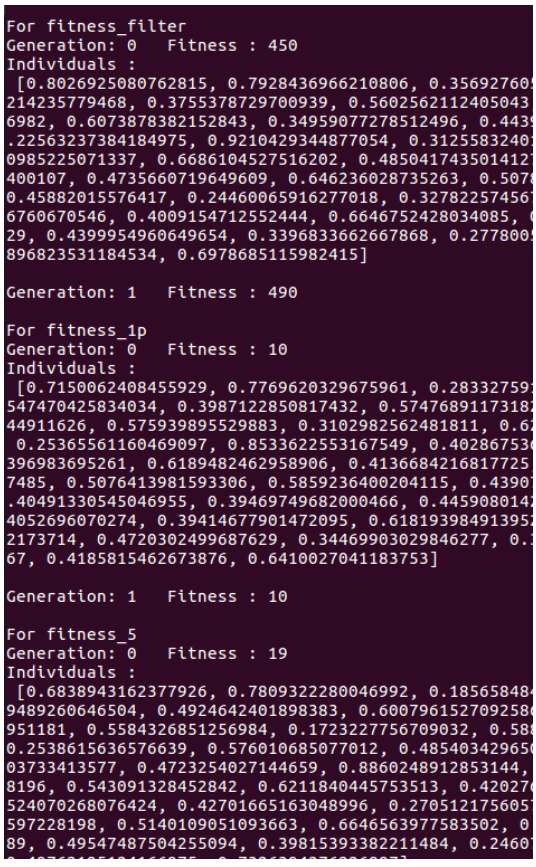
python validate.py Lang_1.csv Lang_2.csv … 를 실행하시면 추가하신 파일들에 대해서 평가를 하게 됩니다. python validate.py 를 실행하시면 fluccs_data 내의 모든 파일들에 대해서 랭킹을 진행하게 됩니다.

python validation.py 를 하시면 fluccs_data 내의 모든 파일들에 대해서 아래에 서술한 각 과정마다 dna 를 평가하게 됩니다.

Python auto_train.py 를 실행하시면 아래에 서술한 모든 과정을 각각 순차적으로 진행하게 됩니다. fluccs POP_SIZE 를 통해 인구 수를 조절할 수 있고, GENERATIONS 를 통해 총 진화 횟수를 조절할 수 있다.

오른쪽은 train.py 를 실행했을 때, 왼쪽은 auto_train.py 를 실행했을 때이다.





2. 프로젝트 상세 설명

이번 프로젝트는 주어진 여러가지 multi-objective fitness 를 평가하는 함수들과 추가 정보들을 토대로 Fault 를 포함하는 라인이 상위에 나타나도록 랭킹을 하는 것이다. 본인은 우선 이들을 이용하여 평가하는 상황을 최대한 단순화 하여 각 함수 및 정보 항목들에 대해 importance 를 주고 이를 이용해서 weighted sum 을 하는 방식을 택하였다. 가령 ochiai, jaccard, gp13 이 [0.5, 0.7, 0.2]로 주어졌을 때, 이 중 어느 정보의 값을 중요하게 여길지 가중치를 정하고, 그 가중치들의 값이 [0.8, 0.1, 0.9] 로 주어졌다면 이들의 weighted sum 인 0.4 + 0.07 + 0.18 = 0.65 값을 계산한다. 이렇게 계산된 값을 토대로 랭킹을 매겨서 에러 라인을 찾을 것이다.

현제 프로그램은 과제에서 주어진 데이터 형식 - 첫번째 column 이 라인명이고, 그 이후 41 개의 라인이 각각의 데이터 값이며 마지막 1 개의 라인이 폴트 여부를 나타내는 라인인 포멧의 csv 데이터를 받아서 처리하는 시스템이다. 이 라인의 길이는 LINE_SIZE 값을 통해 조절할 수 있다.
유전 알고리즘의 인구는 POP_SIZE 변수를 통해 조절할 수 있으며 초기 값은 20 으로 설정해두었다. 진화 세대 수는 GENERATIONS 변수를 통해 조절할 수가 있으며 초기 값은 60 세대로 설정해두었다.

다음은 유전알고리즘 전체에 관한 설명이다.

2.1 CrossOver

crossover : dna dna → dna dna
DNA 2 개를 받고 랜덤한 위치에서 두 유전자를 잘라 서로 교차하는 방식이다. 가령 DNA [1,2,3,4,5,6,7,8,9] 와 DNA [4,5,6,7,8,9,1,2,3] 에서 2 의 위치에서 교차가 일어난다면 각 DNA 는 [1,2,6,7,8,9,1,2,3], [4,5,3,4,5,6,7,8,9] 가 될 것이다.

2.2 Mutation

mutate : dna → dna
DNA 중 랜덤한 자리 하나를 랜덤변수로 교체함으로써 DNA [0.5, 0.3, 0.2]을 입력했을 때 랜덤위치 1 와 랜덤한 숫자 0.8 이 발생했을 경우 아웃풋이 [0.5, 0.8, 0.2] 가 출력이 된다. 현재 다루는 DNA 는 길이가 40 이 넘으며 이중 하나의 변수를 바꾸는 과정이기 때문에 매번 mutation 이 일어나도록 한다.

2.3 Selection

weighted_choice : population → dna
각 DNA 마다 fitness value 를 두고 이를 역수를 취한 값들과 함께 population 들을 입력해준다. 이후 각 DNA 마다 입력받은 fitness 역수값들을 기준으로 이 값이 큰 값들(최종적으로는 fitness value 가 작은 값들)을 우선적으로 뽑도록 한다.

2.4 Genetic algorithm

GA_process : dna integer → population

초기 값으로 삼을 dna seed 와 진화 모드를 설정한 뒤 seed 를 가지고 랜덤 파퓰레이션을 발생시키고, 위에서 언급한 selection 을 이용하여 엘리트들을 선택, 과거 인구의 엘리트 2 개를 포함하여 교차, 돌연변이 과정을 통해 인구 수 만큼의 유전자를 생산해낸다. 이렇게 입력받은 모드에 따라 진화 알고리즘을 진행하여 마지막 파퓰레이션을 출력한다.

3. Approach

유전 알고리즘에 있어서 초기 세대의 설정, 즉 SEED 의 설정은 이후 성능을 크게 좌우하는 요인이 된다. 따라서 여러가지 환경에서 유전 알고리즘을 수행하며 SEED 를 찾는 작업을 수행하도록 설계하였다. 따라서 프로젝트는 다음 4 가지 단계로 나뉘어 진행된다. 아래의 네 단계를 한꺼번에 진행하기 위해선 python auto_train.py 를 실행하면 된다.

3.1 Filter

첫번째 SEED 를 가공하는 작업은 단순하게 에러를 발생시키는 Fault 라인의 평가 값이 크고, 에러가 없는 정상 라인의 평가 값이 작게 나오는 DNA 를 선발하는 과정을 구현하였다. 즉 분류를 잘 해내지 못한 가중치들에 패널티를 매기는 과정이다. 구체적으로는 Fault 라인이 들어오면 라인의 평가 값이 20 을 넘게, 정상 라인들은 10 을 넘지 못하게 조절을 하였다.

초기 랜덤변수로 발생시킨 DNA 들을 이용하여 계산한 평가 값을 보며, Fault line 의 경우 이 값이 20 을 넘지 못하면 패널티 10 을 주고, 정상 라인이 이 값을 10 을 넘긴다면 패널티 1 을 주며 이 패널티의 총합을 fitness 로 사용하였다. 패널티의 크기에 차이가 나는 이유는 대부분의 데이터가 매우 극소수의 Fault line 을 포함하고 있기 때문에, Fault 라인의 평가 값이 크게 나타나는 현상을 좀 더 강조하여 포착하도록 만들기 위함이다. 설명한 fitness 함수가 ‘fitness_filter’ 에 저장되어 있으며, 이 과정을 통해 생성된 SEED 가 SEED_filter 로 저장되어있다. 이 과정을 실행하기 위해선 아래의 fitness 함수를 주석처리하고 이 함수의 이름에서 ‘_filter’를 지워 fitness 로 바꾸면 된다.

3.2 Top 1% selection

위의 과정으로 대략적으로 weight 가 정리된 SEED_filter 를 얻을 수 있었다. 이 SEED_filter 를 토대로 랜덤 파퓰레이션을 지정한 뒤 다시 유전 알고리즘을 구동한다. 사실 위의 과정은 크게 랭킹에 효율이 없는 것이, 위의 평가 값들이 같아도 fault 가 있을 수도 있고 없을 수도 있기 때문에 폴트 라인과 정상 라인을

명확히 구분하는 marginal space 를 찾는 다는 것은 불가능하다고 볼 수 있다. 다만 폴트를 포함하고 있을 것으로 의심이 되는 라인을 찾을 수 있을 뿐이다. 따라서 랭킹을 매겨야 한다.
파트를 평가하는데 필요한 데이터 량이 크고, 파이썬을 이용하는 이유로 시스템을 직접 컨트롤 하지 못해서 C++ 등의 언어에 비해 비교적 느린 등의 이슈로 인해 여기서부터는 랭킹을 매기는 과정도 데이터 포션을 나누어 진행하였다. 진화에서 사용하는 fitness 함수의 경우 전체 모든 파일을 사용하면 너무 진행이 느려 결과를 확인할 수가 없어서 파일의 일부를 이용해 fit 를 평가하였다. 다만 cross validation 형식으로 교차진행을 한다면 전체 데이터가 고르게 진화에 사용되는 것은 맞으나, 순차적이고 규칙적으로 데이터 묶음이 사용되는 탓에 DNA 의 평가가 편향되는 현상을 겪는다. 해서 fitness 에 사용하는 데이터는 전체 150 여개의 데이터중 20%를 매번 랜덤하게 선택하여 평가하도록 했다.

첫 번째 과정은 위의 SEED_filter 를 이용한 랜덤 pop 으로부터 시작해서 폴트를 포함하고 있는 라인을 상위 1 퍼센트에 랭킹 시킬 수 있는지 여부를 통해 패널티를 매기는 피트니스를 설정하였다. 즉 fault 를 포함한 라인이 상위 1 퍼센트에 포함되지 못한다면 패널티를 매기며, 패널티를 낮게 만드는 유전자를 다음 세대로 넘기는 fitness 이다. 이 과정을 통해 만들어진 것이 SEED_rank 이다. 테스트해본 결과 이렇게 얻어진 SEED_rank 를 이용하면 모든 데이터에 대해서 Fault 라인이 상위 1 퍼센트 안에 랭크되는 것을 확인할 수 있었다. 이 함수는 fitness_1p 에 저장되어 있다.

3.3 Top 5 selection

두번째 과정은 이렇게 얻어진 SEED_rank 로부터 랜덤파퓰레이션을 생성하고, 폴트 라인을 상위 5 위 안에 랭크시키는 DNA 를 찾는 것이다. 각 평가에는 전체 데이터의 20%를 랜덤으로 선택해서 사용하였고 이 함수는 fitness_5 에 구현되어 있다. 이렇게 얻어진 DNA 를 SEED_5 에 저장하였다.

3.4 Ranking

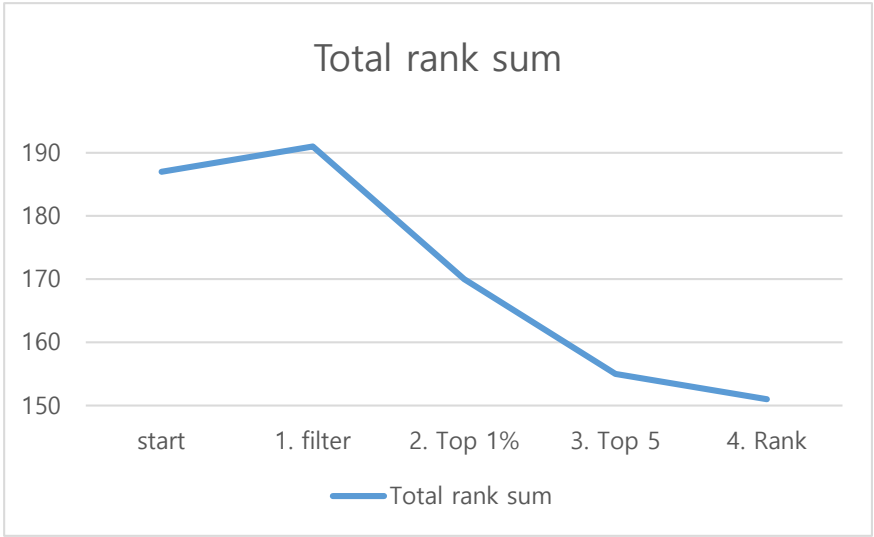
마지막 과정으로 등수를 매기는 과정이다. 이 프로젝트의 궁극적인 목표는 fault 를 포함하고 있는 라인이 높은 랭크에 포함되는 것이다. 따라서 fault 를 포함한 라인이 높게 랭크되면 패널티를 적게 주는 피트니스 함수를 통해 진화한다. fault line 이 1 등을 하면 패널티 1 을, 2 등을 하면 패널티 2 를 주는 식으로 등수 만큼의 패널티를 매기고, fault line 임에도 랭킹이 4 등 바깥으로 넘어간다면 공통적으로 2 의 패널티를 추가적으로 주었다.

```
File: Math_22.csv        Rank: 0
File: Math_3.csv         Rank: 0
File: Math_8.csv         Rank: 0
File: Lang_44.csv        Rank: 0
File: Math_106.csv       Rank: 0
File: Math_50.csv        Rank: 0
File: Math_43.csv        Rank: 2
File: Lang_40.csv        Rank: 0
File: Lang_58.csv        Rank: 0
File: Math_25.csv        Rank: 0
File: Math_75.csv        Rank: 1
File: Math_92.csv        Rank: 3
File: Math_94.csv        Rank: 0
File: Lang_61.csv        Rank: 0
File: Lang_24.csv        Rank: 1
File: Lang_43.csv        Rank: 1
File: Math_15.csv        Rank: 0
File: Lang_25.csv        Rank: 0
File: Lang_28.csv        Rank: 0
File: Math_35.csv        Rank: 0
File: Math_26.csv        Rank: 0
File: Lang_64.csv        Rank: 1
File: Math_28.csv        Rank: 4
Total number of file :   156
Total rank sum :         151
Num of rank under 3:     113
>>>
```

```
navi@navi-H81M-DS2V:~/Dropbo
For DNA SEED_rand
Total number of file :   156
Total rank sum :         187
Num of rank under 3:     105

For DNA SEED_filter
Total number of file :   156
Total rank sum :         191
Num of rank under 3:     110

For DNA SEED_rank
Total number of file :   156
Total rank sum :         170
Num of rank under 3:     107

For DNA SEED_5
Total number of file :   156
Total rank sum :         155
Num of rank under 3:     106

For DNA SEED
Total number of file :   156
Total rank sum :         151
Num of rank under 3:     113
```

4. 실험

왼쪽이 최종 DNA 를 이용하여 모든 파일에 대해 랭킹을 매기고 그 성능을 평가한 것이고, 오른쪽은 각 단계마다 나온 DNA 를 이용하여 모든 파일들을 랭킹을 매기고 DNA 를 평가한 것이다. (실험을 실행하기 위해선 python validate.py 를 실행하시면 fluccs_data 폴더 내의 모든 파일에 대해서 평가를 합니다. 또한 최종 과정을 통해 얻어낸 DNA 는 validate.py 내부에 dna 에 들어있으며, 이 값을 통해 다른 dna 들을 이용해 랭킹을 매길 수 있습니다.) (오른쪽의 사진처럼 각 단계별 DNA 평가를 위해선 python validation.py 를 실행하시면 됩니다.) 위에서 서술한 모든 과정을 주어진 파일의 수가 156 개이고 총 랭크들의 합이 151, fault 를 포함한 라인의 랭크가 3 이하로 기록된 파일의 수가 156 개중 113 개이다. 또한 DNA 가 가공될 수록 성능이 나아지는 것을 확인할 수 있었는데, 위에서 언급한 처리의 순서대로 엘리트 DNA 를 각각 보관하여 비교해본 결과 다음과 같이 다른 형태롤 진화를 할 수록 점점 성능이 좋아지는 것을 확인할 수 있다. Random DNA 의 경우 운에 따라 가끔씩 total rank sum 이 170 까지 내려가기도 하지만 일반적으로는 다소 높은 값을 보인다. 이후 올바른 filtering 을 기준으로 진화한 DNA 에 대해서는 total rank sum 이 191 로 다소 나아지고, 점점 처리를 할 수록 DNA 가 fault 를 포함한 라인을 좀 더 높은 랭크에 올려 놓고, 3 위 안쪽으로 랭크하는 수도 늘어나는 것을 확인할 수가 있다. (최종적으로 진화를 시킨 것이 SEED 이다.)



5. 결론

설계한 모델이 잘 작동하여 각 단계별로 이전 단계에서 얻은 DNA 보다 좋은 성능을 발휘하는 DNA 들을 얻을 수 있었다. 최종적으로 RANK 를 고려한 진화모델을 통해 얻은 폴트를 포함한 라인을 3 등안에 넣는데 성공한 파일이 156 개 중에서 113 개로 꽤나 괜찮은 성능을 보이는 것을 확인할 수 있었다.

다만 개선점으로는, 뒤늦게 실험을 하면서 깨닫게 된 부분이지만 제공받은 데이터가 너무나 잘 정제되어 있어서 아무 것이나 기준으로 잡고 Naive 하게 정렬을 해도 상당히 괜찮은 형태로 정렬이 되는 경향이 있다는 것을 알게 되었다. 본인이 설계한 프로젝트의 경우, 제공된 데이터 가운데 평가에 방해되는 요소가 되는 factor 가 포함되어 있는 등 좀 더 랜덤한 데이터에 대해서는 상당히 괜찮은 성능을 발휘했을 것이지만, 본 데이터에 관해서는 그렇게까지 놀라운 성능을 발휘하지는 못하는 것 같다. 즉 이미 상당히 좋은 형태로 정제된 데이터라서, 솔루션 모델이 선형 그저 단순한 결합인 형태로는 크게 놀라운 성능 개선을 기대하기 어렵다고 생각된다.
가령 변수를 가지고 ochiai 와 같은 복잡한 수식으로 구성된 평가 메트릭을 생성하는 방법 혹은 커널을 사용하는 방법을 알았더라면, 솔루션 스페이스를 단순한 선형 결합이 아닌 좀더 복잡한 모델을 사용하여 성능 개선을 할 수 있었을 것이라 생각된다. 혹은 머신러닝 툴을 사용해 비선형 블랙박스 함수를 구해내는 시도를 통해 좀 더 좋은 성능을 낼 수 있었을 것이라 생각된다.