SymbolicRegression (симболичка регресија)

1.Увод

Ово је пројекат из симболичке регресије у коме смо се потрудили да од нуле саградимо пројекат и да имамо тестерски програм на коме можемо да видимо перформансе.

Овај пројекат је рађен на рачунару следећих конфигурација:

OS	System type	RAM	CPU	Graphic card	Speed:
Windows10	64-bit OS, x64-based processor	16 GB	AMD Ryzen 5 5600H	NVIDIA GrForce RTX 3050	3.30 GHz

2.Main.py

Ово је класа одакле покрећемо пројекат, ту можемо да правимо неки насумичан улаз помоћу неке наше функције коју треба да апроксимира нап симболички регресор.

3.SymbolicRegression.py

Ово је фолдер у којима се налазе две класе са којима суштински радимо пројекат и једна помоћна класа. Овде декларишемо да све исписујемо у `output.txt` фајл.

3.1. Class Node

Класа помоћу којих можемо да правимо симболичка дрва. Нама су генерално симболичка дрва скуп чворова које могу бити следећег типа (`class Type(Enum)`):

1. `FIRST` - то нам је dummy чвор који служи да показује на први елемент у стаблу.
2. `TERM` - то нам је чвор који представља променљиву или број.
3. `OPERATOR` - то нам је чвор који представља бинарну операцију(+,*,-,%).
4. `TRIGONOMETRY` - то нам је чвор који представља косинусну/синусну операцију.

Класа се конструише на следећи начин: `__init__(self,type,level,char = None,value=None)`.

Методе које имамо су следеће у овој класи:

1. `getSubTreeFromPath(self,path,GP)` - рекурзивна метода која узима путању(`path`) и враћа чвор који је резултат путање од почетног чвора стабла.
2. `putSubTree(self,path,node)` - рекурзивна метода која ставља на путању(`path`) дати чвор(`node`).
3. `getRandomPath(self,GP)` - метода која генерише насумичну путању која се прави до максималне могућности дубљине стабла. Та путања нам се чува у `GP.localPath`.
4. `setChild1(self,node), setChild2(self,node)` - ставља на лево или десно дете нови чвор.
5. `stringNode(self)` - враћа нам ниску која представља наш математички израз.

6. `mutateInPath(self,path,GP,nodeNumForMutation)` - метода која у односу на нашу случајну путању(`path`) мења елемент у стаблу тако да прави на његовом месту насумично подстабло које је дубине `nodeNumForMutation`.
7. `getDepthOfNode(self)` - враћа дубину стабла/подстабла.
8. `getValue(self,xValue)` - враћа вредност стабла ако је променљива x једнако `xValue`.

3.2. Class GP(short for Genetic Programming)

Класа у којима се чува популација симболичких дрвета, где се налазе алгоритми конструисања насумичних дрвета, мутација, укрштања, као и сам *еволутивни алгоритам*.

Класа се контруише на следећи начин:
`__init__(self,goals,POPULATION_NUMBER,ITERATION_NUMBER,TOURNAMENT_SIZE,ELITISM_SIZE,MUTATION_RATE)`. Где је:

- `goals` - низ елемената која имају x вредност и f(x) вредност функције коју требамо да апроксимирамо.
- `POPULATION_NUMBER` - број популације.
- `ITERATION_NUMBER` - број итерације.
- `TOURNAMENT_SIZE` - број турнирске селекције.
- `ELITISM_SIZE` - број најбољих елемената која преживљавају следећу генерацију.
- `MUTATION_RATE` - вероватноћа са којом се ради мутација или укрштање.

Методе које се налазе:

1. `GP(self)` - метода која нам служи да узме нашу популацију `population` и да је кроз генерације мења тако да што више може да служи као апроксиматор функције.
2. `tournamentSelection(self)` - метода турнирске селекције, враћа индекс из тренутне генерације за укрштање/мутацију следећег хромозона.
3. `createRandomPopulation(self)` - у атрибуту `population` ствара насумичну популацију.
4. `betterMutation(self,index,GP)` - мутира `index`-ти елеменат популације.
5. `betterCrossover(self,index1,index2)` - метода која узима два хромозона, налази им насумичну путању `localPath` и онда им размењује елементе.
6. `calculateFitness(self,index)` - рачунамо фитнес нашег хромозона на следећи начин:

$$Fitness = \sum_{i=1}^{TrainingSet} |aproximateValue_i - realValue_i|$$

7. `generateRandomNode(self, depth, xValue,isParentTrig = False)` - генеришемо насумичан чвор у једном хромозону по томе која је његова дубина у којом се налази `depth`.
8. `generateSubTree(self,currentNode,depth,nodeNum,xValue,areWeGenerateFullTree = False)` - рекурзивна метода која генерише ново стабло/подстабло. `currentNode` нам означава чвор у коме се метода тренутно налази. `depth` је дубина до које смо генерисали дрво. `nodeNum` помоћни број за генерисање операција. `areWeGenerateFullTree` је индикатор којем наглашавамо јер правимо ново стабло(морамо да генеришемо `FIRST` тип чвора) или не.

Имамо и помоћну класу `DepthOfNode`, она сам служи да бисмо могли добро рекурзивно да имамо појам о томе на којој смо дубини стабла.

4.gplearn_test.py

Ово је пајтон програм који служи за упоређивање резултата нашег пројекта. Овде користимо `sklearn` и `gplearn` библиотеке које нам служе да правимо симболички регресор. Укратко:

```
import numpy as np #Biblioteka sa rad sa matematickim problemima
import pandas as pd #Biblioteka za pravljenje DataFrame-ova
import matplotlib.pyplot as plt #Biblioteka ya graficki prikaz podataka

from gplearn.genetic import SymbolicRegressor #Simbolicki regresor
from sklearn.model_selection import train_test_split #metoda za deljenje ulaznih
podataka na trening i test skup
from sympy import * #biblioteka za dobro formatiranje teksta.
```

5. Резултати

Сви извештаји од `SymbolicRegression` се налазе у `output.txt` фајлу када год се покрене пројекат. У суштини, најбољи fitness у просеку за око 30 генерације иде до око 600-800.