Rezolvarea problemelor cu ajutorul metodelor de învățare

Obiective

Dezvoltarea sistemelor care învaţă singure. Algoritmi de învăţare. Specificarea, proiectarea şi implementarea sistemelor care învaţă singure cum să rezolve probleme de regresie.

Aspecte teoretice

Proiectarea şi dezvoltarea sistemelor care învaţă singure.
Algoritmi de învăţare de tipul:
- metoda celor mai mici pătrate
- *stocastic gradient descent*
- algoritmi evolutivi

Probleme abordate

1. Scurta prezentare a problemei
    a. ce se da (input X, output Y, un input xnou), ce se cere (functia care transforma X in Y: f(X) = Y, astfel incat sa poata fi calculat ynou=f(xnou))
    b. ce poate fi X ? -->
        i. o lista de valori numerice (regresie simpla) X = (x1), x1 = x11, x21, ..., xn1), unde n e nr de exemple de antrenare),
        ii. vector cu mai multe dimensiuni de valori numerice (regresie multipla): daca avem 2 dimensiuni: X = (x1, x2), x1 = (x11, x21, ..., xn1), x2=(x12, x22, x32, ..., xn2), unde n e nr de exemple de antrenare
    c. ce poate fi Y? -->
        i. o lista de valori numerice (pt un exemplu, trebuie prezis un singur output), Y = (y1), y1 = y11, y21, ..., yn1), unde n e nr de exemple de antrenare),
        ii. vector cu mai multe dimensiuni de valori numerice: daca avem 3 dimensiuni: Y = (y1, y2, y3), y1 = (y11, y21, ..., yn1), y2=(y12, y22, y32, ..., yn2), y3 = (y13, y23, ..., yn3), unde n e nr de exemple de antrenare (pt un exemplu, trebuie prezise mai multe (3) output-uri)
2. Metode de identificare a functiei f pt cazul in care f este functie liniara, $X = (x_i)_{i=1,n}$, $x_i = (x_{i,1})$ - un exemplu are un singur atribut, $Y = (yi)_{i=1,n}$, $y_i=(y_{i,1})$ - un exemplu are un singur output numeric
    a. metoda celor mai mici patrate (least square root)
    b. gradient descent
    c. algoritmi evolutivi
3. Exemplu de problema

    Enunt

    Se cunosc următoarele n (n = 5) informații aferente unei anumite perioade de timp: numărul de ore însorite dintr-o zi și numărul de beri consumate pe o terasă.

Nr exemplu	Nr ore însorite (X)	Nr beri (Y)
i = 1	2	4
i = 2	3	5
i = 3	5	7

i = 4	7	10
i = 5	9	15

Să se aproximeze (folosind un model liniar) câte beri se vor consuma într-o zi cu 8 ore însorite.

Rezolvare:

Se identifică dreapta Y = a X + b (trebuie calculați coeficienții a și b)

a. Metoda least square root - identificare (exacta) a coeficientilor a si b

1. Pentru fiecare cuplu (x, y) se calculează x^2 și xy.

Nr exemplu	Nr ore însorite (X)	Nr beri (Y)	x^2	xy
i = 1	2	4	4	8
i = 2	3	5	9	15
i = 3	5	7	25	35
i = 4	7	10	49	70
i = 5	9	15	81	135

2. Se însumează, pe rând, valorile fiecărei coloane

Nr exemplu	Nr ore însorite (X)	Nr beri (Y)	x^2	xy
i = 1	2	4	4	8
i = 2	3	5	9	15
i = 3	5	7	25	35
i = 4	7	10	49	70
i = 5	9	15	81	135
	∑=26	∑=41	∑=168	∑=263

3. Se calculează a și b
   a = $(n\Sigma xy - \Sigma x\, \Sigma y)/(\mathbf{n(\Sigma x^2) - (\Sigma x)^2})$
   a = (5 * 263 - 26 * 41) / (5 * 168 - 26*26)
   a = 1.51

   b = $(\Sigma y - a(\Sigma x))/\mathbf{n}$
   b = (41 - 1.51 * 26) / 5
   b = 0.3

   deci funcția de aproximare va fi y = 1.51 * x + 0.3

4. predicția consumului de bere pentru ziua cu 8 ore însorite va fi:
   1.51 * 8 + 0.3 = 12.38

```
import numpy as np
from sklearn import linear_model
from sklearn.linear_model.stochastic_gradient import SGDRegressor
from math import exp
from math import log2
```

```
def myLSR(x, y):
    sx = sum (i for i in x)
    sy = sum (i for i in y)
    sx2 = sum (i * i for i in x)
    sy2 = sum(i * i for i in y)
    sxy = sum (i * j for (i,j) in zip(x, y))
    a = (len(x) * sxy - sx * sy) / (len(x) * sx2 - sx * sx)
    b = (sy - a * sx) / len(x)
    return b, a

def LSRTool(x, y):
    reg = linear_model.LinearRegression()
    xx = []
    for i in x:
        l = [i]
        xx.append(l)
    reg.fit (xx, y)

#     model = ""
#     for c in reg.coef_:
#         model += str(c) +"x"+"+"
#     model += str(reg.intercept_)
#     print("model: ", model)

    return  reg.intercept_, reg.coef_


def testLSR():
    input = [2, 3, 5, 7, 9]
    output = [4, 5, 7, 10, 15]
    print(myLSR(input, output))
    print(LSRTool(input, output))

testLSR()
```

b. Metoda SGD

Modelarea coeficienților (a, b) ai drepte de regresie:
- la iterația 0: valori random (sau 0) pt a și b
- la iterația t + 1 (t = 0, 1, 2, ...)
a(t+1) = a(t) - learning_rate * error(t) * x(t)
b(t+1) = b(t) - learning_rate * error(t)
unde error(t) = computed - realOutput = a(t) * x + b(t), iar learning_rate e parametru al SGD

Discutie: asemanari si diferente intre Stocastic GD (erorarea se calculeaza pentru un singur exemplu din setul de antrenare) si Batch GD (erorarea se calculeaza pentru mai multe exemple din setul de antrenare)

```
import numpy as np
from sklearn import linear_model
from sklearn.linear_model.stochastic_gradient import SGDRegressor
from math import exp
from math import log2

def predict(data, coef):
    s = 0.0
    for i  in range(0, len(data)):
        s += coef[i] * data[i]
    s += coef[len(data)]
    return s

def mySGD(x, y, learningRate, noEpoch):
    coef = [0.0 for i in range(len(x[0]) + 1)]
    for epoch in range(noEpoch):
        sumError = 0.0
        for i in range(0, len(x)):
            ycomputed = predict(x[i], coef)
            crtError = ycomputed - y[i]
            sumError += crtError**2
            for j in range(0, len(x[0])):
                coef[j] = coef[j] - learningRate * crtError * x[i][j]
            coef[len(x[0])] = coef[len(x[0])] - learningRate * crtError
    return coef



def SGDTool(x, y, learningRate, noEpoch):
    reg = linear_model.SGDRegressor()
    reg.max_iter = noEpoch
    reg.fit (x, y)
    return  reg.coef_, reg.intercept_

def testSGD():
    input = [[2], [3], [5], [7], [9]]
    output = [4, 5, 7, 10, 15]
    print(mySGD(input, output, 0.05, 2))
    print(SGDTool(input, output, 0.05, 2))


    #input = [[2, 3], [3, 7], [5, 2], [7, 4], [9, 1]]
    #output = [4, 5, 7, 10, 15]
    #print(mySGD(input, output, 0.001, 2))
    #print(SGDTool(input, output, 0.001, 2))

testSGD()
```

c. Metoda bazata pe Algoritmi Evolutivi

Cromozom

- reprezentare: cromozom = (a,b), a,b sunt nr reale
- fitness: patratul erorii calculată pe tot setul de antrenare

```
from random import random, randrange, uniform

class Chromosome:
    def __init__(self, v = []):
        self.representation = v
        self.fitness = 0.0
    def eval(self, trainInData, trainOutData):
        for i in range(0, len(trainInData)):
            computedOutput = 0.0
            for j in range(0, len(self.representation) - 1):
                computedOutput += self.representation[j] *
trainInData[i][j]
            computedOutput +=
self.representation[len(self.representation) - 1]
            self.fitness += (trainOutData[i] - computedOutput) ** 2

def init(pop, noGenes, popSize):
    for i in range(0, popSize):
        #coefs = [random() for i in range(0, noGenes)]
        coefs = [uniform(0,2) for i in range(0, noGenes)]
        indiv = Chromosome(coefs)
        pop.append(indiv)

def eval(pop, trainInput, trainOutput):
    for indiv in pop:
        indiv.eval(trainInput, trainOutput)

def selection(pop):
    pos1 = randrange(len(pop))
    pos2 = randrange(len(pop))
    if (pop[pos1].fitness < pop[pos2].fitness):
        return pop[pos1]
    else:
        return pop[pos2]

def crossover(M, F):
    off = Chromosome([])
    for i in range(0, len(M.representation)):
        off.representation.append((M.representation[i] +
F.representation[i]) / 2.0)
    return off

def mutation(off):
    pos = randrange(len(off.representation))
    off.representation[pos] = uniform(0,2)
    return off

def bestSolution(pop):
    best = pop[0]
    for indiv in pop:
        if indiv.fitness < best.fitness:
            best = indiv
    return best

def EA(noGenes, popSize, noGenerations, trainIn, trainOut):
```

```
    pop = []
    init(pop, noGenes, popSize)
    eval(pop, trainIn, trainOut)
    for g in range(0, noGenerations):
        popAux = []
        for k in range(0, popSize):
            M = selection(pop)
            F = selection(pop)
            off = crossover(M, F)
            off = mutation(off)
            popAux.append(off)
        pop = popAux.copy()
        eval(pop, trainIn, trainOut)
    sol = bestSolution(pop)
    return sol.representation



def testEA():
    input = [[2], [3], [5], [7], [9]]
    output = [4, 5, 7, 10, 15]
    print(EA(2, 100, 100, input, output))

    #input = [[2, 3], [3, 7], [5, 2], [7, 4], [9, 1]]
    #output = [4, 5, 7, 10, 15]
    #print(EA(3, 100, 100, input, output))

testEA()
```