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 clasificare.

Aspecte teoretice

Proiectarea şi dezvoltarea sistemelor care învaţă singure.
Algoritmi de învăţare de tipul:
- *programare genetica*

Probleme abordate

1. *Remember* problema de regresie
   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 (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: 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 - Programare genetica
3. Problemă
   Se cunosc următoarele informaţii pentru o perioadă de timp trecută: nivelul umidităţii - U, nivelul radiaţiilor solare - RS, intensitatea vântului – V – şi consumul orar de energie electrică – EE (datele normalizate aferente unui set de 10 înregistrări se găsesc în Tabel 1). Să se estimeze consumul orar de energie electrică pentru un tuplu de informaţii (umiditate=0.31, radiaţii solare = 0.55, intensitate vânt=0.82).

U	RS	V	EE
0.74	0.42	0.97	-0.33911
0.04	0.76	0.79	-0.73327
0.72	0.89	0.13	1.1539
0.13	0.26	0.14	-0.07017
0.65	0.49	0.79	-0.14347
0.43	0.44	0.70	-0.31482
0.86	0.68	0.99	0.17052
0.73	0.39	0.29	0.27971

0.08	0.96	0.56	-0.41447
0.47	0.12	0.72	-0.60652

Tabel 1 Date normalizate privind nivelul umidităţii, nivelul radiaţiilor solare şi intensitatea vântului

Încercaţi să rezolvaţi problema folosind un algoritm de programare genetica cu următorii operatori:

- selectie ruleta
- incrucisare cu punct de taietura
- mutatie la nivel de nod

```
import random

MAX_DEPTH = 2
FUNCTION_SET = ["+", "-", "*"]
TERMINAL_SET = [0, 1]   # no of features = 2

class Chromosome:
    def __init__(self):
        self.representation = []
        self.fitness = 0.0

    def grow(self, crtDepth):
        if (crtDepth == MAX_DEPTH):    #select a terminal
            terminal = random.choice(TERMINAL_SET)
            self.representation.append(terminal)
        else:   #select a function or a terminal
            if (random.random() < 0.5):
                terminal = random.choice(TERMINAL_SET)
                self.representation.append(terminal)
            else:
                function = random.choice(FUNCTION_SET)
                self.representation.append(function)
                self.grow(crtDepth + 1)
                self.grow(crtDepth + 1)

    def eval(self, inExample, pos):
        if (self.representation[pos] in TERMINAL_SET):
            return inExample[self.representation[pos]]
        else:
            if (self.representation[pos] == "+"):
                pos += 1
                left = self.eval(inExample, pos)
                pos += 1
                right = self.eval(inExample, pos)
                return left + right
            elif (self.representation[pos] == "-"):
                pos += 1
                left = self.eval(inExample, pos)
                pos += 1
                right = self.eval(inExample, pos)
                return left + right
            elif (self.representation[pos] == "*"):
                pos += 1
                left = self.eval(inExample, pos)
                pos += 1
                right = self.eval(inExample, pos)
                return left + right

    def __str__(self):
        return str(self.representation) # + " fit = " + str(self.fitness)

    def __repr__(self):
        return str(self.representation) #+ " fit = " + str(self.fitness)

def init(pop, noGenes, popSize):
    for i in range(0, popSize):
        indiv = Chromosome()
        indiv.grow(0)
        pop.append(indiv)
```

```python
def computeFitness(chromo, inData, outData):
    err = 0.0
    for i in range(0, len(inData)):
        crtEval = chromo.eval(inData[i], 0)
        crtErr = abs(crtEval - outData[i]) ** 2
        err += crtErr
    chromo.fitness = err

def evalPop(pop, trainInput, trainOutput):
    for indiv in pop:
        computeFitness(indiv, trainInput, trainOutput)

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


#roulette selection
def selectionRoulette(pop):
    sectors = [0]
    sum = 0.0
    for chromo in pop:
        sum += chromo.fitness
    for chromo in pop:
        sectors.append(chromo.fitness / sum + sectors[len(sectors) - 1])
    r = random.random()
    i = 1
    while ((i < len(sectors)) and (sectors[i] <= r)):
        i += 1
    return pop[i - 1]

def traverse(repres, pos):
    if (repres[pos] in TERMINAL_SET):
        return pos + 1
    else:
        pos = traverse(repres,pos + 1)
        pos = traverse(repres,pos)
        return pos

#cutting-point XO
#replace a sub-tree from M with a sub-tree from F
def crossover(M, F):
    off = Chromosome()
    #a sub-tree of M (starting and ending points)
    startM = random.randrange(len(M.representation))
    endM = traverse(M.representation, startM)
    #a sub-tree of F (starting and ending points)
    startF = random.randrange(len(F.representation))
    endF = traverse(F.representation, startF)

    for i in range(0, startM):
        off.representation.append(M.representation[i])
    for i in range(startF, endF):
        off.representation.append(F.representation[i])
    for i in range(endM, len(M.representation)):
        off.representation.append(M.representation[i])
    return off

#change the content of a note (function -> function, terminal -> terminal
def mutation(off):
    pos = random.randrange(len(off.representation))
    if (off.representation[pos] in TERMINAL_SET):
        terminal = random.choice(TERMINAL_SET)
        off.representation[pos] = terminal
    else:
        function = random.choice(FUNCTION_SET)
        off.representation[pos] = function
    return off

def bestSolution(pop):
    best = pop[0]
    for indiv in pop:
```

```
        if indiv.fitness < best.fitness:
            best = indiv
    return best

def EA_generational(noGenes, popSize, noGenerations, trainIn, trainOut):
    pop = []
    init(pop, noGenes, popSize)
    evalPop(pop, trainIn, trainOut)
    for g in range(0, noGenerations):
        popAux = []
        for k in range(0, popSize):
            #M = selection(pop)
            #F = selection(pop)
            M = selectionRoulette(pop)
            F = selectionRoulette(pop)
            off = crossover(M, F)
            off = mutation(off)
            popAux.append(off)
        pop = popAux.copy()
        evalPop(pop, trainIn, trainOut)
        #print("best sol at gener ", g, " has fitness = ", bestSolution(pop).fitness)
    sol = bestSolution(pop)
    return sol

def EA_steadyState(noGenes, popSize, noGenerations, trainIn, trainOut):
    pop = []
    init(pop, noGenes, popSize)
    evalPop(pop, trainIn, trainOut)
    for g in range(0, noGenerations):
        for k in range(0, popSize):
            #M = selection(pop)
            #F = selection(pop)
            M = selectionRoulette(pop)
            F = selectionRoulette(pop)
            off = crossover(M, F)
            off = mutation(off)
            computeFitness(off, trainIn, trainOut)
            crtBest = bestSolution(pop)
            if (off.fitness < crtBest.fitness):
                crtBest = off
        #print("best sol at gener ", g, " has fitness = ", bestSolution(pop).fitness)
    sol = bestSolution(pop)
    return sol

def runEA(inputTrain, outputTrain, inputTest, outputTest):

    learntModel = EA_generational(2, 10, 10, inputTrain, outputTrain)
    print("Learnt model: " + str(learntModel))
    print("training quality: ", learntModel.fitness)
    computeFitness(learntModel, inputTest, outputTest)
    print("testing quality: ", learntModel.fitness)

    learntModel = EA_steadyState(2, 10, 10, inputTrain, outputTrain)
    print("Learnt model: " + str(learntModel))
    print("training quality: ", learntModel.fitness)
    computeFitness(learntModel, inputTest, outputTest)
    print("testing quality: ", learntModel.fitness)

tinnyInputTrain = [[2, 3], [3, 7], [5, 2]]
tinnyOutputTrain = [4, 5, 7]

tinnyInputTest = [[7, 4], [9, 1]]
tinnyOutputTest = [10, 15]
TERMINAL_SET = [0, 1]   # no of features = 2

inputTrain  = [[0.74, 0.42, 0.97],
               [0.04, 0.76, 0.79],
               [0.72, 0.89, 0.13],
               [0.13, 0.26, 0.14],
               [0.65, 0.49, 0.79],
               [0.43, 0.44, 0.70],
               [0.86, 0.68, 0.99],
               [0.73, 0.39, 0.29],
               [0.08, 0.96, 0.56],
               [0.47, 0.12, 0.72]]
outputTrain = [-0.33911, -0.73327, 1.1539, -0.07017, -0.14347, -0.31482, 0.17052, 0.27971, -0.41447, -0.60652]

inputTest = [[0.31, 0.55, 0.82]]
```

```
outputTest = [0.80]
TERMINAL_SET = [0, 1, 2]   # no of features = 3

#runEA(tinnyInputTrain, tinnyOutputTrain, tinnyInputTest, tinnyOutputTest)

runEA(inputTrain, outputTrain, inputTest, outputTest)
```