

Rezolvarea problemelor cu ajutorul metodelor de învățare



Obiective

Dezvoltarea sistemelor care învață singure. Algoritmi de învățare. Specificarea, proiectarea și implementarea sistemelor care învață singure cum să rezolve probleme de clasificare.



Aspecte teoretice

Proiectarea și dezvoltarea sistemelor care învață singure.

Algoritmi de învățare de tipul:

- *programare genetica*



Probleme abordate

1. *Remember* problema de regresie
 - a. ce se da (input X , output Y , un input x_{nou}), ce se cere (funcția care transformă X în Y : $f(X) = Y$, astfel încât să poată fi calculat $y_{nou}=f(x_{nou})$)
 - b. ce poate fi X ? -->
 - i. o listă de valori numerice (regresie simplă) $X = (x_1)$, $x_1 = (x_{11}, x_{21}, \dots, x_{n1})$, unde n e nr de exemple de antrenare),
 - ii. vector cu mai multe dimensiuni de valori numerice (regresie multiplă): dacă avem 2 dimensiuni: $X = (x_1, x_2)$, $x_1 = (x_{11}, x_{21}, \dots, x_{n1})$, $x_2 = (x_{12}, x_{22}, x_{32}, \dots, x_{n2})$, unde n e nr de exemple de antrenare
 - c. ce poate fi Y ? -->
 - i. o listă de valori (pt un exemplu, trebuie prezis un singur output), $Y = (y_1)$, $y_1 = (y_{11}, y_{21}, \dots, y_{n1})$, unde n e nr de exemple de antrenare),
 - ii. vector cu mai multe dimensiuni de valori: dacă avem 3 dimensiuni: $Y = (y_1, y_2, y_3)$, $y_1 = (y_{11}, y_{21}, \dots, y_{n1})$, $y_2 = (y_{12}, y_{22}, y_{32}, \dots, y_{n2})$, $y_3 = (y_{13}, y_{23}, \dots, y_{n3})$, unde n e nr de exemple de antrenare (pt un exemplu, trebuie prezise mai multe (3) output-uri)
2. Metode de identificare a funcției f - Programare genetica
3. Problemă

Se cunosc următoarele informații pentru o perioadă de timp trecută: nivelul umidității - U , nivelul radiațiilor solare - RS , intensitatea vântului - V - și 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 informații (umiditate=0.31, radiații 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 umidității, nivelul radiațiilor solare și intensitatea vântului

Încercați să rezolvați problema folosind un algoritm de programare genetica cu următorii operatori:

- selecție ruleta
- încrucișare cu punct de tăietura
- mutație la nivel de nod

```
import random

MAX_DEPTH = 2
FUNCTION_SET = ["+", "-", "*"] # it can be extended
TERMINAL_SET = [0, 1] # no of features = 2; it can be extended

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)
```

```

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 node (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
        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)
```