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:
- *stocastic gradient descent*
-

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 etichete (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 etichete: 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/etichete)
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 eticheta
    a. gradient descent - regresie logistică

    Presupunem tot un model liniar de clasificare
    f(x) = a0 + a1 x1 +a2 x2 + a3 x3 + ...

    Scopul este gasirea acelor coeficienti a = (a0, a1, a2, ...) care maximizeaza probabilitatea clasificarii corecte. Putem presupune ca avem clasificare binara (clasa pozitiva si clasa negativa)

    X sunt numere reale, deci f(x) va fi un nr real. Putem asocia fiecarui exemplu x o probabilitate (probabilitatea ca exemplul curent sa apartina clasei pozitive):
    $P_{poz}(x) = 1 / (1 + exp(-x))$ - o functie sigmoid (care este convexa, deci putem sa ii gasim optimul global); dar putem alege si alta functie care sa modeleze probabilitatea

Fiind dat $P_{poz}(x)$, putem calcula probabilitatea ca un exemplu sa apartina clasei negative:
$P_{neg}(x) = 1 - 1 / (1 + \exp(-x))$

Clasificarea va fi cu atat mai buna cu cat probabilitatile asociate tuturor exemplelor (in numar de n) sunt mai apropiate de 1, adica putem maximiza

$$max \prod_{i=1}^{n} P(x^i) = \max\left(\prod_{i=1}^{n} P_{poz}(x^i) \prod_{i=1}^{n} P_{neg}(x^i)\right)$$
$$= \max\left(\prod_{i=1}^{n} P_{poz}(x^i) \prod_{i=1}^{n} (1 - P_{poz}(x^i))\right)$$
$$= \min -\left(\prod_{i=1}^{n} P_{poz}(x^i) \prod_{i=1}^{n} (1 - P_{poz}(x^i))\right)$$
$$= \min -\left(\prod_{i=1}^{n} h(x^i) \prod_{i=1}^{n} (1 - h(x^i))\right)$$

Prin logaritmarea expresiei de mai sus, produsul se transforma in suma si se poate obtine expresia:

$$E = -\sum_{y^i=1\ (clasa\ pozitiva)} y^i \log\left(h(x^i)\right) - \sum_{y^i=0\ (clasa\ negativa)} (1 - y^i) \log\left(1 - h(x^i)\right)$$

Pentru a se gasi punctul de optim al acestei expresii, trebuie calculate derivatele partiale (in functie de coeficientii a0, a1, a2, ...)

Se folosesc formulele de derivare:
'(log(x)) = 1 / x
g(z) = 1 / (1 + exp(-z)), g'(z) = exp(-z) / (1 + exp(-z))^2 = 1 / (1 + exp(-z)) * (1 - 1 / (1 + exp(-z))) = g(z) * (1- g(z))
$f(x) = a_0 + a_1x_1 + a_2x_2 + ...$
$\delta f / \delta a_0 = 1$
$\delta f / \delta a_1 = x_1$
$\delta f / \delta a_2 = x_2$

h(x)= sigm(f(x)) = 1 / (1 + exp(-a0-a1x1-a2x2-...))

$$\frac{\partial E}{\partial a_0} = -\sum_{i=1}^{n}\left(\frac{y^i}{h(x^i)} + \frac{1-y^i}{1-h(x^i)}\right)\frac{\partial h(x^i)}{\partial a_0}$$

$$= -\sum_{i=1}^{n}\left(\frac{y^i h(x^i)\left(1-h(x^i)\right)}{h(x^i)} + \frac{(1-y^i)h(x^i)\left(1-\ h(x^i)\right)}{1-h(x^i)}\right)\frac{\partial f(x^i)}{\partial a_0}$$

$$= -\sum_{i=1}^{n}\left(\frac{y^i h(x^i)\left(1-h(x^i)\right)}{h(x^i)} + \frac{(1-y^i)h(x^i)\left(1-\ h(x^i)\right)}{1-h(x^i)}\right) * 1$$

$$= -\sum_{i=1}^{n}\left(\frac{y^i h(x^i)\left(1-h(x^i)\right)^2 + (1-y^i)h^2(x^i)\left(1-\ h(x^i)\right)}{h(x^i)\left(1-\ h(x^i)\right)}\right)$$

$$= -\sum_{i=1}^{n}\left(\frac{h(x^i)\left(1-h(x^i)\right)\left[y^i * \left(1-h(x^i)\right) + \ (1-y^i)h(x^i)\right]}{h(x^i)\left(1-\ h(x^i)\right)}\right)$$

$$= -\sum_{i=1}^{n}\left(\left[y^i * \left(1-h(x^i)\right) + \ (1-y^i)h(x^i)\right]\right) = -\sum_{i=1}^{n}\left(y^i - h(x^i)\right)$$

$$= \sum_{i=1}^{n}\left(h(x^i) - y^i\right)$$

Similar

$$\frac{\partial E}{\partial a_1} = \sum_{i=1}^{n}\left(h(x^i) - y^i\right)x_1^i$$

$$\frac{\partial E}{\partial a_2} = \sum_{i=1}^{n}\left(h(x^i) - y^i\right)x_2^i$$

Functia de cost E este convexa, deci putem sa ii gasim punctul de optim cu metoda gradientului: pornim cu coeficienti a0,a1,a2 random si ii imbunatatim pe baza formulei

$$a_i = \ a_i - \ learninRate * \frac{\partial E}{\partial a_i}$$

1. scalarea lui f la intervalul (0,1) - se poate cu ajutorul unei functii sigmoid sigmoid(z) = 1 / (1 + exp(0 - z))
2. discretizarea intervalului
   a. pt clasificare binara (2 etichete) fixarea unui prag (De ex Theta = 0.5) astfel încât f(x) sub Theta va însemna eticheta 1, iar f(x) peste prag va însemna eticheta 2
   b. pt clasificare cu mai multe clase se pot fixa mai multe praguri (nr de praguri = nr de clase - 1)

3. Exemplu de problema
   Enunt

Se cunosc următoarele n (n = 5) informații aferente unei anumite perioade de timp: numărul de minute însorite dintr-o zi, nr de meciuri dintr-o zi și cantitatea de bere consumata pe o terasă.

Nr exemplu	Nr ore însorite (x_1)	Nr meciuri	Nr beri (Y)
i = 1	120	2	Mica
i = 2	180	1	Mica
i = 3	300	4	Mare
i = 4	420	6	Mare
i = 5	540	5	Mare

Să se aproximeze (folosind un model liniar multiplu) câte beri se vor consuma într-o zi cu 240 minute însorite și 3 meciuri.

Rezolvare:

1. Normalizarea datelor

$$\text{valNormalizată} = \frac{\text{val - medie}}{\text{deviatia standard}} \qquad (1)$$

Medie = suma valorilor / nr de valori

Deviația[1] = sqrt (suma patratelor diferentelor (valoare - medie) / (nr valori - 1))

Nr exemplu	Nr ore însorite (x_1)	Nr meciuri (x2)	Nr beri (Y)
i = 1	120	2	Mica
i = 2	180	1	Mica
i = 3	300	4	Mare
i = 4	420	6	Mare
i = 5	540	5	Mare
Media	312	5.6	
Deviația	171.81	2.07	

Media x1 = (120 + 180 + 300 + 420 + 540) / 5 = 312
Media x2 = (2 + 1 + 4 + 6 + 5) / 5 = 3.6
Deviația x1 = sqrt(((120 - 312)^2 + (180 - 312)^2 + (300 - 312)^2 + (420 - 312)^2 + (540 - 312)^2) / (5 - 1)) = 171.81
Deviația x2 = sqrt((2 - 3.6)^2 + (1 - 3.6)^2 + (4 - 3.6)^2 + (6 - 3.6)^2 + (5 - 3.6)^2) / (5 - 1)) = 2.07

Valorile normalizate

X1 pt i =1 : (120 - 312) / 171.81 = -1.11

Nr exemplu	Nr ore însorite (x_1)	Nr meciuri (x2)	Nr beri (Y)

[1] la nivel de eșantion (de aceea e cu n - 1 la numitor)

i = 1	-1.11749	-0.77159	Mica
i = 2	-0.76827	-1.25383	Mica
i = 3	-0.06984	0.192897	Mare
i = 4	0.628587	1.157383	Mare
i = 5	1.327018	0.67514	Mare

2. Se identifică dreapta Y = a0 + a1 x1 + a2x2 (trebuie calculați coeficienții a) folosind metoda gradientului.

$$a_j = a_j - learninRate * \sum_{i=1}^{n} \left(\frac{1}{1 + e^{-(a0+a1x_1^i+a2x_2^i)}} - y^i \right) * x_j^i$$

```
from sklearn import linear_model
from random import random
import numpy as np
import math

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

def sigmoidFunction(z):
    return 1.0 / (1.0 + math.exp(0.0 - z))

def cost_function(input, output, coef):
    noData = len(input)
    totalCost = 0.0
    for i in range(len(input)):
        example = input[i]
        predictedValue = sigmoidFunction(prediction(example, coef))
        realLabel = output[i]
        class1_cost = realLabel * math.log(predictedValue)
        class2_cost = (1 - realLabel) * math.log(1 - predictedValue)
        crtCost = - class1_cost - class2_cost
        totalCost += crtCost
    return totalCost / noData

def updateCoefs(input, output, coef, learningRate):
    noData = len(input)
    predictedValues = []
    realLabels = []
    for j in range(noData):
        crtExample = input[j]
        predictedValues.append(sigmoidFunction(prediction(crtExample, coef)))
        realLabels.append(output[j])
    for i in range(len(coef)):
        gradient = 0.0
        for j in range(noData):
            crtExample = input[j]
            gradient = gradient + crtExample[i] * (predictedValues[j] - realLabels[j])
        coef[i] = coef[i] - gradient * learningRate
    return coef


def train(input, output, learningRate, noIter):
    coef = [random() for i in range(len(input[0]))]
    costs = []
    for it in range(noIter):
        coef = updateCoefs(input, output, coef, learningRate)
        crtCost = cost_function(input, output, coef)
        costs.append(crtCost)
```

```
    return costs, coef

def test(input, coef):
    predictedLabels = []
    for i in range(len(input)):
        predictedValue = sigmoidFunction(prediction(input[i], coef))
        if (predictedValue >= 0.5):
            predictedLabels.append(1)
        else:
            predictedLabels.append(0)
    return predictedLabels

def accuracy(computedLabels, realLabels):
    noMatches = 0
    for i in range(len(computedLabels)):
        if (computedLabels[i] == realLabels[i]):
            noMatches += 1
    return noMatches / len(computedLabels)

def myLogisticRegression(input, output, learningRate, noIter):
    costs, coeficients = train(input, output, learningRate, noIter)
    computedLabels = test(input, coeficients)
    acc = accuracy(computedLabels, output)
    return acc


def SGDLogisticTool(x, y, learningRate, noEpoch):
    logreg = linear_model.LogisticRegression()
    logreg.max_iter = noEpoch
    logreg.fit (x, y)
    correct = sum(y == logreg.predict(x))
    return  correct / len(x)

def testLogisticSGD():
    input = [[-1.117488473, -0.771588515], [-0.768273325, -1.253831338], [-0.06984303,
0.192897129], [0.628587266, 1.157382773], [1.327017562,  0.675139951]]
    output = [1, 0, 1, 0, 1]
    print("tool acc =  ", SGDLogisticTool(input, output, 0.001, 4))
    print("my acc = ", myLogisticRegression(input, output, 0.001, 4))

testLogisticSGD()
```