Instituto Politécnico Nacional

Escuela Superior de Cómputo

Regresión Lineal

Natural Language Processing

Estudiante:
Nicolás Sayago Abigail

Profesora:
Olga Kolesnicova

March 31, 2020

RESULTADOS TRAINING SET

Puedo observar que tal como mencionaba el profesor, en la iteración 0 y 1 se ve un cambio muy brusco, pero después empieza a cambiar por muy poco.

```
Cost Function initial value: 209140051559.17117
TRAINING TEST:
Iteration: 0 Cost Function value: 49428828425.961
Iteration: 50 Cost Function value: 34573271677.767265
Iteration: 100 Cost Function value: 33913544981.65546
Iteration: 150 Cost Function value: 33721282258.287045
Iteration: 200 Cost Function value: 33623826641.194653
Iteration: 250 Cost Function value: 33562970622.217136
Iteration: 300 Cost Function value: 33521985088.313515
Iteration: 350 Cost Function value: 33493339445.0792
Iteration: 400 Cost Function value: 33472844160.6985
Iteration: 450 Cost Function value: 33457939810.132267
Iteration: 500 Cost Function value: 33446976130.231293
Iteration: 550 Cost Function value: 33438846176.872955
Iteration: 600 Cost Function value: 33432783939.47736
Iteration: 650 Cost Function value: 33428246292.591095
Iteration: 700 Cost Function value: 33424841017.15943
Iteration: 750 Cost Function value: 33422281052.927032
Iteration: 800 Cost Function value: 33420354292.007168
Iteration: 850 Cost Function value: 33418902961.99316
Iteration: 900 Cost Function value: 33417809167.575188
Iteration: 950 Cost Function value: 33416984535.49037
```

RESULTADOS TESTING SET

Después de haber entrenado, se aplica la hipótesis y se compara con el valor real del conjunto de prueba:

```
    ...:     j = j + 1
Prediction: 524695.7522827049 Real: 245500.0
Prediction: 313391.8761328119 Real: 230000.0
Prediction: 769313.7615031959 Real: 1770000.0
Prediction: 482236.953004864 Real: 425000.0
Prediction: 369683.8958635811 Real: 220000.0
Prediction: 384677.18561883114 Real: 370000.0
Prediction: 689550.9880488135 Real: 390000.0
Prediction: 410385.5804063771 Real: 239000.0
Prediction: 337455.0066295272 Real: 345000.0
Prediction: 300277.08189784863 Real: 289000.0
Prediction: 146205.16434202378 Real: 285000.0
Prediction: 567994.0105406654 Real: 147400.0
Prediction: 514607.95932153397 Real: 600000.0
Prediction: 475689.40187220654 Real: 245000.0
Prediction: 475689.40187220654 Real: 520000.0
Prediction: 493646.3955494128 Real: 352500.0
Prediction: 585263.1108050327 Real: 448000.0
Prediction: 791353.8909967549 Real: 270000.0
Prediction: 277046.67811581213 Real: 872500.0
Prediction: 1139189.8292430635 Real: 2150000.0
Prediction: 560472.7970308239 Real: 225000.0
Prediction: 463138.39902912016 Real: 735000.0
Prediction: 574196.0412853383 Real: 375000.0
Prediction: 239982.15854712797 Real: 169000.0
Prediction: 239982.15854712797 Real: 279950.0
Prediction: 1372506.789729353 Real: 1950000.0
Prediction: 368553.4312885988 Real: 310000.0
```

✓Código fuente

```
import csv
import numpy as np

def readData(nameFile):
    auxMatrix = list()  #Matrix of numpy arrays
    with open(nameFile, newline = '') as csvfile:
        reader = csv.reader(csvfile)
        headers = next(reader)  #List of headers
    with open(nameFile, newline = '') as csvfile:
```

```
        reader = csv.DictReader(csvfile)
        cont = 0
        porcent = 15118
        for row in reader:
            if cont < porcent:
                auxList = list()
                auxList.append(1)   #Adding 1 for convenience
                for i in range(3, len(headers)):
                    intAux = float(row[headers[i]])
                    auxList.append(intAux)
                auxNum = np.array(auxList)  #Numpy array
                auxNum = np.absolute(auxNum)
                auxMatrix.append(auxNum)
            else:
                break
            cont = cont + 1

    matrix = np.array(auxMatrix)  #Numpy matrix
    return matrix

def readTest(nameFile):
    auxMatrix = list()  #Matrix of numpy arrays
    with open(nameFile, newline = '') as csvfile:
        reader = csv.reader(csvfile)
        headers = next(reader)  #List of headers
    with open(nameFile, newline = '') as csvfile:
        reader = csv.DictReader(csvfile)
        cont = 0
        porcent = 15118
        for row in reader:
            if cont > porcent:
                auxList = list()
                auxList.append(1)   #Adding 1 for convenience
                for i in range(3, len(headers)):
                    intAux = float(row[headers[i]])
                    auxList.append(intAux)
                auxNum = np.array(auxList)  #Numpy array
                auxNum = np.absolute(auxNum)
                auxMatrix.append(auxNum)
            else:
                cont = cont + 1

    matrix = np.array(auxMatrix)  #Numpy matrix
    return matrix
```

```
def getY(nameFile):
    with open(nameFile, newline = '') as csvfile:
        reader = csv.DictReader(csvfile)
        auxList = list()
        cont = 0
        porcent = 15118
        for row in reader:
            if cont < porcent:
                listaux = list()
                intAux = float(row["price"])
                listaux.append(intAux)
                n = np.array(listaux)
                auxList.append(n)
            else:
                break
            cont = cont + 1
        npArray = np.array(auxList)
    return npArray

def getYTest(nameFile):
    with open(nameFile, newline = '') as csvfile:
        reader = csv.DictReader(csvfile)
        auxList = list()
        cont = 0
        porcent = 15118
        for row in reader:
            if cont > porcent:
                listaux = list()
                intAux = float(row["price"])
                listaux.append(intAux)
                n = np.array(listaux)
                auxList.append(n)
            else:
                cont = cont + 1
        npArray = np.array(auxList)
    return npArray

def featureScaling(matrix):
    U = np.sum(matrix, axis = 0) #Sum of columns

    for i in range(0, len(U)):
        U[i] = U[i] / len(matrix)
```

```
    S = list() #Get Standar D.
    for i in range(0, len(matrix[0])):
        column = [row[i] for row in matrix]
        column = np.array(column)
        std = column.std()
        S.append(std)
    S = np.array(S)

    newMatrixAux = list()
    for i in range(0, len(matrix)): #row
        aux = list()
        for j in range(0, len(matrix[i])): #column
            res = matrix[i][j] - U[j]
            res = abs(res)
            if S[j] != 0:
                res = res / S[j]
            if j == 0:
                res = 1
            aux.append(res)
        auxNum = np.array(aux)
        newMatrixAux.append(auxNum)

    newMatrix = np.array(newMatrixAux)
    return newMatrix

def initializeTheta(n):
    m = list()
    for i in range(0, n):
        aux = list()
        aux.append(0)
        auxNum = np.array(aux)
        m.append(auxNum)

    theta = np.array(m)
    return theta

def printMatrix(matrix, n):
    for i in range(0, n):
        print(matrix[i])

def sumaCostFunction(H_theta, Y):
    auxSub = H_theta - Y
    ansSum = 0
    for i in auxSub:
```

```
        for j in i:
            j = j * j
            ansSum = ansSum + j
    ansSum = ansSum / (2 * len(Y[0]))
    return ansSum

def gradientDescent(theta, H_theta, Y, matrix, learningRate):
    temp = list()
    for j in range(0, len(matrix)):
        ans = 0
        for i in range(0, len(matrix[j])):
            res = ((H_theta[0][i] - Y[0][i])) * matrix[j][i]
            ans = ans + res
        ans = ans / len(H_theta[0])
        aux =  theta[j][0] - (learningRate * ans)
        listAux = list()
        listAux.append(aux)
        temp.append(listAux)

    tempNum = np.array(temp)
    return tempNum

#####################################################
#                    MAIN
#####################################################
# Read data
nameFile =
 ↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Practice/17/in.csv
matrixN = readData(nameFile) #Training set
auxY = getY(nameFile)
Y = auxY.transpose()

matrixTestN = readTest(nameFile) #Testing set
auxY = getYTest(nameFile)
YTest = auxY.transpose()

#Feature scaling
matrixN = featureScaling(matrixN)
matrixTestN = featureScaling(matrixTestN)

#Get transpose matrix of data
matrix = matrixN.transpose()
matrixTest = matrixTestN.transpose()
```

```
#Get initial theta vector
theta = np.zeros(shape = (len(matrix), 1)) # n x 1

#H(theta) = thetaT * matrix
thetaT = theta.transpose() # 1 x n
H_theta = thetaT.dot(matrix) # 1 x m

#Get Cost Function
price = sumaCostFunction(H_theta, Y)
print("Cost Function initial value:", price)

tempTheta = initializeTheta(len(matrix))
learningRate = 0.1

print("TRAINING TEST:")
for ite in range(0, 1000):
    tempTheta = gradientDescent(theta, H_theta, Y, matrix, learningRate)
    theta = tempTheta
    thetaT = tempTheta.transpose()
    H_theta = thetaT.dot(matrix)
    price = sumaCostFunction(H_theta, Y)
    if((ite % 50) == 0):
        print("Iteration:", ite, "Cost Function value:", price)



print(YTest) # m x 1
#H(theta) = thetaT * matrix
thetaT = theta.transpose() #Now, this has been trained 1 x m
print(thetaT)
# matrixTest 19 x m
#H_theta = thetaT.dot(matrixTest) #Get H(theta)
#price = sumaCostFunction(H_theta, YTest) #Get Cost Function
print("TESTING SET, first 50 elements:")
matrixTest = matrixTest.transpose()
print(matrixTest)
Ytest = list()
for i in YTest:
    for j in i:
        Ytest.append(j)
print(Ytest)
type(Ytest)
j = 0
for i in range(0, 50):
```

```
    aux = np.array(matrixTest[i]) # m x 1
    matrixIndexAux = list()
    for i in aux:
        listAuxiliar = list()
        listAuxiliar.append(i)
        lA = np.array(listAuxiliar)
        matrixIndexAux.append(lA)
    matrixIndexTest = np.array(matrixIndexAux)
    # print(matrixIndexTest) # m x 1
    prediction = thetaT.dot(matrixIndexTest)
    print("Prediction:", prediction[0][0], "Real:", Ytest[j])
    j = j + 1
    # price = sumaCostFunction(H_theta, YTest)
    # print("Cost Function value:", price)
```