Alumno:	García González Aarón Antonio
Grupo:	3CV9
Unidad de Aprendizaje:	Procesamiento de lenguaje natural
Profesora:	Kolesnikova Olga
Tarea 1:	Algoritmo de aprendizaje automático por modelo de regresión lineal multivariable con descenso gradiente
Fecha:	Jueves 02 de abril de 2020

Valores de la función de costo en 1000 iteraciones

```
^[[A(base) Aarons-MacBook-Pro:PLN aarongarcia$ python index.py
50 50129103766.2371
100 39794209664.47317
150 33734818191.89084
200 30028930757.228462
250 27675337164.87491
300 26124673469.242596
350 25064555288.711926
400 24312399083.70143
450 23758977027.48157
500 23337525079.234306
550 23006356654.335564
600 22738863887.455006
650 22517663702.50992
700 22331115899.435276
750 22171221277.121284
800 22032334343.28108
850 21910363245.649433
900 21802264803.500042
950 21705720497.185627
1000 21618924824.910053
```

Valores de predicción de algunos elementos del conjunto de pruebas

Real	Prediccion	Error
572000	558267	2.40082
1e+06	742226	25.7774
642000	552055	14.0102
275000	349524	27.0996
292500	327585	11.9949
400000	401843	0.460833
230000	240524	4.57551
527500	470717	10.7645
339000	309376	8.7387
317000	402171	26.8679
365000	509343	39.5459
915000	1.00124e+06	9.42487
1.78e+06	1.37849e+06	22.557
450000	863378	91.8617
410000	833750	103.354
350000	291956	16.5839
1.06e+06	1.24683e+06	17.6256
394000	347609	11.7744
682000	718260	5.3167
365000	291309	20.1893
295000	261356	11.4047
580000	573230	1.16725
570000	335338	41.1688
1.05e+06	1.10081e+06	4.83915
297000	388048	30.6558
346290	386798	11.6977
310000	322520	4.03875
450000	575415	27.8701
345000	407163	18.0182
263000	186427	29.1152
1.08e+06	739125	31.5625
331500	441791	33.2704
321950	160514	50.1433
289571	247791	14.4282
390000	411564	5.52933
160000	82946.4	48.1585
2.21e+06	980968	55.6123
252000	236047	6.33048
552250	519680	5.89765
251000	299124	19.1728
254000	374674	47.5096
590000	565130	4.21531
319950	494952	54.6968
397380	399572	0.551624
310000	280227	9.60427
302000	177756	41.1405
635000	606759	4.44737
562500	572344	1.75009
472000	751330	59.1801
771000	774484	0.451931
808000	1.01999e+06	26.2361

Valor de error general del programa y valor de función de costo en conjunto de datos de pruebas

```
Valor de la funcion de costo:  22049707513.578587
Error:  25.05190071742754
(base) Aarons-MacBook-Pro:PLN aarongarcia$
```

Código

```python
from datetime import datetime
import pandas as pd
import numpy as np
import random
import math
from tabulate import tabulate

θs = []

# funcion que agrega uno's en la primera posicion de cada fila de la matriz X
def addOnes(vector_x):
    return  np.insert(vector_x, 0, [1], axis=1)


def getData(name_file):
    global θs

    datos = pd.read_csv(name_file)

    matriz = np.array(datos)
    matriz = matriz[:,1:] # quitamos la columna id

    for f in matriz:
        f[0] = datetime.strptime(f[0], '%M/%d/%Y').date() # convertimos a fecha

    minimo = min(matriz[:, 0]) # fecha mas antigua

    for f in matriz:
        f[0] = int(abs(minimo - f[0]).days) # favorecemos en nuemro a las fechas mas cercanas

    Y = matriz[:, 1]
    X = matriz[:, [0]+[f for f in range(2,len(matriz[0]))]] # quitamos la columna del id

    θs = np.array( [0]*(len(X[0])+1) )

    return X,Y
```

```
def featureScaling(X):

    resultado = []

    for col in np.transpose(X):
        total_sum = np.sum(col)
        minimum = min(col)
        maximum = max(col)
        average = total_sum / len(col)
        col = (col - average) / (maximum - minimum)
        resultado.append(col)

    resultado = np.transpose(resultado)
    resultado = addOnes(resultado) # aniade 1 al principio de cada vector

    return resultado


def separateSet(X,Y,test_percentage):

    data_zipped = list(zip(X, Y))
    random.shuffle(data_zipped) # revuelve la data
    X, Y = zip(*data_zipped) # descomprime el iterable dado

    total_test = math.ceil(len(Y) * test_percentage)
    total_train = len(Y) - total_test

    X_train = X[:total_train]
    X_test = X[total_train:]
    Y_train = Y[:total_train]
    Y_test = Y[total_train:]

    return np.array(X_train), np.array(Y_train), np.array(X_test), np.array(Y_test)


# la funcion orginal es θ^T . X, pero dado que θ es un vector, entonces solo cambia si es
# vector columna o vector fila, y eso al vovler a transponer nos da el resultado correcto.
# DIM(θ) = 1 * N+1 y DIM(X) = M * N+1, y al hacer θ^T . X, hay error en cuanto a las dimenciones,
# por otro lado hacer: θ^T . X, genera un vector de m * N+1
def h(X):
    return np.dot(X, np.transpose(np.array(θs)))


def error(X, Y):
    return (h(X)-Y)


# costo
def J(X, Y):
    return np.sum(error(X, Y)**2) / (2*len(Y))
```

```
# Es lo mismo que la suma, pero dado que hacemos producto punto, en cada multiplicacion de
# fila por columna se hace la suma, y nuevamente al multiplicar por un vector columna, no
# importa el orden, esto con el objetivo de ajustar las dimenciones de la multiplicacion
def J_parcial(X, Y):
    return np.dot(np.transpose(X), error(X, Y)) / len(Y)


# funcion gradiente que actualiza los valores de th
def gradient(X, Y, α):
    global θs

    θs_temp = θs - (α * J_parcial(X, Y))

    θs = θs_temp


def train(X,Y,learning_rate,iterations):
    for i in range(0,iterations):
        if (i+1)%50 == 0:
            print(i+1, J(X, Y))
        gradient(X, Y, learning_rate)

def test(X,Y):
    Y_predicciones = h(X)

    tabla = []
    error_total = 0
    for i in range(0,len(Y)):
        eror_particular = abs(100-(100/(Y[i])*Y_predicciones[i]))
        error_total += eror_particular
        tabla.append([Y[i],Y_predicciones[i],eror_particular])

    print(tabulate(tabla,headers=['Real','Prediccion', 'Error' ],tablefmt="grid",
numalign="center"))
    print("Valor de la funcion de costo: ", J(X,Y))
    print("Error: ", error_total/len(Y))

def main():
    name_file = "data.csv"
    α = 0.1
    X, Y = getData(name_file)
    X = featureScaling(X)
    X_train, Y_train, X_test, Y_test = separateSet(X,Y,0.3)
    train(X_train, Y_train, α, 1000)
    test(X_test,Y_test)


if __name__ == "__main__":
    main()
```