Alumno:	García González Aarón Antonio
Grupo:	3CV9
Unidad de Aprendizaje:	Procesamiento de lenguaje natural
Profesora:	Kolesnikova Olga
Tarea 2.	Algoritmo de aprendizaje automático por modelo de regresión logística en clasificación binaria con descenso gradiente
Fecha:	Domingo 05 de abril de 2020

Valores de la función de costo en 1000 iteraciones

```
(base) Aarons-MacBook-Pro:2_regresion_logistica aarongarcia$ python index.py
50 -0.3593061474206523
100 -0.7273376320637897
150 -0.7303646955743993
200 -0.7322706772380487
250 -0.7336039444109248
300 -0.7345984009447708
350 -0.7353720637989284
400 -0.7359923684765162
450 -0.7365013155180985
500 -0.7369266819319207
550 -0.7372876590578648
600 -0.7375979597762389
650 -0.7378676515780095
700 -0.7381042947247722
750 -0.7383136775757857
800 -0.7385003071248949
850 -0.7386677451024181
900 -0.7388188435986425
950 -0.7389559135890578
1000 -0.7390808476299899
```

Valores de predicción de algunos elementos del conjunto de pruebas

Real	Prediccion
0	0.00428486
0	0.00501844
0	0.00387365
0	0.00886802
0	0.00329051
1	0.0345642
1	0.00502599
0	0.00348193
1	0.687916
1	0.967923
0	0.00328001
0	0.00424998
0	0.0033595
1	0.997332
1	0.0930724
0	0.00489291
0	0.00301617
0	0.00526314
0	0.00636846
1	0.0416491
0	0.00382477
0	0.00416961
0	0.0045329
0	0.00370289
0	0.00304921

0	0.00484676
0	0.00357856
1	0.700368
0	0.00478857
0	0.00384928
0	0.00279492
0	0.00132168
0	0.00834589
0	0.00424155
0	0.000733072
1	0.132742
0	0.00386763
1	0.387734
1	0.985604
1	0.962337
0	0.00481675
0	0.00498452
0	0.00494269
0	0.00395378
0	0.00495565
0	0.00247314
0	0.00236297
0	0.00500049
0	0.0037397
1	0.0575092
0	0.00443234

0	0.00284962
0	0.00528089
0	0.00346362
0	0.00290431
0	0.00440578
0	0.0041438
0	0.00429591
0	0.00374615
0	0.00181128
0	0.00461449
1	0.0207515
0	0.00390936
0	0.00482384
1	0.0461698
1	0.0169083
1	0.0142144
0	0.0026772
0	0.00203059
0	0.0033447
0	0.00405517
0	0.00228238
0	0.00318606
0	0.00495565
1	0.23619
0	0.00276481
0	0.00364482

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

```
Valor de la funcion de costo:  -0.3735410018368852
Exactitud:  86.4321608040201 %
(base) Aarons-MacBook-Pro:2_regresion_logistica aarongarcia$
```

Código

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

0s = []

# leer de archivo .txt, entrega una lista de lineas
def getText(path):
    from nltk.corpus import stopwords
    stopwords = stopwords.words('english')
    resultado = []

    with open(path, 'r') as f:
        lines = f.readlines()

    tags = ['a', 'n', 'r', 'v']
    for i in range(0,len(lines)):
        lines[i] = nltk.word_tokenize(lines[i].lower()) # minuscula y tokenizacion
        lines[i].pop(-2) # quiar la coma que separa el tipo de mensaje
        lines[i] = nltk.pos_tag(lines[i]) # hacer el POS tag

        row = []
        for j in range(0,len(lines[i])):
            tag = lines[i][j][1][0].lower() # convertir a minuscula el tag[0]
            if tag in tags: # lematizar de acuerdo a tags
                lines[i][j] = nltk.WordNetLemmatizer().lemmatize(lines[i][j][0],tag)
            else:
                lines[i][j] = lines[i][j][0]
            # quitamos stopwords
            if lines[i][j] not in stopwords:
                row.append(lines[i][j])
        resultado.append(row)

    return resultado


# obtiene el vocabulario a partir de cada linea
```

```
def getvocabulary(text):
    vocabulary = []

    for line in text:
        vocabulary.extend(line)

    return sorted(set(vocabulary))


# funcion que añade la columa de 1's en el vector X en la primera posicion
def addOnes(vector_x):
    return  np.insert(vector_x, 0, [1], axis=1)


# regresa el vector X y Y
def getData(text,vocabulary):
    np.seterr(divide='ignore', invalid='ignore')
    global θs

    X = []
    Y = []

    for line in text:
        if line[-1] == "spam":
            Y.append(1)
        else:
            Y.append(0)
        line.pop(-1) # eliminar dato objetivo
        row = []
        for token in vocabulary:
            row.append(line.count(token))
        X.append(np.array(row))

    X = np.array(X)
    Y = np.array(Y)

    # realizar normalizacion de frecuencia o "probabilidad"
    sumas = np.sum(X, axis=0)

    x = []
    for i in range(0,len(X)):
        x.append(np.divide(X[i],sumas))

    x = np.array(x)
    x = addOnes(x)
    θs = np.array([0]*len(x[0]))


    df = pd.DataFrame(data=x)
    df = df.replace(np.nan, 0)
```

```
    x = df.to_numpy()

    return x,Y


# funcion que separa los datos para entrenamiento y prueba
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 [ 1/(1+np.power(np.e,-1*z)) for z in np.dot(X, np.transpose(np.array(θs)))]


# funcion de error, es decir prediccion menos realidad
def error(X, Y):
    return (h(X)-Y)


#Funcion de costo
def J(X, Y):
    predict = np.array(h(X))
    sumando_1 = np.sum(np.log(predict) * Y)
    sumando_2 = np.sum(((predict * (-1)) + 1) * ((Y * (-1)) + 1))

    return (-1)*(sumando_1 + sumando_2)/len(Y)

# funcion derivada parcial
def J_parcial(X, Y):
    return np.dot(np.transpose(X), error(X, 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

# funcion de entrenamiento con base al nuemro de iteraciones
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 = []
    correctos = 0
    for i in range(0,len(Y)):
        eror_particular = abs(100-(100/(Y[i])*Y_predicciones[i]))
        tabla.append([Y[i],Y_predicciones[i]])
        if Y[i] == 1:
            if Y_predicciones[i] >= 0.5:
                correctos += 1
        else:
            if Y_predicciones[i] < 0.5:
                correctos += 1

    print(tabulate(tabla,headers=['Real','Prediccion'],tablefmt="grid", numalign="center"))
    print("Valor de la funcion de costo: ", J(X,Y))
    print("Exactitud: ", 100*correctos/len(Y),"%")


def main():
    name_file = "SMS_Spam_Corpus_big.txt"
    α = 0.1
    text = getText(name_file) # texto por linea ya lematizado con base a POS tag
    vocabulary = getvocabulary(text)

    X,Y = getData(text,vocabulary)
    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()
```