NEURONA LOGÍSTICA

By: Jared Isaías Monje Flores

20 DE FEBRERO DE 2024
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
Inteligencia Artificial II

Neurona Logística

d = 8

η = 0.1

epochs = 10000

Resultados obtenidos:

W: [-0.67585917 -0.59510698 0.21373198 -0.52831287 -0.13956815 0.31548662

-0.48334791 -0.34342004]

W: [1.79368414 5.87596869 -1.51040994 0.14385852 -0.23675301 4.09193227

1.85887691 1.06692296]

Accurancy: 0.7721354166666666

```python
#
# By: Dexne
#
# Este código es la implementación simple de una neurona logistica para clasificación binaria
# Es entrenado con un dataset de datos proporcionado y se informa la presicion
#
# para obtener mejores resultados ajusta las epocas y el learning rate
#


# Importamos los modulos y librerías necesarias para los calculos matematicos y para el tratado de los datos
import numpy as np
from sklearn import preprocessing



# Definimos la clase de la neurona logistica
class LogisticNeuron:

    # Atributos:
    # w = array de pesos
    # b = sesgo
    # eta = taza de aprendizaje utilizado durante el entrenamiento


    def __init__( self, n_inputs, learning_rate=0.1 ) -> None:
        self.w = -1 + 2 * np.random.rand( n_inputs )
        self.b = -1 + 2 * np.random.rand()
        self.eta = learning_rate


    def predict_proba( self, X ):
        Z = np.dot( self.w, X ) + self.b
        return ( 1 / (1 + np.exp(-Z)) )
        #
        #          1
```

```
        #    --------------

        #              -z

        #       1 + e


    # Función prediccion con umbral a 0.5

    # hacemos prediccion basada en el calculo de probabilidades

    def predict( self, X, umbral=0.5 ) -> int:

        Z = np.dot( self.w, X ) + self.b

        Y_est = ( 1 / (1 + np.exp(-Z)) )

        return 1.0 * ( Y_est > umbral )


    # funcion entrenamiento

    def fit( self, X, Y, epochs=500 ):

        p = X.shape[1]

        for _ in range( epochs ):

            Y_est = self.predict_proba( X )

            # Actualizamos los pesos y el sesgo usando el gradiente descendente

            self.w += ( self.eta/p) * np.dot( (Y - Y_est), X.T ).ravel()

            self.b += ( self.eta/p) * np.sum( (Y - Y_est) )


if __name__=="__main__":

    # Definimos nuestro numero de epochs, lr y las columnas

    epochs, learning_rate, I = 10000, 0.1, 8


    # Cargamos los datos

    # En caso no estar en la misma ruta, ajustarla

    X = np.genfromtxt("diabetes.csv", delimiter=',', skip_header=1, usecols=[i for i in range(I)]).T

    Y = np.genfromtxt("diabetes.csv", delimiter=',', skip_header=1, usecols=[I]).T


    for i in range(I):

        X[i, :] = preprocessing.minmax_scale( X[i, :] )
```

```
    # Creamos una neurona
    neuron = LogisticNeuron( X.shape[0], learning_rate )

    print('W:\t', neuron.w )
    neuron.fit( X, Y, epochs=epochs)
    print('W:\t', neuron.w )

    predic = neuron.predict( X )
    good = 0
    for i in range( X.shape[1] ):
        if ( predic[ i ] == Y[ i ] ):
            good += 1

    # Imprimimos el porcentaje de precision
    print( "Accurancy:\t", good/X.shape[1] )

## Resultados ------------------------------------
# W:        [-0.67585917 -0.59510698  0.21373198 -0.52831287 -0.13956815  0.31548662
#  -0.48334791 -0.34342004]
# W:        [ 1.79368414  5.87596869 -1.51040994  0.14385852 -0.23675301  4.09193227
#   1.85887691  1.06692296]
# Accurancy:        0.7721354166666666
```