David Rodriguez Fragoso

A01748760

Momento de Retroalimentación: Módulo 2 Implementación de una técnica de aprendizaje máquina sin el uso de un framework

El propósito de esta entrega es realizar una implementación de un modelo de aprendizaje sin hacer uso de frameworks tales como scikit learn. Hemos optado por crear un perceptrón que haga uso del gradiente descendiente estocástico y la función de activación signo, ya que su implementación es relativamente sencilla.

En esta ocasión haremos uso de un dataset creado por nosotros mismos, ya que el formato que hemos decidido es algo específico. En la primera fila se encuentra nuestra x0, en la cual todos sus valores serán de 1. Las siguientes filas serán una cantidad indefinida de x de entrada hasta llegar a la penúltima fila, la cual será nuestro valor esperado y finalmente, la última fila será nuestro coeficiente de aprendizaje (Alpha). Recordemos que, ya que nuestra función de activación es signo, los valores ingresados y de salida serán 1 y 0 únicamente.

Lo primero que hicimos fue cambiar los 0 por -1, ya que estos no nos sirven con la función signo. Seguido de esto, obtendremos el valor de Alpha y lo retiraremos de nuestro dataset para poder mezclar los datos y asegurar que al momento de dividir entre datos de aprendizaje y datos de entrenamiento, estos sean siempre diferentes y evitar así overfitting.

```
for i in data:
    for j in range(0, len(i)):
        i[j] = float(i[j])
        if i[j] == 0.0:
            i[j] = -1

#define learning rate, t, and vector x
alpha = data[len(data)-1]
data.pop()
alpha = np.array(alpha)
```

Después de esto, inicializaremos el vector de pesos inicial en un valor cercano a 0 (en esta implementación será 0.1) A continuación haremos uso de la función signo y del producto punto para obtener nuestra primera aproximación (o) a los valores esperados (t). Compararemos si estos son iguales o si nuestro vector de pesos se ha repetido más de 20 veces y si alguna de estas condiciones es verdadera, entonces se detendrá el programa e imprimirá los valores calculados, de lo contrario, se volverán a calcular los pesos según indica el gradiente descendiente estocástico y se iterará de nuevo. Para verificar que los vectores de pesos no se hayan repetido más de 20 veces, estos se guardarán en un diccionario.

```python
def obtain_o(x,w):
    result = []
    #print(f'{"x":=^60}')
    #print(x)
    for i in range(len(x)):
    #for i in range(x.shape[1]):
        result.append(np.sign(np.dot(x[i],w)))
    return result
```

```python
  while o != t and dictW[str(weights)]<21:
      weights = np.array(obtain_w(weights,alpha,t,o,x))

      t = np.array(t)
      o = obtain_o(x,weights)


      t = t.tolist()
      o=np.array(o)
      o = o.tolist()
      if str(weights) not in dictW:
          dictW[str(weights)]=1
      else:
          dictW[str(weights)]+=1
```

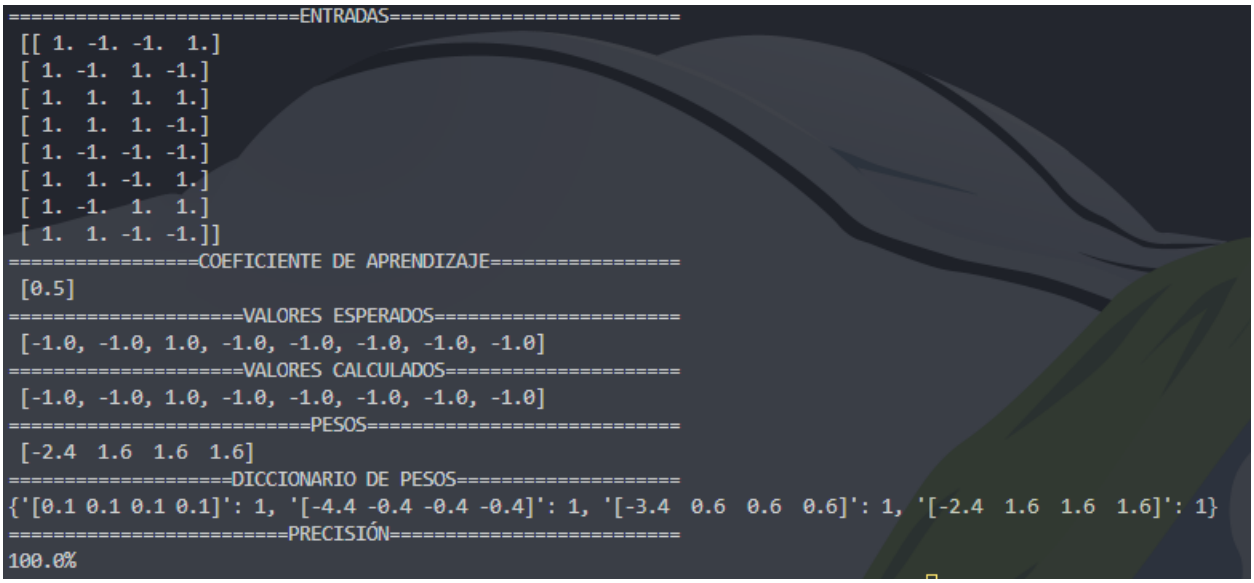
```python
def obtain_w(w,alpha,t,o,x):

    #print(f'w{w}')
    #print(f'o{o}')
    for i in range(len(x)):
        w = w+alpha*(t[i]-o[i])*x[i]
```

Finalmente calcularemos nuestro error y lo desplegaremos

Para nuestra prueba inicial usaremos el siguiente input:

```
1,1,1,1,1,1,1,1
0,0,0,0,1,1,1,1
0,0,1,1,0,0,1,1
0,1,0,1,0,1,0,1
0,0,0,0,0,0,0,1
0.5
```



Y la siguiente salida:

Podemos observar que se calcularon muchos vectores de pesos, pero nos quedamos con el que nos daba el resultado esperado.

Es importante mencionar que Alpha es un hiperparámetro, por lo cual, alterar su valor puede tener un gran impacto (positivo o negativo) en nuestro aprendizaje.

Haremos más pruebas, pero ahora con más datos para comprobar el correcto funcionamiento de nuestro modelo:

Input:

```
1,1,1,1,1,1,1,1,1,1
0,0,0,0,0,0,0,0,1,1
0,0,0,0,1,1,1,1,0,0
0,0,1,1,0,0,1,1,0,0
0,1,0,1,0,1,0,1,0,1
0,0,1,1,1,1,0,1,0,0
0.5
```

```
==========ENTRADAS==========
[[ 1. -1. -1.  1. -1.]
 [ 1. -1. -1. -1.  1.]
 [ 1.  1. -1. -1. -1.]
 [ 1. -1. -1.  1.  1.]
 [ 1.  1. -1. -1.  1.]
 [ 1. -1.  1. -1. -1.]
 [ 1. -1.  1. -1.  1.]
 [ 1. -1.  1.  1.  1.]
 [ 1. -1. -1. -1. -1.]
 [ 1. -1.  1.  1. -1.]]
==========COEFICIENTE DE APRENDIZAJE==========
[0.5]
==========VALORES ESPERADOS==========
[1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, -1.0]
==========VALORES CALCULADOS==========
[-1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0]
==========PESOS==========
[ 0.1 -1.9  2.1  2.1  2.1]
==========DICCIONARIO DE PESOS==========
{'[0.1 0.1 0.1 0.1 0.1]': 1, '[ 0.1 -1.9  0.1  0.1 -1.9]': 1, '[-1.9  0.1  0.1  0.1  0.1]': 1, '[ 3.1 -4.9  1.1  1.1  1.1]': 1, '[ 0.1 -1.9  2.1  2.1  2.1]': 21, '[ 1.1 -2.9  1.1  1.1  1.1]': 20, '[-1.9  0.1  2.1  2.1  2.1]': 20,
'[-0.9 -0.9  1.1  1.1  1.1]': 20, '[ 0.1 -1.9  0.1  0.1  0.1]': 20, '[-2.9  1.1  1.1  1.1  1.1]': 20, '[ 2.1 -3.9  2.1  2.1  2.1]': 20}
==========PRECISIÓN==========
70.0%
```

Input:

```
1,1,1,1,1,1,1,1,1,1
0,0,0,0,0,0,0,0,1,1
0,0,0,0,1,1,1,1,0,0
0,0,1,1,0,0,1,1,0,0
0,1,0,1,0,1,0,1,0,1
0,0,1,1,1,1,0,1,0,0
0.5
```

```
==========COEFICIENTE DE APRENDIZAJE==========
[0.1]
==========VALORES ESPERADOS==========
[-1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0]
==========VALORES CALCULADOS==========
[1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0]
==========PESOS==========
[ 0.3 -0.5  0.3  0.3  0.3]
==========DICCIONARIO DE PESOS==========
{'[0.1 0.1 0.1 0.1 0.1]': 1, '[ 0.1 -0.3  0.1  0.1 -0.3]': 1, '[-0.3  0.1  0.1  0.1  0.1]': 1, '[ 0.7 -0.9  0.3  0.3  0.3]': 1, '[ 0.1 -0.3  0.5  0.5  0.5]': 1, '[ 0.3 -0.5  0.3  0.3  0.3]': 21, '[
 0.1 -0.3  0.1  0.1  0.1]': 20, '[-0.5  0.3  0.3  0.3  0.3]': 20}
==========PRECISIÓN==========
80.0%
```

Finalmente podemos observar que a pesar de haber usado el mismo dataset en las últimas dos pruebas, si cambiamos el coeficiente de aprendizaje, la precisión varía. Para mejorar la precisión de nuestro modelo habría que hallar la mejor combinación de Alpha con nuestras entradas. Probablemente otros modelos o incluso otras implementaciones de éste mismo pudieran ser más precisas, pero este definitivamente ha servido de aprendizaje para entender cómo funciona a detalle un modelo de aprendizaje.