Tecnológico de Monterrey

Momento de Retroalimentación: Módulo 2

Random Forest

Materia:

Inteligencia artificial avanzada para la ciencia de datos I

(Gpo 101)

Alumno:

Alfredo Azamar López - A01798100

Profesor:

Jorge Adolfo Ramírez Uresti

Tec de Monterrey Septiembre 2024

Para la implementación de una técnica de aprendizaje máquina sin el uso de un framework utilizaremos el algoritmo de aprendizaje de Hebb, herramienta que se utiliza dentro del campo de las redes neuronales para el aprendizaje automático. Nos proporciona un acercamiento base al entendimiento de cómo las neuronas aprenden conocimiento.

A continuación se explicara el código implementado para representar el algoritmo de Hebb:

Objetivo

Basándose en un conjunto de entradas y salidas, la función ajusta los pesos iterando sobre cada muestra del dataset, calculando la suma ponderada de las entradas y aplicando la fórmula de Hebb para actualizar los pesos.

Parámetros de Entrada

La función "**hebb_learning**" recibe:

- *num_inputs*: El número de entradas diferentes (o características) que se proporcionan al modelo. Ej. `num_inputs = 2`
- *inputs*: Una lista de listas que contiene los valores de las entradas para cada muestra en el dataset. Ej.
```
inputs = [
                    [-1, -1, 1, 1],  # x1
                    [-1, 1, -1, 1]]  # x2
```
- *output*: Un arreglo que contiene los valores de salida esperados para cada conjunto de entradas. Ej. `output = [-1,-1,-1,1]`
- *init_weights*: El valor inicial con el que se inician todos los pesos, incluyendo el peso del sesgo. Ej. `init_weights = 0`
- *bias_value*: El valor del sesgo que se usa en la función de activación para ajustar la salida. Ej. `bias_value = 1`

Procesamiento

~Inicialización de pesos~

{{ weights = [init_weights] * (num_inputs + bias_value) }}

Se crea una lista de los pesos iniciales de la red neuronal, toma el valor inicial de los pesos y lo multiplica por la suma del número de entradas y el valor del sesgo [esto para tomar en cuenta el peso del sesgo ($w0$)].

~Actualización de pesos~

```
{{
for i in range(len(inputs[0])):
        x = [1] + [inputs[j][i] for j in range(num_inputs)]
        y = output[i]

        for j in range(len(weights)):
            weights[j] += x[j] * y
}}
```

Se va recorriendo cada elemento de las entradas y de las salidas para ir calculando el cambio del peso del producto de la entrada y la salida esperada. Para después actualizar el peso en la lista “weights”. Este ajuste lo hace mediante la regla de Hebb.

~Genera ecuación final~

```
{{
equation = f"y = {weights[0]}"
    for i in range(1, len(weights)):
        equation += f" + ({weights[i]} * x{i})"
}}
```

Se construye un string que representa la ecuación de la red neuronal, inicia la ecuación con el peso que representa al sesgo, después realiza un ciclo for para obtener el valor de los siguientes pesos.

Salida

El código regresa los pesos actualizados y la ecuación generada.

```
Pesos finales: [-2, 2, 2]
Ecuación final: y = -2 + (2 * x1) + (2 * x2)
```

Nota

El algoritmo de Hebb no hace uso de un Dataset dividido en train, validation y test; porque el propósito del algoritmo era más teórico, buscando capturar una idea sencilla de cómo las neuronas podrían aprender y fortalecer sus conexiones basándose en la coactivación. Además que su enfoque se basa en resolver problemas linealmente separables, ajustando los pesos de esos datos sin la necesidad de considerar su generalización a datos no vistos.