ALGORITMO SOM
IMPLEMENTACIÓN EN CARACTERÍSTICAS DE ANIMALES

El algoritmo *SOM* o *Self.Organizing Map*, es una técnica de redes neuronales no supervisada que se utiliza principalmente para reducir la dimensionalidad y poder visualizar datos; su objetivo es *mapear* ciertos datos de alta dimensionalidad en un espacio de menor dimensionalidad (comúnmente 2D).

Pasos del funcionamiento del algoritmo:

- Se inicializa la rejilla y a cada neurona se le asigna un vector de pesos aleatorio de la misma dimensión que los vectores de entrada.
- Entrenamiento:
    - Se selecciona aleatoriamente un vector de datos de la entrada.
    - A la neurona ganadora, que tiene el vector de pesos más cercano al vector de entrada, se denomina *Best Matching Unit (BMU).*
    - Los pesos de la *BMU* y de las neuronas vecinas a la *BMU* se ajustan para que se acerquen más al vector de entrada.
    - La magnitud de la actualización depende de la tasa de aprendizaje y de la distancia entre la *BMU* y las neuronas vecinas.
    - Tanto la tasa de aprendizaje como el radio de influencia de las neuronas se van reduciendo, para ajustarse con el tiempo.

Los datos se visualizan en una rejilla en la que cada neurona tendrá un vector de pesos que representa un patrón determinado en los datos, y posteriormente, los datos de entrada se agruparán de manera que vectores con características similares se *mapeen* en neuronas cercanas entre sí en el mapa.

Aplicación en programa Python

Para comprobar el funcionamiento del algoritmo *SOM* con un determinado conjunto de datos, es este caso, se utilizó una tabla con diferentes características de animales, algunos con características muy similares, cada una con un valor determinado asignado al momento de cumplirse.

TABLE 9.3 Animal Names and Their Attributes

Animal		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

El primer paso que sigue el algoritmo es, después de importar las bibliotecas necesarias para determinadas funciones, declarar el arreglo con los nombres de los animales en la tabla, y desarrollar la matriz de las características y el valor; posteriormente, se normalizan los datos de las características con una función.

```python
import numpy as np
import matplotlib.pyplot as plt

animales = ['paloma', 'gallina', 'pato', 'ganso', 'buho', 'halcon', 'aguila', 'zorro',
            'perro', 'lobo', 'gato', 'tigre', 'leon', 'caballo', 'zebra', 'vaca']

caracteristicas = np.array([
    [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  # pequeño
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1],  # mediano
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0],  # grande
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],  # 2 patas
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],  # 4 patas
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],  # pelo
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],  # pezuñas
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],  # melena, creo
    [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  # plumas
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],  # caza
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],  # corre
    [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  # vuela
    [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]   # nada
]).T

def normalizar(data):
    min_val = np.min(data)
    max_val = np.max(data)
    return (data - min_val) / (max_val - min_val)

caracteristicasNormalizadas = normalizar(caracteristicas)
```

Como segundo paso se inicia la clase *SOM*, la cual contiene métodos necesarios para el aprendizaje y la aplicación del algoritmo, así como la búsqueda de la *neurona BMU*, y la actualización de los pesos. Siguiendo los pasos del algoritmo *SOM*, durante el desarrollo del programa, cada neurona se ajusta a uno de los vectores de características de los animales y se consideran los pesos.

```python
class SOM:
    def __init__(self, x, y, input_len, learning_rate=0.5, radius=None, iterations=10000):
        self.x = x
        self.y = y
        self.input_len = input_len
        self.tazaAprendizaje= learning_rate
        self.iterations = iterations
        self.radius = radius if radius else max(x, y) / 2
        self.pesos = np.random.random((x, y, input_len))

    def aprendizaje(self, data):
        for i in range(self.iterations):
            rand_i = np.random.randint(len(data))
            sample = data[rand_i]
            bmu = self.busqueda(sample)
            self.actualizarPesos(bmu, sample, i)

    def busqueda(self, sample):
        dist = np.linalg.norm(self.pesos - sample, axis=2)
        busquedaResultado = np.unravel_index(np.argmin(dist), dist.shape)
        return busquedaResultado

    def actualizarPesos(self, busquedaResultado, sample, iteration):
        tazaAprendizaje= self.tazaAprendizaje* (1 - iteration / self.iterations)
        radio = self.radius * (1 - iteration / self.iterations)
        for i in range(self.x):
            for j in range(self.y):
                distanciaBmu = np.linalg.norm(np.array([i, j]) - np.array(busquedaResultado))
                if distanciaBmu <= radio:
                    influence = np.exp(-distanciaBmu / (2 * (radio ** 2)))
                    self.pesos[i, j] += influence * tazaAprendizaje* (sample - self.pesos[i, j])
```

Una vez que se declara la clase principal, se cierra el programa con la aplicación de todos estos métodos sobre el conjunto de elementos de la tabla utilizada en el desarrollo, así como el proceso necesario para poder visualizar la rejilla de los resultados, y, utilizando de herramientas de las bibliotecas, generar la pestaña para poder visualizarlos en el mapa.

```
som = SOM(4, 4, caracteristicasNormalizadas.shape[1], iterations=10000)
som.aprendizaje(caracteristicasNormalizadas)

plt.figure(figsize=(10, 10))
for i in range(4):
    for j in range(4):
        plt.text(i, j, animales[np.argmin(np.linalg.norm(som.pesos[i, j] - caracteristicasNormalizadas, axis=1))],
                 ha='center', va='center', bbox=dict(facecolor='white', alpha=0.5))

plt.xlim(-0.5, 3.5)
plt.ylim(-0.5, 3.5)
plt.gca().invert_yaxis()
plt.show()
```

Por último, al ejecutar el programa, se genera una venta con la imagen del mapa resultante, en donde cada neurona es el nombre de un animal.