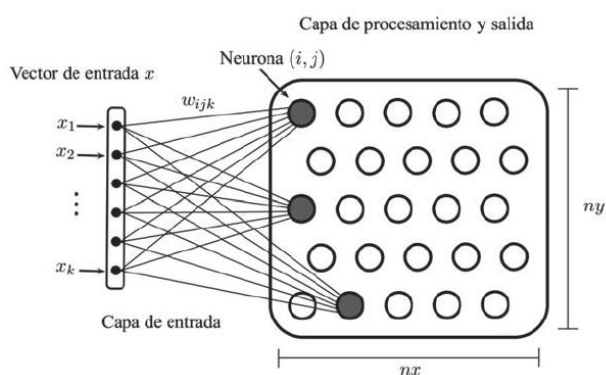




**Centro Universitario de Ciencias Exactas e
Ingenierías**
Universidad de Guadalajara



Práctica 8: SOM
Aprendizaje Máquina



Alumno: Samuel David Pérez Brambila

Código: 222966286

Profesora: Karla Ávila Cárdenas

Sección: D01

Fecha de Entrega: 17 de Noviembre de 2024

Introducción

Un mapa autoorganizado (Self-Organizing Map, SOM) “es un tipo de red neuronal no supervisada que organiza datos multidimensionales en una representación bidimensional o tridimensional, de manera que los datos similares se agrupen cercanamente en el mapa. Esto facilita la visualización y comprensión de patrones en grandes conjuntos de datos y la detección de relaciones entre ellos.” (Foqum, s.f.)

De acuerdo con Foqum (s.f.), la principal aplicación de los mapas autoorganizados (SOM) en la inteligencia artificial es la visualización y la interpretación de datos complejos y de alta dimensión. Los SOM son utilizados para reducir la dimensionalidad de los datos mientras mantienen sus relaciones topológicas, lo que significa que patrones similares en los datos de alta dimensión se mapean a posiciones cercanas en un mapa de dos dimensiones. Esta característica los hace particularmente útiles para la exploración de datos y el descubrimiento de agrupaciones o patrones ocultos en grandes conjuntos de datos. Son herramientas valiosas para el análisis exploratorio, permitiendo a los analistas y científicos de datos obtener insights sobre la estructura inherente y las correlaciones dentro de los datos, lo que puede ser difícil de visualizar y comprender en su forma original de alta dimensión.

Y es posible que surge la duda de ¿Cómo un mapa autoorganizado o SOM se diferencia de otras redes neuronales? Según Foqum (s.f.), los SOM se diferencian de otros tipos de redes neuronales principalmente en su objetivo y en la forma en que aprenden. Mientras que la mayoría de las redes neuronales están diseñadas para realizar tareas de predicción o clasificación, los SOM están orientados a la visualización y exploración de datos de alta dimensión. El aprendizaje en los SOM es no supervisado, es decir, no requieren datos etiquetados para el entrenamiento. En lugar de minimizar un error de predicción como en las redes supervisadas, los SOM aprenden a representar los datos de entrada en una topología de dos dimensiones preservando las propiedades topológicas de los datos originales. Esto significa que intentan asegurar que datos similares en el espacio de entrada sean mapeados a posiciones cercanas en el mapa. Además, los SOM utilizan una técnica de aprendizaje competitivo, donde las neuronas de la red compiten entre sí para representar los datos de entrada, y durante este proceso, la red se autoorganiza de manera que diferentes regiones del mapa se vuelven especializadas en diferentes patrones o características de los datos. Esto contrasta con el aprendizaje basado en gradientes de la mayoría de las redes neuronales, donde todas las neuronas se ajustan en conjunto durante el proceso de aprendizaje.

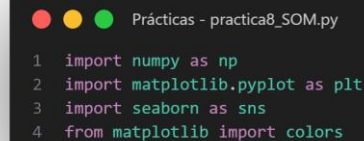
Es así, que en la siguiente práctica se implementará un algoritmo para construir un mapa autoorganizado para un determinado dataset. Para desarrollar este modelo,

emplearemos librerías adicionales como Numpy, Pandas, Matplotlib, entre otras, que facilitarán la carga, procesamiento y visualización de los resultados.

Contenido de la Actividad

El presente código implementa un Mapa Autoorganizado (SOM) para el conjunto de datos de pingüinos, que contiene tres especies diferentes de pingüinos (Adelie, Chinstrap y Gentoo) con cuatro características: longitud del pico, profundidad del pico, longitud de las aletas y masa corporal. Para probar el funcionamiento del SOM, se prueban tres configuraciones de tamaño de red (7x7, 10x10 y 13x13). Cada agrupación muestra la frecuencia de cada especie en cada nodo de la red, y un mapa de colores ayuda a visualizar la predominancia de cada especie en las celdas. Se añaden barras de color para representar la frecuencia de cada especie en cada neurona.

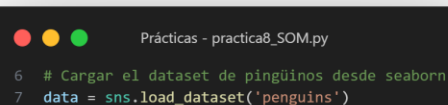
Importación de bibliotecas para tratamiento de datos



```
Prácticas - practica8_SOM.py
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 from matplotlib import colors
```

- numpy: Proporciona funciones para el manejo de datos en matrices, esenciales para cálculos vectoriales y la manipulación de pesos en el SOM.
- matplotlib.pyplot: Facilita la creación de gráficos y visualización de resultados.
- seaborn: Permite cargar conjuntos de datos de muestra como penguins.
- matplotlib.colors: Ayuda en la creación de escalas de colores, utilizadas aquí para diferenciar visualmente la frecuencia de cada especie de pingüino en la cuadrícula SOM.

Carga del dataset



```
Prácticas - practica8_SOM.py
6 # Cargar el dataset de pingüinos desde seaborn
7 data = sns.load_dataset('penguins')
```

Carga el conjunto de datos penguins desde seaborn, que contiene información sobre tres especies de pingüinos.

Preprocesamiento de datos

```
Prácticas - practica8_SOM.py
9  # Preprocesar los datos: eliminar filas con valores nulos y seleccionar características
10 data = data.dropna()
11 features = data[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
12 labels = data['species']
```

- data.dropna(): Elimina filas con valores nulos, evitando errores durante el entrenamiento.
- features: Selecciona las columnas de características (bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g) y las almacena en features.
- labels: Almacena las etiquetas (species) que indican la especie de cada pingüino.

Normalización de características

```
Prácticas - practica8_SOM.py
14 # Normalizar las características
15 normalized_features = (features - features.mean()) / features.std()
```

Escala los datos a una media de 0 y desviación estándar de 1, lo que ayuda a que el entrenamiento SOM sea más efectivo al igualar las escalas de todas las características.

Parámetros del mapa autoorganizado (SOM)

```
Prácticas - practica8_SOM.py
17 # Parámetros para tres tamaños de SOM
18 som_sizes = [(7, 7), (10, 10), (13, 13)] # Tamaños de SOM a probar
19 input_dim = normalized_features.shape[1] # Dimensiones de entrada
20
21 # Definir color para cada especie
22 color_map = {'Adelie': 'red', 'Chinstrap': 'green', 'Gentoo': 'blue'}
```

- som_sizes: Define tres tamaños de red SOM a probar: 7x7, 10x10 y 13x13.
- input_dim: Guarda la cantidad de características (4) del conjunto de datos normalizado, que será el número de entradas de cada nodo.
- color_map: Asigna colores específicos a cada especie de pingüino para distinguirlas en la visualización.

Inicialización de pesos de las neuronas

```
Prácticas - practica8_SOM.py
24 for som_x, som_y in som_sizes:
25     # Inicializar los pesos de cada neurona en la cuadrícula con valores aleatorios entre 0 y 1
26     weights = np.random.rand(som_x, som_y, input_dim)
```

Para cada tamaño de red en `som_sizes`, se inicializan aleatoriamente los pesos de las neuronas en una cuadrícula de tamaño `(som_x, som_y)`, y `input_dim` determina el número de valores en cada vector de peso.

Función `find_bmu` (neurona ganadora o BMU)

```
Prácticas - practica8_SOM.py
28 # Función para encontrar la neurona ganadora (BMU)
29 def find_bmu(input_vector, weights):
30     distances = np.sqrt(((weights - input_vector) ** 2).sum(axis=2))
31     bmu_index = np.unravel_index(np.argmin(distances, axis=None), distances.shape)
32     return bmu_index
```

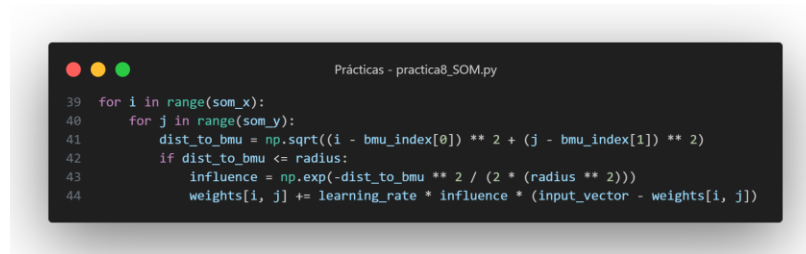
- Calcula la distancia euclidiana entre cada neurona (o nodo) y el `input_vector` (vector de características de un pingüino).
- `np.sqrt(((weights - input_vector) ** 2).sum(axis=2))`: Resta el `input_vector` a cada vector de peso en `weights`, eleva al cuadrado cada elemento y calcula la raíz cuadrada de la suma para obtener la distancia euclidiana.
- `np.argmin(distances, axis=None)`: Encuentra el índice de la neurona con la menor distancia (BMU).
- `np.unravel_index`: Convierte este índice en una coordenada `(x, y)` dentro de la cuadrícula `weights`.
- La función devuelve `bmu_index`, la coordenada de la neurona ganadora.

Función `update_weights` (actualización de pesos)

```
Prácticas - practica8_SOM.py
34 # Función para actualizar los pesos
35 def update_weights(input_vector, weights, bmu_index, t, max_iter, init_learning_rate=0.5, init_radius=3.0):
36     learning_rate = init_learning_rate * (1 - t / max_iter)
37     radius = init_radius * (1 - t / max_iter)
```

- Calcula una tasa de aprendizaje dinámica y un radio de influencia decrecientes, ajustando ambos a medida que `t` se aproxima a `max_iter` (total de iteraciones).

- learning_rate: La tasa de aprendizaje inicial se reduce linealmente con cada iteración.
- radius: El radio de influencia se reduce de manera similar, limitando la influencia de la BMU en otras neuronas cercanas a medida que la red converge.



```

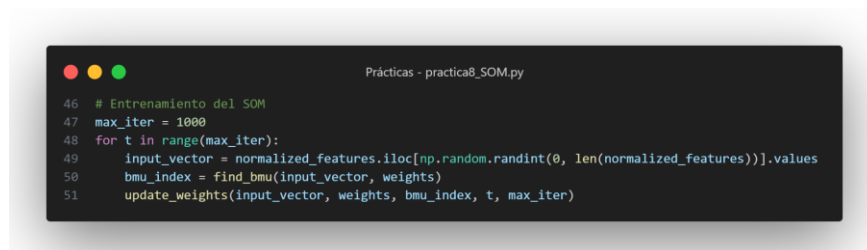
Prácticas - practica8_SOM.py

39 for i in range(som_x):
40     for j in range(som_y):
41         dist_to_bmu = np.sqrt((i - bmu_index[0]) ** 2 + (j - bmu_index[1]) ** 2)
42         if dist_to_bmu <= radius:
43             influence = np.exp(-dist_to_bmu ** 2 / (2 * (radius ** 2)))
44             weights[i, j] += learning_rate * influence * (input_vector - weights[i, j])

```

- Para cada neurona en la red, calcula la distancia euclidiana entre la neurona y la BMU.
- Si la distancia a la BMU es menor que radius, la neurona se actualiza.
- influence: Define el grado de cambio en función de la distancia a la BMU, decayendo exponencialmente.
- Los pesos de la neurona se ajustan en función de la tasa de aprendizaje y el grado de influencia.

Entrenamiento del SOM



```

Prácticas - practica8_SOM.py

46 # Entrenamiento del SOM
47 max_iter = 1000
48 for t in range(max_iter):
49     input_vector = normalized_features.iloc[np.random.randint(0, len(normalized_features))].values
50     bmu_index = find_bmu(input_vector, weights)
51     update_weights(input_vector, weights, bmu_index, t, max_iter)

```

- max_iter: Define el número total de iteraciones.
- En cada iteración, selecciona un `input_vector` aleatorio del conjunto de datos y encuentra la neurona ganadora (`bmu_index`).
- Luego, llama a `update_weights` para ajustar los pesos de la red.

Preparación para la visualización

```
Prácticas - practica8_SOM.py
53 print(f"Entrenamiento completo para SOM de tamaño {som_x}{som_y}.")
54
55 # Crear un mapa de colores basado en la frecuencia de cada especie en cada neurona
56 frequency_map = np.zeros((som_x, som_y))
57 species_map = np.empty((som_x, som_y), dtype=object)
58 species_frequency = {species: np.zeros((som_x, som_y)) for species in color_map.keys()}
```

- frequency_map: Registra la cantidad de vectores que se asignan a cada neurona.
- species_map: Guarda las especies asignadas a cada neurona.
- species_frequency: Almacena un mapa de frecuencia para cada especie en cada neurona.

Asignación de especies a cada neurona en función de sus características

```
Prácticas - practica8_SOM.py
60 for idx in range(len(normalized_features)):
61     input_vector = normalized_features.iloc[idx].values
62     bmu_index = find_bmu(input_vector, weights)
63     if species_map[bmu_index] is None:
64         species_map[bmu_index] = labels.iloc[idx]
65     else:
66         species_map[bmu_index] += f", {labels.iloc[idx]}"
67     frequency_map[bmu_index] += 1
68     species_frequency[labels.iloc[idx]][bmu_index] += 1
```

- Para cada vector, encuentra la BMU y aumenta la cuenta en frequency_map.
- species_map almacena las especies observadas en cada neurona, y species_frequency incrementa el conteo de cada especie.

Creación del gráfico SOM

```
Prácticas - practica8_SOM.py
70 # Visualización del SOM
71 fig, ax = plt.subplots(figsize=(10, 10))
```

Inicializa una figura de 10x10 para la visualización.

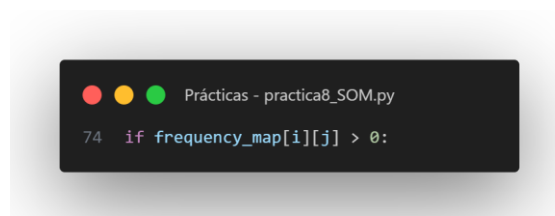
Visualización detallada

Iteración sobre la cuadrícula



Este bloque inicia dos bucles for anidados que recorren cada celda de la cuadrícula SOM. i representa la columna (eje X) y j representa la fila (eje Y) de la red SOM.

Verificación de frecuencia



Se verifica si hay al menos un pingüino asignado a la neurona en la posición (i, j). frequency_map[i][j] contiene el número de datos (pingüinos) que se han asignado a esta celda.

Inicialización del conteo de especies



Se inicializa un diccionario species_count para llevar el registro de la cantidad de pingüinos de cada especie en la celda actual. Las claves del diccionario son las especies de pingüinos y los valores comienzan en 0.

Contar pingüinos por especie en la celda

```
Prácticas - practica8_SOM.py
76 for idx in range(len(normalized_features)):
77     input_vector = normalized_features.iloc[idx].values
78     if find_bmu(input_vector, weights) == (i, j):
79         species_count[labels.iloc[idx]] += 1
```

- Se itera a través de cada pingüino en el conjunto de datos normalizado.
- Se obtiene el input_vector correspondiente al pingüino en la posición idx.
- Se llama a find_bmu(input_vector, weights) para encontrar la neurona ganadora (BMU) para ese vector de entrada.
- Si la BMU coincide con la celda actual (i, j), se incrementa el contador de la especie correspondiente en species_count.

Determinación de la especie predominante

```
Prácticas - practica8_SOM.py
80 predominant_species = max(species_count.items(), key=lambda x: x[1])[0]
```

Se utiliza max para encontrar la especie que tiene el mayor conteo en species_count.

- species_count.items() devuelve un conjunto de tuplas (especie, conteo).
- key=lambda x: x[1] indica que el máximo se determina por el segundo elemento de cada tupla (el conteo).
- Se obtiene solo el nombre de la especie predominante.

Cálculo de la intensidad del color

```
Prácticas - practica8_SOM.py
81 color_intensity = min(frequency_map[i][j] / frequency_map.max(), 1)
```

Se calcula la intensidad del color que se aplicará a la celda:

- Se obtiene el porcentaje de pingüinos en la celda actual dividiendo la frecuencia en frequency_map[i][j] por el máximo valor en frequency_map.

- Se asegura que el valor esté limitado a un máximo de 1 con `min(..., 1)` para evitar problemas de saturación de color.

Dibujo del rectángulo en la celda



```

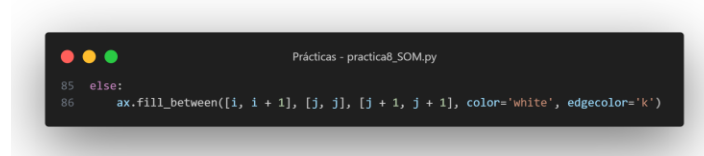
Prácticas - practica8_SOM.py
82 ax.fill_between([i, i + 1], [j, j], [j + 1, j + 1],
83                 color=color_map[predominant_species], alpha=color_intensity,
84                 edgecolor='k')

```

Se dibuja un rectángulo que representa la celda (i, j) en el gráfico:

- `fill_between` crea un área rellena entre las coordenadas especificadas.
- `color=color_map[predominant_species]` establece el color del rectángulo basado en la especie predominante.
- `alpha=color_intensity` aplica la intensidad de color calculada.
- `edgecolor='k'` establece el borde del rectángulo en negro.

Manejo de celdas vacías



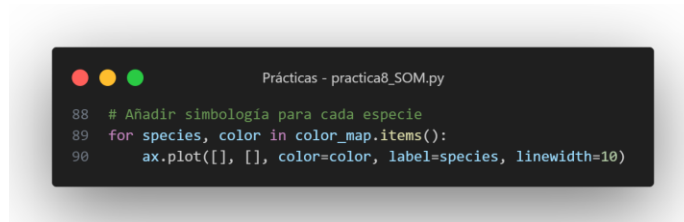
```

Prácticas - practica8_SOM.py
85 else:
86     ax.fill_between([i, i + 1], [j, j], [j + 1, j + 1], color='white', edgecolor='k')

```

Si no hay pingüinos en la celda (es decir, `frequency_map[i][j]` es 0), se dibuja un rectángulo blanco para indicar que la celda está vacía.

Añadir simbología para cada especie



```

Prácticas - practica8_SOM.py
88 # Añadir simbología para cada especie
89 for species, color in color_map.items():
90     ax.plot([], [], color=color, label=species, linewidth=10)

```

Este bloque itera sobre el diccionario `color_map`, que asocia cada especie de pingüino con un color específico.

- `ax.plot([], [], ...)` se utiliza para crear líneas invisibles (sin puntos) en el gráfico.
- `color=color` establece el color de la línea que representa a la especie.

- `label=species` asigna una etiqueta a la línea para que aparezca en la leyenda.
- `linewidth=10` establece el grosor de la línea.

Configuración de la leyenda

```
Prácticas - practica8_SOM.py
91 ax.legend(loc='upper right', title="Especie", bbox_to_anchor=(1.12, 1))
```

Se configura la leyenda del gráfico:

- `loc='upper right'` posiciona la leyenda en la esquina superior derecha.
- `title="Especie"` establece un título para la leyenda.
- `bbox_to_anchor=(1.12, 1)` ajusta la posición de la leyenda para que no se superponga con el gráfico.

Crear barras de color para cada especie

```
Prácticas - practica8_SOM.py
93 # Crear barras de color para cada especie a la izquierda de la cuadrícula
94 cbar_ax_adelie = fig.add_axes([0.1, 0.15, 0.02, 0.7])
95 cbar_ax_chinstrap = fig.add_axes([0.15, 0.15, 0.02, 0.7])
96 cbar_ax_gentoo = fig.add_axes([0.2, 0.15, 0.02, 0.7])
```

Se crean tres ejes adicionales (`cbar_ax_adelie`, `cbar_ax_chinstrap`, `cbar_ax_gentoo`) en el gráfico para las barras de color que representan a cada especie.

- `fig.add_axes(...)` permite especificar la posición y el tamaño de cada barra de color mediante un arreglo `[x, y, width, height]`.

Normalización de datos para la barra de colores

```
Prácticas - practica8_SOM.py
97 norm = colors.Normalize(vmin=0, vmax=frequency_map.max())
```

Se normalizan los datos de frecuencia en `frequency_map` para que se puedan asignar colores de manera adecuada en las barras de color.

- `vmin=0` y `vmax=frequency_map.max()` establecen el rango de valores que se utilizarán para mapear los colores.

Creación de las barras de color

```
Prácticas - practica8_SOM.py
98 plt.colorbar(plt.cm.ScalarMappable(norm=norm, cmap='Reds'), cax=cbar_ax_adelie, label='Adelie')
99 plt.colorbar(plt.cm.ScalarMappable(norm=norm, cmap='Greens'), cax=cbar_ax_chinstrap, label='Chinstrap')
100 plt.colorbar(plt.cm.ScalarMappable(norm=norm, cmap='Blues'), cax=cbar_ax_gentoo, label='Gentoo')
```

Se crean las barras de color para cada especie utilizando `plt.colorbar(...)`:

- `plt.cm.ScalarMappable(norm=norm, cmap='Reds')` crea un mapeo de color basado en la normalización y el colormap especificado (en este caso, 'Reds' para los pingüinos Adelie).
- `cax=cbar_ax_adelie` especifica el eje en el que se dibujará la barra de color.
- `label='Adelie'` agrega una etiqueta a la barra de color.

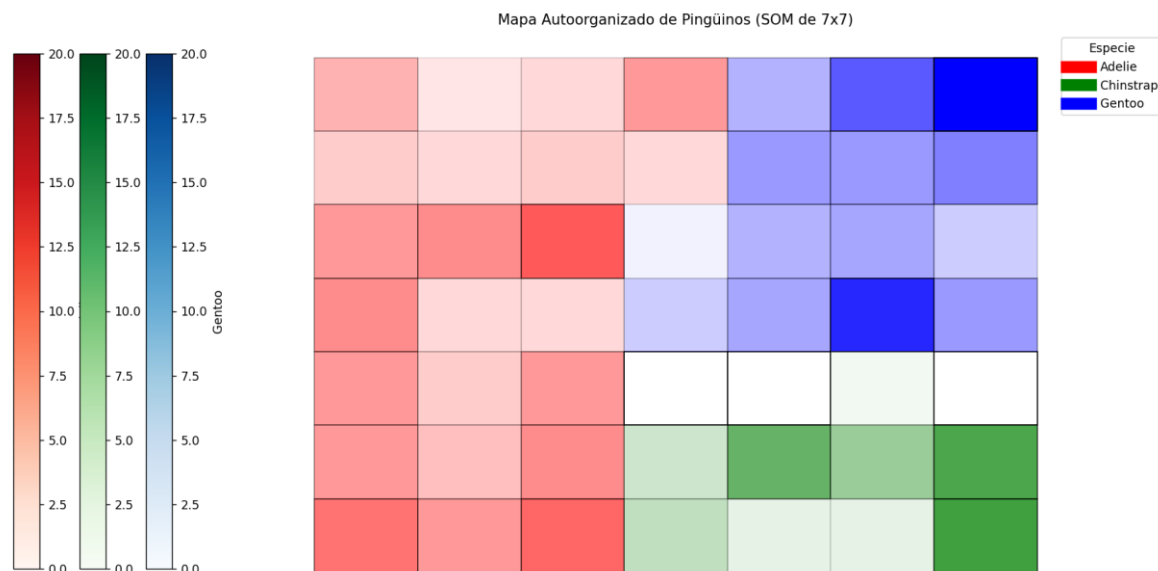
Configuración del título y ajustes finales

```
Prácticas - practica8_SOM.py
102 ax.set_title(f"Mapa Autoorganizado de Pingüinos (SOM de {som_x}x{som_y})")
103 ax.axis('off')
104 plt.subplots_adjust(left=0.3)
105 plt.show()
```

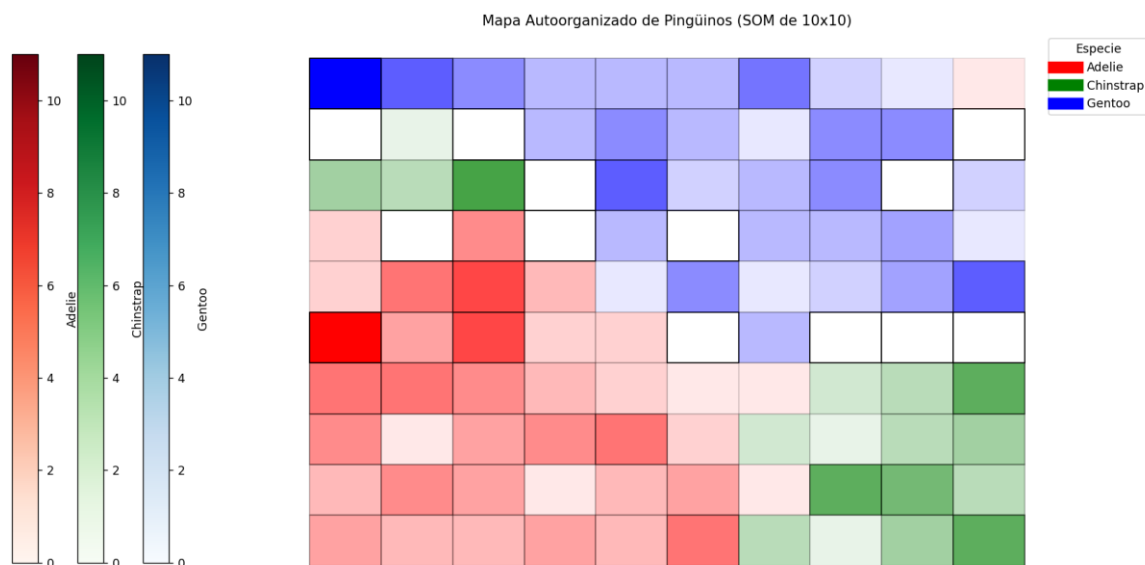
- `ax.set_title(...)` establece el título del gráfico, incluyendo las dimensiones del SOM.
- `ax.axis('off')` oculta los ejes para que la atención se centre en el gráfico en sí.
- `plt.subplots_adjust(left=0.3)` ajusta el espaciado del gráfico para dar más espacio a las leyendas y barras de color a la izquierda.
- `plt.show()` muestra el gráfico final en la pantalla.

Obteniendo así los siguientes resultados, con los 3 tamaños de SOM previamente definidos para que haga el proceso de manera automática:

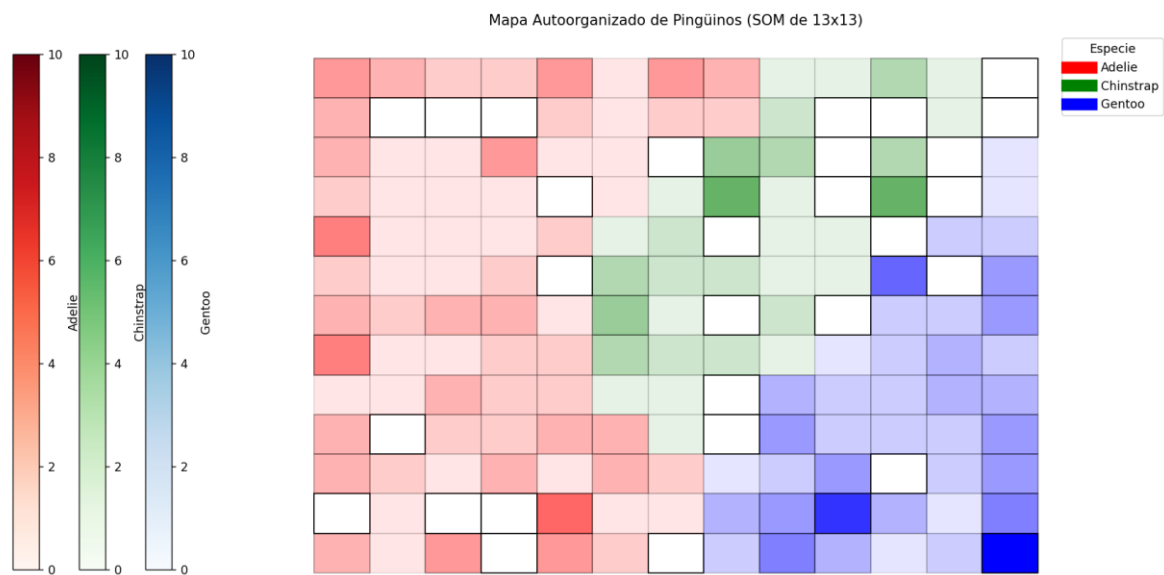
Entrenamiento completo para SOM de tamaño 7x7.



Entrenamiento completo para SOM de tamaño 10x10.



Entrenamiento completo para SOM de tamaño 13x13.



Conclusiones

Al concluir la práctica de implementación del algoritmo de Mapa Autoorganizado (SOM) en Python, se ha evidenciado que esta técnica es una herramienta poderosa para abordar problemas de clasificación no supervisada. Durante el proceso, se destacó la importancia de seleccionar adecuadamente la dimensión de la cuadrícula del SOM, ya que este parámetro impacta directamente en la capacidad del modelo para representar la estructura subyacente de los datos. Al experimentar con diferentes dimensiones (7x7, 10x10 y 13x13), se pudo observar cómo cada configuración afecta la organización y utilización de cada neurona.

La preparación adecuada de los datos fue fundamental, enfatizando la normalización de características y la selección de atributos relevantes, lo que influyó significativamente en el rendimiento del SOM. Además, la identificación de los Best Matching Units (BMUs) y la visualización de las frecuencias de activación de cada neurona permitió una comprensión más profunda de las relaciones entre los grupos, lo que resulta esencial para la interpretación de los resultados obtenidos.

Las visualizaciones gráficas facilitaron la identificación de las especies en el SOM y sus correspondientes frecuencias, proporcionando una representación clara de cómo se agrupan los datos en el espacio de características. Esto fue particularmente útil para observar la distribución de las distintas especies en el conjunto de datos de Penguins de Searborn y cómo el SOM las clasifica, mostrando la capacidad del modelo para capturar la complejidad de los datos.

En resumen, esta práctica ha demostrado que el Mapa Autoorganizado (SOM) es un enfoque robusto para la agrupación de datos en un contexto no supervisado. Se enfatizó la importancia de la elección adecuada de las dimensiones de la cuadrícula y la visualización de resultados. La experiencia adquirida ha sido fundamental para comprender mejor los fundamentos del aprendizaje no supervisado, ofreciendo una base sólida para futuros proyectos en el ámbito de la ciencia de datos y el aprendizaje automático. Herramientas como NumPy y Matplotlib han sido cruciales en el desarrollo del modelo, subrayando su utilidad en el análisis y visualización de datos.

Referencias

- Foqum. (s.f.). *Mapa autoorganizado (Self-organizing map)*. Foqum IO. Recuperado el 31 de Octubre de 2024 de: <https://foqum.io/blog/termino/mapa-autoorganizado-self-organizing-map/>