SISTEMAS INTELIGENTES I

Práctica 2: Análisis y Diseño de Arquitecturas Neuronales No Supervisadas (Self-Organizing Map) 2020/21

27 de enero de 2021

José María Amusquívar Poppe | Prashant Jeswani Tejwani

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería en Informática

Índice

Estudio del Funcionamiento, Topología y Neurodinámica de la Red SOM

La red SOM es un tipo de red neuronal artificial, que es entrenada usando aprendizaje no supervisado para producir una representación discreta del espacio de las muestras de entrada, llamado mapa. Los mapas autoorganizados usan una función de vecindad para preservar las propiedades topológicas del espacio de entrada y son útiles para visualizar vistas de baja dimensión de datos de alta dimensión.

El mapa autoorganizado está formado por componentes llamadas neuronas. Asociado con cada neurona hay un vector de pesos, de la misma dimensión de los vectores de entrada, y una posición en el mapa. La configuración de las neuronas es, normalmente, un espacio de dos dimensiones en una rejilla rectangular.

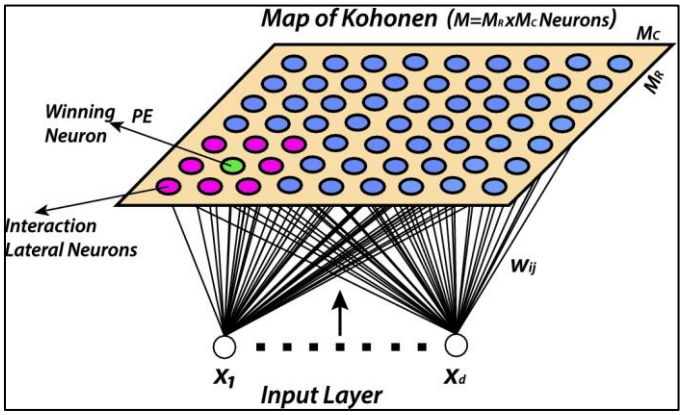

Figura 1. Arquitectura SOM (1)

El objetivo del aprendizaje es provocar que diferentes partes de la red respondan similarmente a ciertos patrones de la entrada (esto es parcialmente motivado por el manejo en partes separadas de la corteza cerebral del cerebro humano de la información sensorial, como la visual y la auditiva).

Los pesos de las neuronas son inicializados con valores aleatorios y el entrenamiento utiliza aprendizaje competitivo. Cuando un dato de entrada es presentado a la red, su distancia euclidiana a todos los vectores de pesos es calculada. La neurona cuyo vector de pesos es más similar a la entrada es llamada unidad de mejor correspondencia o neurona ganadora (BMU). Los pesos del BMU y las neuronas cercanas al mismo en la cuadrícula del SOM son ajustados hacia el vector de entrada. La magnitud de los cambios se decrementa con el tiempo y con la distancia desde el BMU.

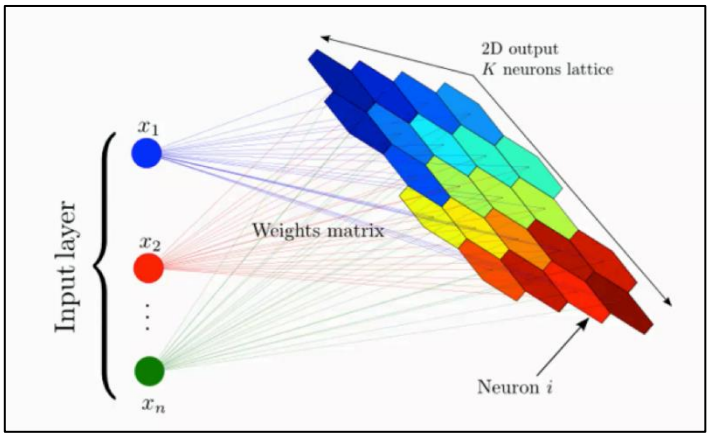

Figura 2. Arquitectura SOM (2)

Además, la función de vecindad se contrae con el tiempo, es decir, al inicio, cuando la vecindad es completa, la autoorganización tiene lugar a escala global, pero cuando la vecindad ha sido ajustada a solo unas cuantas neuronas, los pesos irán convergiendo a estimaciones locales.

El aprendizaje se puede resumir de la siguiente manera:

1. Hacer un mapa de neuronas con vectores de pesos aleatorios
2. Tomar un vector de entrada D(t)
    1. Iterar por cada neurona del mapa
        - Calcular la distancia euclídea entre el vector de entrada y los vectores de pesos de las neuronas del mapa
        - Mantener la neurona que ha tenido la menos distancia, esta neurona será el BMU
    2. Actualizar las neuronas en la vecindad del BMU:

$$W_v(s+1) = W_v(s) + \Theta(u, v, s)\alpha(s)(D(t) - W_v(s))$$

3. Incrementar ***s*** y volver al paso 2, mientras que ***s*** < número de iteraciones

Análisis del conjunto de datos

Se ha hecho uso del conjunto de datos: **Análisis y Clasificación de la criminalidad en la ciudad de Chicago en el año 2018**. El conjunto de datos se compone de las siguientes características:

ID, Case Number
Date
Block
IUCR
Primary Type
Description
Location Description
Arrest
Domestic
Beat
District
Ward
Community Area
FBI Code
X Coordinate
Y Coordinate
Year
Updated

Figura 3. Características de los datos.

Se ha preprocesado los datos de la siguiente manera:

Se han eliminado las columnas *ID* y *Case Number* ya que son valores únicos. También se ha eliminado la columna *Update* ya que no aporta mucha información al conjunto de datos.

Además, se han eliminado otras columnas que aportan información repetitiva como *Year* (todas las observaciones son del 2018) y *Location* (es una concatenación de la columna *X Coordinate*, *Y Coordinate*).

Con la columna Date, se extraído el día, mes y la hora y se han formado tres nuevas columnas, eliminando así la columna *Date*.

Finalmente, las columnas de tipo no numéricas se han realizado un *decode*, es decir, se ha mapeado los valores a un diccionario para así convertir las columnas en numéricas.

A continuación, se visualizan algunas de las características importantes de los datos para entender el conjunto de datos:

1. ¿Qué tipos de delitos conllevan a más arrestos?

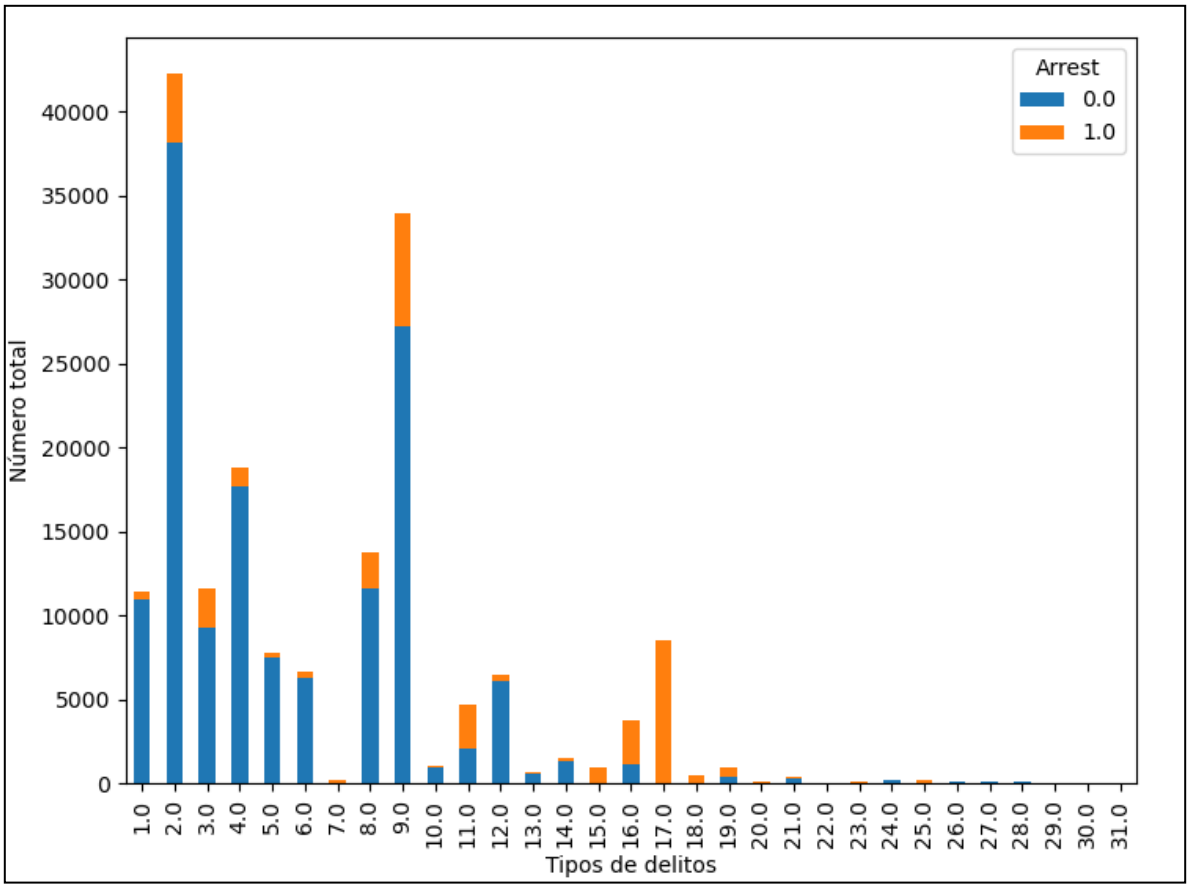

Figura 4. Comparativa entre delitos cometidos y arrestos.

Si se representa el número de veces que se ha cometido cada tipo de delito, siendo el eje X los tipos de delitos, cada tipo representado por un valor y el eje Y, el número de delitos. Observamos que el tipo de arresto más común es el 2 (el cual corresponde a *THEFT* con 4112 arrestos), el cual le sigue el 9 (delito de tipo *BATTERY* con 6727 arrestos).

Pero el delito que siempre termina siendo arrestado (8487 arrestos) es el 17 que corresponde con el crimen de tipo *NARCOTICS*.

2. ¿En qué distritos suceden más delitos?

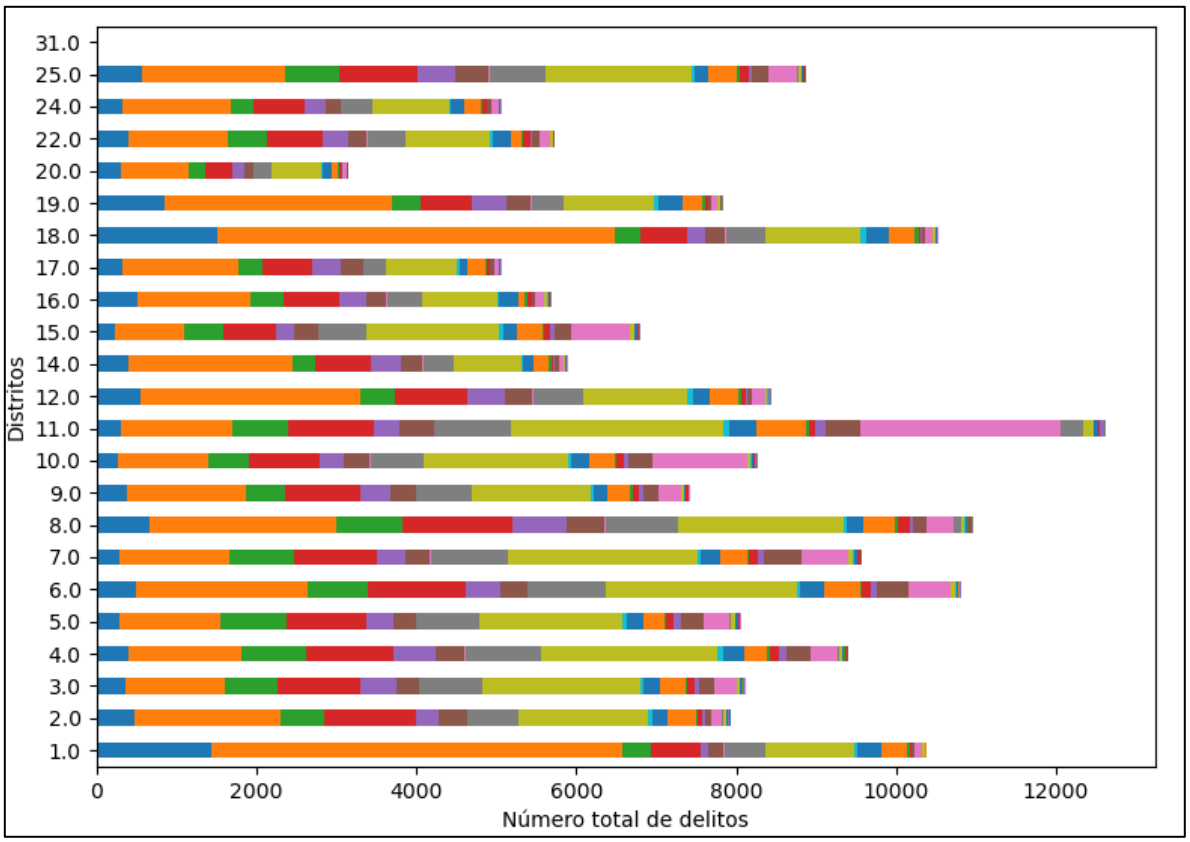

Figura 5. Distritos en los que suceden más delitos.

Los distritos en el conjunto de datos son representados por tres dígitos. Observamos que en el distrito en el cual se producen más delitos es en el distrito 011 con más de 12.000 delitos, el cual le sigue los distritos 018 y 001 con más de 10.000 delitos.

3. ¿Qué tipo de delito es el más común cada mes?

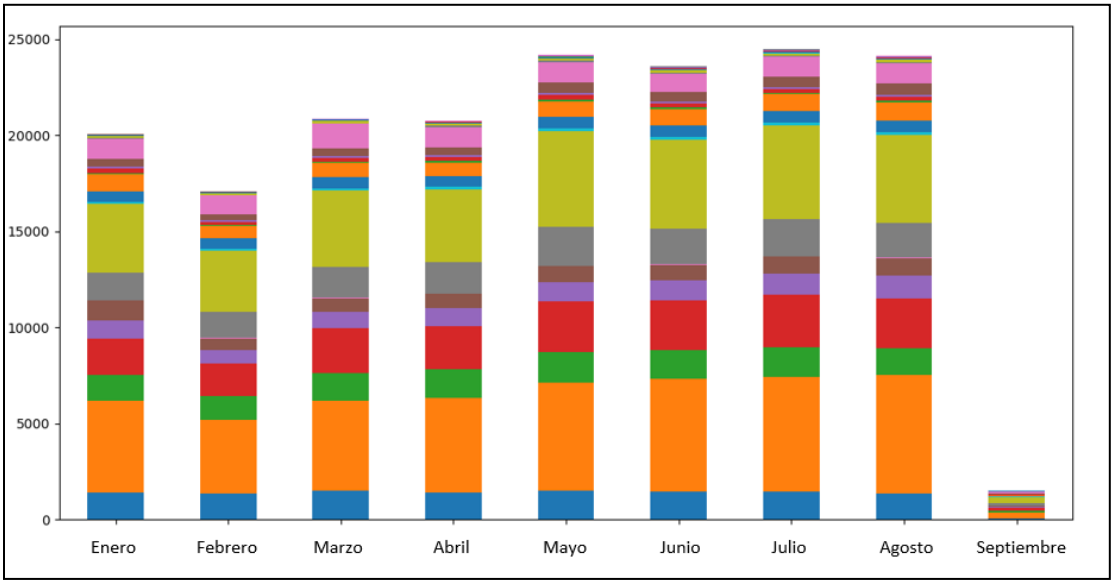

Figura 6. Comparativa entre tipo de delitos y el mes en el que se cometieron.

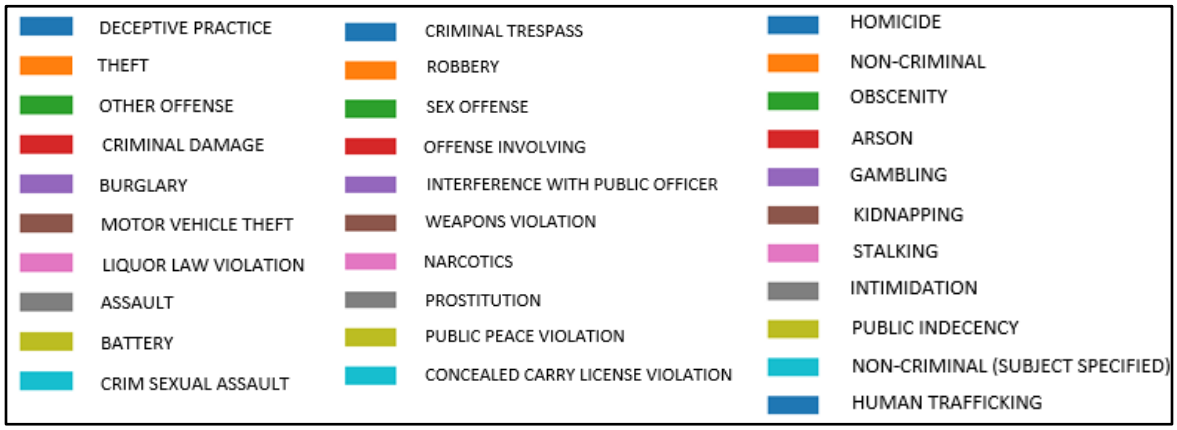

Figura 7. Etiquetas transcritas de número a texto.

Finalmente, si se representa el tipo de crimen según los meses del año, observamos que:

- Enero: el delito más común es THEFT (4783 casos).
- Febrero: el delito más común es THEFT (3853 casos).
- Marzo: el delito más común es THEFT (4689 casos).
- Abril: el delito más común es THEFT (4925 casos).
- Mayo: el delito más común es THEFT (5628 casos).
- Junio: el delito más común es THEFT (5898 casos).
- Julio: el delito más común es THEFT (6000 casos).
- Agosto: el delito más común es THEFT (6197 casos).
- Septiembre: el delito más común es BATTERY (331 casos).

Desarrollo en Python y TensorFlow

Tal como se mencionó anteriormente, para la realización de esta práctica se ha empleado el "dataset" de criminalidad en Chicago, sin embargo, como medida de comprobación del correcto funcionamiento del código, se han empleado otros "datasets", tales como uno de colores "RGB" o el "dataset" de "Iris".

Dado que se ha realizado un "decode" a los datos de tipo texto, se ha empleado un diccionario para el mapeo de estos datos a tipos numéricos, tal como se representa en la siguiente figura:

```
values_dict = dict()
for col in df.columns:
    if isinstance(df[col][0], str):
        aux = df[col].unique()
        d = dict(enumerate(aux.flatten(), 1))
        d = dict((V, K) for K, V in d.items())
        values_dict[col] = d
        df[col] = df[col].map(d)
```

Figura 8. Mapeo de atributos tipo texto a valores numéricos.

El paso siguiente, una vez se han leído los datos y almacenado como un "dataframe", es coger una porción aleatoria de estos datos, para poder procesarlos y ejecutarlos en la red "SOM" desarrollada. Se ha cogido, únicamente, una porción de los datos puesto que el número de observaciones total del conjunto de datos es muy elevado, y ralentizaría el aprendizaje a la vez que acapararía todo el uso de la memoria.

```
df, df_dict = process_data(path_crime)
df = df.sample(frac=self.batch_data)
```

Figura 9. La variable "batch_data" representa la porción.

PCA

Una vez los datos han sido almacenados en un array "numpy", se procede a obtener el "PCA" de estos datos. Se realiza este procesamiento puesto que el conjunto de datos posee muchas características, lo que requiere de múltiples dimensiones para ser representados.

El "PCA" o análisis de componentes principales es una técnica de aprendizaje no supervisado, cuyo principal objetivo es la de reducir las dimensiones de los datos perdiendo la menor cantidad de información posible. Cada componente de la nueva matriz se corresponde con una componente principal y que son el resultado de una combinación lineal de las variables originales, y, además, son independientes o no relacionadas entre sí.

Pues bien, se ha desarrollado una clase en Python que se encarga de realizar el “PCA” de los datos, obteniendo como resultado la nueva matriz de componentes principales, además de poder obtener los porcentajes de aportación de información de cada componente. Esta clase está desarrollada empleando la librería “Tensorflow” y, empleando el uso de autovectores y autovalores para el cálculo de la matriz “PCA”.

```
def compute_PCA(self):
    # Encuentra autovectores y autovalores
    self.eigen_values, self.eigen_vectors = tf.linalg.eigh(self.data_cov)

    # Se reordenan los autovalores en forma decreciente, al igual que los autovectores
    sorted_index = tf.argsort(self.eigen_values)[::-1]
    self.eigen_values = tf.gather(self.eigen_values, sorted_index)
    self.eigen_vectors = tf.gather(self.eigen_vectors, sorted_index, axis=1)
```

Figura 10. Función que realiza el "PCA" de la matriz de covarianza.

Esta función devuelve una matriz cuyas dimensiones son idénticas a la matriz original, sin embargo, bastará con representar dos o tres columnas de esta nueva matriz para obtener un alto porcentaje de información y, consiguiendo así, reducir las dimensiones.

Aplicando “PCA” a los datos y cogiendo, únicamente, las tres primeras componentes, se obtiene un total de 99,9998% de información, es decir, lo que antes requeriría más de diez dimensiones para representar, ahora se puede hacer empleando sólo tres.

Finalmente se ha obtenido una matriz cuyas filas son las mismas que la porción de datos cogida, pero con sólo tres columnas. El siguiente paso es realizar el entrenamiento de la red “SOM”.

SOM

Puesto que “Tensorflow” ha sufrido varias actualizaciones entre la versión 1 y la versión 2, se requiere utilizar la función: “tf.compat.v1.disable_eager_execution()” para evitar ciertas advertencias y poder ejecutar el código con normalidad.

Se ha desarrollado una clase denominada “SOM”, la cual se encarga de procesar y entrenar la red con los datos que se han preparado usando “PCA”. En el constructor de la clase se reciben parámetros como la dimensión del mapa, la tasa de aprendizaje “alpha”, o el radio que determinada el rango de afectación a las neuronas vecinas “sigma”. En este constructor también se crea el mapa de neuronas, iniciando sus pesos aleatoriamente, así como las localizaciones de éstas.

Dentro de esta clase se tienen varios métodos que se encargan de generar las localizaciones de las neuronas, obtener el mapa de pesos, actualizar los pesos de neuronas aplicando la distancia euclídea, y un método que se encarga de entrenar la red aplicando los métodos antes descritos.

El método que se encarga de entrenar la red recibe como parámetro los datos de entrenamiento, el número de épocas y el número de observaciones disponibles en los datos. Este método realiza unas iteraciones igual al número de épocas definido, en la que, encada época, “alimenta” al mapa con el “dataset”, obligando, de este modo, a que estas neuronas modifiquen sus pesos y se adapten cada vez más al vector de entrada.

Una vez se ha entrenado la red “SOM”, ésta devuelve el mapa de pesos de las neuronas junto a las localizaciones de los vectores. Ambos parámetros serán de utilidad para poder representar los datos, empleando para ello la función “matplotlib.pyplot.imshow()”.

Tal como se mencionó anteriormente, se ha comprobado el correcto funcionamiento de la red empleando el “dataset” de “Iris”, la cual está compuesta por 4 características y una quinta columna que tiene el papel de clasificador. Se ha procedido a entrenar la red con este “dataset”, y se ha obtenido la siguiente imagen:

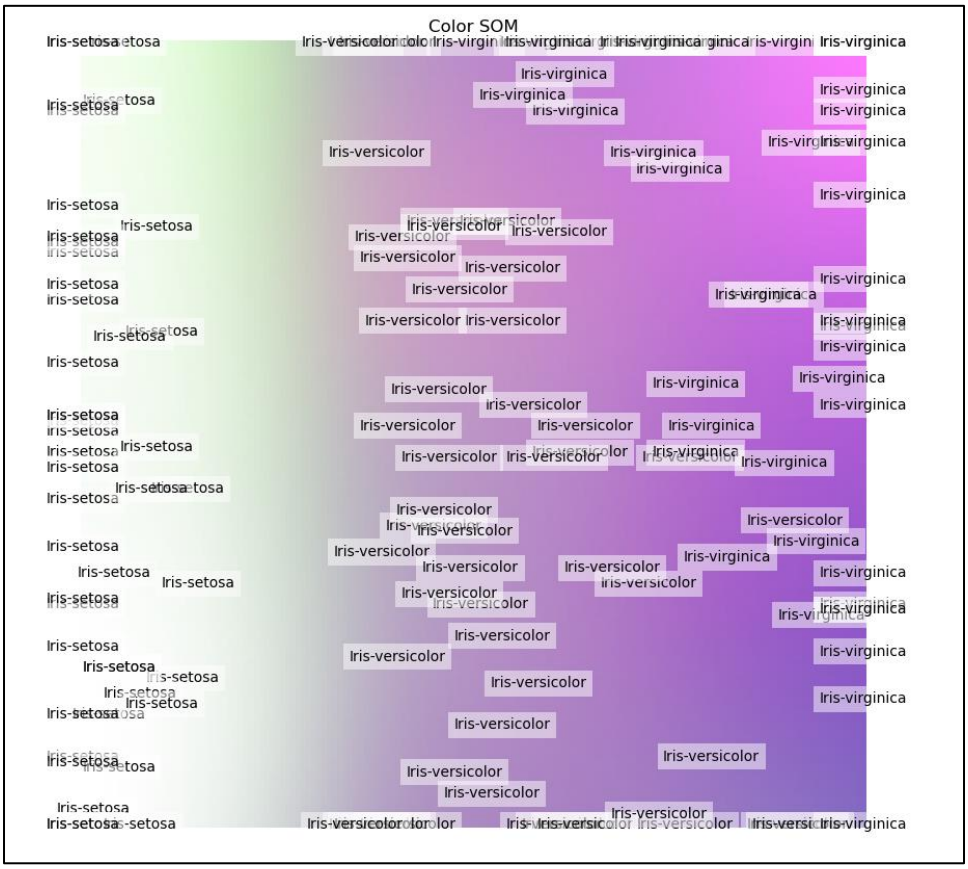

Figura 11. Clasificación de los distintos tipos de flores empleando la red "SOM".

Además de este “dataset”, se ha empleado uno mucho más simple compuesto de código de colores “RGB”, el cual posee 15 observaciones y 3 características, además de una cuarta columna que representa el color que representa.

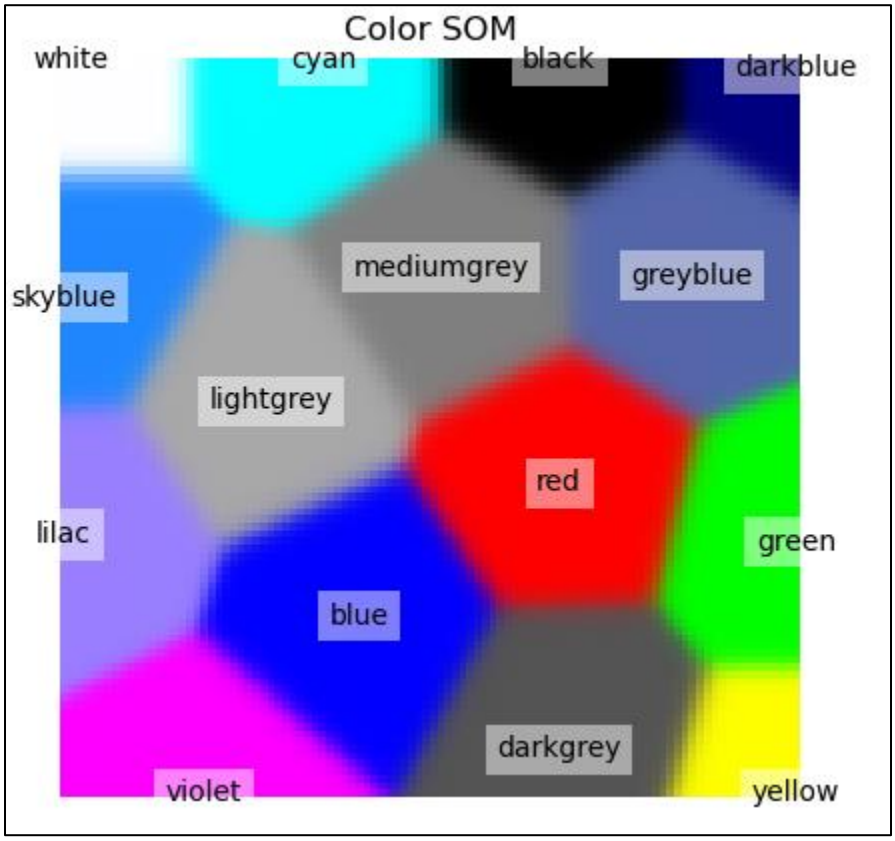

Figura 12. Mapa de colores obtenido mediante la red "SOM".

Y, finalmente, se representan varias imágenes del “dataset” de criminalidad representando distintas características.

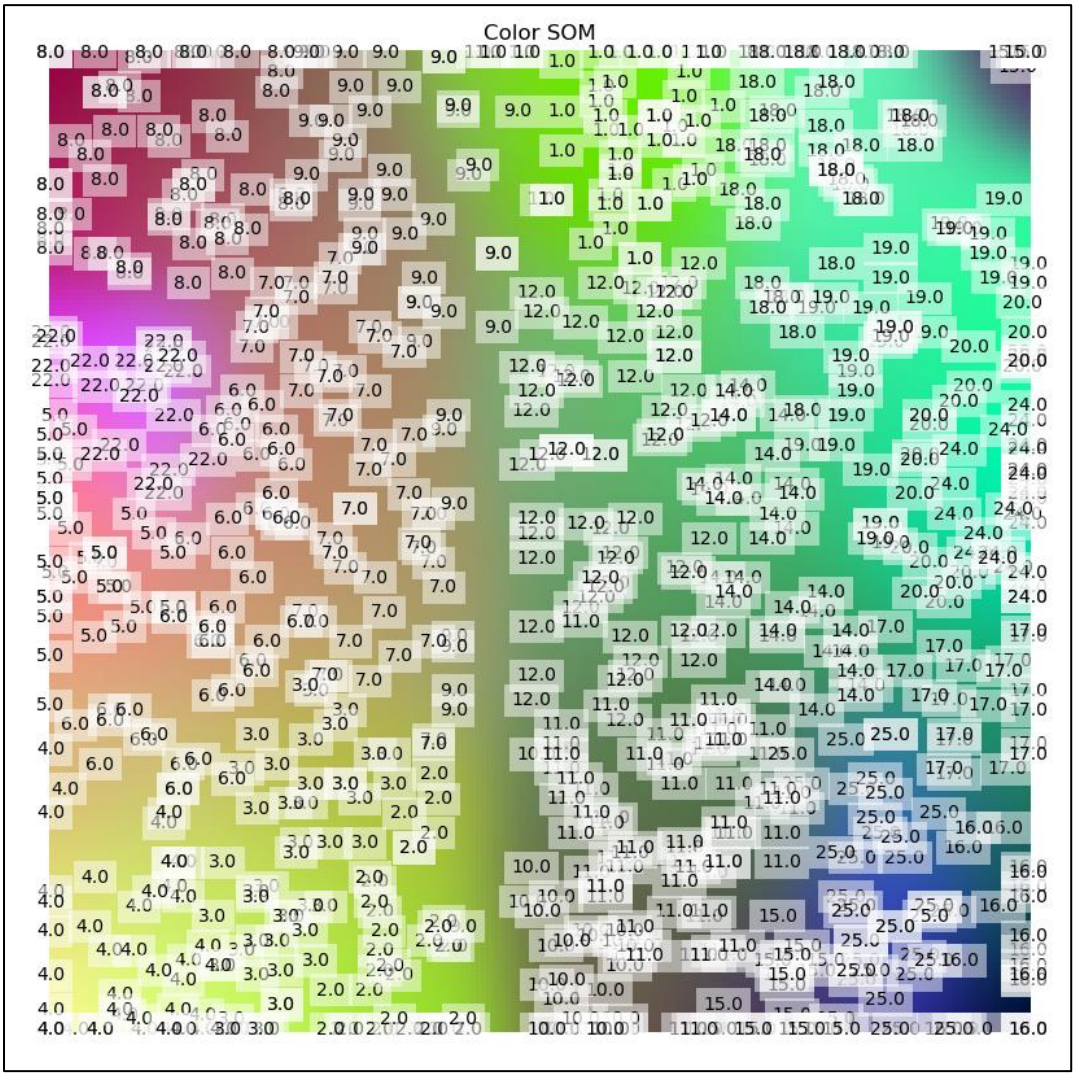

Figura 13. Representación de la característica "District".

Tal como se puede apreciar en la imagen superior, la red “SOM” ha sido capaz de crear 23 “clusters”, agrupando en cada uno de ellos los distintos distritos observados en el conjunto de datos.

Referencias

TensorFlow. (s.f.). *Manual.* Obtenido de https://www.tensorflow.org/guide

ULPGC. (s.f.). *Práctica.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/mod/folder/view.php?id=258360

ULPGC. (s.f.). *ULPGC.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/1971057/mod_resource/content/3/Gu%C3%ADaTema5_SI_basadosenRNA_2020-21.pdf