Mapas Autoorganizados (Self-Organizing Maps)

Daniel Carpio Martín

Trabajo Fin de Grado
dirigido por Miguel Alejandro Fernández Temprano

Grado en Estadística
Universidad de Valladolid
Septiembre 2020

Universidad de Valladolid

Agradecimientos

Me gustaría dedicar unas palabras de agradecimiento a aquellas personas que me han ayudado, apoyado y acompañado en la realización de este Trabajo de Fin de Grado.

En primer lugar, a mi familia por haberme apoyado y animado a dar cada uno de los pasos que he dado hasta aquí aunque a veces no los entendiesen, por haber hecho de mí quien soy y por ser mi principal apoyo en estos tiempos llenos de rareza e incertidumbre. Especial dedicatoria a los pequeños Óliver, Samuel y Rafael y a la futura Nora por ser siempre fuente de alegría y felicidad.

A los amigos con los que llegué a este Grado y siguen estando ahí, especialmente a los que aguantaron mis dudas, me apoyaron en todo momento y a día de hoy siguen haciéndolo, sabiendo cómo sacar una sonrisa en cualquier circunstancia y que me enseñan a ser alguien mejor a diario. Espero poder estar siempre a la altura de lo que os merecéis.

A los compañeros y amigos que me encontré al empezar este capítulo de mi vida, y que hicieron posible sentirme un veterano más y un novato más a la vez. Entre todos han hecho que los momentos de clases, estudio y exámenes hayan sido mucho más llevaderos.

Al resto de amigos que he podido conocer durante estos años y que me han acompañado a lo largo de este camino tanto en Valladolid como en mi estancia en Granada y también, al resto de la familia Martín por dar significado a lo que son.

Por último, un especial agradecimiento a Miguel Alejandro Fernández Temprano por aceptar ser mi tutor en este Trabajo Fin de Grado y por todo el tiempo dedicado a ayudarme a mejorar y todo lo aprendido.

Índice general

Índice de figuras

Resumen

En el análisis de datos, tratar conjuntos de alta dimensión puede ser complicado ya que a veces es difícil encontrar estructuras o representar esos datos. Los mapas autoorganizados son un tipo de red neuronal efectivo pero sencillo basado en aprendizaje no supervisado y competitivo que permite representar estos datos en sencillos mapas a través de proyecciones no lineales a espacios de baja dimensión, generalmente 2D, preservando la topología del espacio original. Este método, que fue desarrollado por Teuvo Kohonen, ofrece por un lado la representación visual de los datos manteniendo las similitudes entre ellos, y, por otro, la agrupación de observaciones similares en el espacio original.

Este trabajo desarrolla los aspectos teóricos más importantes de los mapas autoorganizados, se explica su funcionamiento y se estudian diferentes posibilidades para su implementación. Finalmente, se presentan dos aplicaciones de esta técnica, una de ellas tratando con datos de la pandemia del COVID-19

Abstract

In data analysis, dealing with high dimensional datasets can be complicated as it is sometimes challenging to find structures or visualize them. Self-organizing maps are a simple yet effective type of neural network based on unsupervised and competitive learning that allows representing these data in simple maps through nonlinear projections to low dimensional spaces, usually 2D, preserving the topology of the original space. This method, which was developed by Teuvo Kohonen, offers, on the one hand a visual representation of the data that preserves similarities among them, and, on the other hand, clusters of similar observations in the original space.

This work develops the most important theoretical aspects of self-organizing maps, explains how they work and studies the different posibilities for its implementation. Finally, two applications od the technique are presented, one of them dealing with the COVID-19 pandemia data.

Capítulo 1

Introducción

El análisis exploratorio de datos, introducido por Tukey en 1977 [30], es un conjunto de procedimientos y técnicas que permiten estudiar un conjunto de datos y las relaciones existentes entre las variables analizadas o las observaciones disponibles.

Con el constante aumento de la capacidad de los ordenadores y el interés por analizar conjuntos de datos cada vez con más cantidad de variables, son necesarios métodos más sofisticados que permitan desarrollar este tipo de análisis con técnicas para resumir, visualizar o simplificar el conjunto de datos original así como encontrar estructuras y patrones que permitan relacionar las variables en estudio.

Dentro de este análisis se pueden diferenciar dos tipos. Por un lado, el análisis o aprendizaje supervisado, en el que las técnicas están destinadas a analizar y sacar conclusiones sobre el comportamiento de los datos con respecto a una variable *'clase'* concreta. Y, por otro lado, el análisis no supervisado, consistente en buscar comportamientos, patrones, relaciones,... en el conjunto de los datos completo sin destacar ninguna variable.

Algunos de los objetivos más comunes del análisis de datos no supervisado son encontrar grupos o clústeres en el conjunto de datos (métodos de clustering), lo que permite reducir el conjunto de observaciones al considerar los grupos como observaciones, o poder reducir la dimensionalidad de las observaciones para simplificar los datos mientras se mantienen las estructuras de los datos originales y poder representar la información si el nuevo espacio generado es de una dimensión suficientemente pequeña (métodos de proyección).

Un método especialmente interesante es el conocido como Mapas Autoorganizados (*Self-Organizing Maps*), propuesto y desarrollado por Teuvo Kohonen [20] y que se puede utilizar al mismo tiempo tanto para reducir la cantidad de datos mediante clustering como para realizar un proyección no lineal en un espacio de dimensión menor que permite su representación.

Los objetivos principales de este Trabajo de Fin de Grado son:

1. Conocer el planteamiento del método y explicar el algoritmo principal.
2. Explicar brevemente el fundamento teórico del método.
3. Estudiar otras posibles variantes para su implementación y sus medidas de error.
4. Comparar el método con otros ya conocidos.
5. Analizar el software para su implementación en distintos lenguajes, particularmente en R y los distintos paquetes existentes.
6. Comparar estos paqutes a través de su utilización en un conjunto de datos sencillos y determinar cuál es mejor.
7. Construir un mapa autoorganizado en un conjunto de datos reales de alta dimensión para poder representarlos graficamente y encontrar comportamientos similares entre las observaciones.

Para conseguir estos objetivos, la memoria se ha estructurado de la siguiente forma:

- En primer lugar, en el capítulo 2 se comentará la motivación y origen del método y se explicará su funcionamiento y algoritmo. También se incluirá un breve estudio teórico, así como la existencia de distintas versiones y una pequeña comparación con algunos de los métodos más conocidos utilizados para clustering y proyección.
- A continuación, en el capítulo 3 se hará un repaso a las diferentes posibilidades ya existentes para su implementación en distintos lenguajes como R, Python o Matlab, destacando un análisis de los distintos paquetes de R más utilizados.

- En el capítulo 4, se aplicará el método de los mapas autoorganizados a dos conjuntos de datos diferentes. La primera aplicación será para ver su funcionamiento y comparar los distintos paquetes de R comentados y, la segunda será una aplicación a un conjunto de datos reales en la que se analizará una situación de máximo interés en la actualidad como son los datos de incidencia de COVID-19 a nivel mundial.

- Por último, en el capítulo 5, se incluyen algunas de las conclusiones obtenidas, así como un resumen de lo aprendido e ideas sobre posibles trabajos futuros sobre este tema.

Este trabajo ha servido para complementar los conocimientos obtenidos a lo largo del Grado de Estadística, especialmente en las asignaturas relacionadas con las técnicas de análisis de datos y redes neuronales como Análisis de Datos, Análisis Multivariante o Técnicas de Aprendizaje Automático y conocer una nueva técnica no estudiada en ellas pero que requiere de lo aprendido en dichas asignaturas.

Capítulo 2

Mapas autoorganizados

2.1. Introducción

Los Mapas Autoorganizados son un tipo de red neuronal artificial que fueron introducidos por Teuvo Kohonen en 1986 [20]. Alternativamente, también son llamados Redes de Kohonen, al ser este su principal desarrollador.

El objetivo de los Mapas Autoorganizados o *SOMs* (Self-Organizing Maps) es poder representar conjuntos de datos de alta dimensión en un sistema de coordenadas (generalmente de dos dimensiones) de tal forma que se mantenga la topología del espacio original. Es decir, que puntos que se encuentren próximos en el espacio original sigan estando cerca en el espacio de dimensión reducida creado.

Generalmente, las redes neuronales utilizan un método de aprendizaje supervisado. Esta es la primera diferencia con respecto los SOMs, puesto que estos están basados en aprendizaje no supervisado. Por otro lado, los SOMs utilizan un aprendizaje competitivo mientras que otras redes neuronales aprenden por corrección del error. El aprendizaje competitivo consiste en que, en cada paso, todas las neuronas que forman la red compiten por ser la 'ganadora' y sólo una se activa en cada iteración.

Al ser una técnica de aprendizaje no supervisado, los SOMs son utilizados para encontrar estructuras y comportamientos comunes en los datos, así como para reducir la dimensión de la información y, gracias a que esta técnica preserva la topología original, se utilizan también para realizar sencillas visualizaciones en mapas de dos dimensiones de datos procedentes de espacios de dimensión mayor.

En cuanto a las aplicaciones donde se utilizan los Mapas Autoorganizados, destacan los campos de las finanzas [11], [10], la economía [21] o el marketing [38], así como en sistemas de reconocimiento de voz [12], la minería de textos [23] o en procesamiento del lenguaje [16].

A continuación, se explicará el funcionamiento de dicha técnica.

2.2. Funcionamiento

Como ya se ha mencionado anteriormente, los Mapas Autoorganizados o SOMs pueden interpretarse como un tipo de red neuronal en la que sólo existe una capa de neuronas y están dispuestas, generalmente, en un espacio de dos dimensiones. El número de neuronas, K, que forman el mapa se fijará previamente y estas pueden tener una disposición cuadriculada o hexagonal en el espacio de salida.

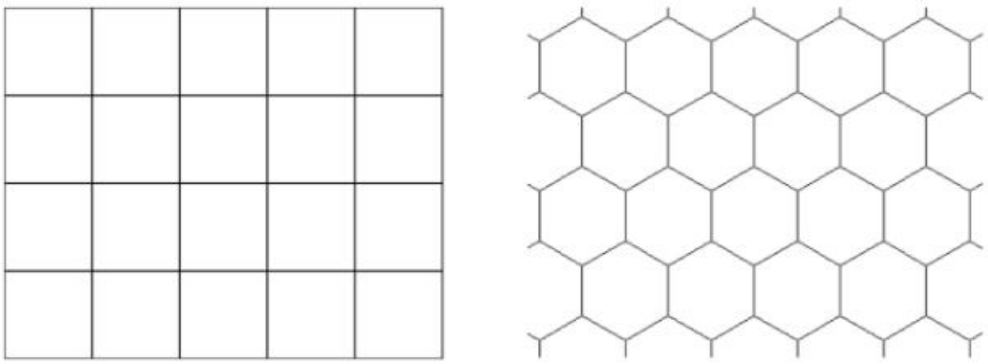


Figura 2.1: Disposición en cuadrícula o hexagonal

Si se dispone del conjunto de datos formado por n observaciones representadas cada una por un vector p-dimensional (p variables) el mapa debe asignar a cada una de las observaciones del espacio $\mathbb{R}^p$ de entrada una neurona del espacio de salida.

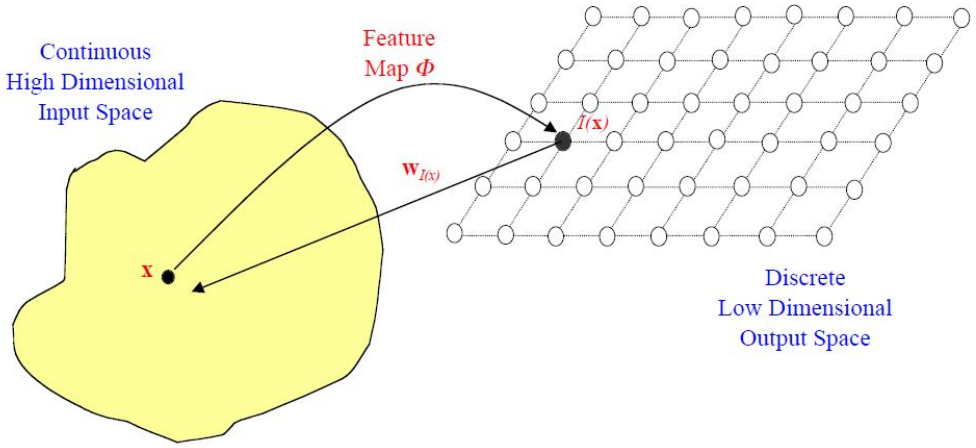


Figura 2.2: Proyección del espacio de entrada en el espacio de salida. De [5]

Para conseguir esto, cada una de las neuronas del espacio de salida, tendrá asociado un vector de pesos p-dimensional de forma que a través de un proceso iterativo de 'autoorganización', estos vectores de pesos se irán modificando de forma que la red creada se asemeje lo más posible al conjunto de datos original. La clave de este proceso reside en que al ir modificando los vectores de pesos de una neurona, también se modificarán los correspondientes a las neuronas más cercanas en el espacio de salida, lo que permite conservar la topología que ya se mencionó.

Este proceso de autoorganización, está compuesto por cuatro fases que se describirán brevemente a continuación.

2.2.1. Inicialización

El primer paso será determinar el número K de neuronas que formarán la red, así como su disposición en el espacio bidimensional. Este número K debe elegirse de tal forma que el mapa no sea ni demasiado grande ni demasiado pequeño. Existen distintos criterios [28] para determinar el valor de K y en la mayoría de las ocasiones puede depender del caso concreto y los datos a los que se aplique. Sin embargo, Vesanto [32], [33] propuso una regla para determinar el número de neuronas que se ha convertido en una de las más utilizadas y será la que se utilice en las aplicaciones del presente trabajo. Según esta regla, un buen tamaño del mapa será:

$$K = 5\sqrt{n} \tag{2.1}$$

siendo n el número de observaciones del conjunto de datos con el que se va a construir el mapa.

Una vez elegida la configuración, será necesario inicializar los vectores de pesos de cada una de las neuronas. Una de las formas más empleadas puesto que funciona bien y su simplicidad, es generar K vectores aleatorios de $\mathbb{R}^p$. Otra forma utilizada es inicializar los vectores en el plano de las primeras componentes principales de los datos iniciales. Esta segunda opción mejora la inicialización de la red, puesto que los vectores iniciales serán más similares a los datos, pero es más costosa, por lo que no siempre es útil.

2.2.2. Competición

Tras la inicialización, empieza el proceso iterativo. En cada repetición, el primer paso será tomar una observación del conjunto de datos de entrada y localizar aquella neurona cuyo vector de pesos sea más próximo o similar al vector de entrada. Es decir, se produce una competición entre todas las neuronas por ver cual es la más cercana, que se considerará la neurona ganadora o BMU (Best Matching Unit).

2.2.3. Cooperación

Como ya se ha mencionado, el objetivo de los Mapas Autoorganizados es preservar la topología del espacio, y para ello, una vez encontrada la neurona ganadora en cada paso (2.2.2), no sólo se moverá su vector de pesos en dirección al vector de entrada, si no que se moverán también todos los vectores de pesos de las neuronas cercanas a la BMU o neurona ganadora. Si, en el caso contrario, no se eligiesen vecinos y en cada iteración sólo la neurona ganadora fuese modificada, el proceso resultante sería similar al método de K-medias.

Uno de los puntos clave de este método será precisamente el criterio para determinar que neuronas son consideradas vecinas de la neurona ganadora en cada paso. En cada paso, se definirá N_c como el conjunto de neuronas vecinas de la neurona ganadora c, que estará centrado en la propia neurona ganadora. Generalmente, se considerarán como vecinas de la neurona ganadora aquellas

que se encuentren a una distancia menor de un radio determinado que se irá modificando (decrecientemente) en cada paso.

Inicialmente, puesto que la red estará completamente desordenada, deberán considerarse radios amplios para estructurar la red modificando los vectores de pesos de varias neuronas y posteriormente, los radios irán decreciendo hasta que en los últimos pasos, las modificaciones sólo afectarán a la neurona ganadora.

2.2.4. Adaptación

Por último, una vez elegida la neurona ganadora y su conjunto de vecinos, deben modificarse sus vectores de pesos para que la red se adapte al conjunto de datos.

Para ello, en cada paso de adaptación, los vectores de pesos de las neuronas que se vayan a modificar, se 'moverán' de tal forma que se acerquen al vector de entrada. La proporción en la que se moverán vendrá dada por un parámetro denominado *tasa de aprendizaje* α que tomará valores entre 0 y 1 y que será decreciente con el tiempo. A mayores valores, se producen cambios más fuertes en la estructura de la red y a medida que el valor de α va decreciendo, los cambios serán menores, proporcionando sólo pequeños ajustes en la red.

Por otro lado, en la mayoría de implementaciones, se introduce también una *función de vecindad* h, de forma que no todos los vecinos se modifiquen en la misma proporción y los cambios que se produzcan sean menores a medida que las neuronas están más alejadas de la neurona ganadora.

En la siguiente sección, se detallará el algoritmo a seguir y se introducirán las ecuaciones y fórmulas matemáticas que se ven envueltas en los procesos de Competición, Cooperación y Adaptación.

2.3. Algoritmo y cuestiones matemáticas

En esta sección, se explica el algoritmo y su funcionamiento en términos matemáticos, determinando las operaciones necesarias a realizar en cada uno de los pasos mencionados anteriormente.

Dado el conjunto de datos de entrada, denotaremos a cada observación por $\mathbf{x}(t)$, con $t = 1, ..., n$ donde n es el número de observaciones disponibles. Cada entrada tendrá valores para p variables, por lo que se puede ver cada observación como un vector p-dimensional, es decir, $\mathbf{x}(t) \in \mathbb{R}^p$.

Por otro lado, se establece un número K, que será el número de neuronas del mapa. Estas neuronas tendrán asociado un vector de pesos p-dimensional $\mathbf{m}_i \in \mathbb{R}^p$, $\forall i \in \{1, ..., K\}$, denominados vectores modelo, que se irán actualizando en cada paso del algoritmo. El primer paso del algoritmo será inicializar estos vectores $\mathbf{m}_i$, por ejemplo, de manera aleatoria.

Una vez inicializados los vectores de pesos, llega el proceso iterativo, que se irá repitiendo para cada $t = 1, ...n$. Cada iteración está caracterizada por los procesos de competición, cooperación y adaptación que ya se mencionaron anteriormente.

Dada una observación $\mathbf{x}(t)$ se buscará aquella neurona cuyo vector de pesos asociado $\mathbf{m}_i$ sea lo más parecido posible. Para ello, se utilizará, casi siempre, la distancia Euclídea y se elegirá la neurona ganadora c, tal que sea la más cercana:

$$\|\mathbf{x}(t) - \mathbf{m}_c\| = \min_i \{\|\mathbf{x}(t) - \mathbf{m}_i\|\} \tag{2.2}$$

Por último, los procesos de cooperación y adaptación, estarán caracterizados por la fórmula de actualización de los vectores de pesos que estará determinada por la tasa de aprendizaje y la función de vecindad. La función de vecindad sirve para determinar qué neuronas se consideran vecinas de la neurona ganadora y la proporción en la que serán modificadas. Se utilizará una función con forma de campana de Gauss que simula el comportamiento de la transmisión de información en capas cerebrales, de forma que irá decreciendo a medida que la neurona está más lejos de la neurona ganadora c. Por tanto, para dos neuronas i y j, se define la función de vecindad como:

$$h_{ij}(t) = \exp\left(-\frac{d^2}{2\sigma(t)^2}\right) \tag{2.3}$$

donde d representa la distancia euclídea entre las neuronas en el espacio de dos dimensiones y $\sigma(t)$ será una función decreciente con t que representa el radio del espacio de vecinos. Generalmente, esta función decreciente será de tipo exponencial:

$$\sigma(t) = \sigma_0 \exp\left(\frac{-t}{\tau_0}\right) \tag{2.4}$$

donde τ_0 será el radio inicial del vecindario.

Una vez determinada la función de vecindad (2.3), se define la función de actualización de los vectores de pesos $\mathbf{m}_i$ como

$$\mathbf{m}_i \leftarrow \mathbf{m}_i + \alpha(t)h_{ci}(t)\left[\mathbf{x}(t) - \mathbf{m}_i\right] \tag{2.5}$$

con $\alpha(t)$ una función decreciente con t que tomará valores entre 0 y 1. Comúnmente se utiliza para la tasa de aprendizaje una función exponencial:

$$\alpha(t) = \alpha_0 \exp\left(\frac{-t}{\alpha_n}\right) \tag{2.6}$$

donde α_0 es un valor inicial en $[0, 1]$ y α_n el número de iteraciones.

En cada iteración, la fórmula de actualización (2.5), se aplica a aquellas neuronas consideradas dentro del vecindario de la neurona ganadora. De esta forma, tanto el vector de pesos de la neurona ganadora como de sus vecinas son actualizados de forma que se acercan al vector de entrada empleado en dicha iteración.

Por tanto, el algoritmo del método se puede resumir en:

1. Determinar número de neuronas K que tendrá el mapa.

2. Inicializar los vectores de pesos $\mathbf{m}_i$ de las neuronas.

3. Dada una observación $\mathbf{x}(t)$, se busca entre todas las neuronas aquella con vector de pesos más cercana según (2.2) (Competición).

4. Actualización de los $\mathbf{m}_i$ según la ecuación (2.5) (Cooperación y Adaptación).

5. Modificación de los parámetros α y σ.

6. Repetir los pasos 3 a 5 hasta alcanzar un número fijado de iteraciones.

Pese a ser un algoritmo bastante sencillo y de fácil implementación, su estudio teórico es muy complejo y existen pocos resultados salvo en el caso unidimensional.

2.4. Estudio teórico de los SOMs

A lo largo de las secciones anteriores se ha explicado el proceso de construcción de los Mapas Autoorganizados. A pesar de ser un método aparentemente sencillo, que es fácilmente entendible e implementable y que en la práctica funciona bien, contrasta fuertemente con la dificultad encontrada a la hora de demostrar matemáticamente su eficacia.

La red o mapa se define como

$$\mathbf{m}(t) = (\mathbf{m}_1(t), \mathbf{m}_2(t), ..., \mathbf{m}_k(t)) \tag{2.7}$$

con $\mathbf{m}_i(t)$ el vector modelo asignado a la neurona i en el instante t. Es decir, $\mathbf{m}(t)$ representa el estado de la red completa en el instante t. El objetivo es demostrar que existe un estado al que converge la red que preserve la topología del espacio de entrada. Como ya se vio antes, el paso de un estado al siguiente puede escribirse como

$$\mathbf{m}(t+1) = \mathbf{m}(t) - \alpha(t)H\left(\mathbf{x}(t+1), \mathbf{m}(t)\right) \tag{2.8}$$

y puede verse como un proceso estocástico.

A lo largo de la literatura, pueden encontrarse numerosos artículos desarrollando algunos aspectos parciales de los SOMs, pero que suelen limitarse a casos muy particulares con diferentes condiciones impuestas, limitándose los desarrollos más extensos a los casos de una dimensión, tanto en el espacio de entrada como en el de salida (una red lineal). Algunos de los más destacados son de Kohonen, [19], Bouton y Pagès, [3], [4], y Cottrell y Fort, [7], [8] y [9]. Uno de los motivos que dificultan el desarrollo matemático en más dimensiones es debido a que en dimensión mayor a 1 no se puede definir un orden único que garantice una configuración ordenada y estable.

La construcción de los mapas autoorganizados puede dividirse en dos etapas: una en la que se produce la organización de la red, es decir, se ordena y otra en la que se ajusta (los vectores modelo convergen a un estado estable). Esta característica hace que los estudios teóricos también se dividan en dos partes: por un lado buscar la existencia de una configuración ordenada, y por otro, que una vez alcanzada esta, haya una convergencia a un estado estable.

En [9] se recogen algunos de los resultados parciales obtenidos, especialmente los correspondientes a dimensión 1, que se detallan a continuación.

2.4.1. Autoorganización en dimensión 1

En esta situación, se considera que el espacio de entrada es $[0, 1]$ (tras normalizar), es decir, las observaciones corresponden a una sola variable. Además, también se supone una dimensión en la disposición de las neuronas del mapa, por lo que se disponen en un array lineal. En estas condiciones, la autoorganización se demostraría si se encuentra un estado absorbente y ordenado al cual el mapa $\mathbf{m}(t)$ llegue con probabilidad 1 en un tiempo finito.

Se definen los siguientes conjuntos

$$\mathcal{F}_k^+ = \{m \in \mathbb{R}/ \quad 0 < m_1 < m_2 < ... < m_k < 1\} \tag{2.9}$$
$$\mathcal{F}_k^- = \{m \in \mathbb{R}/ \quad 0 < m_k < m_{k-1} < ... < m_1 < 1\} \tag{2.10}$$

Si la función de vecindad h es una función decreciente con la distancia entre dos neuronas, puede considerarse el siguiente teorema:

2.1 Teorema. (i): $\mathcal{F}_k^+$ y $\mathcal{F}_k^-$ son conjuntos absorbentes.
(ii): Si α es constante y la función h es decreciente como función distancia, entonces el tiempo, τ, en el que se alcanza $\mathcal{F}_k^+ \cup \mathcal{F}_k^-$ es casi seguro finito y, existe $\lambda > 0$ tal que $\sup_{m\in[0,1]^k} E_m(\exp(\lambda\tau))$ es finito siendo E_m la esperanza con $\mathbf{m}(0) = m$.

Este teorema, que se demuestra en [7] a través de técnicas de Cadenas de Markov, garantiza que, dadas las condiciones fijadas, el algoritmo ordenará los m_i casi seguro. Es decir, dadas las condiciones necesarias, dada una red iniciada con valores $\mathbf{m}_i$ en $[0, 1]$, el número de modificaciones de estos hasta conseguir que queden de manera ordenada, ya sea creciente o decrecientemente, se realizará en tiempo finito, alcanzando un estado ordenado.

2.4.2. Convergencia en dimensión 1

Tras asegurar la autoorganización alcanzando un estado ordenado $\mathcal{F}_k^+$ ó $\mathcal{F}_k^-$ es necesario estudiar la convergencia del algoritmo. Al haber demostrado que se alcanza el estado ordenado, se puede considerar $\mathbf{m}(0) \in \mathcal{F}_k^+$ (Lo mismo para $\mathcal{F}_k^-$).

A partir de la ecuación (2.8), los posibles estados límite deberán ser solución de

$$h(m) = 0 \tag{2.11}$$

con $h(m) = E(H(x,m)) = \int H(x,m)d\mathcal{P}(x)$ y por tanto será necesario estudiar la ODE

$$\frac{dm}{dt} = -h(m) \tag{2.12}$$

y encontrar sus equilibrios.

El siguiente teorema, [9], demuestra la convergencia en el caso de que el parámetro de adaptación $\alpha(t)$ sea decreciente y bajo ciertas condiciones necesarias.

2.2 Teorema. Sea $\alpha(t)$ una función decreciente y definida en $(0,1)$ que cumple $\sum_t \alpha(t) = +\infty$ y $\sum_t \alpha^2(t) < +\infty$. Sea, además, la función de vecindad h una función decreciente a partir de un $k_0 < \frac{k-1}{2}$. Se supone también que la distribución de entrada $\mathcal{P}$ tiene una densidad asociada f positiva en $(0,1)$ y que cumple que $\ln(f)$ es estrictamente cóncava (o sólo cóncava si cumple $\lim_{0^+} f + \lim_{1^-} f$ es positivo). Entonces:

1. La función

$$h(m) = E(H(x,m)) = \int H(x,m)d\mathcal{P}(x) \tag{2.13}$$

   tiene al menos un cero m^* en $\mathcal{F}^+$.

2. Todo punto estacionario m^* es atrayente. Es decir, si $m(0) \in \mathcal{F}^*$, $\mathbf{m}(t) \xrightarrow{c.s.} m^*$

3. Si además la distribución es uniforme, el zero m^* es único.

En el mismo sentido, el teorema anterior es igualmente aplicable en el caso de $\mathcal{F}^-$.

Como se puede ver, los resultados teóricos requieren de fuertes condiciones y complicaciones hasta en el caso más simple y sencillo de dimensión 1. En cuanto al caso multidimensional, el problema se hace muy difícil sin que se hayan alcanzado muchos resultados, y los que hay son de una gran complejidad. Este problema contrasta con la creciente aparición de aplicaciones en las que se utilizan los SOMs pese a no existir rigurosos estudios teóricos que garanticen la calidad igual que con otros métodos clásicos.

2.5. Otras versiones

En la sección 2.3 se ha visto el caso del algoritmo desde el punto de vista más sencillo y común. Sin embargo, existen diferentes versiones entre las que se puede diferenciar entre versiones con modificaciones en alguna de las funciones empleadas o versiones en las que cambia la forma de computación.

Respecto a las primeras, se ha explicado que el algoritmo básico selecciona la neurona ganadora escogiendo aquella que está más cercana al dato de entrada en términos de distancia Euclídea, pero pueden elegirse otros criterios para elegir que neurona está más "cercana", siempre y cuando, se modifique también la fórmula de actualización de pesos para que sea compatible con esa métrica. Por ejemplo, con datos binarios, es común utilizar el producto interno para medir la similaridad entre vectores, tomando como neurona ganadora aquella que maximiza la similaridad. En este caso, las fórmulas empleadas para el emparejamiento y la actualización serían las siguientes:

$$\rightarrow \text{Emparejamiento:} \quad \mathbf{x}^T(t)\mathbf{m}_c(t) = \max_i \{\mathbf{x}^T(t)\mathbf{m}_i(t)\}$$

$$\rightarrow \text{Actualización:} \quad \mathbf{m}_i(t+1) = \begin{cases} \dfrac{\mathbf{m}_i(t) + \alpha'(t)\mathbf{x}(t)}{\|\mathbf{m}_i(t) + \alpha'(t)\mathbf{x}(t)\|} & i \in N_c(t) \\ \mathbf{m}_i(t) & i \notin N_c(t) \end{cases}$$

donde $N_c(t)$ corresponde al conjunto de neuronas vecinas de la neurona c en el instante t.

2.5.1. Versión batch

Tanto en el algoritmo visto en la sección anterior (2.3) como en las posibles modificaciones vistas ahora, el proceso básicamente es el mismo: Iterativamente, se toma una observación del conjunto de datos, se busca la neurona ganadora siguiendo cierto criterio de similaridad y se actualizan su peso y el de los vecinos correspondientes. Sin embargo, existe una versión alternativa, denominada versión *batch* (o en bloque) donde en cada paso del entrenamiento, se utiliza todo el conjunto de datos.

En esta versión, que es más rápida y segura, se cambia la idea principal del algoritmo. Aquí, la forma de proceder será utilizar en cada paso del algoritmo

todo el conjunto de datos, comparando todas las entradas con los vectores modelo $\mathbf{m}_i$ y asignando a cada uno de los vectores modelo el subconjunto de observaciones para las cuales ese $\mathbf{m}_i$ es su vector más cercano. De esta forma, cada una de las observaciones $\mathbf{x}_j$ estará asociada a uno y sólo uno de los vectores modelo $\mathbf{m}_i$, y por tanto a una de las neuronas del mapa.

A continuación, para cada uno de los vectores $\mathbf{m}_i$, se buscan cuáles de las neuronas de su alrededor se consideran vecinas y se procede a actualizar el peso de $\mathbf{m}_i$ a partir de todas las observaciones $\mathbf{x}_j$ asignadas a dicha neurona y sus vecinas.

Este proceso, será necesario repetirlo varias veces hasta que los $\mathbf{m}_i$ sean estables.

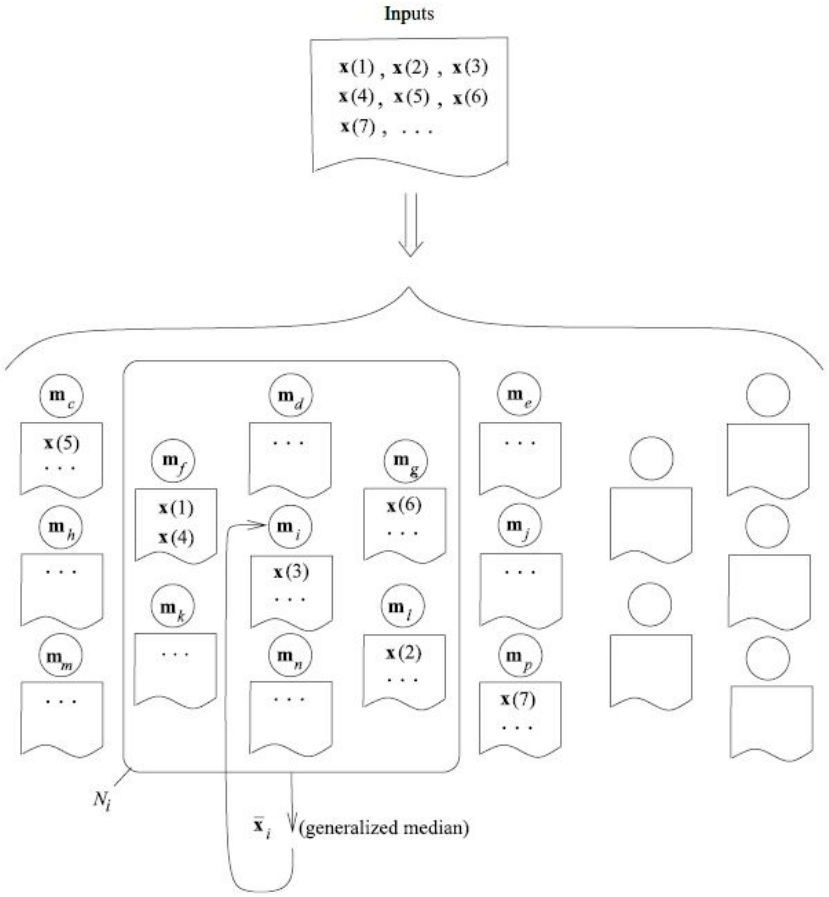


Figura 2.3: Funcionamiento de la versión batch de los SOMs. De [21]

Para obtener la fórmula de actualización de este método es necesario

recurrir al método iterativo y a la suposición de existencia de un estado estable de éste.

Así, asumiendo estabilidad, se debe de cumplir que cuando $t \to \infty$, los valores de $\mathbf{m}_i(t+1)$ y $\mathbf{m}_i(t)$ serán iguales, $E_t\left(\mathbf{m}_i\left(t+1\right)\right) = E_t\left(\mathbf{m}_i(t)\right)$. Entonces, para cada neurona i:

$$\begin{aligned} E_t\left(\mathbf{m}_i\left(t+1\right)\right) &= E_t\left(\mathbf{m}_i(t) + \alpha(t) h_{ci}\left(\mathbf{x}(t) - \mathbf{m}_i(t)\right)\right) \\ &= E_t\left(\mathbf{m}_i(t)\right) + \alpha(t) E_t\left(h_{ci}\left(\mathbf{x}(t) - \mathbf{m}_i(t)\right)\right) \\ &\Rightarrow E_t\left(h_{ci}\left(\mathbf{x}(t) - \mathbf{m}_i(t)\right)\right) = 0 \end{aligned} \tag{2.14}$$

Si se denota por $\mathbf{m}_i^*$ al estado asintótico de $\mathbf{m}_i(t)$, desarrollando en (2.14):

$$\begin{aligned} E_t\left(h_{ci}\left(\mathbf{x}(t) - \mathbf{m}_i(t)\right)\right) &= \frac{1}{t}\sum_t \left(h_{ci}\left(\mathbf{x}(t) - \mathbf{m}_i^*\right)\right) = 0 \\ \Rightarrow \sum_t h_{ci}\mathbf{x}(t) &= \sum_t h_{ci}\mathbf{m}_i^* \\ \Rightarrow \mathbf{m}_i^* &= \frac{\sum_t h_{ci}\mathbf{x}(t)}{\sum_t h_{ci}} \end{aligned} \tag{2.15}$$

Con esta fórmula del estado asintótico $\mathbf{m}_i^*$, volviendo al método batch, si una vez asignados a cada vector $\mathbf{m}_i$ los $\mathbf{x}(t)$ más cercanos, se calcula para cada vector modelo la media $\mathbf{x}_{\mathbf{m}_i}$ como la media de todos los vectores $\mathbf{x}(t)$ de su vecindario, en cada paso del algoritmo se pueden actualizar los vectores modelo como:

$$\mathbf{m}_i^* = \frac{\sum_j n_j h_{ij} \mathbf{x}_{\mathbf{m}_j}}{\sum_j n_j h_{ij}} \tag{2.16}$$

siendo n_j el número de vectores $\mathbf{x}(t)$ asociados a la neurona j.

Esta versión, a pesar de no seguir la idea original del algoritmo es utilizada comúnmente debido a las mejoras en la implementación que presenta frente a la versión secuencial. Sin embargo, como se verá posteriormente, ambas son utilizadas.

2.6. Calidad del ajuste. Medidas de error

Una vez construido el mapa autoorganizado, como en el resto de métodos, es necesario evaluar como de bueno es el resultado obtenido y saber si los datos se ajustan adecuadamente al modelo obtenido permitiendo extrapolar las conclusiones obtenidas del mapa a los datos originales. Existen distintas medidas que permiten analizar la calidad del modelo, siendo dos de ellas las más utilizadas y estudiadas.

Como en otros métodos de clustering, será necesario comprobar si las observaciones del espacio de entrada se ajustan bien a sus proyecciones en el espacio de salida (es decir, a los vectores de referencia asociados a cada una) a través del *error de cuantificación*. Por otro lado, puesto que el método de los SOM busca preservar la topología del espacio original, podrá medirse la calidad del modelo en este aspecto a partir del *error topográfico*, con el que se podrá medir la calidad con la que se mantienen en el mapa las características topológicas del conjunto de entrada.

Estas dos propiedades a medir están relacionadas entre ambas ya que ajustar el mapa mejorando el error de cuantificación puede empeorar la conservación de las propiedades topológicas y al revés, por lo que será interesante mantener un equilibrio entre ambas.

A continuación, se describirán ambas medidas así como la forma de obtenerlas.

2.6.1. Error de cuantificación

El error de cuantificación es una medida utilizada en todos los algoritmos de clustering y cuantificación vectorial. Esta medida no tiene en cuenta las características relacionadas con la topología y sólo evalúa la calidad de la proyección de las observaciones en el espacio de salida, es decir, cómo de bien se ajustan a los vectores de referencia. Se mide como el promedio de las distancias entre las observaciones y los vectores modelo de las neuronas a las que están asociados.

$$QE = \frac{1}{n} \sum_{i=1}^{n} \parallel \mathbf{x}_i - \mathbf{m}_c(t) \parallel \tag{2.17}$$

Este error disminuirá a medida que el mapa sea más grande, ya que las observaciones pueden estar repartidas entre más neuronas y que estas se ajusten mejor a las observaciones correspondientes. En el caso extremo, si cada observación estuviese asociada a una neurona distinta, el error podría ser 0.

Al ser una medida que depende del tamaño de la red, no puede emplearse para comparar mapas de distinto tamaño. Por otro lado, al ser simplemente un promedio de los errores entre cada observación y su vector asociado, es equivalente al error de cuantificación en otros métodos de clustering, por lo que podría utilizarse para comparar métodos.

2.6.2. Error topográfico

Por otro lado, el error topográfico es una de las medidas mas sencillas que permiten ver si se mantienen las características topológicas del espacio de entrada. Para ello, se considerará que un mapa que reproduzca bien la topología será aquel que para neuronas vecinas sus vectores de referencia sean cercanos entre ellos. Para ello, será necesario calcular para cada una de las observaciones $\mathbf{x}_i$ las dos neuronas más cercanas (denominadas $BMU_1(\mathbf{x}_i)$ y $BMU_2(\mathbf{x}_i)$ respectivamente) y comprobar si estas dos son vecinas. El error topográfico estará determinado por el siguiente promedio:

$$TE = \frac{1}{n}\sum_{i=1}^{n} err\left(\mathbf{x}_i\right) \tag{2.18}$$

siendo

$$err(\mathbf{x}_i) = \begin{cases} 0 & \text{si } BMU_1(\mathbf{x}_i) \text{ y } BMU_2(\mathbf{x}_i) \text{ son adyacentes} \\ 1 & \text{en otro caso} \end{cases} \tag{2.19}$$

En el caso de una conservación perfecta, este error sería 0, por lo que interesa mantener un error bajo. Sin embargo, el error topográfico tiende a crecer a medida que aumenta la dimensión del mapa, por lo que es necesario buscar mapa con unas dimensiones apropiadas para lograr un equilibrio entre ambos errores (2.17) y (2.18).

De cómo se calcula este error, se puede concluir que no es una medida a considerar en casos de mapas muy pequeños al tener pocos nodos. Este error no se puede comparar con otros métodos que no sean SOMs.

Aparte de estas dos medidas del error, que son las más empleadas, existen algunas otras que permiten estudiar la calidad desde diferentes puntos de vista [26]. Algunas de estas otras medidas son el *producto topográfico* que sólo emplea los vectores modelos y se utiliza para determinar el tamaño del mapa o la *integridad* y la *conservación del vecindario* que comparan si puntos que son vecinos en el mapa son vecinos en el conjunto de datos de entrada.

2.7. Comparación con otros métodos

Los Mapas Autoorganizados o *SOMs* están relacionados o pueden compararse con algunos otros métodos de reducción de dimensionalidad o de clustering. A continuación, se verán algunos de los casos más importantes.

K medias

Se ha visto que una de las ventajas de los SOMs es que son apropiados para realizar agrupaciones de datos en varios subconjuntos por lo que se puede comparar con otros métodos de clustering.

Dado el algoritmo visto en 2.3, si se prescinde de los vecinos de la neurona ganadora (tomando un radio de vecindad suficientemente pequeño) de tal forma que el único vector modelo $\mathbf{m}_i$ que se actualiza en cada paso es el correspondiente a la neurona más próxima al vector de entrada, estaríamos ante un algoritmo que asigna cada vector de entrada a un $\mathbf{m}_i$ que se actualizarían en función de los datos asignados. Por tanto, se puede ver esta versión del SOM como un equivalente del algoritmo de K medias y por extensión, la versión general de los SOMs, como una extensión del algoritmo de K medias.

Sin embargo, aunque ambos métodos son utilizados para agrupar datos, la elección de un método u otro es diferente. Por un lado, en K medias el número de clústeres K ha de elegirse de acuerdo al número de grupos que existen en los datos. En cambio, en el caso de los SOMs, el número de neuronas puede ser mucho mayor, ya que el número final de clústeres puede elegirse a partir de la disposición final de la red.

Curvas Principales

El algoritmo SOM crea una represntación organizada de los datos que sigue su misma distribución y puede verse como una caracterización del método de curvas principales. Cada punto de una curva principal se puede ver como el promedio de todas las observaciones que se proyectan en él. Es decir, la curva se forma a partir de las esperanzas condicionadas de los datos. Por otro lado, la versión batch de los Mapas Autoorganizados define los vectores modelo $\mathbf{m}_i$ como las esperanzas condicionadas locales de los datos, por lo que se podrían interpretar las Curvas Principales como la extensión continua en el espacio de los SOMs.

MDS

Otra de las utilidades de los SOMs es la de realizar una reducción de la dimensionalidad de los datos, pasando generalmente de un espacio de alta dimensión a uno de dos dimensiones.

Por un lado, el objetivo del MDS es proyectar los datos en espacios de baja dimensión preservando directamente las distancias entre todos los puntos. Al no poderse cumplir esta condición para cada par de puntos, el algoritmo proyectará la mayoría de las distancias de una foma bastante aproximada mientras que algunas se transformarán de manera bastante pobre, que serán aquellas que más afecten a la medida del error.

Por otro lado, el SOM lo que intenta es realizar proyecciones también en un espacio de baja dimensión (normalmente 2D) de forma que las proyecciones sean localmente correctas. De esta forma, el algoritmo mantiene las distancias cercanas para garantizar que los puntos cercanos en el espacio original lo estén también en el espacio transformado. Con este método sólo las distancias cercanas son aquellas que se consideran para medir el error.

Por tanto, la diferencia entre ambos métodos de reducción de dimensionalidad es que los SOMs garantizan que las observaciones cercanas se proyectan en zonas cercanas preservando el orden y la agrupación local mientras que el MDS mantiene el orden global de las distancias (o disimilaridades).

Capítulo 3

Software para su implementación en R

Una vez comentada la base teórica de los SOMs, en este capítulo se procederá a estudiar su aplicación y comprobar su funcionamiento. Como todos los métodos de análisis de datos, pueden ser implementados en varios lenguajes como R, Python o Matlab, pero este trabajo se centrará en estudiar su implementación en R.

A lo largo de este capítulo se explicará brevemente el manejo de los SOMs detallando el funcionamiento de distintas librerías en R y se mencionarán algunas de las posibilidades para implementar los mapas autoorganizados en otros lenguajes como Python y Matlab.

Aunque una vez conocido el algoritmo, podría construirse una función propia para la construcción de los SOMs, se ha comprobado que ya existen librerías, y además, más de una, en la que estos algoritmos ya están implementados. Por tanto, en este capítulo se comentarán las librerías o paquetes más frecuentes y empleados para ver sus diferencias y poder utilizarlos en el siguiente capítulo en diferentes aplicaciones.

3.1. Librería `class`

La librería `class`, desarrollada por Brian Ripley [31], incluye diferentes funciones que implementan distintos métodos de clasificación como son

k-vecinos próximos, LVQ (Learning Vector Quantization) o los Mapas Autoorganizados. Su última actualización fue publicada en 26/04/2020 en su versión 7.3-17.

Respecto a la implementación de los Mapas Autoorganizados, `class` implementa los SOMs tanto en su versión secuencial (algoritmo original) como en su versión batch a través de las funciones **SOM** y **batchSOM** respectivamente.

Ambas funciones requieren que se pase por parámetros el conjunto de datos y un elemento de tipo *grid*, que se genera mediante la función **somgrid**, y los parámetros que definen las dimensiones del mapa y el tipo de disposición de las neuronas (rectangular o hexagonal). Además también serán necesarios los valores relativos a α, al número de iteraciones (sólo en la versión secuencial), el radio de vecindad y los valores iniciales de los vectores $\mathbf{m}_i$ asociados a cada una de las neuronas, que por defecto tomarán valores aleatorios.

Estas dos funciones devuelven objetos del tipo SOM, que están formados por dos elementos. Por un lado, *grid* que almacena la información relativa a las dimensiones y disposición y por otro lado, *codes* donde se almacenan los vectores asociados a cada neurona.

Por último, `class` dispone de una función **plot** para los objetos de tipo SOM que representa en el mapa de dos dimensiones un gráfico poligonal para cada una de las neuronas y las distintas variables existentes.

3.2. Librería `som`

Otra librería que permite la implementación de los SOMs en R es `som`, creada por Jun Yan en 2010, y cuya versión actual, la 0.3-5.1 fue lanzada en 2016 [37]. Esta librería está creada exclusivamente para la implementación de los mapas autoorganizados y por tanto, todas las funciones están relacionadas con su construcción.

Esta librería dispone de dos funciones para el preprocesamiento que permiten filtrar (**filtering**) y normalizar (**normalize**) los datos.

A la hora de la construcción de los SOMs, existen varias funciones, siendo **som** la principal, que permite la inicialización y entrenamiento mediante

el algoritmo batch con una sola orden a partir de los datos, dimensiones, topología, forma de inicialización de los vectores y los parámetros α y el radio. Además, **som.init** permite la inicialización del mapa creando una matriz con los valores iniciales de los vectores $\mathbf{m}_i$ y **som.train** el entrenamiento a partir de dicha matriz.

Tanto las funciones **som** como **som.train** crean un objeto de clase "som" formado por una lista de elementos que incluye todos aquellos que son argumentos de las funciones (datos, dimensiones, tipo de inicialización, topología, parámetros α y radio,...) y los creados por la propia función como *code*, que almacena los $\mathbf{m}_i$ finales, *visual*, una matriz que asigna a cada una de las observaciones las coordenadas de la neurona más próxima y la distancia a ella y *qerror* que es una medida de distorsión promedio.

Una vez construido el mapa, **som.update** permite modificar el mapa a partir de nuevos parámetros y **som.project**, permite clasificar nuevos datos a partir de un SOM ya construido.

A la hora de analizar los resultados, las funciones **summary** o **print** permiten mostrar los detalles del objeto "som" construido. Por otro lado, mediante la función **plot**, se puede representar el mapa con el número de observaciones asignados a cada neurona así como una linea poligonal en cada una de las neuronas mostrando el valor de los vectores $\mathbf{m}_i$ para cada componente.

3.3. Librería `yasomi`

La librería `yasomi` es otra de las que permiten implementar los SOMs en R. Su nombre viene de las siglas de 'Yet Another Self Organizing Map Implementation' (Otra implementación de los mapas autoorganizados) y fue creado por F. Rossi en 2012 [27].

Esta librería es bastante completa, ya que incluye implementaciones de distintos algoritmos para distintos tipos de datos, buenas representaciones gráficas y dispone de medidas del error para medir la calidad de los SOMs construidos.

En primer lugar, para determinar la estructura del mapa, se utilizará la función **somgrid** para especificar las dimensiones y la disposición de las neuronas en el mapa. Una vez construida la estructura del mapa, es necesario

entrenar el SOM. Una de las ventajas de esta librería frente a otras es que dispone de la función **som.tune** que permite entrenar el SOM para distintos valores de los parámetros (**somcontrol**) y elegir aquellos parámetros que construyen el mejor mapa en términos de un cierto tipo de error.

Además, `yasomi` dispone de muchas opciones de visualización para los SOMs. La representación más simple es representar los vectores modelo $\mathbf{m}_i$ asociados a cada neurona en el mapa. Al ser vectores multidimensionales, esta representación suele hacerse mediante polígonos o *glyphs*, coordenadas paralelas o gráficos de barras. Otra de las posibilidades es representar por separado cada variable mediante una escala de colores para poder analizar el comportamiento individual de cada una de ellas a lo largo del mapa y poder extraer conclusiones y patrones. Otra posibilidad es representar el tamaño de cada neurona en función del número de observaciones que tenga asignadas en un gráfico denominado *hitmap*.

Otra de las ventajas de `yasomi`, es que el objeto construido contiene la matriz de distancias entre los vectores de referencia, lo que permite aplicar directamente la función **hclust** que puede resultar interesante para buscar clústeres dentro del SOM utilizando clustering jerárquico y combinar los clústeres encontrados con el SOM para visualizar en el mapa los distintos grupos de neuronas encontrados.

3.4. Librería `SOMbrero`

La librería `SOMbrero`, toma su nombre de las iniciales de *'SOM Bound to Realize Euclidean and Relational Outputs'* y está creada por Natalie Vialaneix, Jerome Marriette y Madalina Olteanu en 2015, y cuya versión actual (1.3-1) fue lanzada en Agosto de 2020 [35].

En `SOMbrero` se implementa el algoritmo secuencial del SOM en diferentes versiones que permiten su entrenamiento en el caso de datos numéricos, datos relacionales o tablas de contingencia (para estos casos, no se ha desarrollado en este trabajo el algoritmo pertinente, por lo que se detallará sólo el caso de los datos numéricos). Por otro lado, incluye una gran variedad de representaciones gráficas y, además, dispone de una interfaz gráfica de usuario que facilita mucho su implementación sin tener que escribir comandos para usuarios que no se desenvuelvan bien con el lenguaje `R`.

En el caso numérico, se utiliza la función **trainSOM**, que requiere como argumento el conjunto de datos (puede ser como matriz o como data frame). Además, se podrán especificar otros argumentos correspondientes a las dimensiones, topología, tipo de función de vecindad, tipo de distancia,...que permiten determinar cada uno de los posibles parámetros del SOM y personalizarlo completamente.

Esta función construye objetos de la clase *somRes* con varios componentes: *clustering* (cuál es la neurona asociada a cada observación), *prototypes* (coordenadas de los vectores referencia), *energy* (medida que permite medir la estabilidad y el error), *parameters*,... que facilitarán su interpretación y representación.

En cuanto a las opciones gráficas, existe un gran abanico de posibilidades que permiten tener una idea muy clara del mapa obtenido y poder extraer conclusiones sobre el comportamiento de las variables, relaciones entre ellas y/o clústeres. Por un lado, se puede representar la energía (una medida del error) para comprobar si el mapa es estable y adecuado. Por otro lado, se puede representar en el mapa tanto el comportamiento de las variables (escalas de colores, gráfico de barras, de lineas, sectores,....) o el comportamiento en sí de las neuronas (número de observaciones asociadas, representación de los vectores de referencia,...). Por último, la función **SuperClass** permite aplicar clustering jerárquico sobre las neuronas.

3.5. Librería `kohonen`

Por último, la librería `kohonen` dispone de todas las funciones necesarias para crear Mapas Autoorganizados de manera fácil. Entre otras funciones, destacan las que permiten la construcción del SOM así como representar gráficamente el resultado. Su versión actual, la 3.0.10, fue publicada el 26/11/2019 [36].

Para entrenar el SOM, será necesario emplear la función **som** que construye un objeto con los resultados deseados. Para ello, necesita el conjunto de datos con el que se pretende entrenar, la malla con las características que se elijan y el tipo de algoritmo (online o batch).

Para la generación de la malla, se utilizará la función **somgrid** que permite generar una malla en dos dimensiones con el número de filas y columnas

de neuronas deseado así como especificar el tipo de función de vecindad y la forma de disposición de las neuronas (rectangular o hexagonal).

Tras entrenar y construir el mapa, este podrá representarse gráficamente con el comando **plot**, que en función del parámetro *type*, permite realizar distintos tipos de visualizaciones:

codes: (Tipo por defecto). Representa las neuronas y en cada una de ellas un gráfico que muestra el valor de cada una de las variables. Esto permite visualizar el comportamiento de cada variable a lo largo del mapa.

mapping: Representación de los datos asignados a cada neurona.

counts: Muestra la cantidad de puntos asociados a cada neurona.

dist.neighbours: Muestra la suma de las distancias a las neuronas vecinas. Cuando existen clústeres en los datos, neuronas fronterizas, mostrarán valores más altos de esta distancia, mientras que si una neurona tiene un valor pequeño, significará que se encuentra cerca de todas sus neuronas vecinas.

property: Con este tipo de gráfico se puede representar el comportamiento de una variable (especificada mediante el comando *property* a lo largo del mapa. Esto permitirá ver el comportamiento o situación en el mapa en función del valor de dicha variable.

quality: De esta forma, se puede ver la distancia de los puntos asignados a cada neurona al vector modelo correspondiente. Cuanto menor sea el valor, más aproximado será el vector modelo a los datos.

changes: Muestra la evolución de la distancia durante el entrenamiento.

3.6. Implementación en otros lenguajes

Como se ha podido ver, debido a la utilidad de los SOMs, existen distintos desarrollos que permiten su implementación en R. Sin embargo, su popularidad extiende sus implementaciones a otros lenguajes como pueden ser Python o Matlab. A continuación, se enumeran algunas posibilidades en estos lenguajes aunque no se desrrollan en detalle al trascender de los objetivos de este trabajo que se limita a la implementación en R.

En Python, destacan algunas librerías como son `sompy` [24], `SimpSom` [6] o `MiniSom` [34].

En Matlab, ya en 1999 se diseñó una herramienta que permitía su implementación, 'SOM Toolbox' [33], que fue desarrollada por Juha Vesanto a partir de los desarrollos de Kohonen y que permite tanto el entrenamiento como la visualización de mapas autoorganizados.

Capítulo 4

Aplicaciones

Una vez visto el contexto teórico, el funcionamiento y algunos de los paquetes empleados en R para su implementación, en esta sección se verán algunas aplicaciones en distintos conjuntos de datos.

En primer lugar, se aplicará un SOM a un conjunto de datos sencillo en $\mathbb{R}^3$ cuyas observaciones tendrán asociado un color en la escala RGB con el objetivo de explorar las distintas opciones en la implementación de esta técnica. A continuación se aplicará un mapa autoorganizado a un caso más real y actual, como pueden ser los datos de Covid-19.

4.1. Datos generados: Colores RGB

En este primer ejemplo, puede pensarse en los colores como observaciones en $\mathbb{R}^3$ donde sus coordenadas representan el tono de rojo(R), verde(G) y azul(B) y toman valores enteros dentro del intervalo $[0, 255]$. El motivo de elegir este tipo de datos es que, a pesar de que los datos podrían representarse originalmente en el espacio $\mathbb{R}^3$, la caracterización por el color, permite una visualización mucho más sencilla y cómoda. Esta ventaja hace que aplicaciones de este tipo de datos sean las utilizadas en ocasiones para explicar el funcionamiento de los SOMS [1], [2] .

En este caso, los datos se generarán de manera aleatoria y a partir de ellos se construirán los mapas mediante los distintos paquetes de R para poder ver las diferencias entre ellos y cuales producen una mejor implementación.

Se generan 400 observaciones con sus tres coordenadas entre 0 y 255. Puesto que los mapas van a permitir disponer los datos en rejillas de dos

dimensiones, para una visualización inicial, se pueden representar los 400 colores en una rejilla de 20×20.

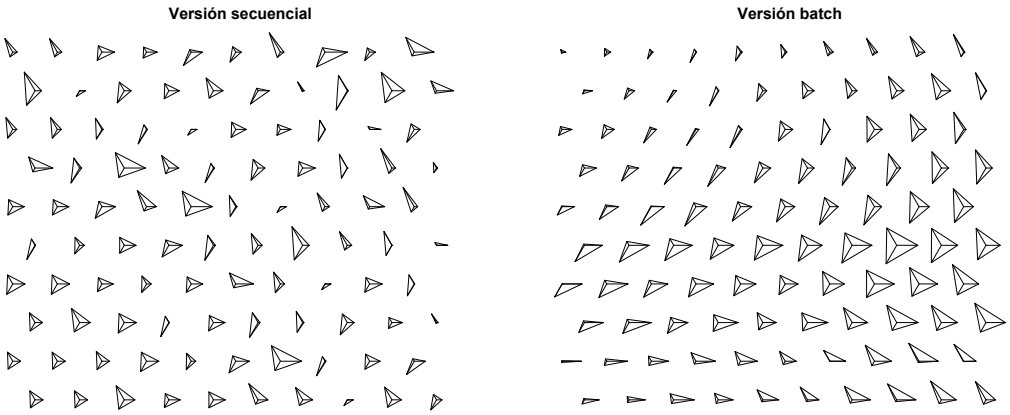
Figura 4.1: Representación de los datos generados aleatoriamente

Para construir el mapa, lo primero será determinar el tamaño y disposición de las neuronas. En primer lugar, siguiendo el criterio de Vesanto 2.1 de que un número adecuado de neuronas sería $5\sqrt{N}$ (como se vio en el capítulo 2), se construirá un SOM de 100 neuronas (10×10) dispuestas de forma hexagonal. Con este ejemplo se verá la implementación de los mapas autoorganizados con los distintos paquetes de R estudiados en el capítulo anterior para analizar las posibilidades que ofrece cada uno de cara a aplicaciones más complejas.

En primer lugar, el paquete `class` permite construir el mapa tanto con la versión secuencial del algoritmo como con la versión batch. Sin embargo, los resultados gráficos son muy pobres y sólo ofrece una representación mediante gráficos poligonales para los vectores modelo de las neuronas donde cada segmento representa el valor de la neurona para cada variable.

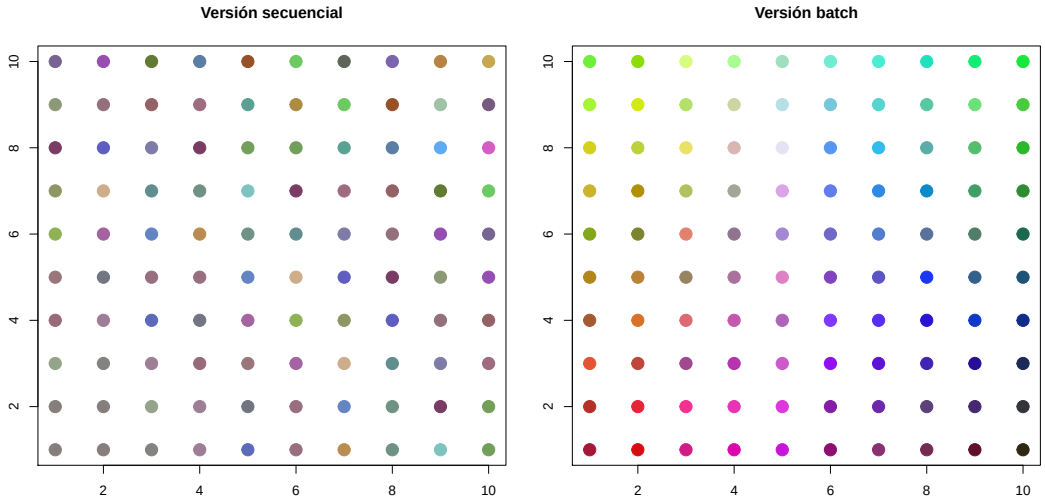
En la figura 4.2 se puede ver como la versión secuencial no ofrece buenos resultados puesto que no parece que las variables sigan una distribución creciente o decreciente a lo largo del mapa. En cambio, en la versión batch sí que se puede apreciar este comportamiento, viendo como por ejemplo en la esquina superior izquierda se encontrarían las observaciones con valores bajos en las tres variables mientras que la variable representada en la esquina superior del polígono crece según se avanza en el mapa de izquierda a

derecha, la variable del extremo derecho del polígono aumenta a medida que se baja en el mapa y la del extremo inferior toma valores bajos tanto en la parte superior como inferior del mapa, tomando los valores altos en la franja intermedia del mapa.



(a) Versión secuencial (b) Versión batch

Figura 4.2: SOMs con el paquete `class`



(a) Versión secuencial (b) Versión batch

Figura 4.3: Representación RGB del SOM con `class`

Por otro lado, el objeto que se construye, sólo almacena la información relativa a los vectores modelo, pero, redondeando las coordenadas a números enteros, puede emplearse para representar las neuronas con su color en escala RGB asociado.

Como se puede apreciar en la Figura 4.3, con el paquete `class` los resultados obtenidos a partir de la versión secuencial no son buenos mientras que con la versión batch, las neuronas parecen preservar un orden que se aprecia en que los colores tienen un orden. Para saber a que neurona está asociada cada observación, sería necesario volver a calcular la distancia de cada una de ellas a todos los vectores modelo ya que en este paquete esa información no se almacena.

A continuación, se verá como otros paquetes sí que ofrecen esta posibilidad y otras que permiten una mejor visualización.

El segundo de los paquetes que se ha visto en el capítulo anterior es el paquete `som`. Este paquete sólo permite construir la versión batch que se implementa a través de la función **som**. Este paquete aunque almacena más información útil del mapa, también es bastante pobre en cuanto a visualización y, de hecho, la función que permite representar gráficamente el SOM sólo es válida si los datos están normalizados.

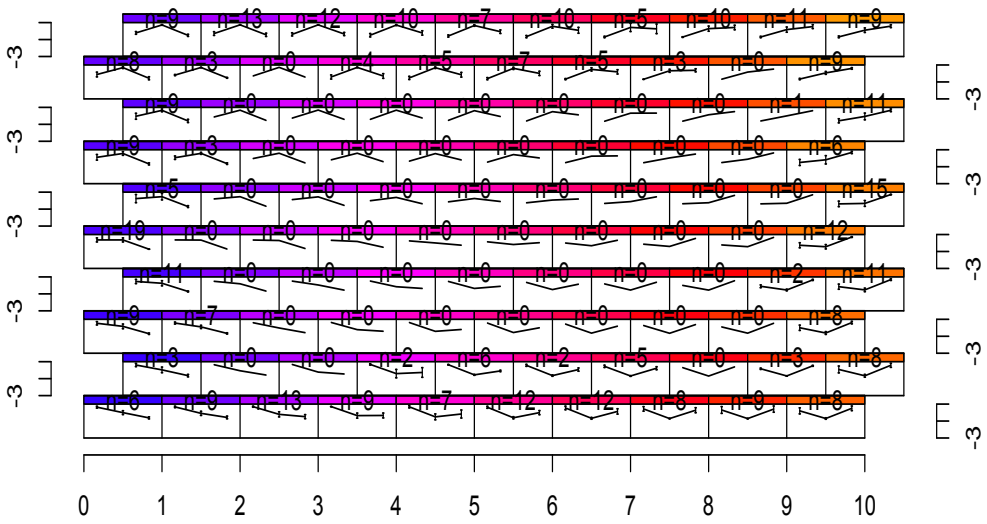


Figura 4.4: SOM con el paquete `som`

En la Figura 4.4 se representan las 100 neuronas con el número de observaciones asociadas a cada una de ellas y la representación de sus vectores modelo donde cada vértice representa el valor de una de las variables. Por ejemplo, se puede observar como la primera variable toma los valores más bajos en la esquina superior derecha y aumenta a medida que se avanza hacia la izquierda y hacia abajo. Por otro lado, la segunda variable tomará los valores máximos en la esquina superior izquierda mientras que la última de las variables decrece de derecha a izquierda. Sin embargo, puede observarse que las neuronas del centro del mapa no tienen observaciones asociadas y que se reparten a lo largo de los extremos. Como en este caso los vectores modelo están normalizados, no se pueden representar en la escala RGB y sería necesario entrenar un mapa con los datos en el intervalo $[0, 255]$ para poder representarlos gráficamente como en la Figura 4.5, donde se representan las neuronas en escala RGB y con un tamaño proporcional al número de observaciones.

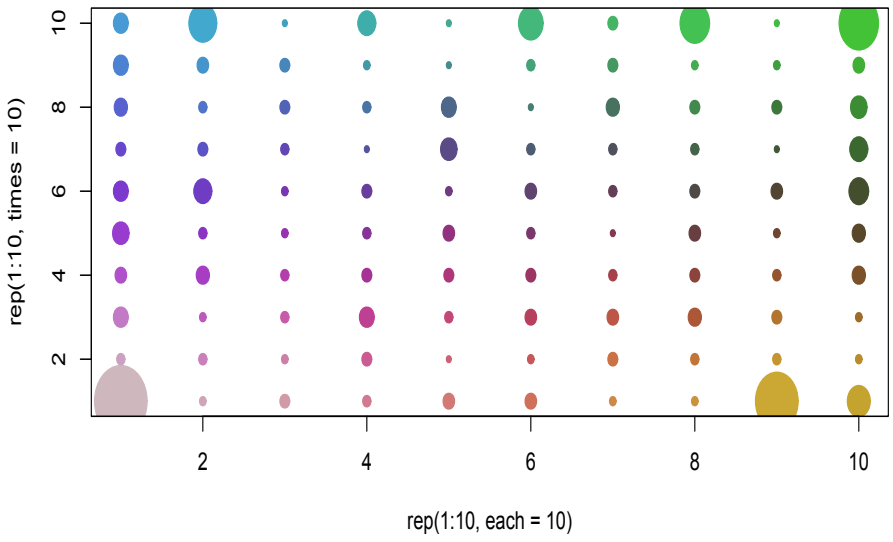


Figura 4.5: Representación de las neuronas en escala RGB

El tercero de los paquetes vistos es `yasomi`, que también implementa la versión batch y además permite a través de la función **som.tune** construir varios SOMs con distinto radio de vecindad inicial y elegir aquel SOM que

menor error tenga. En este caso se construirán 20 SOMs con radios que irán de 2 al radio máximo del mapa y se elegirá aquel que minimiza el error.

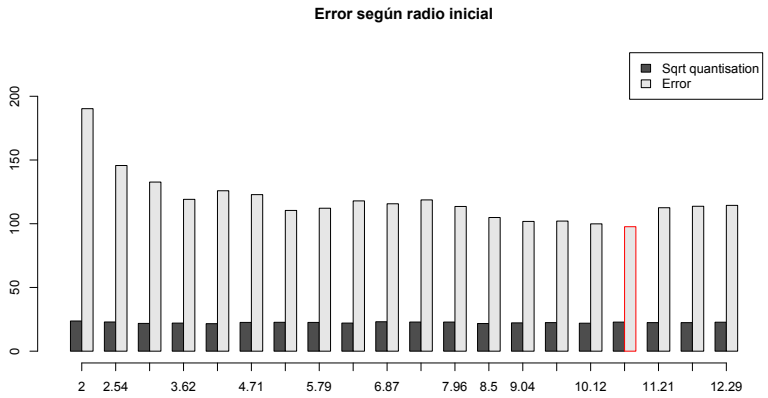


Figura 4.6: Elección del radio incial del SOM con `yasomi`

En la gráfica 4.6 se representan los errores para los 20 SOMs construidos. De esta forma se puede elegir aquel radio inicial que proporciona el mapa con los menores errores. En la gráfica se pueden observar los valores de la raíz del error de cuantificación 2.17 y el error de Kaski y Lagus propuesto e implementado en el paquete `yasomi` que combina el error de cuantificación con que se mantengan las cualidades topológicas. En este caso, se puede observar como escoger un radio inicial pequeño puede provocar más error que elegir uno más elevado.

Este paquete ofrece más opciones de visualización que permiten ver como decrece el error en cada paso o que permiten representar en el mapa tanto los vectores modelo de las neuronas como los datos asociados a cada una de ellas de distintas formas como puede ser con representaciones poligonales o de líneas. Además permite construir un 'hitmap' en el que se representa cada neurona con un tamaño proporcional al número de observaciones asignadas y que sirve para ver si hay regiones del mapa separadas. También se pueden generar gráficos que permitan evaluar el comportamiento de cada variable a lo largo del mapa individualmente o generar la matriz U, en la que se representa la distancia de cada neurona al resto de sus vecinas.

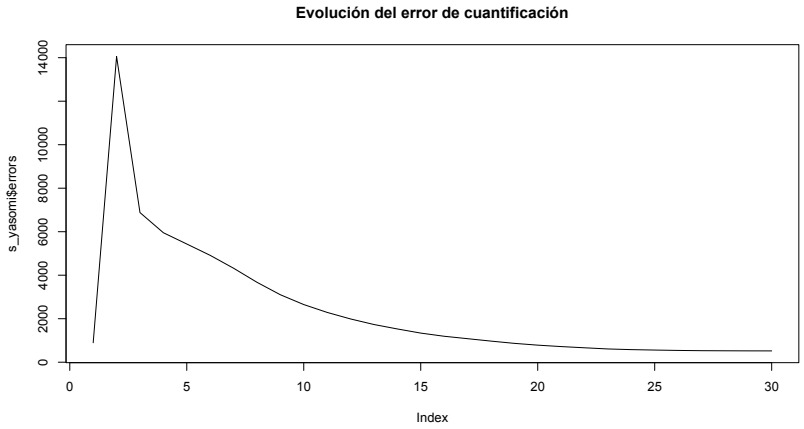


Figura 4.7: Evolución del error en cada paso

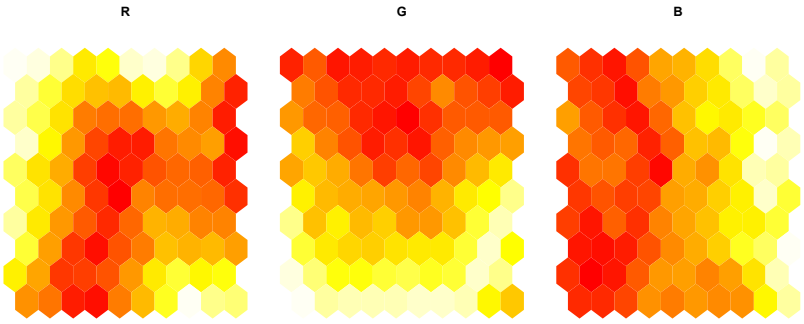
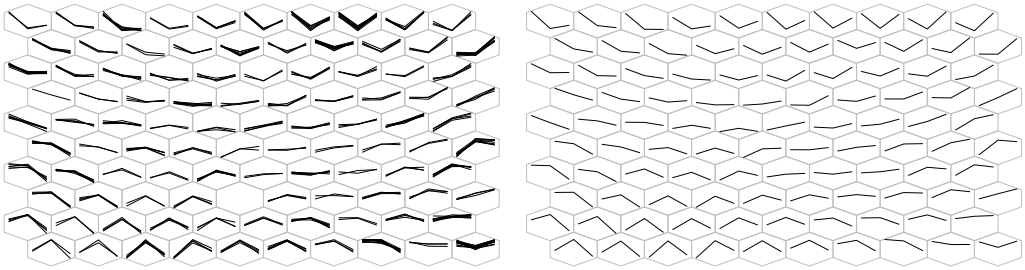


Figura 4.8: Comportamiento de cada componente con `yasomi`

En la Figura 4.7 se observa como en pocas iteraciones del algoritmo batch el error se estabiliza. En la figura 4.8 se representa el comportamiento de cada una de las variables a lo largo del mapa en una escala que va desde el color rojo (valores bajos) al blanco (valores más altos). En ella se puede ver como la primera de las componentes toma principalmente valores bajos en el centro creciendo hacia distintos extremos del mapa mientras que las otras dos variables siguen un comportamiento más uniforme, aumentando la segunda de la parte superior a la inferior del mapa y la tercera creciendo de izquierda a derecha.



(a) Observaciones (b) Vectores modelo

Figura 4.9: Representación de los datos y los vectores modelo

En la gráfica 4.9 se representan mediante segmentos las observaciones asociadas a cada neurona (izquierda) y los vectores modelo de las neuronas (derecha). Puede observarse como en 4.9a, aunque haya neuronas con unas cuantas observaciones, sus valores son muy similares entre ellas y que las formas de éstas están muy bien reflejadas en los vectores modelo de las neuronas a las que están asociadas (4.9b) por lo que el mapa se ajusta bastante bien a los datos.

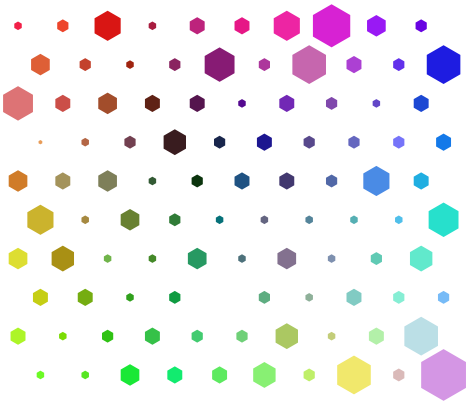


Figura 4.10: Nº de observaciones y representación RGB del SOM con `yasomi`

Empleando la representación RGB de los vectores asociados a las neuronas, se puede observar en las Figuras 4.10 y 4.11 como los vectores quedan bastante ordenados preservando el orden topológico y en la matriz U pueden observase las distancias entre las neuronas, donde se ve que existe una neurona que guarda más distancia con sus vecinas (color blanco) mientras que otras están muy próximas (color rojo).

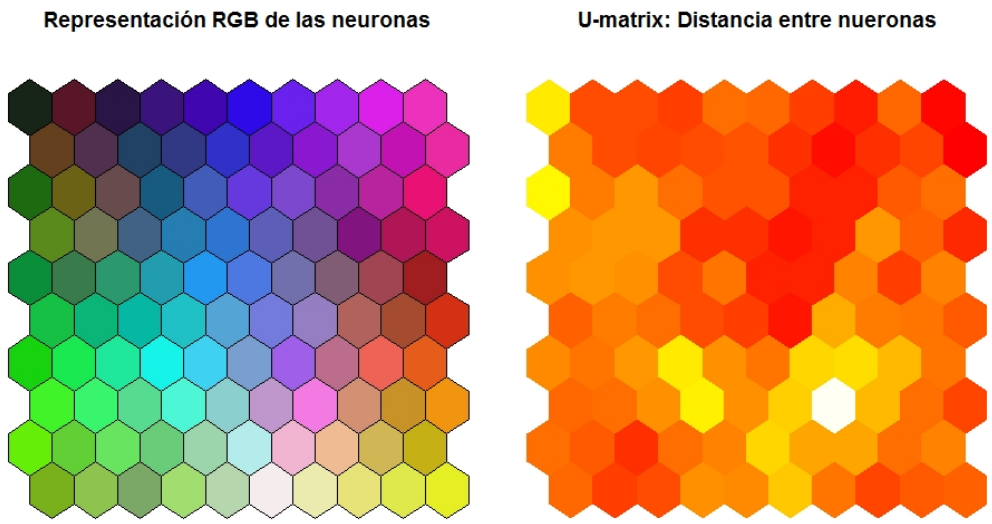
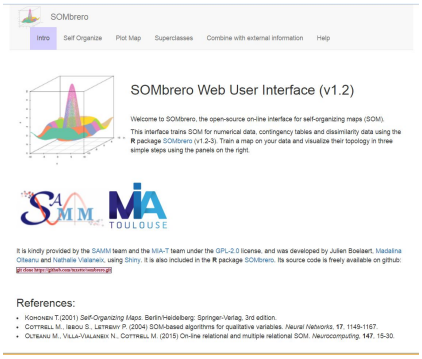


Figura 4.11: Representación del SOM y U-matrix

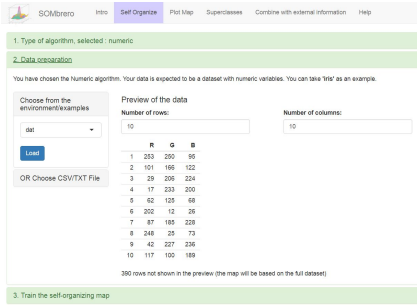
Por otro lado, visualmente hablando, el paquete `SOMbrero` ofrece las mejores opciones, ya que puede abrirse un entorno gráfico que permite crear el SOM mediante el algoritmo secuencial y obtener gran cantidad de gráficos de manera sencilla sin la necesidad de escribir código.

Para ello, una vez cargado el paquete en R, será necesario ejecutar la orden **sombreroGUI()**. Este comando abrirá una ventana en la que paso a paso se podrá elegir los datos, seleccionar los parámetros para el entrenamiento y después representar el mapa gráficamente con múltiples opciones. Además, a lo largo de los pasos, se muestra el código ejecutado, lo cual permitirá replicar los gráficos desde el propio R.

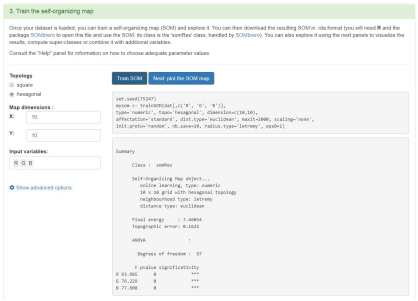
Este paquete, además, tiene la posibilidad de crear un objeto *SuperClass* que permite dividir el SOM en distintos clústeres construidos mediante clustering jerárquico a partir de los vectores modelo asociados a las neuronas. Para ello, utiliza las distancias cuadráticas entre cada par de neuronas y a continuación realiza la agrupación mediante cualquiera de los métodos que acepta la función **hclust** (método de Ward por defecto).



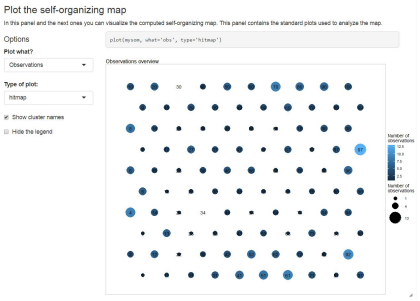
(a) Inicio interfaz gráfica



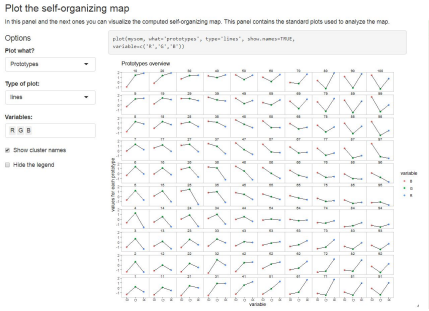
(b) Lectura de datos



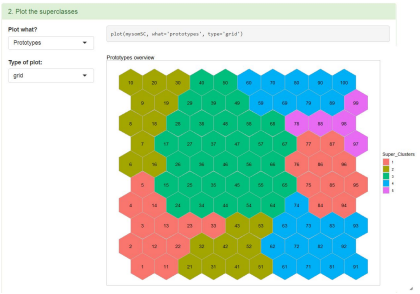
(c) Entrenamiento SOM



(d) Hitmap



(e) Vectores modelo

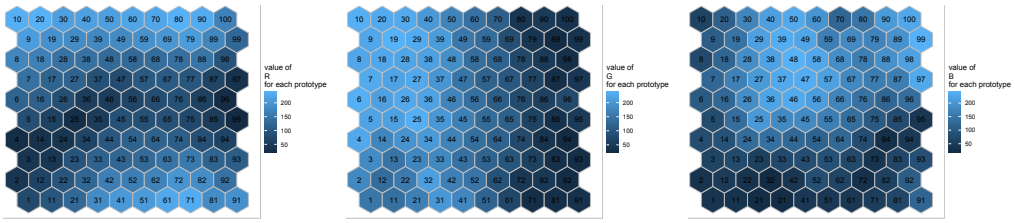


(f) Clustering

Figura 4.12: Interfaz gráfica del paquete `SOMbrero`

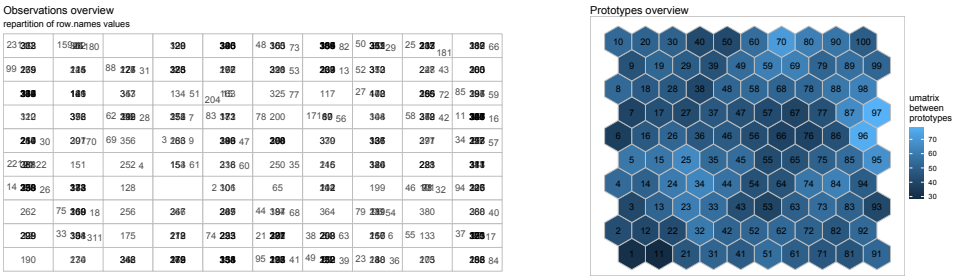
En la Figura 4.12 se pueden ver distintos momentos del proceso de construcción del SOM mediante la interfaz gráfica, así como el *hitmap* (4.12d) con el número de observaciones asociadas a cada neurona o la representación de las componentes de cada vector modelo (4.12e). Por último, en 4.12f se puede observar el resultado obtenido de aplicar un clustering de 5 grupos a las neuronas del mapa permitiendo dividirlo en 5 regiones distintas formadas por neuronas con valores similares.

Además, existen más representaciones gráficas que permiten ver la evolución de cada componente (Figura 4.13), qué observaciones están asociadas a cada neurona 4.14a o la matriz U (4.14b).



(a) Componente R (b) Componente G (c) Componente B

Figura 4.13: Comportamiento componentes del SOM con `SOMbrero`



(a) Observaciones en cada neurona (b) U-matrix

Figura 4.14: Otras visualizaciones del paquete `SOMbrero`

Además, el paquete `SOMbrero` ofrece la posibilidad de crear copias intermedias de los vectores modelo que permiten ver la evolución del entrenamiento del mapa. En este caso, al emplear la representación RGB de los vectores puede representarse fácilmente la evolución. En la Figura 4.15 se puede ver como se inician los vectores aleatoriamente y van modificándose hasta llegar al estado final.

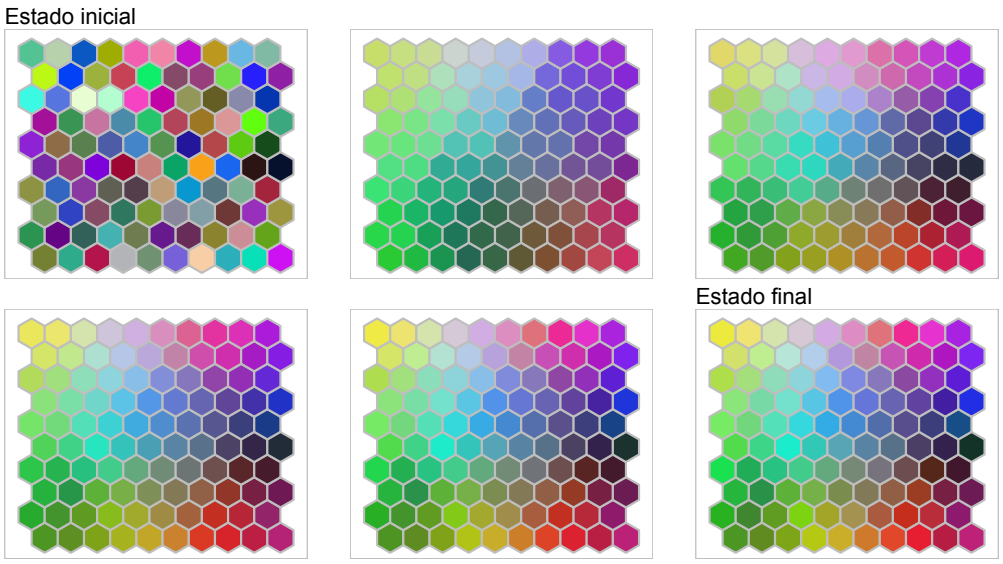


Figura 4.15: Evolución de los vectores modelo durante el entrenamiento

Por último, también se han implementado dos mapas con el paquete `kohonen` de R que permite construir mapas autoorganizados con las dos versiones (secuencial y batch) del algoritmo.

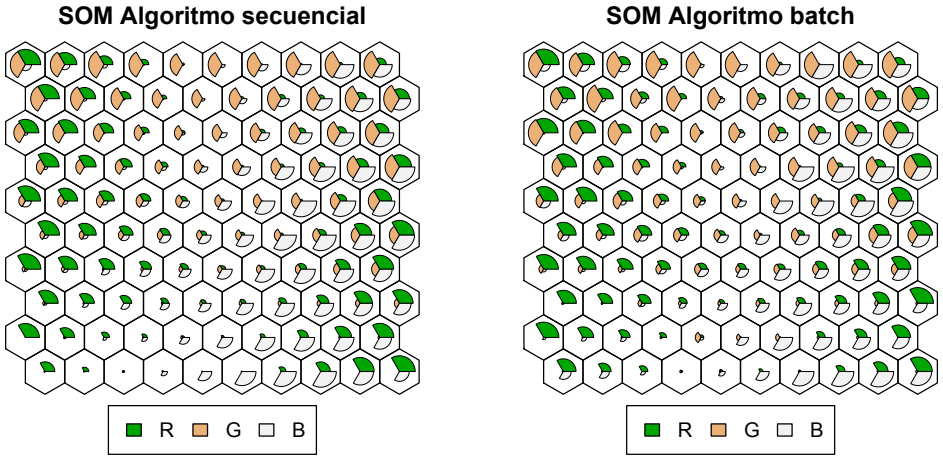
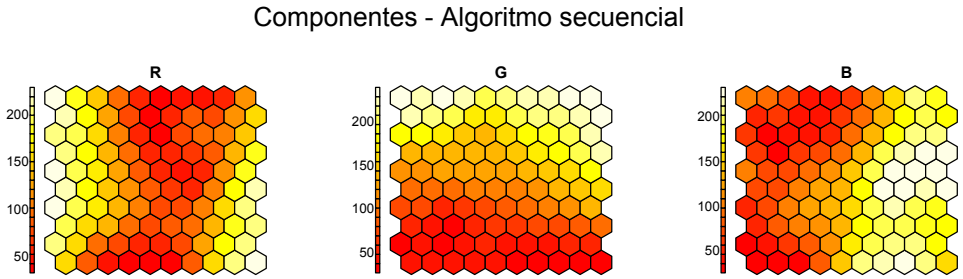
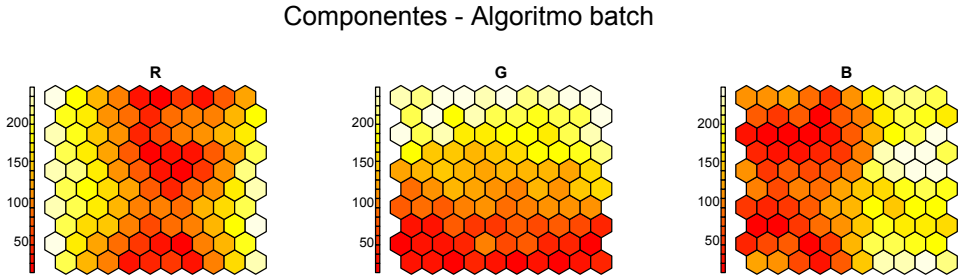


Figura 4.16: Representación de los SOM con `kohonen`



(a) Componentes en el SOM secuencial



(b) Compontentes en el SOM batch

Figura 4.17: Comportamiento de las componentes en los dos SOMs

En las Figuras 4.16 y 4.17 se puede observar como en ambos mapas la primera variable crece de la parte central hacia los laterales, la segunda crece de la parte inferior a la superior y la tercera y última variable aumenta desde la parte izquierda del mapa hacia la derecha. Para observar que mapa se ajusta mejor a los datos, en la Figura 4.18 se puede ver que en la versión batch las distancias de los puntos a sus neuronas asociadas es menor.

Otras posibles representaciones son el hitmap con el número de observaciones asociadas a cada neurona o la u-matrix donde se puede apreciar la distancia de cada neurona con sus vecinas (Figura 4.19).

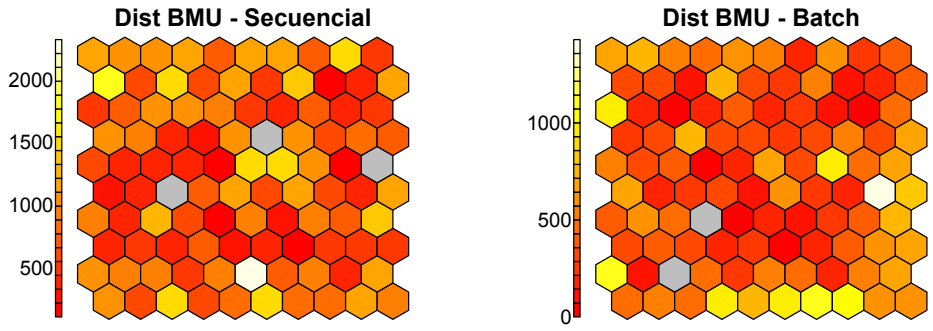
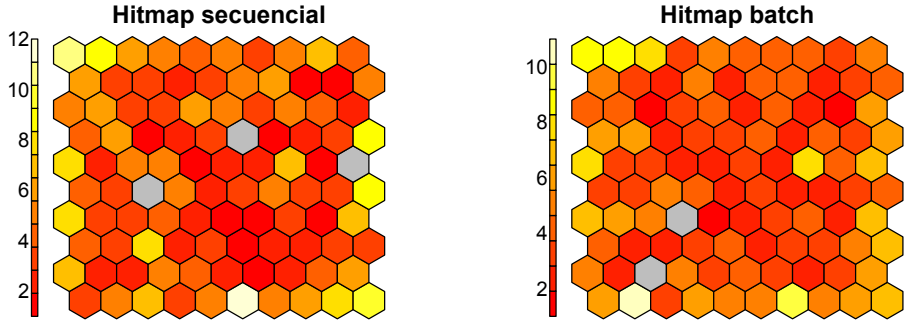
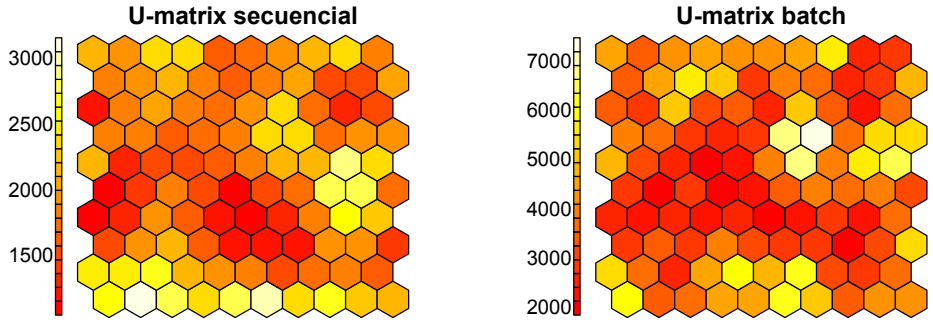


Figura 4.18: Distancia media de los datos a las neuronas asociadas



(a) Hitmap de los SOMs



(b) U-matrix de los SOMs

Figura 4.19: Hitmap y U-matrix de los SOMs de `kohonen`

Por último, en este caso, se puede utilizar la representación RGB de los vectores para 'colorear' las neuronas, y además en este caso, pueden representarse en cada neurona los puntos asociados correspondientes. En la Figura

4.20 puede observarse como aparte de quedar la malla claramente organizada según los colores, los puntos asociados a cada neurona son muy similares al vector modelo de la neurona.

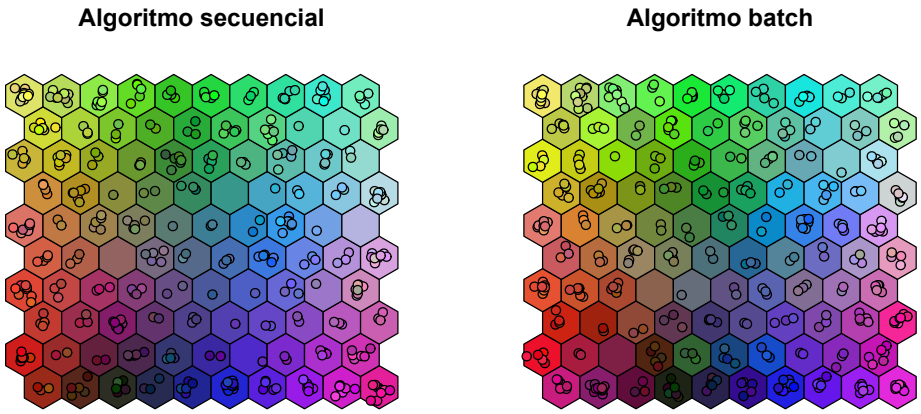


Figura 4.20: Representación RGB del mapa y las observaciones

Una vez vistos todos los paquetes y las opciones que ofrecen, se puede concluir que el mejor es `SOMbrero` por todas las posibilidades gráficas que ofrece seguido de `kohonen`. Por tanto, `SOMbrero` será el empleado en el siguiente apartado, donde se construirá un SOM en el caso de datos reales mucho más complejos.

4.2. Datos reales: COVID-19

El desarrollo de este trabajo, al igual que toda actividad en cualquier ámbito durante este año 2020, se ha visto marcada por la crisis sanitaria provocada por el coronavirus Sars-Cov-2. Por ello, se consideró oportuno e interesante poder aplicar el método estudiado a lo largo de esta memoria a los datos a los que hemos vivido condicionados durante estos últimos meses.

Conjunto de datos

Para ello, se han tomado como conjunto de datos los datos diarios de cada país a lo largo del tiempo recogidos y publicados por el CSSE de la

universidad de Johns Hopkins ("COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University") en https://github.com/CSSEGISandData/COVID-19 a fecha de 6 de Julio de 2020.

Este repositorio dispone de gran cantidad de datos tanto a nivel mundial como a nivel estatal en el caso de EEUU. En esta aplicación se tomarán los datos a nivel mundial para intentar construir un mapa autoorganizado que permita representar visualmente los países con comportamientos más similares a lo largo de la pandemia.

El conjunto de datos original contiene 34 columnas con información de diferentes indicadores para cada uno de los países entre el 31 de Diciembre de 2019 y el 5 de Julio de 2020. Las primeras cuatro variables del conjunto de datos corresponden al código ISO del país, el continente, el nombre del país y la fecha. A continuación, se dispone de algunas variables relativas a la evolución de la pandemia como son el número de casos confirmados, fallecidos y tests realizados de forma acumulada y diaria tanto en términos absolutos como por millón de habitantes. Por último, se incluyen algunas variables socioeconómicas de los países no relacionadas con la pandemia como pueden ser los habitantes, la esperanza de vida, la edad media, el PIB..., que en esta aplicación no serán utilizadas.

Preprocesamiento

Para la construcción de un SOM en este trabajo se ha elegido como variable de interés el número de casos diarios por millón de habitantes para cada país. Por tanto, será necesario tratar el conjunto de datos original en el que cada registro corresponde a cada uno de los días para cada uno de los países y pasar a un conjunto de datos en el que cada fila corresponda únicamente a un país y las columnas sean el valor de la variable elegida para cada uno de los días. Esta conversión se realiza de manera sencilla con la orden **pivot_wider()** del paquete `tidyr` de R.

Tras convertir el conjunto de datos al formato de interés, se suprimirán aquellas filas que puedan corresponder a países de los que no se dispone de información, así como de dos filas que corresponden a los datos a nivel mundial (identificados con los nombres "World" e "International"). Finalmente se dispondrá de un conjunto de datos formado por 209 observaciones y 178

columnas, de las cuales las dos primeras corresponden al código ISO y al nombre del país y el resto a los datos de la variable. En el Anexo I (A.1) se puede encontrar una tabla con la relación entre estos códigos y sus correspondientes países.

Construcción del SOM

En esta aplicación, se utilizará el paquete `SOMbrero` por ofrecer más opciones gráficas y la aplicación de clustering sobre las neuronas que permitirá agrupar los países en función de su semejanza en el comportamiento. Tras el preprocesamiento de los datos, el número de observaciones (países) de los que se dispone información es de 209, por lo tanto, siguiendo el criterio de Vesanto (2.1), un número apropiado de neuronas para el mapa sería $5\sqrt{209}$, es decir, unas 72 neuronas. Por tanto, se construirá el mapa de tamaño 9×8 con las neuronas en disposición hexagonal.

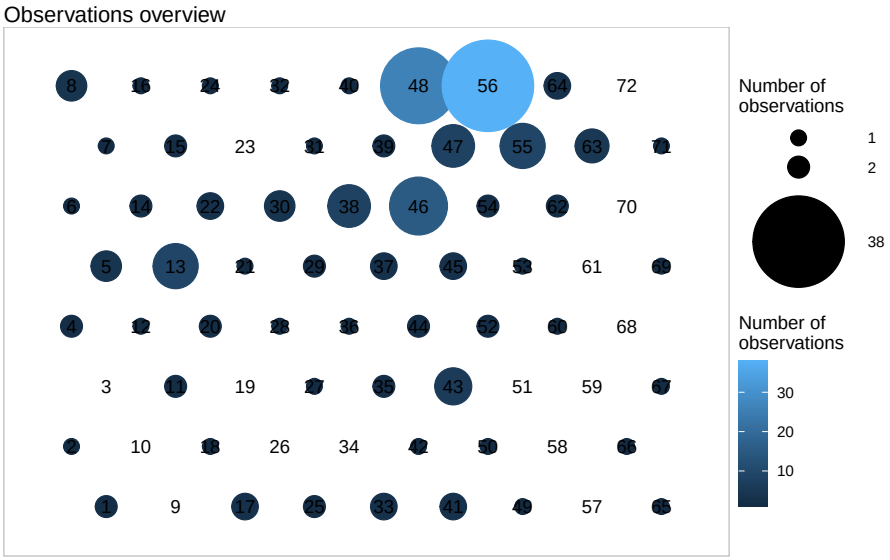


Figura 4.21: hitmap casos Covid

Observando el *hitmap* del mapa construido (Figura 4.21) puede observarse como existe una zona del mapa y, especialmente, una neurona con muchas observaciones asociadas y luego muchas con pocas, por lo que podría interpretarse que existen muchos países con un comportamiento similar (al menos en

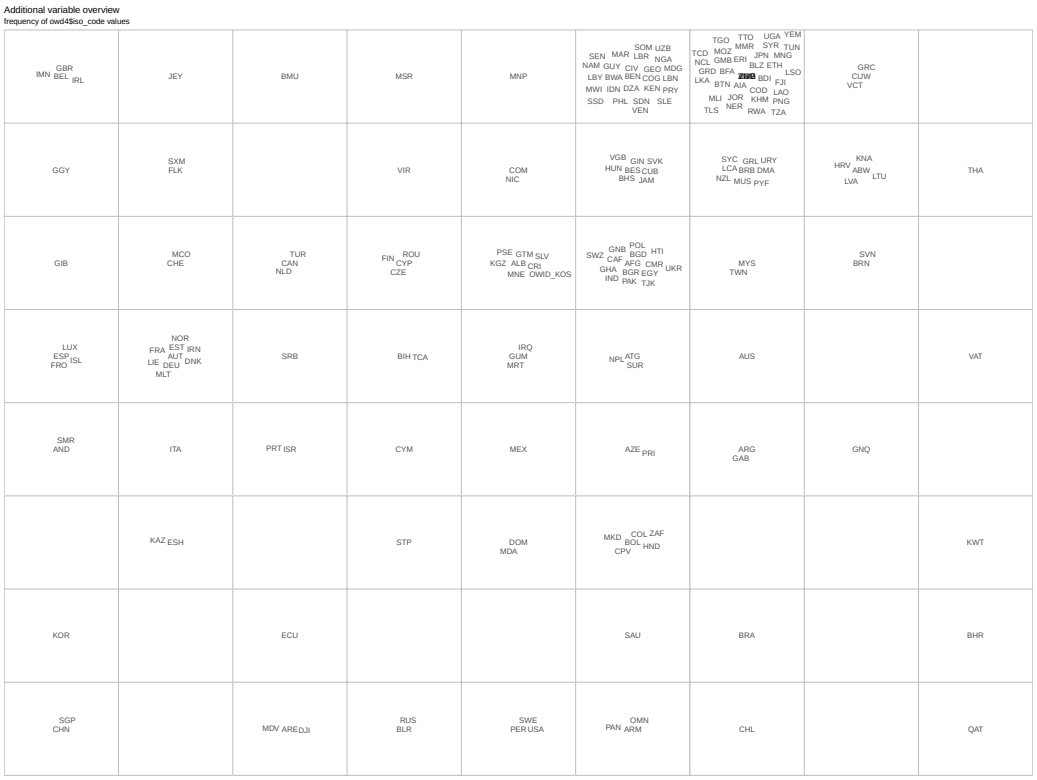


Figura 4.22: Representación de los países en el mapa

cuanto a datos proporcionados). En la Figura 4.22 se representan qué países están asociados a cada neurona (Correspondencia códigos ISO en A.1). En ella se puede observar que las neuronas con muchas observaciones son países con pocos casos notificados. También se puede observar como China figura en la parte inferior izquierda del mapa junto a Singapur y cerca de Corea del Sur. También se puede ver como a medida que se avanza hacia arriba se pueden encontrar los países europeos como Italia, Francia o Alemania y a medida que se avanza hacia la derecha en la parte inferior se encuentran los países que han mantenido un crecimiento más continuo como Rusia, Suecia, Estados Unidos, Suecia, Chile o Brasil. En la esquina inferior derecha se puede encontrar a Qatar y Bahrain que destacan por ser los países con una mayor incidencia en proporción a la población.

Para saber mejor el perfil de cada neurona se puede ver el perfil completo de los vectores modelos asociados a cada neurona que representarán la evolución de la pandemia en términos relativos. En la Figura 4.23 se puede confirmar que la parte superior y central está asociada a los países con poca afectación. En la parte izquierda del mapa, se encuentran en la parte inferior

los países donde se originó la pandemia (China) mientras que ha medida que se sube se puede observar como el pico de la evolución se encuentra en el centro, lo que se correspondería con los meses de marzo y abril. Por último, se puede apreciar que en la parte inferior, el pico de la curva se va desplazando hacia las últimas fechas a medida que se avanza en el mapa hacia la derecha donde se encuentran los casos que actualmente (6 de julio) están siendo los más golpeados.

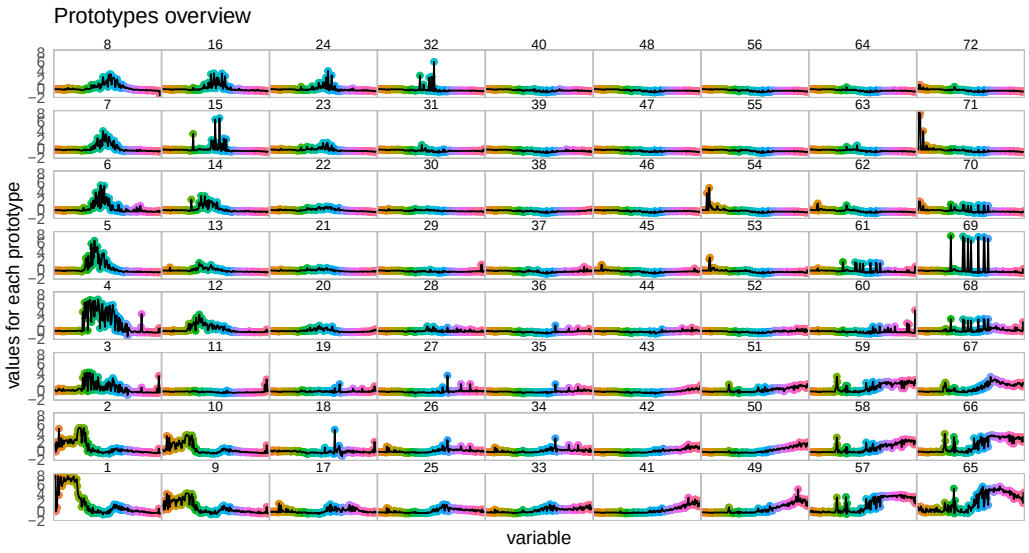


Figura 4.23: Perfil de evolución de cada neurona del mapa

También se puede observar el comportamiento individual de algunos días a lo largo del tiempo para observar como se desplazan los valores altos de unas neuronas a otras, viendo claramente en la Figura 4.24 como la esquina inferior izquierda corresponde al origen de la pandemia, el resto del lateral izquierdo está asociado a los países con mayor incidencia en los meses de marzo y abril mientras que los países con una afectación más tardía (mayo, junio y julio) se encuentran en la parte inferior derecha.

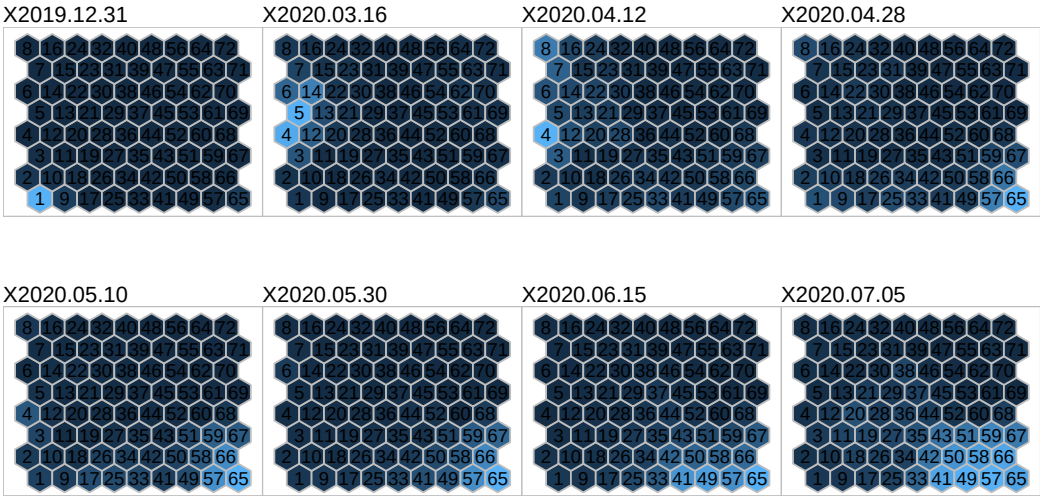


Figura 4.24: Representación del mapa para distintas componentes (días)

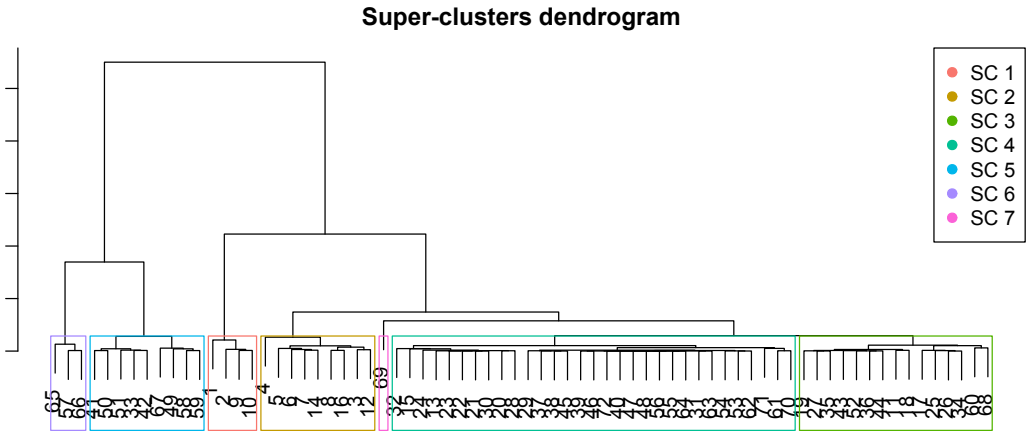


Figura 4.25: Dendograma del clustering jerárquico

Una vez construido el mapa, se pueden agrupar las neuronas en un menor número de grupos, pasando de las 72 a un número escogido aplicando clustering jerárquico siguiendo el criterio de Ward (Figura 4.25). En este caso se han escogido 7 grupos permitiendo agrupar las 72 neuronas del mapa en 7 clústeres (Figura 4.26). Observando el dendograma, existen grupos con muy

pocas neuronas, incluso hay uno que sólo incluye una por lo que podría haberse planteado una división en menos grupos. Sin embargo, en caso de reducir el número de clústeres, se agruparían los dos de la parte derecha formando un clúster demasiado grande que reduciría la calidad e interpretabilidad de los resultados obtenidos.

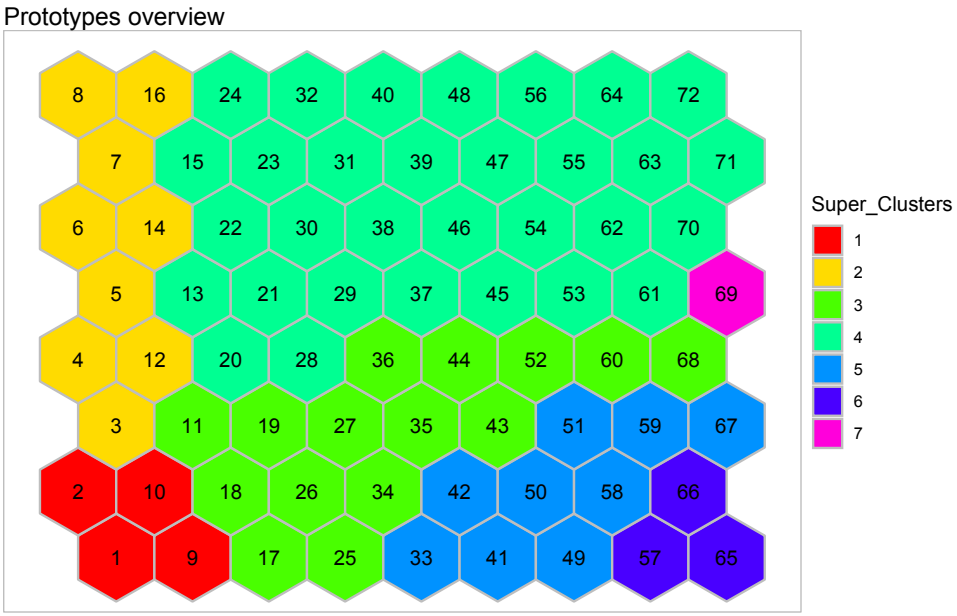


Figura 4.26: Agrupación de neuronas del SOM

Una vez se ha determinado qué países están asociados a cada grupo, puede calcularse directamente el perfil medio de cada clúster como la media de los datos de sus observaciones. En la tabla A.1 del Anexo A con la correspondencia entre los códigos ISO y los países, se incluye una columna que indica a que clúster está asociado cada uno de ellos.

En la Figura 4.27 se representan estos perfiles medios para poder comparar el comportamiento promedio de cada grupo. En esta gráfica se ha omitido el séptimo y último grupo ya que se corresponde únicamente con El Vaticano y no aporta información relevante más allá de crear algo de confusión en el gráfico debido a sus valores tan dispares.

Por último, con la asignación de cada país a un clúster se puede representar en un mapa del mundo cada país con el color del cluster asignado utilizando el paquete `rworldmap` de R, [29] de manera sencilla.

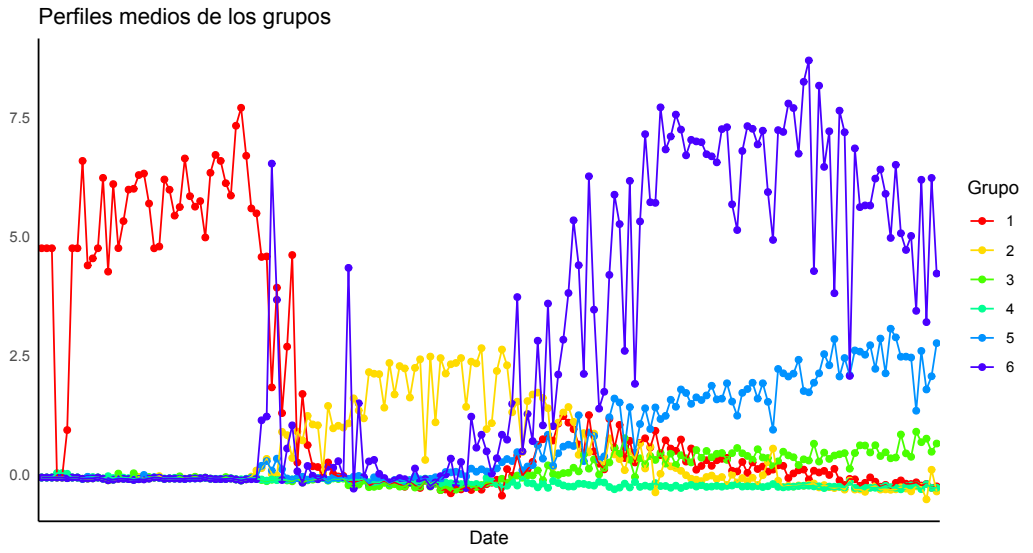


Figura 4.27: Perfiles medios de los clústeres

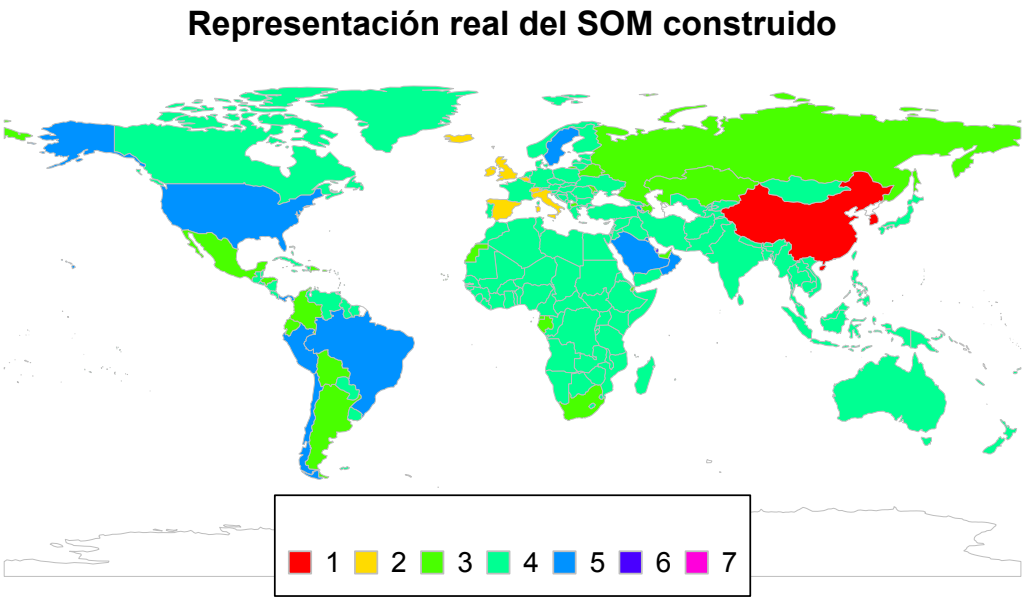


Figura 4.28: Representación de los países

Tanto en el mapa mundial (Figura 4.28) como en la gráfica de los perfiles medios de los clústeres construidos (Figura 4.27) se pueden observar los distintos comportamientos de los países a lo largo del tiempo. Por un lado, el primero de los grupos corresponde a aquellos países que tuvieron su pico en el momento inicial como China, Singapur y Corea del Sur, ya que sus curvas son muy diferentes a las del resto de países, teniendo su pico máximo en el momento inicial del coronavirus.

A continuación, el segundo grupo es aquel que está asociado a las neuronas del lateral izquierdo del mapa (Figura 4.26). En este grupo se encuentran algunos países, principalmente europeos como Italia, España, Bélgica, Gran Bretaña, Irlanda o Islandia. Observando los perfiles de las neuronas asociadas, se puede observar como estos países son los que sufrieron una mayor incidencia tras el primer grupo en los meses de marzo y abril y que corresponden con el momento en el que el coronavirus se extendió a lo largo y ancho del planeta.

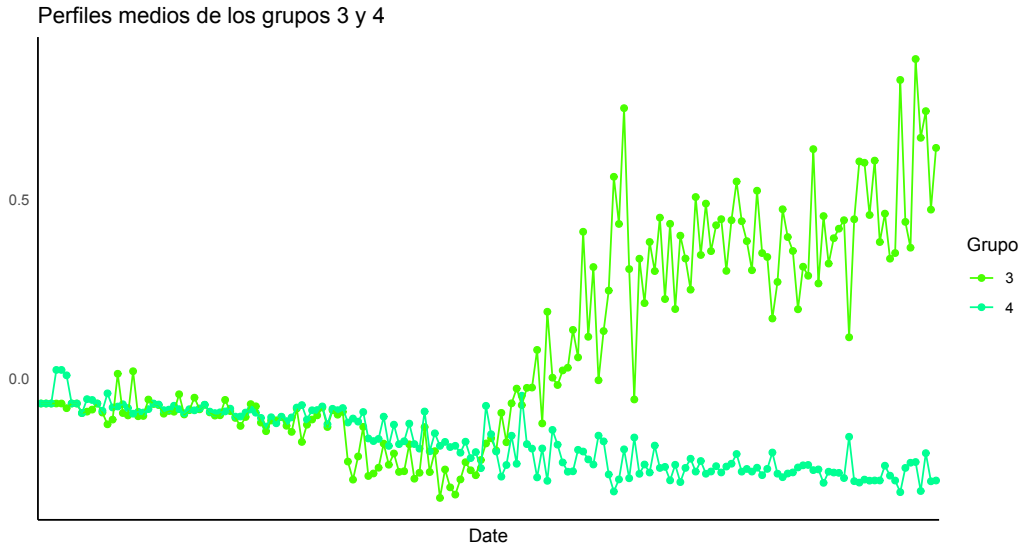


Figura 4.29: Perfil medio clústeres 3 y 4

Los dos siguientes grupos, el 3 y el 4, son los correspondientes a la parte más central del SOM y son los dos grupos más cercanos entre ellos observando el dendograma. Estos dos clústeres son los que se asocian con unas series más estables y de una incidencia menor en función de la población. Para poder encontrar las diferencias de comportamiento entre estos dos grupos, en la

figura 4.29 se han representado sólo los perfiles medios de estos dos grupos para que el gráfico no se vea afectado por los altos valores del resto de grupos, comprobando su distanciamiento a medida que avanza el tiempo.

Como se puede apreciar, de entre los 2 grupos, el cuarto puede asociarse con aquellos países con menor incidencia del virus en todo momento mientras que en el tercer clúster se encuentran países que a pesar de no haber sufrido una gran afectación tienen algo más de incidencia en las últimas fechas que en el cuarto grupo como pueden ser Rusia, Sudáfrica, México, Colombia o Argentina que son países que han terminado reportando un considerable número de casos. En el grupo de los países menos afectados (grupo 4) se encuentra gran parte de África y el sur de Asia probablemente asociado a la poca capacidad de diagnóstico que puedan tener estos países o países del centro de Europa o Francia y Portugal que lograron contener la ola de una manera mejor que sus países vecinos. También destaca en este grupo India ya que, a pesar de ser uno de los países con más casos reportados, la incidencia respecto a la gran población que tiene sigue manteniéndose en valores bajos.

El quinto grupo y el sexto (hacia la esquina inferior derecha del mapa), se corresponden con los países con una mayor afectación en las últimas fechas de las que se tienen datos, pudiéndose observar en los perfiles de estos grupos los valores más altos en la parte final del gráfico y una tendencia creciente. En el grupo 6 se encuentran los casos extremos de Qatar y Bahrain por ser aquellos con valores más altos y que se corresponden con los dos países con mayor incidencia acumulada. En el quinto grupo, se encuentran el resto de países con un elevado número de casos diarios en junio y principios de julio como son Brasil, Estados Unidos, Chile o Suecia, que destacaban por ser países que seguían en crecimiento cuando la primera ola parecía haberse reducido en gran parte del planeta.

Por último, en el séptimo y último de los grupos se encuentra exclusivamente El Vaticano, al cual le corresponde un perfil bastante diferente al resto al tener valores altos en días alternos (Figura 4.23). Esto se debe a que es un caso especial debido a que algún caso afecta mucho en proporción a la población que tiene (De hecho, en este país sólo se han diagnosticado 12 casos para una población de 809 habitantes a lo largo de todo el tiempo).

Analizando la situación por continentes, es apreciable que, a fecha de 7 de julio, América es el continente con peor proyección ya que una gran mayoría de países (especialmente los más poblados) están en fase de crecimiento. Los resultados obtenidos en África están en gran parte condicionados con la limitada capacidad de diagnosticar casos. Por otro lado, Europa y Asia,

salvo Rusia y la Península Arábiga, parecen haber conseguido controlar la evolución de la pandemia, al menos en una posible primera ola.

En el desarrollo de este caso se ha podido implementar el método de los mapas autoorganizados para conseguir una representación visual en dos dimensiones del conjunto de datos permitiendo mantener cerca los países con comportamientos similares a lo largo del tiempo.

Capítulo 5

Conclusiones

El mundo de las técnicas de análisis de datos no se reduce a los métodos más conocidos y estudiados, continuamente aparecen nuevos métodos que se van popularizando y que aportan nuevas ventajas y visiones respecto a los más habituales. Uno de estos métodos es el de los Mapas Autoorganizados, que gracias al desarrollo de Teuvo Kohonen fueron poco a poco aceptados y utilizados cada vez por más gente.

Este método destaca por partir de una idea básica y sencilla que funciona bien en la práctica pero cuya base teórica, como se ha visto en este trabajo, está desarrollada sólo para los contextos más sencillos. A lo largo de este Trabajo de Fin de Grado también se ha constatado que es un método que se puede utilizar con distintos objetivos, como la representación de datos de alta dimensión, el clustering e incluso pueden utilizarse los mapas para clasificar nuevos datos.

Adicionalmentese se ha visto que, debido a su creciente popularidad, existen diferentes opciones para su implementación en el lenguaje R, y tras la construcción de mapas con ellos, se ha podido concluir en este trabajo que uno de los mejores paquetes es `SOMbrero` debido al gran abanico de opciones que ofrece tanto en la construcción como en la visualización y en la facilidad de uso.

Por un lado, este trabajo ha permitido constatar el continuo desarrollo y avance de nuevas técnicas y su relación con otros campos de la ciencia como la biología. Por otro lado, más concretamente, se ha encontrado en los SOMs una herramienta potente y útil que se puede emplear para tareas y objetivos diferentes como la búsqueda de patrones, proyección en espacios gráficamente visualizables, clustering,...

En este trabajo se han considerado dos aplicaciones prácticas de este método. La primera de ellas, trata con un caso clásico con datos sencillos y está enfocada a entender mejor las posibilidades y funcionalidades del método, así como sus implementaciones en R. La segunda aplicación considera un conjunto de datos de muy alto interés en el momento actual como es la evolución a nivel mundial de la pandemia de COVID-19, tomando como indicador los casos diarios notificados por millón de habitantes de cada uno de los países. Para estos datos el método, junto con un procesado posterior mediante clustering, permite una clasificación de los países en función de la evolución de la pandemia en cada uno de ellos durante el periodo en estudio que podría utilizarse para comprender, por ejemplo, cómo las diferentes medidas adoptadas por los países y el momento en el que estas medidas se han adoptado, han influido en la evolución de la enfermedad en los mismos o interpretar como se ha ido desplazando el foco de la pandemia entre los distintos continentes a lo largo de este periodo.

El desarrollo de este trabajo, abre la puerta a realizar otros estudios que quedan fuera del alcance del desarrollo de esta memoria como podrían ser la comparación de rendimientos de los distintos paquetes estudiados, el estudio más detallado de su implementación en otros lenguajes o la utilización de estos métodos para otro tipo de datos no vectoriales como pueden ser las tablas de contingencia.

Anexos

Anexo A

Códigos Países

ISO	País	Grupo	ISO	País	Grupo
AFG	Afghanistan	4	KGZ	Kyrgyzstan	4
ALB	Albania	4	LAO	Laos	4
DZA	Algeria	4	LVA	Latvia	4
AND	Andorra	2	LBN	Lebanon	4
AGO	Angola	4	LSO	Lesotho	4
AIA	Anguilla	4	LBR	Liberia	4
ATG	Antigua and Barbuda	4	LBY	Libya	4
ARG	Argentina	3	LIE	Liechtenstein	4
ARM	Armenia	5	LTU	Lithuania	4
ABW	Aruba	4	LUX	Luxembourg	2
AUS	Australia	4	MKD	Macedonia	3
AUT	Austria	4	MDG	Madagascar	4
AZE	Azerbaijan	3	MWI	Malawi	4
BHS	Bahamas	4	MYS	Malaysia	4
BHR	Bahrain	6	MDV	Maldives	3
BGD	Bangladesh	4	MLI	Mali	4
BRB	Barbados	4	MLT	Malta	4
BLR	Belarus	3	MRT	Mauritania	4
BEL	Belgium	2	MUS	Mauritius	4
BLZ	Belize	4	MEX	Mexico	3
BEN	Benin	4	MDA	Moldova	3
BMU	Bermuda	4	MCO	Monaco	2
BTN	Bhutan	4	MNG	Mongolia	4
BOL	Bolivia	3	MNE	Montenegro	4
BES	Bonaire Sint Eustatius and Saba	4	MSR	Montserrat	4
BIH	Bosnia and Herzegovina	4	MAR	Morocco	4
BWA	Botswana	4	MOZ	Mozambique	4
BRA	Brazil	5	MMR	Myanmar	4
VGB	British Virgin Islands	4	NAM	Namibia	4
BRN	Brunei	4	NPL	Nepal	4

*continúa en la siguiente página**

ISO	País	Grupo	ISO	País	Grupo
BGR	Bulgaria	4	NLD	Netherlands	4
BFA	Burkina Faso	4	NCL	New Caledonia	4
BDI	Burundi	4	NZL	New Zealand	4
KHM	Cambodia	4	NIC	Nicaragua	4
CMR	Cameroon	4	NER	Niger	4
CAN	Canada	4	NGA	Nigeria	4
CPV	Cape Verde	3	MNP	Northern Mariana Islands	4
CYM	Cayman Islands	4	NOR	Norway	4
CAF	Central African Republic	4	OMN	Oman	5
TCD	Chad	4	PAK	Pakistan	4
CHL	Chile	5	PSE	Palestine	4
CHN	China	1	PAN	Panama	5
COL	Colombia	3	PNG	Papua New Guinea	4
COM	Comoros	4	PRY	Paraguay	4
COG	Congo	4	PER	Peru	5
CRI	Costa Rica	4	PHL	Philippines	4
CIV	Cote d'Ivoire	4	POL	Poland	4
HRV	Croatia	4	PRT	Portugal	4
CUB	Cuba	4	PRI	Puerto Rico	3
CUW	Curacao	4	QAT	Qatar	6
CYP	Cyprus	4	ROU	Romania	4
CZE	Czech Republic	4	RUS	Russia	3
COD	Democratic Republic of Congo	4	RWA	Rwanda	4
DNK	Denmark	4	KNA	Saint Kitts and Nevis	4
DJI	Djibouti	3	LCA	Saint Lucia	4
DMA	Dominica	4	VCT	Saint Vincent and the Grenadines	4
DOM	Dominican Republic	3	SMR	San Marino	2
ECU	Ecuador	3	STP	Sao Tome and Principe	3
EGY	Egypt	4	SAU	Saudi Arabia	5
SLV	El Salvador	4	SEN	Senegal	4
GNQ	Equatorial Guinea	3	SRB	Serbia	4
ERI	Eritrea	4	SYC	Seychelles	4
EST	Estonia	4	SLE	Sierra Leone	4
ETH	Ethiopia	4	SGP	Singapore	1
FRO	Faeroe Islands	2	SXM	Sint Maarten (Dutch part)	4
FLK	Falkland Islands	4	SVK	Slovakia	4
FJI	Fiji	4	SVN	Slovenia	4
FIN	Finland	4	SOM	Somalia	4
FRA	France	4	ZAF	South Africa	3
PYF	French Polynesia	4	KOR	South Korea	1
GAB	Gabon	3	SSD	South Sudan	4
GMB	Gambia	4	ESP	Spain	2
GEO	Georgia	4	LKA	Sri Lanka	4

*continúa en la siguiente página**

ISO	País	Grupo	ISO	País	Grupo
DEU	Germany	4	SDN	Sudan	4
GHA	Ghana	4	SUR	Suriname	4
GIB	Gibraltar	2	SWZ	Swaziland	4
GRC	Greece	4	SWE	Sweden	5
GRL	Greenland	4	CHE	Switzerland	2
GRD	Grenada	4	SYR	Syria	4
GUM	Guam	4	TWN	Taiwan	4
GTM	Guatemala	4	TJK	Tajikistan	4
GGY	Guernsey	2	TZA	Tanzania	4
GIN	Guinea	4	THA	Thailand	4
GNB	Guinea-Bissau	4	TLS	Timor	4
GUY	Guyana	4	TGO	Togo	4
HTI	Haiti	4	TTO	Trinidad and Tobago	4
HND	Honduras	3	TUN	Tunisia	4
HUN	Hungary	4	TUR	Turkey	4
ISL	Iceland	2	TCA	Turks and Caicos Islands	4
IND	India	4	UGA	Uganda	4
IDN	Indonesia	4	UKR	Ukraine	4
IRN	Iran	4	ARE	United Arab Emirates	3
IRQ	Iraq	4	GBR	United Kingdom	2
IRL	Ireland	2	USA	United States	5
IMN	Isle of Man	2	VIR	United States Virgin Islands	4
ISR	Israel	4	URY	Uruguay	4
ITA	Italy	2	UZB	Uzbekistan	4
JAM	Jamaica	4	VAT	Vatican	7
JPN	Japan	4	VEN	Venezuela	4
JEY	Jersey	2	VNM	Vietnam	4
JOR	Jordan	4	ESH	Western Sahara	3
KAZ	Kazakhstan	3	YEM	Yemen	4
KEN	Kenya	4	ZMB	Zambia	4
KOS	Kosovo	4	ZWE	Zimbabwe	4
KWT	Kuwait	5			

Cuadro A.1: Equivalencia entre códigos ISO y Países

Anexo B

Código en R

B.1. Colores RGB

```
#Código Aplicación colores 4.1


#generación de datos
dat<-cbind(x=sample(0:255,400,replace=T),
           y=sample(0:255,400,replace=T),
           z=sample(0:255,400,replace=T))
colnames(dat)<-c("R","G","B")

#representación (usando libreria SOMbrero)
library(SOMbrero)

firstg<-initGrid(dimension=c(20,20),topo="square")
plot(firstg,show.names=FALSE) +
  scale_fill_manual(values = rgb(dat,max=255))


#####################################
#####        c l a s s          ####
#####################################


library(class)

#Construcción mapa versión secuencial
gr<-class::somgrid(10,10,"hexagonal")
s_class<-SOM(dat,gr,alpha = list(seq(0.8, 0, len = 1e5),
```

```
    seq(0.4, 0, len = 1e6)),
             radii = list(seq(15, 1, len = 1e5), seq(7,
                1, len = 1e6)))

#Visualización
plot(s_class)
title(main="Versión secuencial")

#Representación neuronas del som en escala RGB
cols<-rgb(round(s_class$codes),max=255)
plot(rep(1:10,each=10),rep(1:10,times=10),col=cols,
     pch=19,cex=2,xlab="",ylab="")
title(main="Versión secuencial")


#Construcción mapa versión batch
s_classb<-batchSOM(dat,gr,radii=c
   (10,8,6,6,5,5,4,4,2,2,2,1,1,1,0))

plot(s_classb)
title("Versión batch")


#Representación RGB
plot(rep(1:10,each=10),rep(1:10,times=10),
     col=rgb(round(s_classb$codes),max=255),
     pch=19,cex=2,xlab="",ylab="")
title("Versión batch")



########################################
#####           s o m              #####
########################################

library(som)

#conversión datos
dat[,1]<-as.numeric(dat[,1])
dat2<-som::normalize(dat)


#construcción som con datos normalizados
```

```
s_som<-som::som(dat2,xdim=10,ydim=10,topol="hexa",
                rlen=10000,init="random")

#Visualización
plot(s_som,sdbar=4,yadj=0.2)


#som sin normalizar ->representación RGB

s_som2<-som::som(dat,xdim=10,ydim=10,topol="hexa",
                 rlen=10000,init="random")

plot(rep(1:10,each=10),rep(1:10,times=10),
     col=rgb(round(s_som2$code),max=255),
     pch=19,cex=0.2*s_som2$code.sum[,3]+0.6)



########################################
#####         y a s o m i          #####
########################################

install.packages("yasomi", repos="http://R-Forge.R-
   project.org",dependencies=TRUE, INSTALL_opts = c('--
   no-lock'))
library(yasomi)

#Construcción 20 soms con distinto radio inicial
somt<-som.tune(dat,sg,som.tunecontrol(sg, init="random",
   radii=c(2,sg$diam), nradii=20,criterion=error.
   kaskilagus),verbose=TRUE)


#Errores de cada som
plot(somt,relative=FALSE,axes=TRUE )
title("Error según radio inicial")

#Seleccionar el som con menor error
s_yasomi<-somt$best.som

#Error en cada paso del som
plot(s_yasomi$errors,type="l", main="Evolución del error
    de cuantificación")
```

```
#Representación de cada componente
par(mfrow=c(1,3))
for(i in 1:ncol(dat)) {
  componentPlane(s_yasomi,i,global.scale=TRUE)
}

par(mfrow=c(1,1))

#Representación datos en el mapa
plot(s_yasomi,mode="data",type="parallel",with.grid=TRUE
   ,asp=1)

#Representación de las neuronas del mapa
plot(s_yasomi,mode="prototype",type="parallel",with.grid
   =TRUE,asp=1)

#Hitpmap con colores RGB
hitMap(s_yasomi,col= rgb(round(s_yasomi$prototypes), max
   =255),with.grid=FALSE,asp=1)


#Representación RGB y u-matrix del som
par(mfrow=c(1,2))
plot(sg,col=rgb(round(s_yasomi$prototypes),max=255),main
   ="Representación RGB de las neuronas")
umatrix(s_yasomi, main="U-matrix: Distancia entre
   neuronas")


#####################################################
#####            S O M b r e r o            #######
#####################################################


library(SOMbrero)
library(gridExtra)

#Abrir interfaz gráfica
sombreroGUI()

#fijar misma semilla que la de sombreroGUI()
set.seed(75247)
```

```
#Entrenamiento del mapa
mysom2 <- trainSOM(dat[,c('R', 'G', 'B')],type='numeric'
   , topo='hexagonal',dimension=c(10,10),affectation='
   standard', dist.type='euclidean', maxit=2000,scaling=
   'none',init.proto='random',nb.save=10, radius.type='
   letremy', eps0=1)

#Representación hitmap
par(mfrow=c(1,1))
plot(mysom2, what='obs', type='hitmap')


#Representación de cada componente
plot(mysom2, what='prototypes', type='color',show.names=
   TRUE, variable='R')
plot(mysom2, what='prototypes', type='color',show.names=
   TRUE, variable='G')
plot(mysom2, what='prototypes', type='color',show.names=
   TRUE, variable='B')


#Reprentación de cada observación en el mapa
plot(mysom2, what='obs', type='names', show.names=FALSE)

#U-matrix
plot(mysom2, what='prototypes', type='umatrix', show.
   names=TRUE)

#Evolución del mapa en los pasos intermedios
ig<-initGrid(c(10,10),"hexagonal")
p1<-plot(ig,show.names=FALSE) + scale_fill_manual(values
    = rgb(round(mysom2$backup$prototypes[[1]]),max=255))
   + ggtitle("Estado inicial")
p2<-plot(ig,show.names=FALSE) + scale_fill_manual(values
    = rgb(round(mysom2$backup$prototypes[[2]]),max=255))
   +ggtitle("")
p3<-plot(ig,show.names=FALSE) + scale_fill_manual(values
    = rgb(round(mysom2$backup$prototypes[[4]]),max=255))
   +ggtitle("")
p4<-plot(ig,show.names=FALSE) + scale_fill_manual(values
    = rgb(round(mysom2$backup$prototypes[[6]]),max=255))
   +ggtitle("")
```

```
p5<-plot(ig,show.names=FALSE) + scale_fill_manual(values
    = rgb(round(mysom2$backup$prototypes[[8]]),max=255))
   +ggtitle("")
p6<-plot(ig,show.names=FALSE) + scale_fill_manual(values
    = rgb(round(mysom2$prototypes),max=255))+ ggtitle("
   Estado final")

ml<-matrix(c(1:6),ncol=3,byrow=TRUE)
grid.arrange(p1,p2,p3,p4,p5,p6,layout_matrix=ml)



########################################
####          k o h o n e n         ####
########################################

library(kohonen)

#Mapa versión secuencial
s_kohonen<-kohonen::som(dat,grid=somgrid(10,10,"
   hexagonal",neighbourhood.fct="gaussian"),keep.data=
   TRUE,mode="online")

#Mapa versión batch
s_kohonenb<-kohonen::som(dat,grid=somgrid(10,10,"
   hexagonal",neighbourhood.fct="gaussian"),keep.data=
   TRUE,mode="batch")


#Representación de los mapas
par(mfrow=c(1,2))
plot(s_kohonen,shape="straight",main="SOM Algoritmo
   secuencial")
plot(s_kohonenb,shape="straight",main="SOM Algoritmo
   batch")


#Componentes som secuencial
par(mfrow = c(1,3),oma=c(0, 0, 4, 0))
plot(s_kohonen, type = "property", property = getCodes(s
   _kohonen, 1)[,1], main = colnames(getCodes(s_kohonen,
    1))[1],shape="straight")
plot(s_kohonen, type = "property", property = getCodes(s
```

```
    _kohonen, 1)[,2],main = colnames(getCodes(s_kohonen,
    1))[2],shape="straight")
plot(s_kohonen, type = "property", property = getCodes(s
    _kohonen, 1)[,3],main = colnames(getCodes(s_kohonen,
    1))[3],shape="straight")
mtext(side=3, line=-7, cex=1.3, outer=T,"Componentes -
    Algoritmo secuencial")

#Componentes som batch
par(mfrow = c(1,3),oma=c(0, 0, 4, 0))
plot(s_kohonenb, type = "property", property = getCodes(
    s_kohonenb, 1)[,1],main = colnames(getCodes(s_
    kohonenb, 1))[1],shape="straight")
plot(s_kohonenb, type = "property", property = getCodes(
    s_kohonenb, 1)[,2],main = colnames(getCodes(s_
    kohonenb, 1))[2],shape="straight")
plot(s_kohonenb, type = "property", property = getCodes(
    s_kohonenb, 1)[,3],main = colnames(getCodes(s_
    kohonenb, 1))[3],shape="straight")
mtext(side=3, line=-7, cex=1.3, outer=T,"Componentes -
    Algoritmo batch")


#Distancias entre observaicones y neurona asociada
par(mfrow=c(1,2))
plot(s_kohonen,type="quality",shape="straight", main="
    Dist BMU - Secuencial")
plot(s_kohonenb,type="quality",shape="straight",main="
    Dist BMU - Batch")


#Hitmap
par(mfrow=c(1,2))
plot(s_kohonen,type="counts",shape="straight",main="
    Hitmap secuencial")
plot(s_kohonenb,type="counts",shape="straight",main="
    Hitmap batch")


#U-matrix
par(mfrow=c(1,2))
plot(s_kohonen,type="dist.neighbours",shape="straight",
    main="U-matrix secuencial")
```

```
plot(s_kohonenb,type="dist.neighbours",shape="straight",
   main="U-matrix batch")


#Representación RGB de las neuronas y las observaciones
par(mfrow=c(1,2))
plot(s_kohonen,type="mapping",bg=rgb(dat,max=255),col=1,
   pch=21,lwd=1,bgcol=rgb(round(s_kohonen$codes[[1]]),
   max=255),shape="straight",main="Algoritmo secuencial"
   )


plot(s_kohonenb,type="mapping",bg=rgb(dat,max=255),col
   =1,pch=21,bgcol=rgb(round(s_kohonenb$codes[[1]]),max
   =255),shape="straight",main="Algoritmo batch")
```

B.2. COVID-19

```
library(dplyr)
library(tidyr)
library(SOMbrero)
library(ggplot2)
library(rworldmap)
library(gridExtra)



#lectura de datos
owd<-read.csv("COVID-19-master/COVID-19-master/csse_
   covid_19_data/csse_covid_19_time_series/owid-covid-
   data.csv",h=T,sep=",")

#coger columna de interes
owd2<-owd %>% pivot_wider(id_cols = c(1,3),
    names_from = date,values_from=new_cases_per_million,
    values_fill=0,names_sort=TRUE)

#Preprocesamiento de datos
owd3<-data.frame(owd2)
owd4<-owd3[,-c(1:2)]
owd4[is.na(owd4)]<-0
owd4 <- cbind(owd3[,1:2],owd4[,colSums(owd4)>0])
owd4 <- owd4[rowSums(owd4[,-c(1:2)])>0,]
owd4<-filter(owd4,location!="World" & location!="
   International")

#tamaño mapa
dim<-c(9,8)
#normalizar datos
owd4<-cbind(owd4[,1:2],scale(owd4[,-c(1,2)]))

#entrenamiento SOM
s_covid <- trainSOM(owd4[,-c(1,2)],type='numeric',
          topo='hexagonal', dimension=dim,affectation='
             standard', dist.type='euclidean',scaling='
             none',init.proto='random', nb.save=10,
             radius.type='letremy', eps0=1)
```

```
plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE,variable='X2019.12.31')
   + theme(legend.position = 'none') + ggtitle('X2019
   .12.31')

#Hitmap
plot(s_covid, what='obs', type='hitmap')

#Representación de paises en cada neurona
plot(s_covid, what='add', type='names',show.names=FALSE,
     variable=owd4$iso_code,size=4,rm_outside=TRUE)



#Representacion vectores modelo del som
plot(s_covid, what='prototypes', type='lines', show.
   names=TRUE) + theme(legend.position = 'none') + theme
   (axis.text.x=element_blank(),axis.ticks = element_
   blank())

#Representación en el mapa de algunos días
p1<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2019.12.31')
    + theme(legend.position = 'none') + ggtitle('X2019
   .12.31')
p2<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.03.16')
    + theme(legend.position = 'none') + ggtitle('X2020
   .03.16')
p3<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.04.12')
    + theme(legend.position = 'none') + ggtitle('X2020
   .04.12')
p4<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.04.28')
    + theme(legend.position = 'none') + ggtitle("X2020
   .04.28")
p5<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.05.10')
    + theme(legend.position = 'none') + ggtitle("X2020
   .05.10")
p6<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.05.30')
```

```
    + theme(legend.position = 'none') + ggtitle("X2020
   .05.30")
p7<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.06.15')
    + theme(legend.position = 'none') + ggtitle("X2020
   .06.15")
p8<-plot(s_covid, what='prototypes', type='color', show.
   names=TRUE,show.legend=FALSE, variable='X2020.07.05')
    + theme(legend.position = 'none') + ggtitle("X2020
   .07.05")


ml<-matrix(c(1:8),ncol=4,byrow=TRUE)
grid.arrange(p1,p2,p3,p4,p5,p6,p7,p8,layout_matrix=ml)



#### 7 CLUSTERS ####

mysom7k<-superClass(sommap=s_covid, k=7)

#Dendograma
plot(mysom7k, plot.var = FALSE)

#Grupos de las neuronas del mapa
plot(mysom7k, what='prototypes', type='grid')+ scale_
   fill_manual(values = rainbow(7))


#Representación en mapa mediante rworldmap
#añadir al conjunto de datos, columna con la neurona y
   cluster al que pertenece cada observación
covidclasif7<-cbind(owd4,s_covid$clustering)
clust7<-as.data.frame(cbind(1:length(mysom7k$cluster),
   factor(mysom7k$cluster)))
cm7<-merge(x=covidclasif7,y=clust7,by.x="s_covid$
   clustering",by.y="V1")
cm7<-select(cm7,cluster=V2,everything())
cm7$V2<-factor(cm7$cluster)
d<-data("countryExData")
cm7_map<-joinCountryData2Map(cm7,joinCode = "ISO3",
   suggestForFailedCodes = TRUE, nameJoinColumn = "iso_
```

```
   code",verbose=TRUE)

#Mapa
map<-mapCountryData(cm7_map,nameColumnToPlot = "cluster"
   ,numCats=7,catMethod="categorical",colourPalette=
   rainbow(7), missingCountryCol="white",
    mapTitle="Representación real del SOM construido",
       addLegend=FALSE)
do.call(addMapLegendBoxes,c(map,horiz=TRUE,x="bottom",
   title=""))



#Construcción perfiles medios
library(reshape2)
clusters<-unlist(read.table("clusters.txt"))

#Función para calcular los perfiles medios de cada grupo
clust.centroid = function(i, dat, clusters) {
  ind = (clusters == i)
  colMeans(dat[ind,])
}

perf_medios<-sapply(sort(unique(clusters)), clust.
   centroid, arrange(owd4,iso_code)[,-c(1:2)], clusters)
perf_medios<-as.data.frame(perf_medios)
perf_medios$Date<-rownames(perf_medios)
perf_melt <- melt(perf_medios, id.vars="Date")


#Representación grupos 1 a 6
ggplot(perf_melt[perf_melt$variable !="V7",], aes(Date,
   value,group=variable, col=variable)) + geom_point() +
  geom_line() +scale_color_manual(name="Grupo",labels =
     c("1","2","3","4","5","6"),values=rainbow(7))+
     ggtitle("Perfiles medios de los grupos")+
  theme_bw() + theme(axis.text.x=element_blank(),axis.
     ticks = element_blank(),axis.title.y=element_blank
     (), panel.grid.major = element_blank(), panel.grid.
     minor = element_blank(), panel.border = element_
     blank(), axis.line = element_line(colour = "black")
     , panel.background = element_blank())
```

```
#Representación grupos 3 y 4
ggplot(filter(perf_melt,variable %in% c("V3","V4")), aes
   (Date,value,group=variable, col=variable)) +
  geom_point() + geom_line() +scale_color_manual(name="
     Grupo",labels = c("3","4"),values=rainbow(7)[3:4])+
     ggtitle("Perfiles medios de los grupos 3 y 4")+
  theme_bw() + theme(axis.text.x=element_blank(),axis.
     ticks = element_blank(),axis.title.y=element_blank
     (),
  panel.grid.major = element_blank(),panel.grid.minor =
     element_blank(), panel.border = element_blank(),
     axis.line = element_line(colour = "black"), panel.
     background = element_blank())
```

Bibliografía

[1] The bowman lab - tutorial: Self organizing maps in r. https://www.polarmicrobes.org/tutorial-self-organizing-maps-in-r, 2020.

[2] D. Asboth. Self-organising maps: In depth. http://davidasboth.com/2016/11/06/self-organising-maps-in-depth, 2016.

[3] C. Bouton and G. Pagès. Self-organization and a.s. convergence of the one-dimensional kohonen algorithm with non-uniformly distributed stimuli. *Stochastic Processes and their Applications*, 47(2):249 – 274, 1993.

[4] C. Bouton and G. Pagès. Convergence in distribution of the one-dimensional kohonen algorithms when the stimuli are not uniform. *Advances in Applied Probability*, 26(1):80–103, 1994.

[5] J. A. Bullinaria. Self organizing maps: Fundamentals, 2004.

[6] F. Comitani. fcomitani/simpsom: v1.3.4, Apr. 2019.

[7] M. Cottrell and J.-C. Fort. Étude d'un processus d'auto-organisation. *Annales de l'I.H.P. Probabilités et statistiques*, 23(1):1–20, 1987.

[8] M. Cottrell, J.-C. Fort, and G. Pagès. Theoretical Aspects of the SOM Algorithm. *Neurocomputing*, 21:119–138, 1998.

[9] M. Cottrell, J.-C. Fort, and G. Pagès. Two or three things that we know about the kohonen algorithm. 01 1994.

[10] G. Deboeck. Financial applications of self-organizing maps. *Neural Network World*, 8, 05 2000.

[11] G. Deboeck and T. Kohonen. *Visual Explorations in Finance: with Self-Organizing Maps*. Springer Finance. Springer London, 2010.

[12] X.-P. Du and P.-L. He. The clustering solution of speech recognition models with som. In J. Wang, Z. Yi, J. M. Zurada, B.-L. Lu, and H. Yin, editors, *Advances in Neural Networks - ISNN 2006*, pages 150–157, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[13] P. Estevez, J. Principe, and P. Zegers. *Advances in Self-Organizing Maps: 9th International Workshop, WSOM 2012 Santiago, Chile, December 12-14, 2012 Proceedings*. 01 2013.

[14] A. Farooq, A. Jalal, and S. Kamal. Dense rgb-d map-based human tracking and activity recognition using skin joints features and self-organizing map. 07 2018.

[15] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[16] T. Honkela. Self-organizing maps in natural language processing. 12 1998.

[17] J. Kangas, T. Kohonen, and J. Laaksonen. Variants of self-organizing maps. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 1:93–9, 02 1990.

[18] S. Kaski. Data exploration using self-organizing maps, 1997.

[19] T. Kohonen. Analysis of a simple self-organizing process. *Biol. Cybern.*, 44(2):135–140, July 1982.

[20] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480, 1990.

[21] T. Kohonen. Data management by self-organizing maps. In *Computational Intelligence: Research Frontiers: IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008, Plenary/Invited Lectures*, volume 5050, pages 309–332, 06 2008.

[22] T. Kohonen. Essentials of the self-organizing map. *Neural networks: the official journal of the International Neural Network Society*, 37, 10 2012.

[23] K. Lagus, S. Kaski, and T. Kohonen. Mining massive document collections by the websom method. *Information Sciences*, 163(1):135 – 156, 2004. Soft Computing Data Mining.

[24] V. Moosavi, S. Packmann, and I. Vallés. Sompy: A python library for self organizing map (som), 2014. GitHub.[Online]. Available: https://github.com/sevamoo/SOMPY.

[25] F. Murtagh and M. Hernández-Pajares. The kohonen self-organizing map method: An assessment. *Journal of Classification*, 12(2):165–190, Sep 1995.

[26] G. Pölzlbauer. *Survey and comparison of quality measures for self-organizing maps*. na, 2004.

[27] F. Rossi. An introduction to yasomi (yet another self organising map implementation), 2012.

[28] C. Schmidt, S. Rey, and A. Skupin. Effects of irregular topology in spherical self-organizing maps. *International Regional Science Review*, 34:215–229, 03 2011.

[29] A. South. rworldmap: A new r package for mapping global data. *The R Journal*, 3(1):35–43, June 2011.

[30] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[31] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

[32] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *Trans. Neur. Netw.*, 11(3):586–600, May 2000.

[33] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. Self-organizing map in matlab: the som toolbox. In *In Proceedings of the Matlab DSP Conference*, pages 35–40, 1999.

[34] G. Vettigli. Minisom: minimalistic and numpy-based implementation of the self organizing map. GitHub.[Online]. Available: https://github.com/JustGlowing/minisom/.

[35] N. Vialaneix, E. Maigne, J. Mariette, M. Olteanu, F. Rossi, L. Bendhaiba, and J. Bolaert. *SOMbrero: SOM Bound to Realize Euclidean and Relational Outputs*, 2020. R package version 1.3-1.

[36] R. Wehrens and L. M. C. Buydens. Self- and super-organizing maps in R: The kohonen package. *Journal of Statistical Software*, 21(5):1–19, 2007.

[37] J. Yan. Package 'som', 2016. R package version 0.3-5.1.

[38] Z. Zhiming and J. Yingying. Application of som neural network in customer segmentation model in coal enterprises. In *2009 International Forum on Computer Science-Technology and Applications*, volume 3, pages 451–454, 2009.