Análisis Interactivo de Datos: Mapas Autoorganizados

TESIS DOCTORAL

Presentado por:
D. Vicente Buendía Ramón

Dirigido por:
Dr. Emilio Soria Olivas
Dr. José D. Martín Guerrero

Programa de Doctorado en
Informática y Matemática Computacional
Escola Tècnica Superior d'Enginyeria

Octubre 2015

Análisis Interactivo de Datos: Mapas Autoorganizados

Memoria que presenta para optar al título de Doctor

Presentado por:
D. Vicente Buendía Ramón

Dirigido por:
Dr. Emilio Soria Olivas
Dr. José D. Martín Guerrero

Programa de Doctorado en
Informática y Matemática Computacional
Escola Tècnica Superior d'Enginyeria

Octubre 2015

A Mateo y Carmen.
A Darío y Nerea.
A Lydia.

Agradecimientos

Cuando la gratitud es tan absoluta
las palabras sobran.

Álvaro Mutis

Gracias, Emilio. Gracias, José. Este trabajo no podría haberse realizado sin contar con vuestra paciencia, ánimos, conocimiento y capacidad para evitarme casi, casi todos los dislates.

Gracias por el ejemplo y la inspiración, por el apoyo y los medios. Habéis sido muy importantes en esto (también). Gracias de nuevo papás, por mucho que insistáis en que no hay que dároslas.

Las frases *"el papá está trabajando"*, *"el papá está haciendo deberes"* han sido escuchadas frecuentemente en casa. Ésta es la Tesis robada a los niños. En el momento de escribir estas líneas aún no sé cómo compensarlo.

Por último y más importante, no he encontrado palabras para expresar lo que ha significado para mí el hecho de contar con tu comprensión y ayuda en en este largo camino que, de otro modo, sencillamente no habría sido posible. Gracias, Lydia.

Duda que sean fuego las estrellas,
duda que el sol se mueva,
duda que la verdad sea mentira,
pero no dudes jamás de que te amo.

William Shakespeare

...
¿Sabes qué?
...

Resumen y objetivos

El que no posee el don de maravillarse ni de entusiasmarse más le valdría estar muerto, porque sus ojos están cerrados.

Albert Einstein

Los fenómenos físicos, así como los procesos industriales, **producen volúmenes cada vez más cuantiosos de datos**, a menudo de difícil tratamiento. Se hace preceptiva la **generación de sistemas y procedimientos que permitan extraer**, en una primera etapa visual de análisis, **la información subyacente a estos datos**, orientando así los cálculos y estudios posteriores que puedan aplicarse. El análisis visual será reforzado si **se complementa con elementos interactivos que permitan al usuario dirigirse hacia los focos de su interés**. Dentro de las técnicas de visualización de datos para los fines expuestos destacan los *Mapas Autoorganizados* cuya efectividad será potenciada si son dotados de interactividad. Las aportaciones destacadas del presente trabajo son:

1. **Desarrollo de una herramienta de *código abierto*** de ayuda en entornos tecnológicos, académicos e industriales, que incorpora entrenamiento de *Mapas Autoorganizados* y sistema interactivo de visualización de resultados.
2. **Presentación de los *Mapas Autoorganizados* como herramienta de visualización de datos**, a emplear en fases iniciales de análisis.
3. Mejora de los *Mapas Autoorganizados* mediante la **adición de interactividad**, como respuesta a las necesidades actuales de análisis visual de datos. Facilidad para la visualización de muchas capas.
4. **Optimizaciones en el proceso de entrenamiento** de grupos de *Mapas Autoorganizados* para un mismo conjunto de datos para, posteriormente, seleccionar de entre éstos al que reúna mejores condiciones.
5. **Valoración de modelos de entrenamiento y propuesta de variantes** en el ámbito de visualización de datos.

Resum i objectius

El que no posseïx el do de meravellar-se ni
d'entusiasmar-se més li valdria ser mort,
perquè els seus ulls estan tancats.

Albert Einstein

Els fenòmens físics, així com els processos industrials, **produïxen volums cada vegada més quantiosos de dades**, sovint de difícil tractament. Es fa preceptiva la **generació de sistemes i procediments que permeten descobrir**, en una primera etapa d'anàlisi, **la informació subjacent a estes dades**, orientant així els càlculs i estudis posteriors que puguen aplicar-se. L'anàlisi visual serà reforçat si **es complementa amb elements interactius que permeten a l'usuari dirigir-se cap als focus del seu interés**. Dins de les tècniques de visualització de dades per als fins exposats destaquen els *Mapes Autoorganitzatius* l'efectivitat dels quals serà potenciada si són dotats d'interactivitat. Les aportacions destacades del present treball són:

1. **Desenrotllament d'una ferramenta de *codi obert*** d'ajuda en entorns tecnològics, acadèmics i industrials, que incorpora entrenament de *Mapes Autoorganitzatius* i sistema interactiu de visualització de resultats.
2. **Presentació dels *Mapes Autoorganitzatius* com a ferramenta de visualització de dades**, a emprar en fases inicials d'anàlisi.
3. Millora dels *Mapes Autoorganitzatius* per mitjà de **l'adició d'interactivitat**, com a resposta a les necessitats actuals d'anàlisi visual de dades. Facilitat per a la visualització de moltes capes.
4. **Optimitzacions en el procés d'entrenament** de grups de *Mapes Autoorganitzatius* per a un mateix conjunt de dades per a, posteriorment, seleccionar d'entre estos qui reunisca millors condicions.
5. **Valoració de models d'entrenament i proposta de variants** en l'àmbit de visualització de dades.

Summary and objectives

He who can no longer pause to wonder and
stand rapt in awe, is as good as dead;
his eyes are closed.

Albert Einstein

Physical phenomena, as well as industrial processes, **produce increasingly large volumes of data**, often of difficult treatment. It makes mandatory the **generation of systems and procedures that allow to extract**, in the first analysis stage, **the underlying information to this data**, facing this way the calculations and later studies that could be applied. Visual analysis will be strengthened if it **is supplemented with interactive elements that allow the user to move towards the foci of interest**. Among the techniques of visualization of data to the exposed ends are the Self-Organizing Maps, whose effectiveness will be enhanced if they are endowed with interactivity. The contributions of this work are:

1. **Development of an *open source* tool** to help in technological, academic and industrial environments, which incorporates training of *Self-Organising Maps* and an interactive system of visualization of results.
2. **Presentation of the Self-Organising Maps as a data visualization tool** to be used in early stages of analysis.
3. Improvement of Self-Organizing Maps by **adding interactivity**, in response to the current needs of visual data analysis. Ease for the display of many layers.
4. **Optimizations in the process of training** groups of *Self-Organizing Maps* for a same set of data, then select from these the one that meets best conditions.
5. **Assessment of training models and proposal of variants** in the field of data visualization.

Índice

Índice de figuras

Capítulo 1

Mapas Autoorganizados para la Minería Visual de Datos

¿Qué sabe el pez del agua
donde nada toda su vida?

Albert Einstein

1.1. Minería Visual de Datos

El progreso tecnológico que han experimentado los sistemas *hardware* permite el **almacenamiento de inmensas cantidades de datos**. A lo largo de la última década, se ha producido una extraordinaria explosión de datos, potenciada por el crecimiento en el uso de *Internet* y el número de dispositivos conectados en todo el mundo. El volumen de datos generados por las aplicaciones empresariales, así como los obtenidos mediante sistemas automatizados (sensores y sistemas de monitorización) continúa aumentando. Se estima un crecimiento del *universo digital* desde el año 2005 a 2020, de 130 *exabytes* a 40.000 *exabytes* (Hu *et al.*, 2014). Estos datos, actualmente no tratados en profundidad, se extraen y conservan por su interés como potencial fuente de información que pudiera llevar en algún momento a ventajas competitivas.

Discernir la información valiosa implícita en la masa de datos es, sin embargo, una **tarea compleja**. Si no se producen las exploraciones adecuadas, los datos pueden convertirse en inútiles y voluminosos. De la necesidad de estudiarlos, surge el concepto de *Minería de Datos (DM)*.

La *DM* se describió de la siguiente forma (Wegman, 2003): ***La Minería de Datos es una extensión del EDA y tiene básicamente las mismas metas, el descubrimiento de la estructura desconocida y no anticipada de los datos. La distinción principal entre los dos términos reside en el tamaño y dimensionalidad de los conjuntos de datos implicados. DM trata general-***

mente con conjuntos de datos mucho más masivos, con los que el análisis interactivo no es factible. Afortunadamente, la tecnología y procedimientos actuales permiten superar la limitación referida al análisis interactivo, que supone una importante ventaja para la *DM*.

Así pues, la *DM* es un proceso donde **la componente humana es fundamental**. En una definición alternativa (Keim, 2002) la *DM se presenta como el proceso (no trivial) de búsqueda y análisis de datos en búsqueda de información implícita y útil (Walny et al., 2011). Sea $D = d_1, ...d_n$ el conjunto de datos a analizar. El proceso de DM se describe como el proceso de búsqueda:*

- *un subconjunto D´ de D y*
- *hipótesis H_U (D´, C) sobre D' que un usuario U considera útiles en un contexto de aplicación C.*

Cabe notar que D´ no solamente contiene menos elementos que D, sino que puede tener una dimensionalidad menor (m'), considerando D como la unión de relaciones R_1,...R_K ($D = \cup_{i=1}^{k} R_i$) con sus correspondientes dimensiones $(m_1, ..., m_k)$. Las hipótesis expresando aspectos de interés de los datos pueden obtenerse a partir de la base de datos completa o de una relación única (D'= D ó D'= R_i). Las hipótesis pueden ser:

- Propiedades que cumplen todos o la mayoría de los $e_i \in D', (D' \subseteq D)$,
- Clasificaciones de D´ en clases C_i con propiedades diferentes P_i

$$[P_i(e_i) \neq P_j(e_2) \Rightarrow e_1 \in C_i \wedge e_2 \in C_j \wedge i \neq j] \tag{1.1}$$

- Dependencias funcionales F o relaciones R entre dos o más dimensiones

$$[d_{i1} = F(d_{i2}, ..., d_{il}) or R(d_{i1}, ..., d_{il}), l \leq m] \tag{1.2}$$

Para que el proceso de extracción de información a partir de datos, *DM*, sea efectivo, es importante incluir aspectos relacionados con la **percepción humana en el proceso de exploración, de tal modo que se combine la flexibilidad y conocimiento humanos con la capacidad de almacenamiento y potencia de cálculo de los sistemas informáticos actuales**. Por tanto, son especialmente importantes las técnicas capaces de proporcionar una buena visualización general de los datos que empleen las posibilidades de la representación visual para mostrar grandes cantidades de datos multidimensionales.

La idea básica de la Exploración Visual de Datos, *Visual Data Exploration* (VDE) para datos multidimensionales consiste en representar, en un momento dado, todos los datos que sea posible por medio de la conversión de cada valor de los datos a píxeles en pantalla, que serán dispuestos adecuadamente. La *VDE* consiste básicamente en la presentación de datos de manera visual proporcionando al usuario una perspectiva de los datos, posibilitando interactuar con los mismos para poder extraer conclusiones.

Las técnicas de VDE tienen gran valor en el análisis exploratorio de los datos y un alto potencial en el tratamiento de bases de datos voluminosas. *VDE* es especialmente útil cuando se conoce poco de los datos y no existen metas de cálculo específicas.

Es destacable la variedad de técnicas de analítica visual según el objeto de estudio. Sobre éstas se ciernen importantes retos técnicos (Sun *et al.*, 2013; Liu *et al.*, 2014):

- **Escalabilidad visual:** La explosión de datos de las últimas décadas supone un reto para los sistemas de visualización interactiva. El avance de la programación paralela y el empleo de potente hardware - como por ejemplo la utilización para cálculos matemáticos de potentes tarjetas gráficas con cientos de miniprocesadores paralelos (*GPUs, Graphic Processing Units*) - permite el tratamiento de volúmenes cada vez más ingentes de datos, de lo que se deduce la progresiva complejidad en la visualización útil de éstos. Para abordar dicha complejidad emergen métodos como sistemas de recomendación de nodos a visualizar y despliegues interactivos (Crnovrsanin *et al.*, 2011) y de análisis *in situ* (Hadlak *et al.*, 2011). En definitiva, **los mecanismos de visualización deben tener un diseño escalable que permita abordar un significativo volumen de datos objeto de estudio** (Keim *et al.*, 2006). Un tema interesante de investigación, procedente de la propia definición de Minería de Datos, *Data Mining* (DM), es cómo involucrar a los propios usuarios en el proceso de reducción de datos, permitiéndoles responder a sus necesidades de información, y por otra parte contribuyendo con su conocimiento al proceso de reducción de datos. Adicionalmente, resultará de interés el estudio de la combinación de diversas técnicas de reducción de datos de modo que complementen sus debilidades entre sí, pudiendo incluir, además de los métodos comentados, técnicas como *muestreo*, *filtrado*, *cuantización vectorial* o *análisis de componentes principales*.

- **Usabilidad y Evaluación:** Siempre es importante poder valorar la efectividad de los sistemas de visualización. Los diseñadores de técnicas de análisis visual disponen de diferentes elementos comparativos como los estudios de casos conocidos, revisiones de expertos, o estudios formales o informales efectuados por usuarios, para valorar la usabilidad y efectividad de los sistemas. Además de la propia calidad de la visualización del dato, considerando que la *VDE* no es simplemente una presentación de datos, la *interacción humano-máquina* también llamada *experiencia de usuario* puede suponer el éxito (o fracaso) de un experimento, por lo que se puede concluir que este aspecto también debe ser valorado. Las revisiones pueden proporcionar valiosa información que ayude a mejorar las aplicaciones de visualización. Existen diversos casos de estudio de diseño (Bezerianos e Isenberg, 2012; Gomez *et al.*, 2012; Haroz y Whitney, 2012; Hofmann *et al.*, 2012; Kim *et al.*, 2012; Lam *et al.*, 2012). Los diseñadores y desarrolladores deben encontrar métodos de evaluación fiables para los ámbitos específicos de sus aplicaciones, comprendiendo, entre otros (Tory y Moller, 2005):

  a) Realización de pruebas por parte de expertos en presentación de datos y usabilidad.

b) Desarrollo de heurísticos basados en guías de visualización y usabilidad como referencias.

- **Análisis integrado de datos heterogéneos:** La integración y el análisis de datos heterogéneos (procedentes de diversas fuentes y con distintos formatos) constituye un significativo reto técnico; con el desarrollo de la analítica de grandes volúmenes de datos (*Big Data*), esta integración es más importante que nunca en muchas de las funciones de los negocios como, por ejemplo, la *inteligencia de negocio*, la *gestión de recursos humanos* o los estudios generales de *marketing*. Las cuestiones tratadas pueden implicar toma de decisiones a partir de datos que no son fácilmente tratables y actualmente no existen técnicas y herramientas de analítica visual que traten datos heterogéneos.

- **Visualización *in-situ*:** La *visualización in-situ* genera representaciones visuales conforme se van obteniendo nuevos datos. Éste es un sistema efectivo para entender y analizar flujos de datos (*streaming*). Los datos en *streaming* se definen como aquellos que fluyen regularmente a través del *hardware*. Ejemplos clásicos de *streaming* son los *logs* de datos generados a partir de sensores, o datos generados a partir de redes sociales (como los *tweets*). Por ejemplo, durante 2013 se generaron más de 340 millones de tweets diarios (Liu *et al.*, 2014), y una cuestión de interés es cómo discernir aquellos que pueden ser noticias importantes del resto. Como dificultad importante se puede mencionar la necesidad de coordinación del procesamiento continuo de datos junto con la visualización de los resultados pertinentes, lo que requiere la existencia de un canal de comunicación ágil entre ambos procesos.
  Otras aplicaciones de la visualización *in-situ* son aquellas en las que el usuario participa en el propio proceso de representación, ayudándose de gráficos y medidas de calidad provisionales (Pérez *et al.*, 2013).

- **Narrativa:** Este aspecto se refiere a la **inclusión de narraciones sonoras o escritas** junto con las visualizaciones interactivas. De este modo, se pueden comunicar dichas observaciones **de un modo más intuitivo**. Estas técnicas de momento no se encuentran plenamente desarrolladas; su estudio puede suponer una investigación multidisciplinar, involucrando ciencias de la percepción, diseño y arte (Segel y Heer, 2010). En la actualidad, los campos de aplicación principales de estas técnicas emergentes son el periodismo *on-line* y la educación. La extensión al análisis visual de datos sería una combinación entre la construcción visual de datos y la especificación de una estructura narrativa **con anotaciones gráficas o auditivas, técnicas de resaltado, transiciones y controles interactivos**. Nótese que no se ha definido previamente el tipo de usuario de estas aplicaciones; el propósito es acercar las herramientas visuales al especialista en el dominio de la información que se está manejando, **no requiriendo necesariamente de un experto en tecnología**.

- **Fiabilidad: La información incierta** o difusa puede aparecer y extenderse en el proceso analítico (MacEachren *et al.* (2012)), por tanto ésta **debe ser**

considerada en el modelo visual (Wu *et al.*, 2012b). La incertidumbre subestimada puede llevar a decisiones erróneas y la incertidumbre sobreestimada puede hacer desechar observaciones valiosas; si el usuario puede **conocer de manera explícita la información incierta**, podrá tomar decisiones informadas sobre el análisis general.

- **Trazabilidad de la navegación:** Los registros permiten a los analistas conocer las acciones realizadas, dentro del entorno de visualización (Jankun-Kelly *et al.*, 2007), sobre datos gráficos. La trazabilidad en el proceso de visualización permite registrar las diferentes etapas en las que ha estado el usuario dentro de una sesión de trabajo, favoreciendo el entendimiento del proceso seguido, los pasos futuros, la reproducción del proceso (o evitar redundancias, en su caso) e incluso el trabajo colaborativo.

Se pueden citar algunos ejemplos de visualizadores de datos (Liu *et al.*, 2014); aquellos resultantes de aplicaciones específicas de análisis de datos (Cao *et al.*, 2012; Cui *et al.*, 2011; Ma *et al.*, 2012), análisis visual de datos empresariales (Ko *et al.*, 2012; Liu *et al.*, 2008; Cao *et al.*, 2012; Cui *et al.*, 2011; Liu *et al.*, 2012; Shi *et al.*, 2010), o análisis de datos de vídeo (Chen *et al.*, 2012). Las técnicas mostradas en los ejemplos responden a diseños *ad hoc* adaptados a los estudios respectivos para los que van dirigidos. El presente trabajo propone la utilización previa de un estándar de visualización de datos: Los *Mapas Autoorganizados* (en adelante, *SOM*, siglas de *Self Organizing Maps*).

Al igual que otros tipos de redes neuronales, los *SOM* son distinguidos por su robustez en el tratamiento de datos multidimensionales con relaciones no lineales y, especialmente, en su labor de reducción de dimensionalidad. Adicionalmente, los *SOM* son métodos de *clustering*, cuantización, abstracción y visualización (Kohonen *et al.*, 2001), a diferencia de las técnicas estadísticas habituales de reconocimiento de patrones. En comparación con otras técnicas, el *SOM* puede escalarse automáticamente en función de los datos a analizar. Estas ventajas sitúan al *SOM* en el campo de la *VDE*.

1.2. *Processing* para el desarrollo de herramientas de visualización

La analítica visual supone en sí un reto importante en lo que refiere a escalabilidad, usabilidad y evaluación, análisis integrado de datos y fiabilidad. Por tanto, y dada la **complejidad de muchos de los conjuntos de datos existentes**, el empleo de éstos para obtener soluciones significativas requiere tener en cuenta puntos de vista desde diferentes ámbitos (Fry, 2008): ***Estadística*, *DM*, *diseño gráfico* y conceptos de *visualización de información***. Sin embargo, en general, cada campo ha evolucionado por su parte. Así, el diseño visual no suele considerar el modo de manipular miles de datos. Las técnicas de *DM* lo hacen posible aunque suelen estar poco comunicadas con los medios para interactuar con datos. Por otro lado, los entornos

de desarrollo software proporcionan módulos para interactuar y representar diferentes tipos de datos, pero habitualmente se infravaloran los principios estéticos de diseño y percepción visual. En el momento de estudiar un problema de representación de datos, a menudo resulta difícil la elección de un tipo de representación y de elegir qué herramientas emplear o qué bibliografía consultar.

En un esfuerzo por reconciliar los distintos ámbitos en un único proceso, de tal modo que los diseñadores gráficos puedan aprender los métodos de programación necesarios, y que por otro lado, los estadísticos puedan comunicar sus resultados de manera efectiva, surge la herramienta que se presenta a continuación: *Processing*.

Processing[1] **(Reas y Fry, 2005) es un lenguaje de programación, un entorno de desarrollo y una *comunidad on-line***. Desde sus inicios, su utilidad principal es la implementación de aplicaciones interactivas 2D y 3D, para lo cual dispone de funciones que facilitan el diseño gráfico. El empleo como lenguaje de base Java supone la compatibilidad con las librerías desarrolladas en tal lenguaje, lo que confiere al entorno *Processing* gran potencia de desarrollo.

La característica diferenciadora de *Processing* reside en la combinación de conceptos de programación software dentro de un contexto de arte y diseño multimedia. La generalidad y orígenes de la sintaxis de *Processing* (el lenguaje Java) lo constituyen como base para el aprendizaje (Schweitzer *et al.*, 2010). Las aptitudes obtenidas mediante la programación de *Processing* facilitan el aprendizaje de otros lenguajes procedentes de diferentes contextos incluyendo diseño web, comunicaciones, microcontroladores y gráficos por computador.

El **entorno de desarrollo es minimalista (fig. 1.1)**, por lo que es posible comenzar a programar tras recibir pocos minutos de instrucción. Dado que la sintaxis es conocida (lenguaje Java), los usuarios con más experiencia en programación pueden comenzar a escribir programas complejos después de un periodo de aprendizaje corto.

Processing se encuentra asistido por una potente comunidad, lo que ha permitido que el proyecto crezca a través de diversas vertientes, sirviendo el sitio web del proyecto como central de comunicaciones. Otra característica destacable reside en la posibilidad de **exportar a la web los programas de *Processing***, pudiendo emplearse como ejemplo para otros programadores.

Entre otras pueden destacarse la **comunidad Open Processing** (Ascioglu, 2008), que contiene miles de programas, también llamados *sketches* empleando la terminología *Processing*. **Otra comunidad de relieve es *Processing JS*** (Resig, 2008), cuyo objeto principal es un compilador que permite exportar los *sketches* de tal modo que se pueden ejecutar incluidos dentro de páginas web.

Adicionalmente, el hecho de que *Processing* sea ***Open Source*** y gratuito, y **compatible con plataformas Mac, Windows y Linux** lo convierten en el idóneo para la implementación de un sistema de cálculo que emplea el rigor de un lenguaje como Java y, a la vez, dispone de a) **potentes librerías gráficas**, que incluyen un amplio conjunto de primitivas gráficas, funciones de tratamiento de imágenes y filtros globales de imagen, b) **funciones de carga y tratamiento de datos** en distintos

[1] Lenguaje de programación y entorno *Processing* disponible en http://www.processing.org

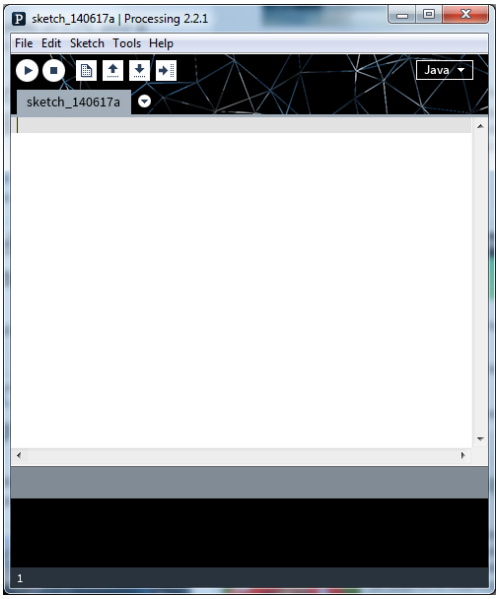


Figura 1.1: Aspecto general del entorno de desarrollo *Processing*

formatos y c) **controladores de eventos de los dispositivos de entrada**, para facilitar la creación de *sketches* interactivos.

Por último, se pone de relieve el hecho de que las aplicaciones generadas en Processing se ejecutan sobre la *Máquina Virtual de Java*, lo que significa que **existe la posibilidad de integrar librerías ya existentes en lenguaje Java**, lo que confiere al sistema toda la potencia y capacidad de reutilización de código ya generado de dicho lenguaje.

1.3. Mapas Autoorganizados

Son una herramienta matemática muy potente, con mucha relevancia para el análisis visual de datos. Para poder valorar sus ventajas, en primer lugar se presenta mediante una descripción general, seguida de una serie de aspectos relevantes a tener en cuenta en la construcción y diseño de esta técnica. **El objeto del *SOM* como red neuronal compuesta por una matriz bidimensional de neuronas, es la proyección y visualización de datos multidimensionales de señal en un espacio de baja dimensionalidad, generalmente un plano bidimensional.**

1.3.1. Introducción al SOM

En esencia, **el *SOM* es un gráfico que refleja las semejanzas existentes entre los datos de entrada**. Convierte las relaciones estadísticas existentes entre datos de alta dimensionalidad en relaciones geométricas proyectadas sobre puntos de una pantalla de baja dimensionalidad, habitualmente, una rejilla de nodos de dos dimensiones, facilitando la exploración visual de estas relaciones. Además del aspecto de visualización, también se produce una cierta abstracción, dado que se produce una compresión de información, preservando las relaciones de vecindad de los datos.

Se describe como un **mapeo no lineal, ordenado y suavizado de los conjuntos de datos de entrada, en los elementos de una matriz de bajas dimensiones**. Este mapeo tiene similitudes con la cuantización vectorial clásica (Makhoul *et al.*, 1985). Se asume por simplicidad que el conjunto de datos de entrada ξ_i se puede representar como un vector real $\mathbf{x} = [\xi_1, \xi_2, ...\xi_n]^T \in \mathbb{R}^n$. Cada elemento - también conocido como neurona o celda - de la matriz del SOM se asocia a un vector $\mathbf{m_i} = [\mu_{i1}, \mu_{i2}, ..., \mu_{in}]^T \in \mathbb{R}^n$ llamado *modelo*. Asumiendo una medida de distancia entre $\mathbf{x}$ y $\mathbf{m_i}$ denotada por $d(\mathbf{x}, \mathbf{m_i})$, la *imagen* de un vector de entrada $\mathbf{x}$ en el array del SOM se define como el elemento $\mathbf{m_c}$ de tal modo que:

$$c = \underset{i}{\operatorname{argmin}}\{d(\mathbf{x}, \mathbf{m_i})\} \tag{1.3}$$

Aunque, a diferencia de la cuantización vectorial tradicional, en este caso la $\mathbf{m_i}$ se define de tal modo que el mapeo está *ordenado*, por lo que se consigue una visualización de la distribución de $\mathbf{x}$.

1.3.2. El algoritmo original SOM

Por su habilidad para proporcionar una proyección topológica de datos de alta dimensionalidad, el algoritmo o *entrenamiento* SOM se puede considerar como un proceso de regresión en el que únicamente se considera un subconjunto de los datos de entrada en cada paso de proceso. El esquema del algoritmo se presenta en la figura 1.2. Tras inicializar la matriz bidimensional (los detalles sobre estas inicializaciones se dan en la sección 1.3.6), se sigue un proceso que se repite iterativamente con cada elemento de la muestra que participa en el entrenamiento:

1. De la muestra original $\mathbf{x}$, se extrae un dato $\mathbf{x_k}$. Lo habitual es realizar esta extracción al azar.

2. Dado un vector de entrada $\mathbf{x_k}$, se compara con todas las referencias $\mathbf{m_1}, ..., \mathbf{m_z}$, y la localización de la *mejor opción o ganador -best matching unit o BMU-* se define como la respuesta o resultado de clasificación de la red. En la figura 1.3 se muestra el mapeo que se realiza desde el espacio $\mathbb{R}^n$ de los elementos de entrada $\mathbf{x}$ hacia una matriz de nodos bidimensional. En la comparación del dato de entrada con la referencia se puede emplear cualquier métrica, aunque en la mayoría de aplicaciones prácticas se utiliza la menor de las *distancias*

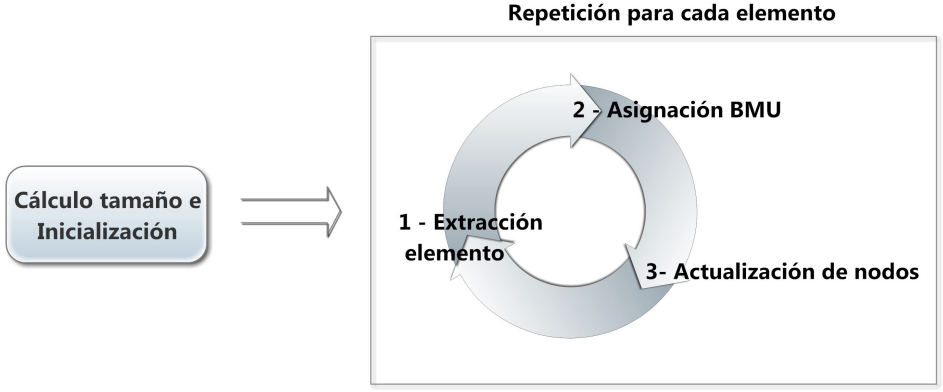


Figura 1.2: Esquema de entrenamiento *SOM*.

Euclidianas $|\mathbf{x} - \mathbf{m_z}|$ para definir el nodo *que es la mejor opción*. En el caso del gráfico 1.3, el nodo c elegido a partir de un dato de entrada x sería:

$$c = \underset{i}{\operatorname{argmin}}\{||\mathbf{x} - \mathbf{m_i}||\}, \textit{ por lo que } ||\mathbf{x} - \mathbf{m_c}|| = \min_i\{||\mathbf{x} - \mathbf{m_i}||\} \quad \forall i \qquad (1.4)$$

3. Durante las etapas de *aprendizaje*, $\mathbf{m_c}$ y los nodos dentro de la matriz que se encuentran cerca de $\mathbf{m_c}$ hasta una determinada distancia geométrica, se adaptan con respecto al dato de entrada $\mathbf{x}$, actualizando sus valores de referencia. Al actualizar no únicamente $\mathbf{m_c}$ sino también sus elementos cercanos, se obtiene un efecto de *suavizado* que además conlleva a una *ordenación global*, que confiere al *SOM* una de sus características más atractivas en el ámbito del presente estudio: Preservación del espacio de alta dimensionalidad original en el espacio transformado de baja dimensión.

Por tanto, siendo $\mathbf{x} \in \mathbb{R}^n$, se podría decir que el SOM constituye una proyección no lineal de la distribución de probabilidad $p(\mathbf{x})$ del vector de alta dimensionalidad $\mathbf{x}$ en una pantalla de dos dimensiones. Una vez que el proceso de aprendizaje ha convergido, los valores $\mathbf{m_1}...\mathbf{m_z}$ tienden a estabilizarse, habiendo partido de valores $\mathbf{m_1}(0)...\mathbf{m_z}(0)$ arbitrarios, que van actualizándose según se muestra en la ecuación (1.5):

$$\mathbf{m_i}(t+1) = \mathbf{m_i}(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m_i}(t)] \qquad (1.5)$$

donde $t \in \mathbb{N}$ es una coordenada discreta de tiempo. La función $h_{ci}(t)$ actúa como *función de vecindad (también llamada neighborhood function)*, constituyendo un *kernel* de suavizado sobre los puntos de la red. Para que haya convergencia, debe cumplirse que $h_{ci}(t) \to 0$ cuando $t \to \infty$. Habitualmente $h_{ci}(t) = h(||\mathbf{r_c} - \mathbf{r_i}||, t)$, donde $\mathbf{r_c} \in \mathbb{R}^n$

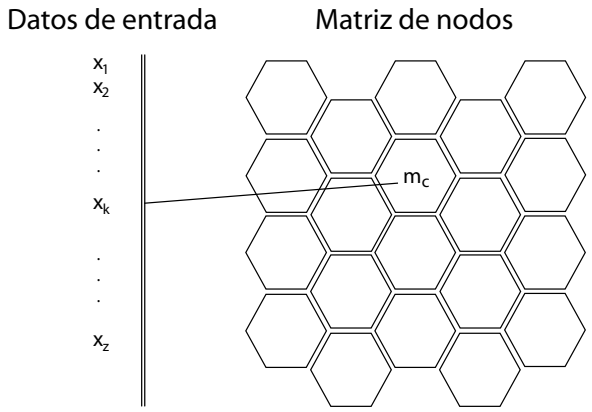


Figura 1.3: Matriz de nodos en un *SOM* bidimensional.

y $\mathbf{r_i} \in \mathbb{R}^n$ son respectivamente los vectores de localización de los nodos c e i. Cuando $||\mathbf{r_c} - \mathbf{r_i}||$ se incrementa, $h_{ci} \to 0$. Las funciones de vecindad más habituales y sus efectos sobre el algoritmo se exponen en el apartado siguiente.

1.3.3. La función de vecindad

Constituye el elemento fundamental del algoritmo de aprendizaje, dado que define las actualizaciones del nodo asociado a $\mathbf{m_c}$, seleccionado como *mejor opción* y de sus nodos cercanos, a partir de un patrón de entrada $\mathbf{x}$. A su vez, dicha *función de vecindad* se encuentra estrechamente relacionada con otro parámetro: el llamado *parámetro de aprendizaje* α, que sirve para ponderar el resultado de dicha función. Dado que este parámetro habitualmente depende del momento o iteración del algoritmo en que se aplique, se puede representar como $\alpha(t)$. Algunas de las funciones de vecindad habituales, son la *vecindad de burbuja*, descrita en la ecuación (1.6), la *gaussiana*, en la (1.7) y la *gaussiana con sesgo*, en la (1.8).

La función de la *vecindad de burbuja* se presenta a continuación:

$$h_{ci}(t) = \begin{cases} \alpha(t) & si \quad |r_c - r_i| <= N \\ 0 & si \quad |r_c - r_i| > N \end{cases} \tag{1.6}$$

En el caso de la vecindad de *burbuja* (1.6) y considerando la ecuación de actualización del proceso de aprendizaje (1.5): Los nodos cuya distancia al nodo *best matching* m_c es menor o igual a un rango N, ven sus valores incrementados en la cantidad expresada por el *parámetro de aprendizaje* $\alpha(t)$.

La *función de vecindad gaussiana* se detalla mediante la expresión:

$$hg_{ci}(t) = \alpha(t) \exp\left[-\frac{|r_c - r_i|^2}{2\sigma^2(t)}\right], \tag{1.7}$$

En el caso de la vecindad *gaussiana* se incorpora además el parámetro $\sigma(t)$, que se corresponde con la desviación típica de la distribución gaussiana considerada. Los parámetros $\alpha(t)$ y $\sigma(t)$ son normalmente funciones monótonas decrecientes del tiempo dado que, de hecho, el proceso se compone habitualmente de dos fases: 1) Una primera parte de ordenación de los elementos, en la que el *kernel* es muy amplio y, por tanto, relaciona gran parte de los elementos de la red, 2) una segunda parte de *afinamiento* con un *kernel* más estrecho (y en contracción, como se comenta anteriormente) donde los datos de entrada afectan a un rango más estrecho de elementos.

La vecindad *gaussiana con sesgo* es una variante de las vecindades *gaussiana* y *burbuja* que se comporta según la ecuación (1.8). Podría definirse como una *gaussiana* que incorpora un rango N a partir del cual no tiene efecto:

$$h_{ci}(t) = \begin{cases} hg_{ci}(t) & si \quad |r_c - r_i| <= N \\ 0 & si \quad |r_c - r_i| > N \end{cases} \tag{1.8}$$

Una variante habitual es la función *triangular*; es similar a la función *gaussiana*, aunque es lineal. Otra variante de la vecindad *gaussiana* es la función *sombrero mexicano*. En ésta, las neuronas que no se encuentran en las proximidades de la neurona *ganadora* se actualizan en sentido contrario, para favorecer las disimilitudes entre distintos grupos. Se adjunta ejemplo en dos dimensiones en la figura 1.4. Dicha función viene dada por la ecuación:

$$h_{ci}(t) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1 - x^2)e^{-x^2/2} \tag{1.9}$$

El tipo de *función de vecindad* tiene efectos sobre la convergencia y sobre la presencia de estados *metaestables*, es decir, aquellos en los que la medida de distorsión empleada converge en un mínimo local (Erwin *et al.*, 1992). Como resultado principal, si la función de vecindad es cóncava, existen estados *metaestables* que pueden provocar retraso en el proceso de convergencia. Así pues, si se emplea una función de vecindad convexa como la función gaussiana, se puede llegar a una buena aproximación de $p(x)$.

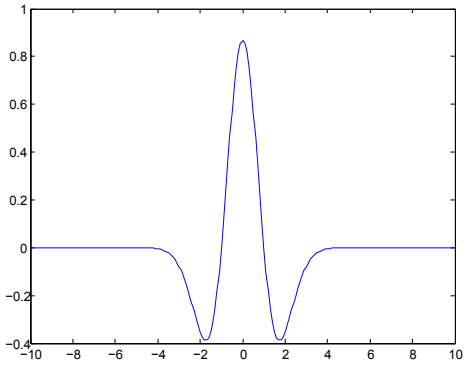


Figura 1.4: Ejemplo de función de vecindad *sombrero mexicano*.

1.3.4. Ajuste del parámetro de aprendizaje

Existen diversos modos de selección de α (Stefanovic y Kurasova, 2011). Siendo t la iteración actual, y T el número máximo de iteraciones:

1. **Lineal**.
$$\alpha(t) = \frac{1}{t} \tag{1.10}$$

2. **Inversa con el tiempo**.
$$\alpha(t,T) = \left[1 - \frac{t}{T}\right] \tag{1.11}$$

3. **Series potenciales**.
$$\alpha(t,T) = (0{,}005)^{\frac{t}{T}} \tag{1.12}$$

4. **Función heurística de Kohonen**. Existen parámetros *optimizados* de aprendizaje α para el SOM (Kohonen *et al.*, 2001). Considerando la función hh_{ci} constituida del siguiente modo:
$$hh_{ci} = \frac{h_{ci}}{\alpha(t)} \tag{1.13}$$
Despejando de la ecuación de actualización de nodos (1.5) resulta:
$$m_i(t+1) = m_i(t) + \alpha(t) hh_{ci}[x(t) - m_i(t)] \tag{1.14}$$
donde hh_{ci} es invariante en la fase final de convergencia, por lo que el parámetro de aprendizaje en función del tiempo resulta:
$$\alpha(t+1) = \frac{\alpha(t)}{1 + hh_{ci}\alpha(t)} \tag{1.15}$$

Esta expresión puede no funcionar del mejor modo en la práctica debido a los diferentes valores $\alpha(t)$ obtenidos a largo plazo, aunque en la práctica no se observan fuertes deformaciones en las densidades de los m_i con respecto a $p(x)$ (Kohonen *et al.*, 2001).

1.3.5. Cálculo de tamaño

Dentro del contexto de visualización de datos, **es importante fijar las medidas adecuadas del SOM para cada conjunto de datos**: Un mapa demasiado pequeño no reflejará suficientemente las cualidades de los datos a representar. Por otro lado, un mapa demasiado grande puede resultar en dificultades innecesarias de visualización, necesidades extra de potencia de cálculo y unidades desaprovechadas. Se reconocen **varias posibilidades de fijación** de las dimensiones que conforman el array SOM:

1. **Fijación de dimensiones por parte del investigador**, previo conocimiento de los datos.

2. **Empleo de heurístico de cálculo de dimensiones**: La cantidad total de nodos z se calcula mediante la ecuación

$$z = 5T^{0,54321} \tag{1.16}$$

   Donde T es el número de vectores de entrada. Una vez se determina el número z de nodos, se calculan los dos mayores valores propios de la muestra de datos, haciendo coincidir la relación de éstos con la relación de altura/anchura de la red (Vesanto *et al.*, 2000).

3. **Creciente de manera adaptativa.** Estas posibilidades se describen con detalle en el capítulo 4 (secciones 4.2 y 4.3). Una contribución del presente trabajo, relacionada con este aspecto, se describe en la sección 4.4.

1.3.6. Inicializaciones

El algoritmo SOM converge incluso con una inicialización aleatoria, sin embargo, en la práctica, **iniciar a partir de un estado en el que las referencias se encuentran ordenadas y cumplen**, aunque vagamente, **con la función de densidad de los datos de entrada, permite acelerar el proceso de convergencia** aún aplicando una función de vecindad estrecha y valores bajos de $\alpha(t)$.

De lo comentado anteriormente se desprende que la red puede ser inicializada de muchas formas; de entre éstas cabe distinguir:

- **Inicialización aleatoria.** El algoritmo SOM puede ser inicializado con valores arbitrarios. Se demuestra que mapas que inicialmente estaban desordenados se ordenarán a largo plazo, aunque posiblemente la solución encontrada no sea la mejor ni la más rápida (Kohonen *et al.*, 2001). Adicionalmente, puede producirse

la llamada *limitación de las redes competitivas*, que consiste en la posibilidad de que existan nodos con pesos muy distanciados de los valores de entrada $\mathbf{x}$, lo que provocaría que estos nodos nunca resultaran seleccionados.

- **Inicialización mediante interpolación de intervalos de datos.** Dados los valores máximo y mínimo de cada uno de los elementos de los datos de entrada $\mathbf{x}$, se asignan correlativamente los valores interpolados proporcionados de éstos a cada uno de los elementos de la matriz, de manera ordenada. Se suele añadir un ruido aleatorio (Mayer *et al.*, 2011).
- **Inicialización directa por vectores de entrada.** Inicialmente, a cada nodo se le asigna el valor de un dato de entrada x elegido aleatoriamente (Mayer *et al.*, 2011).
- **Inicialización mediante el Análisis de Componentes Principales (PCA)** (Akinduko y Mirkes, 2012). Partiendo de la base de que cualquier estado inicial ordenado puede facilitar la convergencia, este método consiste en la determinación de los dos vectores propios principales de la matriz de autocorrelación de los datos de entrada x, para después generar con éstos un espacio bidimensional.

1.3.7. Entrenamiento *batch*

Como variante del algoritmo, independiente del *parámetro de aprendizaje*, el algoritmo de tipo *batch* se asemeja al algoritmo de *Linde-Buzo-Gray* de cuantización vectorial (Linde *et al.*, 2003), donde se dispone de todas las muestras de entrenamiento al inicio del entrenamiento. Los pasos del aprendizaje se definen del siguiente modo:

1. Se toman K elementos de la muestra.
2. Para cada elemento de la red i, se toma una lista de los elementos de la muestra cuyo vector de referencia es próximo a la referencia de i.
3. Se actualiza para cada valor de referencia i la media de dicha lista.
4. Se itera el algoritmo varias veces.

En caso de utilizar una función h_{ji}, esta media sería ponderada con el valor de la función. Esta función es particularmente efectiva si los valores iniciales de la red están ordenados, aunque no aproximen la distribución de la muestra. La figura 1.5 esquematiza este proceso.

1.3.8. Medidas de calidad de la red calculada

A partir de los dos objetivos principales del *SOM* ya comentados (cuantización y ordenación global), se definen dos medidas principales de calidad del mismo: ***Error de cuantización*** **y** ***error topográfico***.

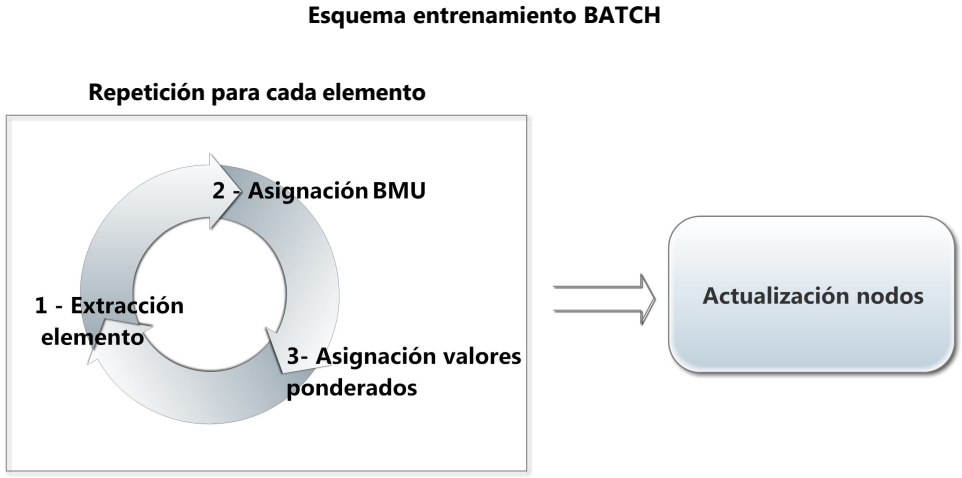


Figura 1.5: Esquema de entrenamiento *batch*.

El *error de cuantización* se obtiene a partir de la suma de los errores cuadráticos resultantes del cuadrado de la distancia entre los datos de entrada $\mathbf{x}$ y los valores de referencia de sus correspondientes elementos $\mathbf{m_i}$ de la red más cercanos. El error topográfico de una muestra de datos X para una red entrenada M se puede definir como:

$$ErrQuan(X, M) = \sum_{x \in X} d^2_{min}(\mathbf{x}, \mathbf{BMU_x}) \tag{1.17}$$

Donde $d^2_{min}(x, BMU_x)$ es el cuadrado de la distancia entre un elemento $\mathbf{x}$ y un elemento $\mathbf{BMU_x} \in M$, siendo $\mathbf{BMU_x}$ el *ganador* para dicha muestra $\mathbf{x}$. Dicho de otro modo, $d_{min}(\mathbf{x}, \mathbf{m_i})$ es la distancia mínima de la muestra $\mathbf{x}$ a la red. Es bastante habitual emplear el valor promedio con respecto a la cantidad de datos de entrada, refiriéndose así al *error cuadrático medio*.

Para medir el *error topográfico*, es decir, el grado en que las observaciones próximas en el espacio de entrada se proyectan en centroides próximos en el SOM, existen varias medidas, como el *Análisis de Componentes Curvilineal* (Demartines y Herault, 1997) que consiste en la realización de un diagrama bidimensional que representa las distancias de los centroides del SOM comparándolas con las distancias del espacio de entrada, o la *Función Topográfica* (Villmann *et al.*, 1997) que mide cómo los vectores próximos del espacio de entrada se mapean en centroides próximos en el *SOM*.

Estos criterios son medidas *globales* que permiten detectar, por comparación, si un mapa *SOM* está en un mínimo local, o si se preserva la topología local.

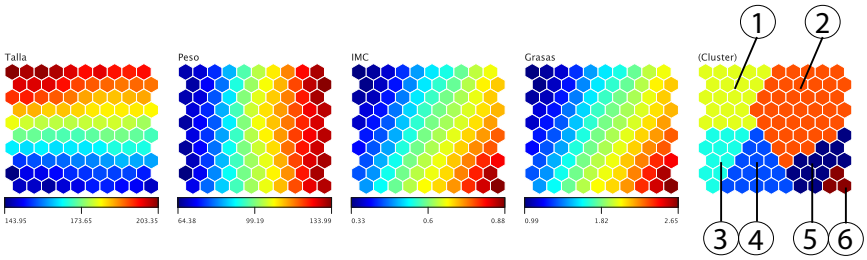


Figura 1.6: Muestra de SOM con ejemplo de agrupamiento en la capa *Cluster*.

Dado que es importante tanto que el *SOM* describa fielmente el espacio original como que la ordenación topológica del mapa sea correcta, es muy habitual utilizar como medida de evaluación del *SOM* el producto de los errores de cuantización y topográfico. Existen otras medidas (Kaski y Lagus, 1996) que combinan ambos errores.

1.3.9. *Clustering*

En ocasiones, el número de unidades del *SOM* es grande, por lo que **para facilitar el análisis cuantitativo de éste y de los datos subyacentes es conveniente separar estas unidades en grupos o *clusters***(Vesanto y Alhoniemi, 2000; Lampinen y Oja, 1992). Cada *cluster* se compone de unidades lo más similares entre sí, y lo más diferentes posibles a las unidades del resto de *clusters*. El clásico *K-Means* (Mishra *et al.*, 2012) se ha utilizado ampliamente para llevar a cabo este tipo de *agrupamientos*, aunque existen otras posibilidades más avanzadas que permiten mejorar este aspecto (Wu y Chow, 2004).

Este sistema sirve de ayuda a la percepción visual al permitir distinguir zonas de las capas que tienen una característica discriminatoria. En la figura 1.6 se muestra un ejemplo de agrupamiento: La capa etiquetada *(Cluster)* muestra diferentes zonas, separadas por similitud. La zona de esta capa etiquetada como *1*, trasladada al resto de capas, permite observar la relación existente entre nodos de la variable "Talla" con valor alto, "Peso" con valor bajo, a los que corresponden valores de "IMC" bajo y "Grasas" bajo. La zona *2* corresponde a nodos con las características de "Talla" alta, "Peso" medio y alto, e "IMC" y "Grasas" intermedias. Cada zona de la capa *(Cluster)* identifica una serie de características definidas. Por último, la zona *6* representa los nodos con valores bajo para la "Talla", alto para el "Peso" y altos para "IMC" y "Grasas".

1.4. Mapas Autoorganizados para la Minería Visual de Datos

La información debe mostrarse de modo que se facilite su percepción y conocimiento; **los propios límites de la atención acotan la habilidad de los observadores de extraer información de las pantallas**. Incluso la respuesta obtenida de una característica visual como el color, fundamental en la percepción, **puede verse afectada negativamente por tareas u organización de la información que requieran una alta demanda de atención y capacidades**. Existe, por tanto, una alta relación entre los límites cognitivos y las tareas demandadas a partir de una información visual. Tener en cuenta esta interacción ayudará a diseñar visualizaciones más efectivas (Haroz y Whitney, 2012). Por tanto, se destaca la importancia de la selección de un sistema de visualización que facilite la realización de diferentes tareas asociadas a la percepción como, por ejemplo, la clasificación de elementos o la búsqueda de patrones discordantes. **Por su método de construcción y características, el SOM es una elección potente para el análisis visual de datos, respondiendo directamente a varios de los retos tecnológicos citados en el apartado 1.1**.

Dadas sus propiedades (proyección gráfica de un espacio de prototipos en una rejilla visual de baja dimensionalidad) el *SOM* es una herramienta excelente para una fase exploratoria inicial y presentación gráfica de datos. Se destaca para ello su capacidad de ordenación general de la información gráfica. Dicha ordenación (Hofmann *et al.*, 2012; Haroz y Whitney, 2012):

- Facilita encontrar elementos discordantes.
- Permite una valoración efectiva de la heterogeneidad de la información así como de la numeración y discriminación de las posibles categorías.
- Para tareas complejas, la agrupación persigue una reducción de la variedad en la vista general sin centrarse en optimizar regiones parciales.

En su modalidad jerárquica, el *SOM* proporciona escalabilidad visual, lo que permite focalizar la atención en grupos determinados de datos, en caso de muestras de datos complejas o muy voluminosas.

El *SOM* se presenta como un modelo gráfico estándar capaz de mostrar gráficamente datos procedentes de distintas vertientes por medio de *vectorización unificada* (Bourennani *et al.*, 2009b). De hecho, se pueden encontrar trabajos sobre integración de datos numéricos con datos textuales (Bourennani *et al.* (2009a)), e integración de datos desde distintas bases de datos empleando como base el SOM (Li y Clifton, 2000).

En resumen, se considera de gran interés la aportación de una herramienta gráfica basada en *SOM* que responda *ad hoc* a los requisitos del usuario, permitiendo la realización de una visualización efectiva de los datos a estudiar. En sucesivos apartados

se presentará la estructura de dicha herramienta, estructurada fundamentalmente en dos partes: la parte de cálculo y la parte de presentación.

1.5. Sistema de ayuda al análisis visual de datos basado en *SOM*

Por lo comentado anteriormente, la propuesta del presente trabajo es preparar un sistema de ayuda al análisis visual de datos, empleando *SOM* y tomando en consideración las necesidades de diseño, generando una estructura que permita una evolución posterior. Una aplicación modular facilitará su entendimiento y modificaciones posteriores. La aplicación se compone de tres partes muy diferenciadas:

1. Un mecanismo de entrada y filtrado de datos.

2. Un algoritmo de cálculo iterativo, de tal modo que el usuario selecciona un conjunto de valores posibles para cada uno de los parámetros de entrenamiento del SOM, y el aplicativo genera las combinaciones posibles de estos valores e iterativamente, entrena con éstos diferentes SOM, seleccionando los que presentan mejores índices de calidad.

3. Un interfaz de usuario interactivo, donde es posible definir los parámetros del experimento, y también consultar SOM calculados que resultaron almacenados en diferentes sesiones de trabajo, en experimentos anteriores.

   Dicho interfaz, o entorno de visualización, permite realizar diferentes acciones (por ejemplo, *zoom*, minimización, reentrenamiento por zonas, selección parcial) que se exponen con detalle en el capítulo 2 y que permitirán extraer mayor información.

Dichas partes se han programado siguiendo un **esquema cliente-servidor, con comunicaciones mediante ficheros XML**. Una de las máximas de desarrollo de la parte de visualización ha sido la productividad en la utilización de recursos máquina. Esto, unido a la programación modular, facilitará posteriores desarrollos cuyo efecto final sea la construcción de un entorno compuesto por un cliente software - no necesariamente con muchos recursos - que se conecte a otra máquina física donde un servidor más potente realice los cálculos pertinentes. Adicionalmente, el entorno de desarrollo empleado permite que el mencionado cliente software pueda generarse a partir de no excesivas modificaciones sobre el código generado siendo compatible con las principales plataformas del mercado.

Capítulo 2

Interacción con el usuario

Lo bello es aquello que es
inteligible sin reflexión.

André Maurois

2.1. Organización

Para ayudar a la percepción de la información relevante, además de facilitar el contraste de características de las diferentes capas, se ha implementado una serie de funciones interactivas que se describen a continuación. La aplicación se ha concebido con una interfaz minimalista de usuario, donde el protagonismo lo toma la visualización de los datos en sí misma. En la figura 2.1 se presenta el aspecto general de la aplicación, dividida en varias zonas. Al inicio de la aplicación por defecto aparece en pantalla el último SOM calculado. A continuación se describen las zonas de pantalla:

1. **Cabecera.** La cabecera contiene el título de la aplicación, autores y versión.
2. **Menú de opciones.** Incorpora todas las opciones que se pueden realizar con la aplicación, incluyendo los modos de selección de datos, opciones de entrenamiento, modos de visualización y navegación por los mapas generados en sesiones anteriores. El menú es retráctil y se muestra plegado inicialmente, para evitar quitar protagonismo a la zona de pantalla dedicada a los datos. Contiene los menús:
   a) **Selección.**
   b) **Visualización.**
   c) **Entrenamiento.**
   d) **Navegación de *SOMs*.**

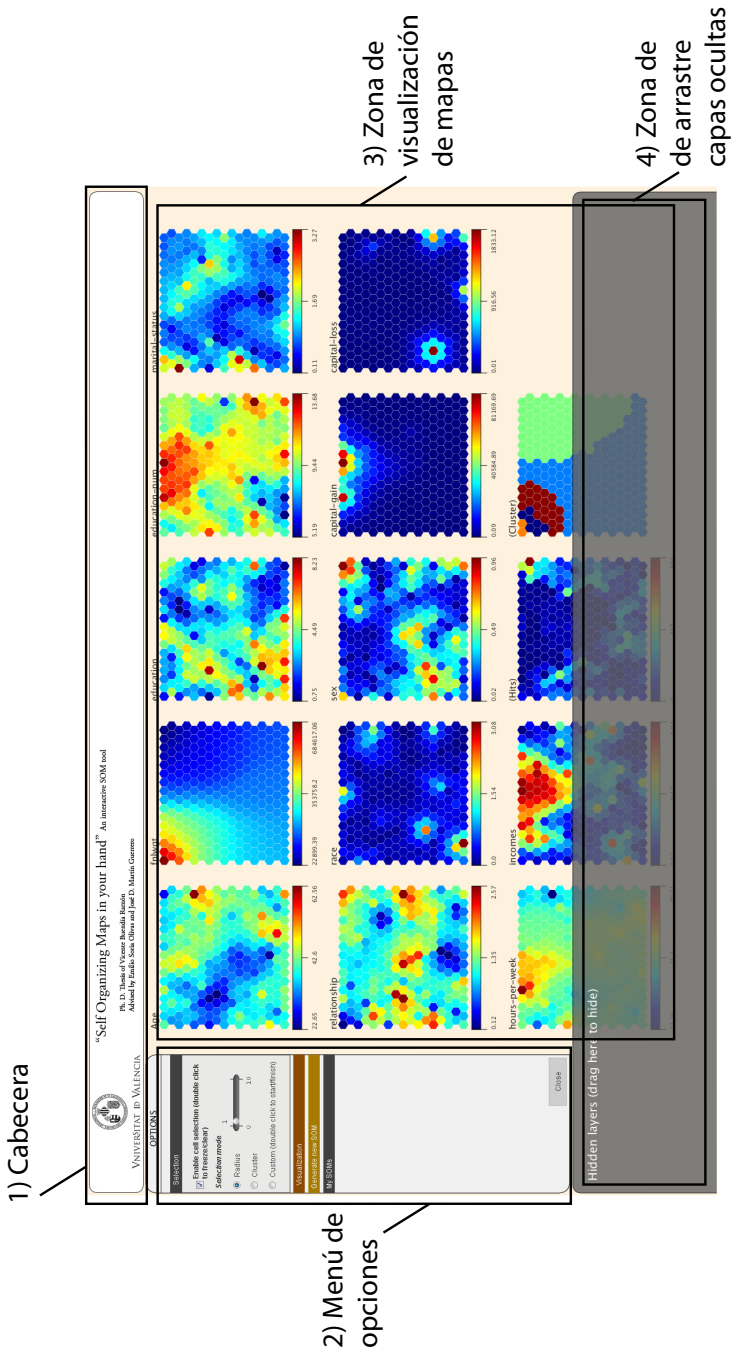


Figura 2.1: Vista general de la aplicación.

3. **Zona de visualización de datos.** La zona de visualización de datos ocupa la mayor parte de la pantalla y es donde se exponen los SOM calculados previamente.

4. **Zona de arrastre de capas ocultas.** Esta zona aloja las capas del SOM previamente ocultadas por el usuario, mostrando de éstas únicamente su nombre o etiqueta, permitiendo ahorrar el espacio que ocuparían en la *zona de visualización de datos*.

A continuación se describen en detalle los contenidos y funcionalidades de las diferentes zonas de la aplicación.

2.2. Menú de opciones

Todas las opciones principales de la aplicación - modos de selección y visualización de datos, parametrización de nuevos entrenamientos y navegación por mapas ya generados - se encuentran concentradas en un único menú desplegable ubicado en la parte izquierda del panel de visualización.

A su vez, cada uno de los submenús es desplegable, de tal modo que en pantalla únicamente se observan las opciones con las que se está trabajando. El menú de opciones se compone de cuatro submenús, representados respectivamente en la figura 2.2, y que se presentan a continuación.

2.2.1. Menú Selección

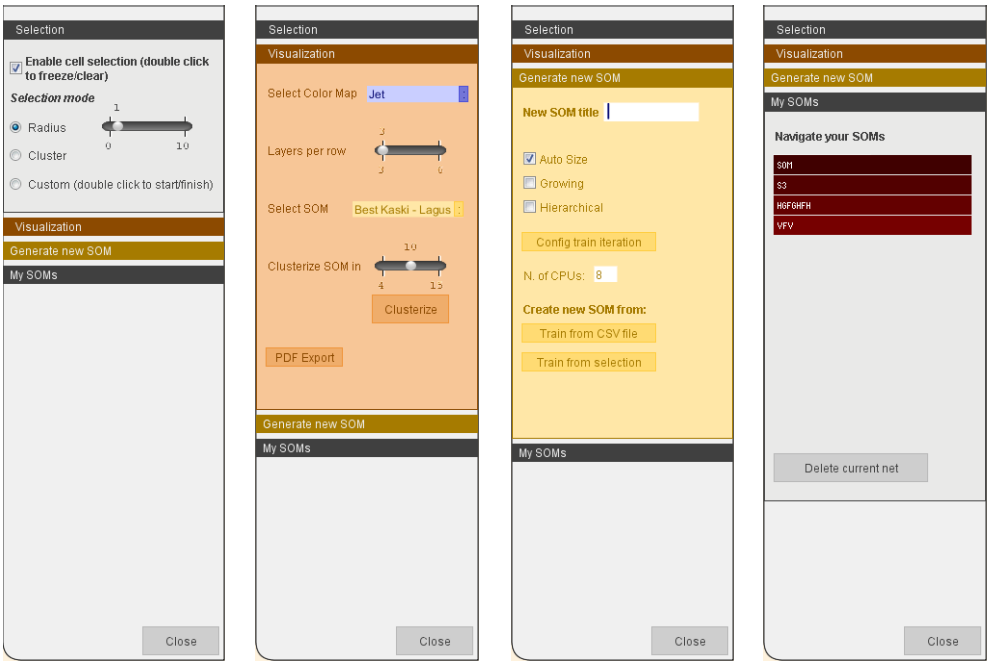
Selección. Las unidades pueden ser seleccionadas gráficamente. Al seleccionar una unidad, o grupo de unidades, esta selección se repite en el resto de capas, de tal modo que se facilita la comparación de una misma zona en las diferentes capas. Adicionalmente, en todos los pies de las capas se muestra el valor promedio de las unidades seleccionadas, el porcentaje de este valor con respecto al valor mínimo de cada capa y el número de muestras *(hits)* asignadas a la unidad, o unidades, seleccionadas (ver fig. 2.3).

Se ofrecen tres modos de selección de unidades:

(I) **Radio.** Selección de unidades presentes en un radio alrededor del cursor. Este radio es configurable y modificable mediante el ratón.

(II) **Cluster.** Se destaca la zona correspondiente al *cluster* mostrado en la capa *Cluster* a la que pertenece el cursor dirigido por el usuario, de tal modo que automáticamente se seleccionan agrupamientos completos (ver figura 2.4).

(III) **A medida.** Existe la posibilidad de dibujar la zona que se desea seleccionar, pudiendo adoptar ésta cualquier forma. Como limitación, debe ser una única zona conexa (ver figura 2.5).



a) Selección b) Visualización c) Entrenamiento d) Navegación

Figura 2.2: Opciones de menú.

En caso de que no se necesite, se puede deshabilitar la opción de seleccionar unidades, evitando así efectos interactivos no deseados.

2.2.2. Menú Visualización

Según el tipo de datos a estudiar puede resultar conveniente que los datos se dispongan en pantalla de un modo u otro por lo que se han creado diferentes opciones de visualización de los SOM calculados:

(I) **Selección de mapa de colores.** Existe la posibilidad de cambiar el juego de colores (también llamado espacio, o mapa, de colores) empleado para mostrar las capas. *Éste debe ser (Sarlin y Rönnqvist, 2013) perceptualmente correcto a la vez que diferenciador e informativo.* El mapa de color por defecto es el *jet*, cuyo rango es desde el color azul al rojo, aunque incorpora azul claro, amarillo y naranja. Dicho mapa está basado en el ideado en el *National Center for Supercomputer Applications* (Sisneros *et al.*, 2008) para la representación de imágenes de combustión. Los gráficos pueden tintarse en escalas de gris o en diferentes

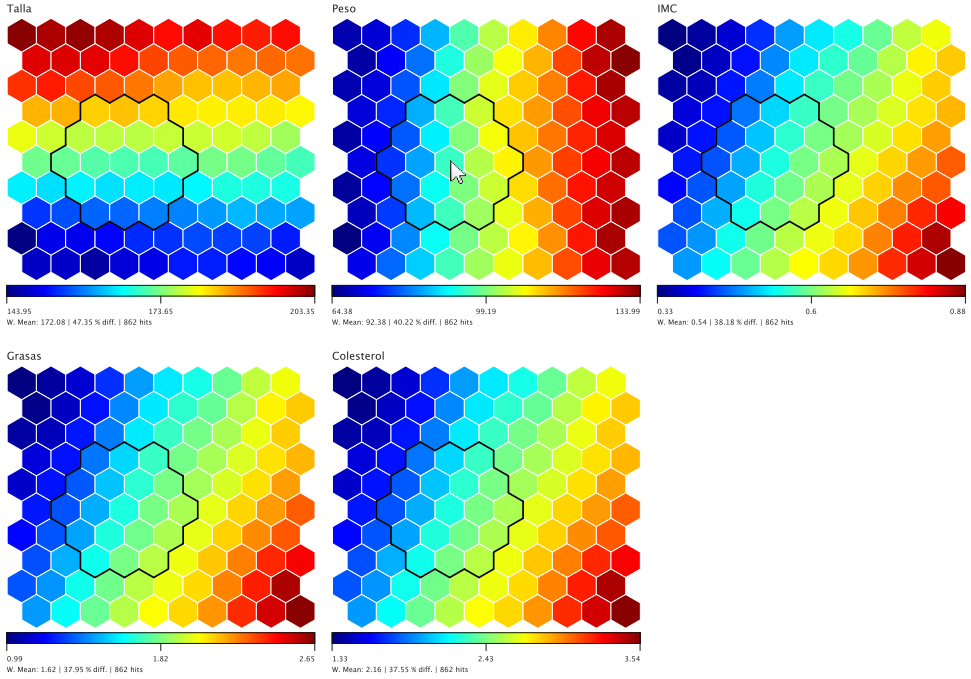


Figura 2.3: Ejemplo de selección de unidades por radio, en el que se seleccionan unidades en un radio de dos unidades a partir de la posición del cursor del ratón.

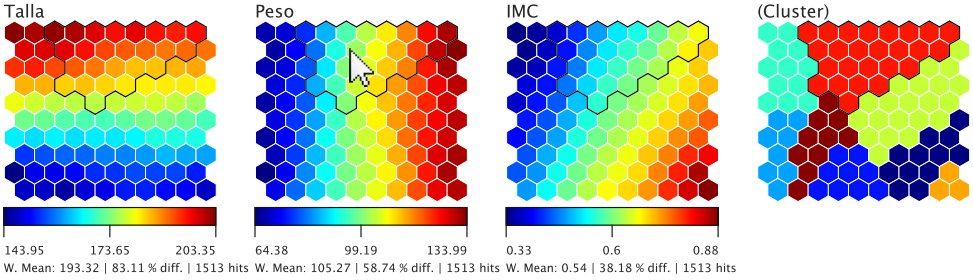


Figura 2.4: Ejemplo de selección por *cluster*. Se seleccionan todas las neuronas que corresponden al *cluster* al que pertenece la neurona bajo el cursor.

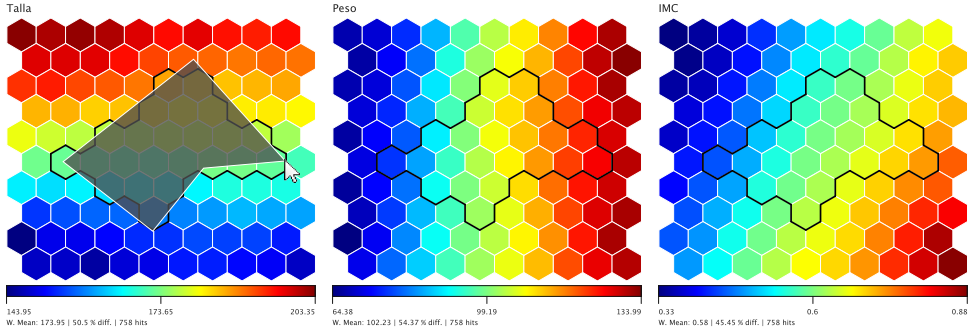


Figura 2.5: Ejemplo de selección a medida. Se seleccionan todas las neuronas que se encuentran bajo una traza realizada por el usuario.

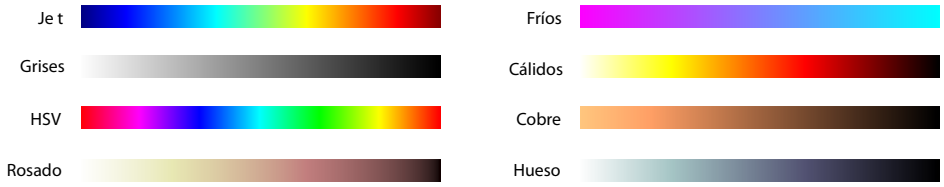


Figura 2.6: Mapas de colores disponibles.

colores con variaciones de brillo. Los juegos de colores disponibles se muestran en la figura 2.6.

(II) **Número de capas por fila.** El número de capas que se muestra en cada fila es configurable, entre un mínimo de tres y un máximo de seis. Además de ser configurable mediante el menú, esta característica de visualización también se puede modificar con el ratón, si éste está provisto de rueda, girando ésta mientras se presiona la tecla *MAYUS*.

(III) **Selección de SOM calculada.** Para un cálculo de SOM realizado para unos datos determinados, se puede visualizar una de entre las tres redes diferentes que maximizan las medidas de calidad expuestas en la sección 1.3.8:

a) La red con menor **Error de Cuantización,** como respuesta al objetivo de mejor cuantización.

b) La red con menor **Error Topográfico,** que maximiza la calidad en lo referido a la ordenación global.

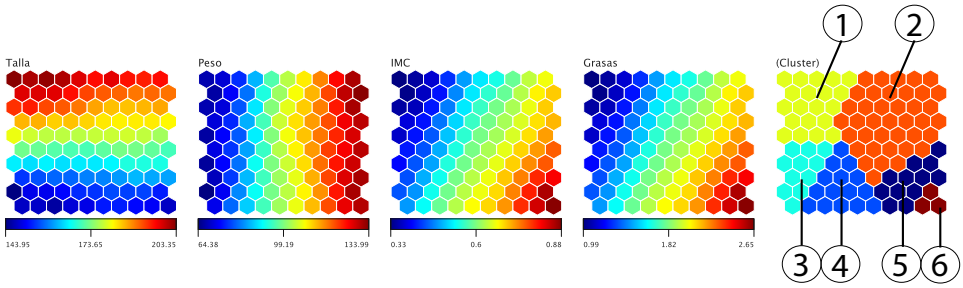


Figura 2.7: Ejemplo de agrupamiento. A partir del mapa se ha generado una propuesta de agrupación en seis zonas distintas.

c) La red con **mejor índice de Kaski and Lagus** (Kaski y Lagus, 1996) que contiene una combinación de las dos primeras.

Agrupamiento. Como ayuda a la visualización, para la delimitación de zonas que contienen datos similares, se ofrece la posibilidad de realizar el agrupamiento (ver figura 2.7) descrito en la sección 1.3.9, donde se ofrece un ejemplo detallado. Se puede seleccionar el número de grupos que se desea ver. Posteriormente podrán realizarse selecciones de unidades empleando como criterio la pertenencia a un cluster.

(IV) **Exportación a PDF.** Al pulsar el botón correspondiente, el contenido de la *Zona de Visualización de Datos* se guarda en un fichero PDF para su posterior tratamiento o publicación.

2.2.3. Menú Entrenamiento

El menú de entrenamiento es, posiblemente, el más importante de la aplicación dado que contiene los parámetros con los que será entrenado el SOM, como se muestra en la figura 2.8. En éste se pueden seleccionar las opciones generales, que son:

(I) **Título.** Cada cálculo de SOM debe ser referenciado de manera unívoca de modo que, posteriormente, pueda ser seleccionado de entre una lista de mapas.

(II) **Tamaño.** Si se conocen las dimensiones idóneas dados unos datos determinados es posible fijarlas. De otro modo se seleccionará tamaño automático. El cálculo del tamaño automático dado un conjunto de datos se calcula como se indica en el apartado 1.3.5.

(III) **Número de CPUs.** Dado que el algoritmo se va a ejecutar repetidas veces con cada una de las combinaciones posibles de los parámetros que se expondrán

a continuación, cabe la posibilidad (e incluso es deseable) del empleo de las capacidades multiproceso de que disponen habitualmente los equipos de trabajo. Dado que las ejecuciones del algoritmo son independientes, se pueden ejeutar de manera concurrente. Por ello, se ofrece la posibilidad de seleccionar el número de CPUs que serán empleadas simultáneamente para la ejecución de las instancias del algoritmo. Esta opción se deja abierta, de modo que el propio usuario puede decidir la potencia a emplear, por si está empleando la máquina para otras tareas. En caso de que por error, el usuario seleccione más CPUs de las que está provisto el equipo, sencillamente se empleará el máximo de CPUs posible.

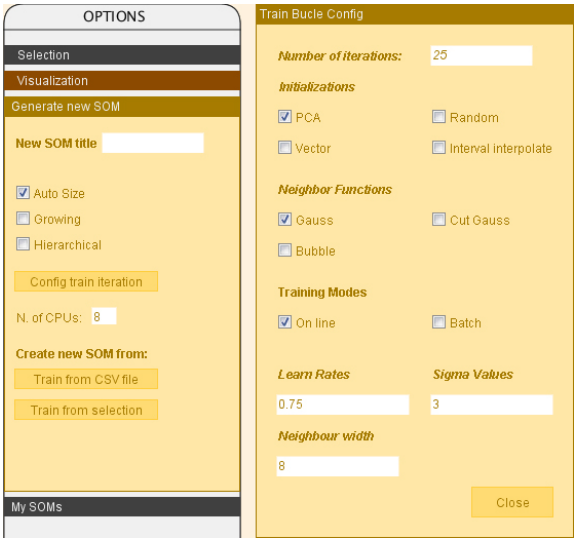


Figura 2.8: Menú de entrenamiento.

Los resultados del algoritmo no son deterministas. Por tanto, el algoritmo se ejecuta de manera repetida para posteriormente seleccionar los resultados que hayan obtenido mejores medidas de calidad (ver apartado 1.3.8). El usuario selecciona el número de iteraciones I a realizar y las opciones de entrenamiento del algoritmo, pudiendo elegir opciones excluyentes entre sí (por ejemplo, dos tipos de inicialización diferentes).
La aplicación realizará I ejecuciones para cada una de las combinaciones factibles de parámetros. Por ejemplo: Si el usuario selecciona veinticinco iteraciones, y ambos modos de cálculo permitidos (*batch* y *on-line*), la aplicación ejecutará veinticinco iteraciones para cada uno de los modos de cálculo, lo que supone cincuenta iteraciones en total. Este método se extiende a todas las combinaciones factibles de los parámetros seleccionados por el usuario. Éste es el llamado *bucle de entrenamiento*, cuyas opciones, seleccionables de manera sencilla mediante menú, son las siguientes:

1. **Número de iteraciones.** Representa la cantidad de veces que se va a ejecutar el algoritmo para cada una de las combinaciones posibles del resto de opciones de entrenamiento que se seleccionen.

2. **Inicializaciones.** Existe la posibilidad de seleccionar las inicializaciones a) *aleatoria*, b) *mediante interpolación de intervalos de datos*, c) *directa por vectores de entrada*, d) *mediante Análisis de Componentes Principales* descritas en el apartado 1.3.6.

3. **Funciones de vecindad.** Se pueden seleccionar las funciones de vecindad: a) *gaussiana* b) *burbuja* c) *gaussiana con sesgo*, descritas en el apartado 1.3.3.

4. **Modo batch.** La aplicación permite seleccionar la variante de *entrenamiento batch* descrita en el apartado 1.3.7.

5. **Constantes de aprendizaje.** Se pueden introducir, separadas por comas, las constantes de aprendizaje iniciales (ver apartado 1.3.4) a emplear para el algoritmo.

6. **Sigma.** Selección del parámetro σ de amplitud del kernel de las funciones de vecindad *gaussiana* y *gaussiana con sesgo* (ver apartado 1.3.3).

7. **Sesgo.** Separados por comas, se pueden elegir diferentes valores de sesgo (ver apartado 1.3.3) a emplear en caso de que se hayan seleccionado las funciones de vecindad *burbuja* o *gaussiana con sesgo*.

2.2.4. Menú Navegación

Todos los SOM calculados previamente son accesibles a través de una lista desplegable. También es posible eliminar el SOM seleccionado. Los mapas calculados se almacenan en carpetas de ficheros (dentro de la carpeta de datos del programa) por lo que son fácilmente transferibles entre diferentes instancias de la aplicación.

2.3. Visualización interactiva

Citando a Jan Miksovsky (McKay, 2013): *En el momento en que un usuario ve tu interfaz, ésta le comunica a dónde ha llegado, lo que puede hacer, y cómo lo debería hacer. El usuario recibe este mensaje desde cada perspectiva de tu diseño: gráfica y textual, silenciosa y audible, estática y en movimiento, intencional y accidental. Imagina qué mensaje quieres que sea, y después haz todo lo que puedas para asegurar que el mensaje de tu interfaz es lo más parecido posible al que pretendías.*

Más allá de la mera presentación en pantalla de la información resultante de un proceso de cálculo, se plantea la realización de una aplicación donde el usuario pueda realizar **acciones sobre la propia presentación de datos**, facilitando así la discriminación de la información presentada visualmente facilitando por tanto el tratamiento y visualización de mayor cantidad de datos frente a la presentación estática. En cuanto al efecto conseguido; el usuario observa que la pantalla, en la mayor parte de su superficie, contiene los gráficos resultantes de los cálculos algorítmicos, que son en sí mismos objetos interactivos. Empleando como herramienta principal el ratón, con el cursor y un único botón se puede interaccionar de manera intuitiva con todos los elementos de pantalla destacando las acciones:

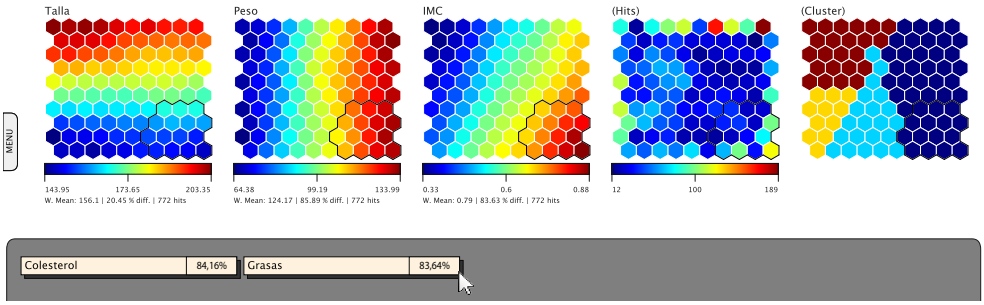


Figura 2.9: Ejemplo de uso de capa oculta. El usuario ha desplazado a la zona inferior dos capas, de las cuales se muestra un resumen con respecto a la zona seleccionada.

- ***Scroll.*** Al mover el cursor del ratón mientras se mantiene el botón pulsado (acción conocida comúnmente como *arrastrar*) las capas se desplazan en pantalla hacia arriba o hacia abajo, según el sentido en que se mueva el cursor.

- **Selección de nodos.** Al pasar el cursor sobre una capa, se preseleccionan los nodos con la topología dependiente del tipo de selección elegido (ver *menú selección*, apartado 2.2.1). En sincronía, se preselecciona el grupo de nodos equivalente en el resto de capas. Al preseleccionar un grupo de nodos, se muestra bajo cada capa:

  - El valor promedio del grupo de nodos.
  - El valor promedio *normalizado* V_{norm} cuya fórmula de cálculo es la siguiente:

    $$V_{norm} = \frac{V_n - V_{min}}{V_{max} - V_{min}} \tag{2.1}$$

    Donde V_n es el valor promedio de los nodos, y V_{min} y V_{max} son los valores mínimo y máximo respectivamente.
  - Número de *hits* o elementos de la muestra original que pertenecen al grupo de nodos seleccionado.

  Al pulsar dos veces seguidas el botón del ratón, los nodos preseleccionados pasan a estar seleccionados, de tal modo que el usuario puede realizar otras acciones sobre la aplicación sin perder la selección y los datos mostrados en pantalla.

- **Selección y desplazamiento de capas.** Al mantener, con el cursor inmóvil, el botón unos segundos sobre una capa, se selecciona una capa entera, dando efecto de *despegue* del lienzo, pudiendo desplazarse a la ubicación que se desee, permitiendo al usuario poner cerca las capas que necesite comparar visualmente.

- **Gestión de capas ocultas.** Al desplazar una capa a la zona inferior de la pantalla, señalizada en otro color, de esta capa se muestra únicamente un resumen con su nombre y el valor promedio *normalizado* V_{norm} de los nodos de la misma zona que el resto de nodos seleccionados en las demás capas (ver fig. 2.9). De este modo se pueden comparar capas sin necesidad de mostrarlas al completo, ahorrando espacio en pantalla.

El manejo en general está inspirado en las aplicaciones móviles, donde con pocos elementos (uno o varios puntos de presión sobre la pantalla) y mediante gestos se desarrolla toda la actividad. Debido a la penetración en mercado de los dispositivos móviles, esta similitud propicia que a) la curva de aprendizaje tenga una pendiente elevada y b) adicionalmente, constituye un avance en el análisis funcional y estudio de experiencia de usuario para la posible generación posterior de aplicaciones móviles para mostrar SOM calculados previamente.

Capítulo 3

Recogida y tratamiento de los datos

La razón es también una pasión.
Eugeni d'Ors

3.1. Estructura interna

La aplicación se ha diseñado de **forma modular**. Este tipo de diseño proporciona diversas ventajas: **Facilita la legibilidad y comprensión del código** y, por ende, su mantenimiento en lo que refiere a corrección de errores y evolución. La modularidad minimiza las dependencias entre bloques de código, lo que supone el **aislamiento de errores y de modificaciones**. Adicionalmente, esta estructura permitirá la **reutilización** por separado de fragmentos del código y su integración dentro de otras aplicaciones, facilitando, por ejemplo, su dimensionamiento a gran escala. El esquema general de la aplicación se presenta en la figura 3.1, donde se muestran sintéticamente tres grandes módulos, muy diferenciados, con intercomunicaciones entre éstos muy definidas. Los módulos son: **a) Entrada de datos, b) cálculos internos y c) visualización**:

- ***Entrada de datos:*** La aplicación es compatible con ficheros .CSV, esto es, ficheros con los datos ASCII (de tipo texto), separados por comas. Cada muestra original de datos a analizar se cargará procedente de un único fichero. El formato de los ficheros de entrada es sencillo: La primera línea contiene los nombres de las características o campos a analizar y, a partir de la segunda línea en adelante, los valores asignados a éstas. Si se proporcionan valores no numéricos, el programa asignará un valor entero a cada cadena diferente para tratarlo como si así lo fuera.

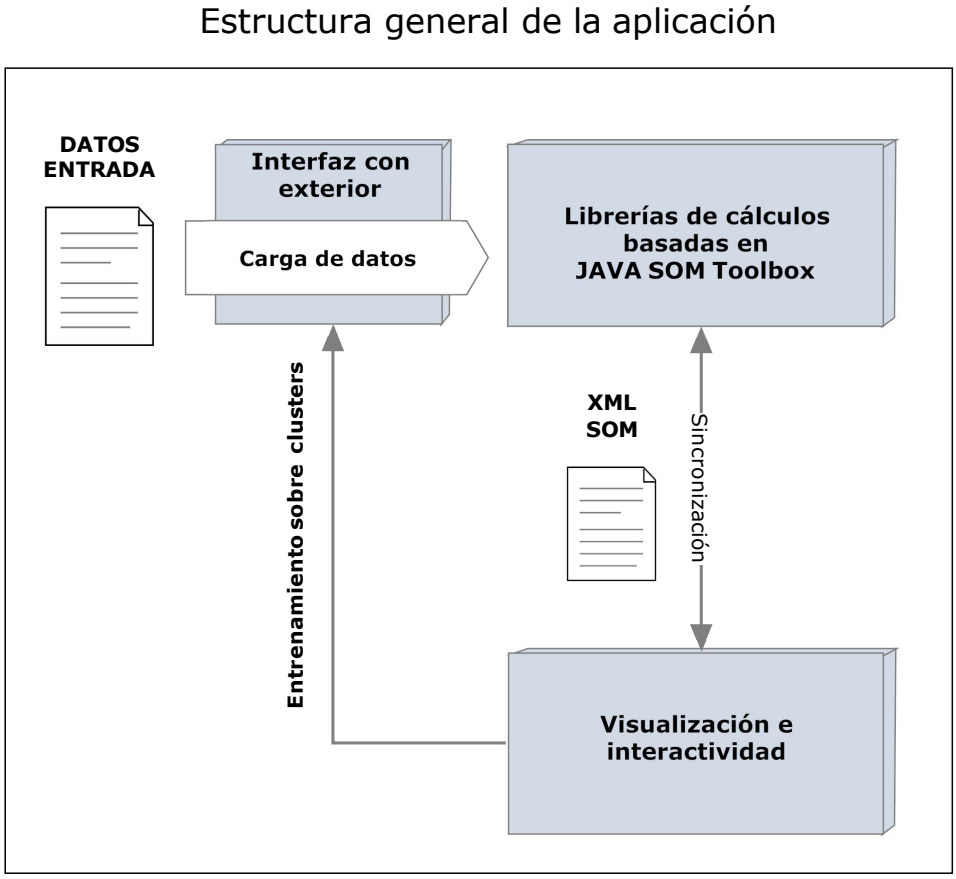


Figura 3.1: Estructura y módulos principales de la aplicación. Se observan tres partes diferenciadas: a) Interfaz con el exterior, módulo encargado de la carga y filtrado inicial de datos. b) Librerías de cálculos internos. c) Visualización e interactividad.

Para la entrada de datos se han empleado las librerías informáticas que proporciona *WEKA* (Hall *et al.*, 2009). La aplicación es compatible con ficheros de datos en formato CSV, para cargar el contenido a analizar en memoria. Asimismo, dado el empleo de las librerías *WEKA*, la aplicación también es compatible con ficheros en los formatos: *Arff*, , *XRFF* (versión *XML* del formato *Arff*), *JSON*, *Matlab* en formato *ASCII* y *C45*. Dichas librerías *WEKA* facilitan la realización de futuras extensiones para permitir la compatibilidad con bases de datos.

- ***Cálculos internos:*** Se ha empleado como modelo la librería *JAVA SOMToolbox* (Mayer *et al.*, 2011). Dicha librería fue creada como prototipo de ayuda para la investigación, y su estructura ha servido de inspiración y base para el proyecto[1]. **A partir de los datos importados en la etapa anterior se ejecuta el experimento de acuerdo con los parámetros de entrenamiento SOM seleccionados**. Finalmente, **los SOM generados se almacenan en ficheros** dentro de la estructura de directorios de la aplicación, de tal modo que podrán ser importados por cualquier aplicación, en este caso, el visualizador interactivo desarrollado a tal efecto. Los SOM almacenados en fichero se codifican en formato *XML* para facilitar la exportación y modularidad. El visualizador interactivo carga posteriormente dichos ficheros para su tratamiento visual. De este modo, se independizan los cálculos de la visualización, facilitando futuras labores de mantenimiento y de intercomunicación entre diferentes aplicaciones que consuman dicha información. La estructura de dichos ficheros se muestra en el apéndice B.

- ***Visualización:*** Tras recibir la señal de sincronización con el sistema de cálculo, los resultados se cargan de los ficheros y se presentan en pantalla, permitiendo al usuario efectuar selecciones parciales de los datos, modificar la posición de los planos, elegir diferentes opciones de visualización y, en definitiva, interactuar con el mapa. Estas opciones se resumen en el Apéndice C.2, y se detallan en el capítulo 2.

A continuación se presentan con más detalle las componentes que intervienen en la estructura de la solución.

3.2. Entrada de datos

Aunque como se ha expuesto anteriormente se soporta variedad de formatos, por defecto se toman los ficheros en **formato *CSV* ASCII**; es decir, ficheros de tipo texto separados por comas. Este tipo de ficheros se emplea de forma habitual dada la **sencillez con que puede ser generado**.

[1] Como referencia: Se han incluido, o modificado, aproximadamente 10.400 líneas de código.

La primera línea, o cabecera del fichero, se compone necesariamente de los nombres de los campos, separados por comas. Cada una de las líneas del resto del fichero representa un elemento de la muestra, conteniendo los valores correspondientes a los campos declarados en la cabecera.

Los datos numéricos son la base del algoritmo; los valores decimales se representan mediante puntos. **También se pueden incluir cadenas de texto como datos de entrada, en este caso discretos**; de manera automática, el algoritmo etiqueta numéricamente cada cadena distinta, y la sustituye internamente por dicha etiqueta. De este modo, el SOM se entrena con todos los valores numéricos, aunque externamente es capaz de trabajar y mostrar dependencias entre valores no numéricos.

La secuencia completa de carga de datos se presenta en la figura 3.2. Para el proceso de entrada de datos se ha modificado la clase original *InputDataFactory* del proyecto *JAVA SOMToolbox* (Mayer *et al.*, 2011), insertando en ésta, entre otras, la clase *WEKA* CSVLoader (Hall *et al.*, 2009), que se encarga de realizar la carga y filtrado inicial. Se han incluido filtros adicionales sobre los datos, con el propósito de optimizar el algoritmo general. Sobre la estructura obtenida, de tipo *WEKA Instances*, se realizan, por orden, los procesos:

1. Eliminación de las variables con desviación típica nula. Se puede deducir que no proporcionan información en lo que se refiere al cálculo de *SOM* (son constantes en todos los patrones). Por tanto, supondrían carga de proceso, sin proporcionar ningún valor.

2. Análisis por Componentes Principales, manteniéndolo precalculado y adjuntando éste como anexo a los datos, dado que será empleado sucesivas veces en cálculos posteriores.

3. Conversión de los datos a la estructura compatible con *JAVA SOMToolbox*.

Una vez cargados y filtrados, los datos están preparados para ser procesados por el algoritmo de entrenamiento SOM. El módulo de carga de datos está claramente delimitado, de este modo se facilita la realización de modificaciones sobre éste que afecten a los datos de entrada, sin afectar necesariamente al resto de los aplicativos.

Además de los datos a tratar, la interfaz convierte las preferencias que el usuario introduce en la aplicación (Apéndice C.1) en propiedades del experimento. Estas propiedades se almacenan en un objeto *Java* cuya clase es denominada *ExpProps (acrónimo de "Propiedades del experimento")*. Éstas serán empleadas, junto con los datos de entrada procesados, en la realización de los cálculos internos que se describe a continuación.

Por lo anterior se puede deducir la opción de diseño elegida: Todos los parámetros de entrada del algoritmo general quedan empaquetados en la clase Java *ExpProps*; este tipo de configuración facilita el mantenimiento y la posterior modificación del algoritmo, dado que para modificar los parámetros de entrada, bastará con incluir los cambios pertinentes en esta clase, que se cumplimenta en la interfaz interactiva y es procesada, posteriormente, por el algoritmo de cálculo. Dicha clase contiene, además

de los valores seleccionados para los distintos parámetros de entrenamiento generales, los procedimientos de consulta de éstos, y los mecanismos de almacenamiento y carga en fichero, almacenándose en formato XML junto con los resultados del experimento. Se puede consultar la estructura XML referida en el Apéndice A.

3.3. Cálculos internos

3.3.1. Esquema general

Exponiendo el proceso en términos de función de transferencia, el procedimiento principal se ha etiquetado como *TrainSelector* (acrónimo de *Entrenamiento* y *Selector*) expuesto en la figura 3.3, que toma como entradas de datos las descritas en el apartado 3.2:

1. Los datos a analizar una vez filtrados y codificados en formato *Java SOMToolbox*, esto es, incluidos en una clase de tipo *Java SOMToolbox InputData*. En esencia, esta clase incluye una matriz de números junto con los nombres de los campos así como funciones auxiliares para la recuperación de datos.

2. Las opciones de entrenamiento del usuario (ver Apéndice C.1), introducidas previamente en la interfaz interactiva, posteriormente almacenadas en memoria codificados en una clase Java de tipo *ExpProps*, descrita anteriormente.

A partir de los citados datos de entrada, *TrainSelector* realiza el entrenamiento de un conjunto de SOM y, sobre el conjunto resultante, una selección posterior, obteniendo:

1. Tres *SOM* seleccionados respectivamente por los mejores índices (expuestos en el apartado 1.3.8):

   a) Kaski and Lagus (Kaski y Lagus, 1996).

   b) Error de cuantización.

   c) Error topográfico.

2. Una tabla que relaciona dichos índices según los parámetros con los que se entrenó cada red, lo que posibilita el estudio de cuáles son los parámetros de entrenamiento que funcionan mejor con un determinado conjunto de datos a procesar.

A la finalización de los cálculos, los SOM seleccionados son almacenados en formato XML (estructura presentada en Apéndice B) y sincronizándose con la interfaz interactiva para indicarle la disponibilidad de los resultados. Dicho interfaz interactivo cargará los SOM mostrándolos al usuario para su consulta y análisis visual.

El proceso principal, *TrainSelector*, se describe a continuación.

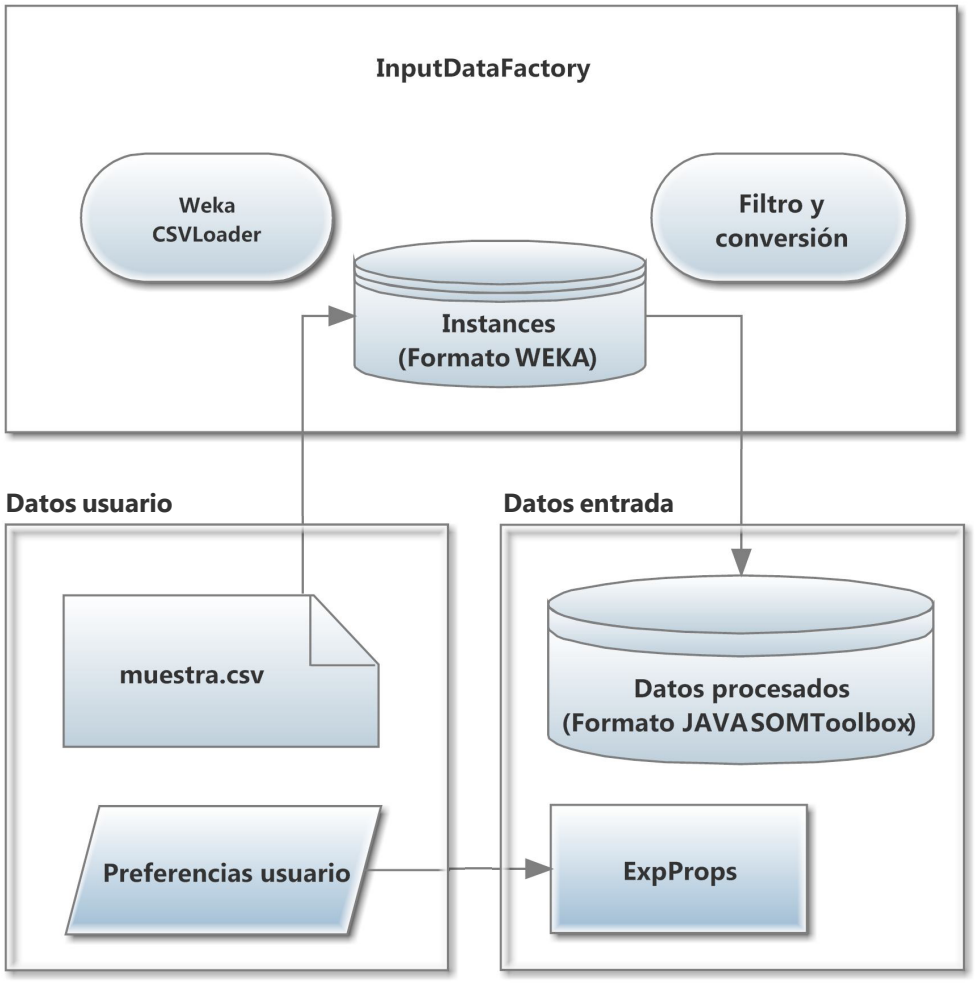


Figura 3.2: Carga de datos seleccionados por el usuario, empleando WEKA y Java SOMToolbox.

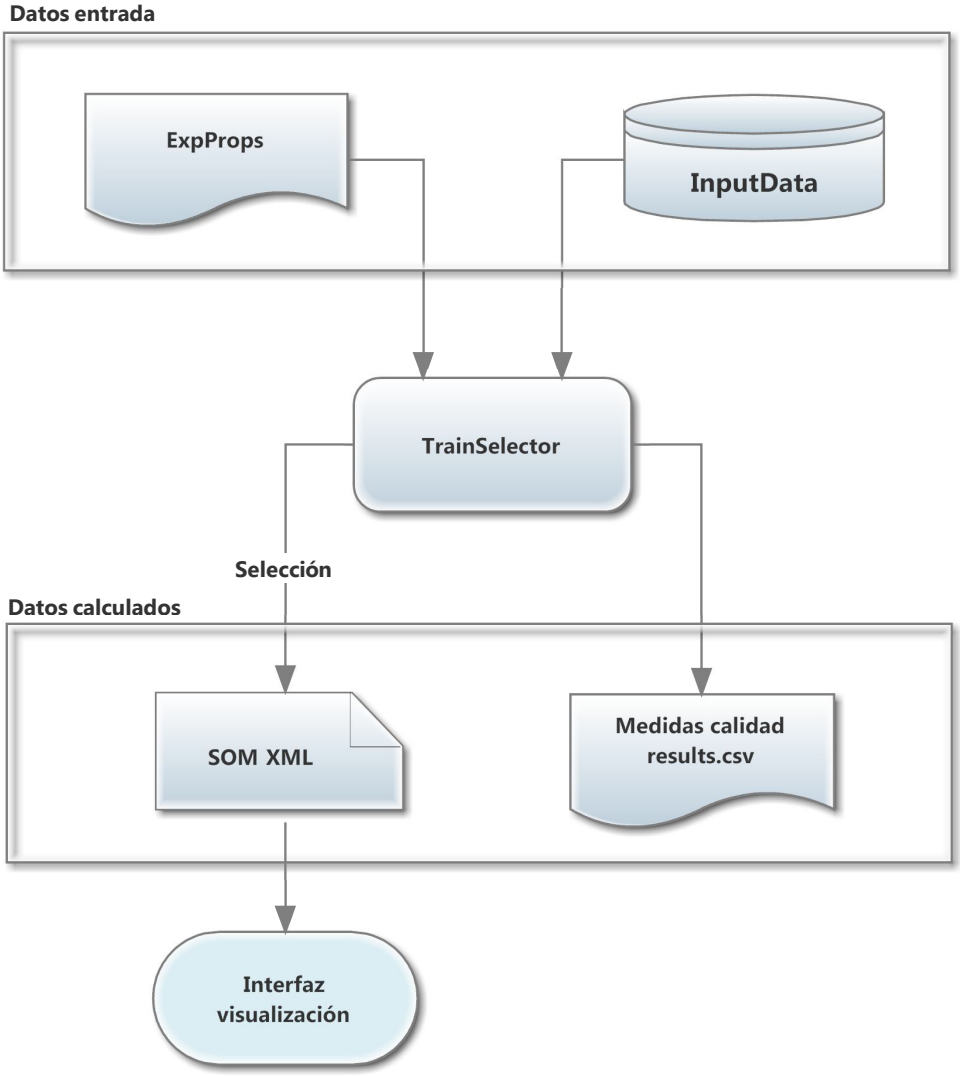


Figura 3.3: Entrada de datos y resultado de los procesos internos de datos.

3.3.2. Organización de los procesos de entrenamiento

El objetivo fundamental del proceso *TrainSelector* (figura 3.4) como elemento fundamental en la organización de los procesos de entrenamiento, es la generación de varios SOM a partir de las preferencias del usuario, y de los datos de entrada, según esquema general expuesto en la figura 3.3.

Las acciones que se realizan en *TrainSelector* se resumen en:

1. **Tratamiento inicial de datos.** Se realiza un proceso que facilitará las tareas siguientes.

   a) Se calcula, si el usuario lo ha elegido previamente, el tamaño que tendrán los SOM calculados, a partir del tamaño de la muestra. La ecuación de cálculo se presenta en el apartado 1.3.5.

   b) Se preparan las incializaciones de SOM (ver apartado 1.3.6) que serán empleadas repetidas veces para los sucesivos entrenamientos, excluyendo la *inicialización aleatoria* que será generada en cada ocasión.

2. **Generación de bucle de entrenamiento.** Las opciones que el usuario ha seleccionado se encuentran almacenadas en un objeto de tipo *ExpProps*. Recordando que estas opciones pueden contener distintas opciones de entrenamiento, se generan todas las combinaciones posibles de valores que se utilizarán para entrenar sucesivos SOM. La información contenida en el objeto de tipo *ExpProps* se transforma, de modo que cada combinación de valores se almacena por separado, en un objeto de tipo *SOMProps* (acrónimo de *Propiedades de SOM*). Se generarán, por tanto, tantos objetos de tipo *SOMProps* como combinaciones posibles de parámetros de entrenamiento se puedan extraer de las preferencias del usuario. Cada *SOMProps* generado se empleará para entrenar redes tantas veces como *iteraciones* haya decidido el usuario (ver Apéndice C.1). Así pues, en cada iteración del proceso se entrena un SOM empleando:

   - Los parámetros de entrenamiento almacenados en un objeto de tipo *SOMProps*.
   - Los datos de entrada, almacenados en un objeto de tipo *InputData*.

   El procedimiento de entrenamiento y el *SOM* en sí se encuentran encapsulados dentro de un objeto de tipo *GrowingSOM*, que será descrito en el apartado 3.3.3. Las ejecuciones se efectúan en paralelo para emplear las capacidades multitarea de los equipos informáticos, lo que supone un ahorro importante en tiempo

3. **Almacenamiento de resultados.** Una vez realizado el entrenamiento, se miden los indicadores de calidad mencionados en el apartado expuestos en el apartado 1.3.8, para seleccionar posteriormente los SOM con mejores medidas, almacenándolos en ficheros con formato XML. Adicionalmente, se almacenan todas las medidas y parámetros para permitir comprobar, si para una muestra determinada existen parámetros que proporcionen mejores resultados.

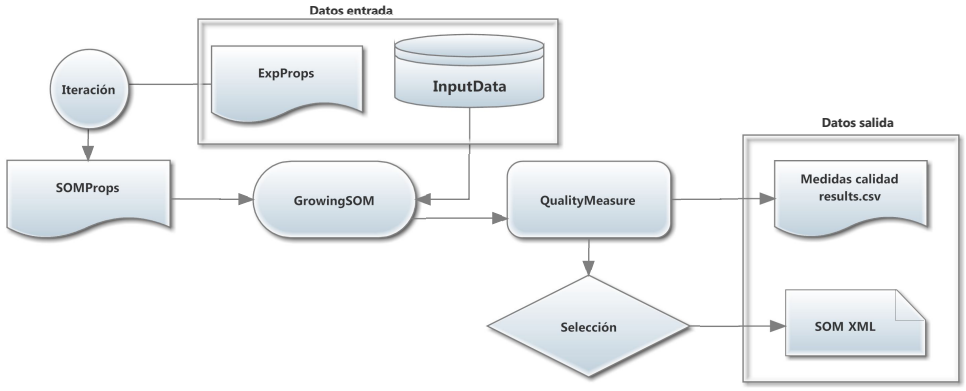


Figura 3.4: TrainSelector: Generación iterativa de entrenamientos y selección de mejores SOM.

4. **Sincronización.** De manera continua, durante todo el proceso de cálculo se envía al entorno interactivo un indicador de progreso, para proporcionar información en tiempo real al usuario. Una vez finalizado, se envía la señal de fin al entorno interactivo (100 % realizado), para indicar que éste ya puede proceder a cargar resultados y presentarlos en pantalla.

3.3.3. Entrenamiento

La clase Java etiquetada como *GrowingSOM* contiene los datos y procedimientos necesarios para realizar el entrenamiento de un SOM. Se distinguen fundamentalmente dos tipos de entrenamiento: Entrenamiento *On Line* y entrenamiento *Batch*, expuestos respectivamente en los apartados 1.3.2 y 1.3.7.

A su vez, *GrowingSOM* se encuentra modularizada, para facilitar su comprensión y mantenimiento, distinguiéndose:

1. Interfaz de comunicación de datos con el resto de la aplicación, por medio de la carga de información procedente de un objeto de tipo *SOMProps* (definido en el apartado 3.3.2).

2. Interfaz de importación y exportación de datos a fichero de tipo XML, para la comunicación con el visor interactivo o cualquier aplicación que haga uso de las estructuras de cálculo. Para ello se ha diseñado un protocolo específico de carga y grabación en ficheros en XML.

3. Objetos de almacenamiento y cálculo de la información; capas y nodos se encuentran modelizados por separado.

4. Algoritmos de *entrenamiento*, tal y como se define en el capítulo 1.

3.3.4. Ejemplo de entrenamiento

A continuación se proporciona un ejemplo de entrenamiento. Para el mismo se ha seleccionado la base de datos *Adult Data Set* [2] procedente del repositorio *Machine Learning Repository* de la Universidad de California, Irvine (Lichman (2013)). El objetivo inicial de la recogida de la muestra era la predicción de características discriminatorias de personas que ingresaban más de 50.0000 dólares al año. Dicha base de datos, de 31.561 registros, contiene un extracto del censo de Estados Unidos de 1994, incluyendo los campos:

- Edad.
- Nivel educativo.
- Ocupación laboral.
- Estado civil(soltería, matrimonio, viudedad, etc.).
- Relación familiar (marido, mujer, hijo único, sin familia, etc.).
- Raza (blanco, asiático, negro, amerindio, etc.).
- Sexo.
- Ganancias de valor de patrimonio.
- Pérdidas de valor de patrimonio.
- País de nacimiento.
- Horas de trabajo a la semana.
- Ingresos. Valor discreto con dos opciones: mayor de 50.000 dólares ó menor o igual a 50.000 dólares al año.

Sobre la base de datos referida se ha efectuado un entrenamiento iterativo variando los siguientes parámetros, definidos en el apartado 1.3 del capítulo 1.1:

- Inicializaciones: a) Mediante *Análisis de Componentes Principales*, b) *directa por vectores de entrada*, c) *aleatoria*, d) *interpolación de intervalos de datos.*
- Funciones de vecindad a) *gaussiana*, b) *gaussiana con sesgo*, c) *burbuja*
- Modos de entrenamiento a) *batch* y b) *on-line.*
- Parámetros de aprendizaje inicial a) $\alpha = 0{,}2$ y b) $\alpha = 0{,}5$.

[2] https://archive.ics.uci.edu/ml/datasets/Adult

- Valores de atenuación de las funciones de vecindad *gaussianas* a) $\sigma = 3$ y b) $\sigma = 5$.
- Sesgos de vecindad (percentuales) a) 0.5 y b) 0.7.

El tamaño de la red se ha calculado mediante el heurístico expuesto en el apartado 1.3.5 del capítulo 1.1, resultando en matrices de 17 filas y 16 columnas. Con cada combinación de parámetros se han entrenado 50 *SOM*. Tomando en consideración que el sesgo de vecindad no afecta a la función de vecindad *gaussiana* y que el parámetro σ no afecta a la función de vecindad *burbuja*, se ha realizado un total de 6.400 entrenamientos[3], seleccionando de entre éstos el *SOM* con mejor índice de *Kaski y Lagus*.

El resultado de entrenamiento se muestra en la figura 3.5. Del gráfico se pueden extraer varias conclusiones:

1. En general, existe relación entre los ingresos y diferentes parámetros: nivel de estudios, raza, ocupación, sexo y tipos de trabajo. Los incrementos de valor de patrimonio se producen en las personas con ingresos altos. *Referencia: Ver en capa Ingresos la zona roja, que se corresponde con los ciudadanos de ingresos altos, y compárese con las neuronas en posiciones equivalentes en las capas mencionadas.*

2. En general, los ciudadanos que trabajan más horas y los ciudadanos que trabajan menos horas, no se encuentran entre los ciudadanos con ingresos altos. *Referencia: En la capa (Horas sem.), observar las neuronas que corresponden con valores altos y con valores bajos, comparándolas con sus respectivas geográficas en la capa* ***"Ingresos"***.

3. La capa ***hits***, donde se muestra una distribución de *"Hits"* o *BMU's* muy asimétrica, pone de relieve que la mayoría de ciudadanos no tiene ingresos superiores a 50.000 dólares al año.

El mapa seleccionado se muestra en la figura 3.5. Éste procede de un entrenamiento efectuado con los parámetros: a) Inicialización: *Vector*; b)Función de vecindad: *Gaussiana*; c) $\sigma = 5$; d) $\alpha = 0{,}2$; e) Entrenamiento en modo *batch*.

La figura 3.6 contiene los resultados de error obtenidos en los entrenamientos, denotando una clara dependencia entre valores de parámetros e índices de error. Se pueden realizar varias observaciones, alguna de éstas intuitivamente lógica: el *índice de Kaski y Lagus* supone un compromiso entre *error topográfico* y *error de cuantización*, que se encuentran enfrentados. Se observa que los valores altos del parámetro *sigma* y función de vecindad *gaussiana*, resultan habitualmente en *Error Topográfico* bajo y *Error de Cuantización* alto. Un ancho de sesgo alto se relaciona con valores elevados del error topográfico. En resumen, se observa que el resultado del entrenamiento es claramente dependiente de los valores de los parámetros de entrada, lo

[3] El tiempo de ejecución del experimento ha sido de 1 hora en un ordenador personal doméstico con procesador Intel I7 y 8 *threads*.

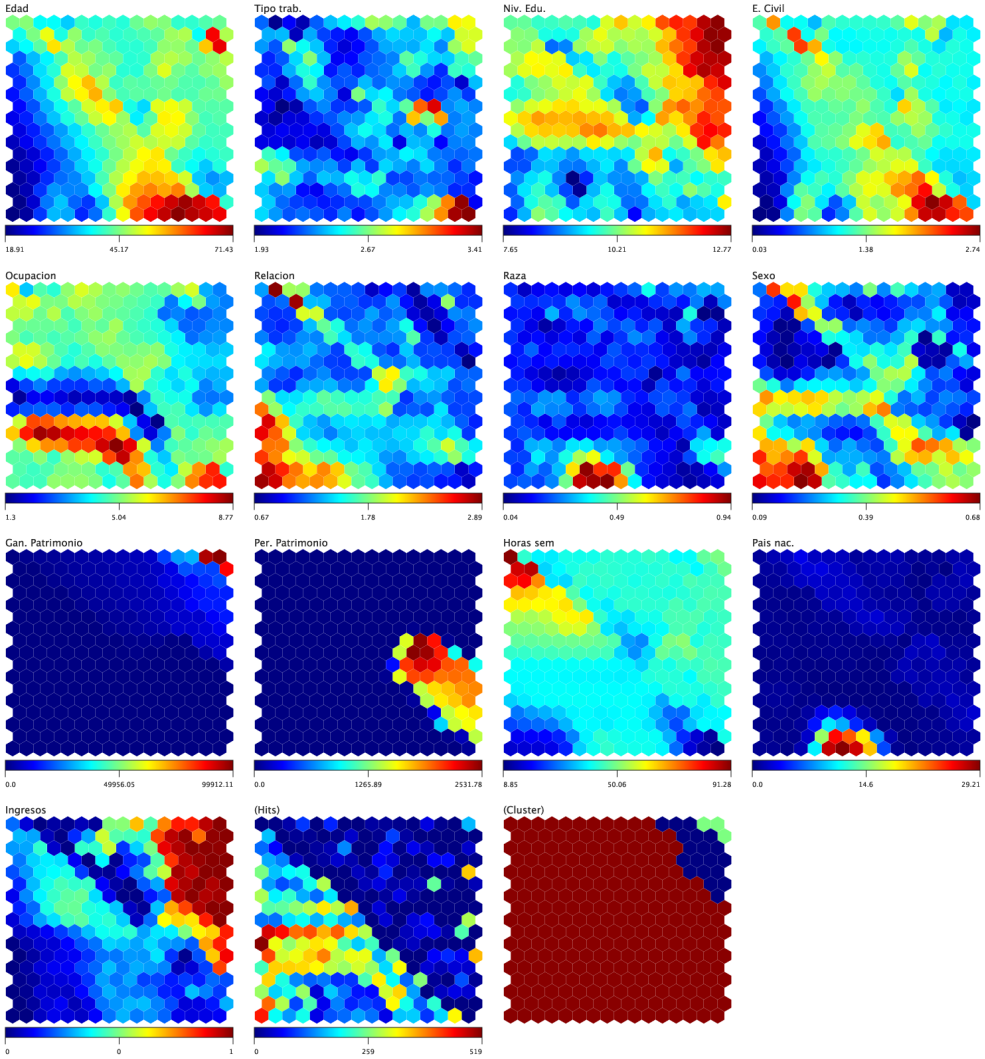


Figura 3.5: Representación de ingresos económicos en USA según situación social, año 1994.

que justifica, en caso de desconocerse a priori los parámetros óptimos, la realización de experimentos de entrenamiento con diversos parámetros para después seleccionar resultados según criterio de error.

3.4. Optimizaciones con respecto al entrenamiento secuencial

Como se ha descrito anteriormente, uno de los objetivos de la aplicación diseñada es el entrenamiento por lotes de un conjunto de SOM. Estos entrenamientos son independientes entre sí. Este hecho, unido a la circunstancia de que cada entrenamiento incluye la ejecución repetitiva de un mismo tipo de acciones cuyos resultados calculados pueden ser almacenados, permite la disposición de diversas optimizaciones que abrevian el proceso completo. Esta posibilidad se convierte en preceptiva si se pretende ejecutar un experimento que cubra un número de opciones suficiente para una cantidad razonable de datos.

En la búsqueda de obtención de costes bajos de ejecución, se ha hecho uso de diversas optimizaciones sobre la implementación del algoritmo, a destacar:

- **Paralelismo.** La tecnología actual permite aprovechar las posibilidades multiproceso de los computadores. La ejecución en paralelo de diversas instancias de cálculo de SOM redunda en una disminución del tiempo total de ejecución. El usuario decide el número de ejecuciones que se realizan en paralelo, según las capacidades de su máquina que planifique dedicar. De este modo, es posible emplear máquinas de uso doméstico para cálculos que antaño habrían requerido de caras estaciones de trabajo.

- **Precálculo de los kernels de vecindad**. Dado un elemento $\boldsymbol{x_k}, 1 < k < n$ de la muestra de entrada, y su correspondiente *BMU*, el cálculo de los ajustes de las z neuronas según la función de vecindad $h_{ci}()$ empleada, supone un coste de ejecución de dicha función zn veces. Sin embargo, si se calcula previamente la función $h_{ci}(t)$ para todos los pares de nodos, transformando el cálculo $h_{ci}(t)$ en una tabla de correspondencia, únicamente hacen falta z^2 ejecuciones. Por tanto, con este método se evitan $zn - z^2$ cálculos, lo cual es positivo siempre que una etapa t incluya un número de muestras $n > z$. Dado que n se puede diferenciar en varios órdenes de magnitud de z, el ahorro es muy importante, permitiendo su ejecución incluso en equipos domésticos. Las etapas t dependen de las variaciones internas en los parámetros de entrenamiento, que diferencian las fases de *ordenación* y *afinamiento* expuestas en el apartado 1.3.3 del capítulo 1.1.

- **Precálculo de las capas con inicialización que depende de una función de los datos**. Existen varias inicializaciones que dependen de los datos, y que no son aleatorias: *Inicialización mediante interpolación de intervalos de datos*

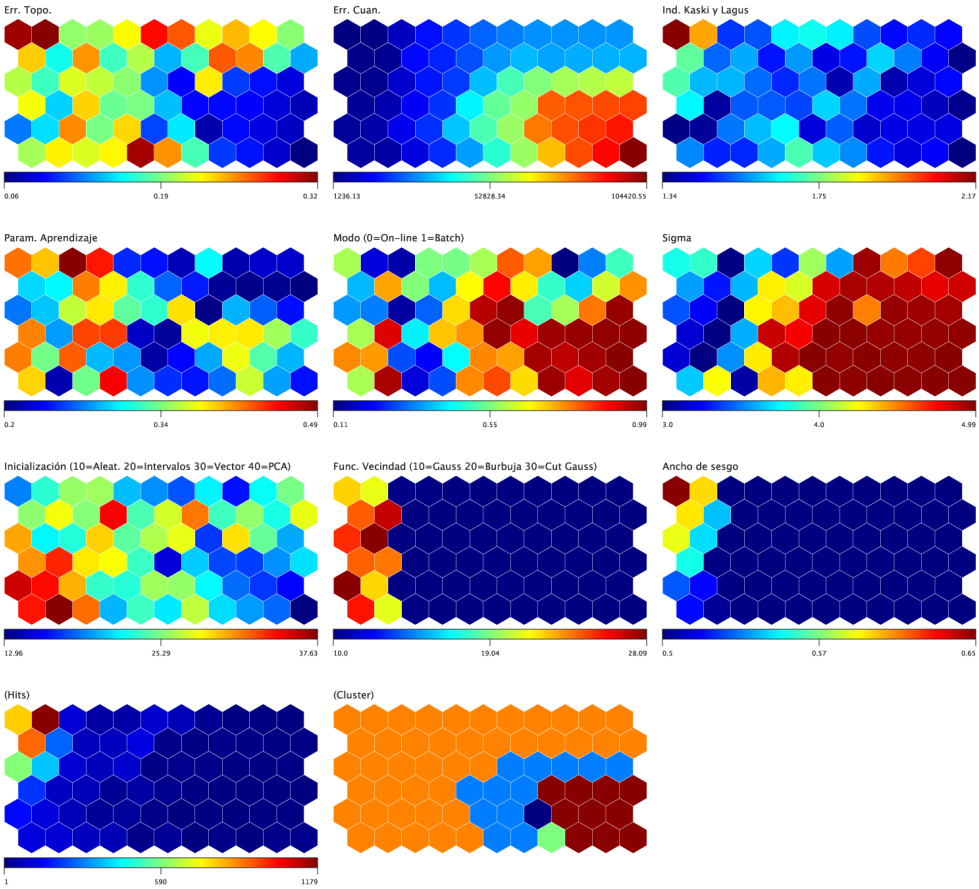


Figura 3.6: Índices de calidad según valores elegidos para los parámetros de entrenamiento *SOM*.

e *Inicialización mediante el Análisis de Componentes Principales*, expuestas en el apartado 1.3.6. En estos casos, se hace el cálculo previo y se deja almacenado para ser empleado en cada entrenamiento que lo requiera.

Capítulo 4

Ampliaciones y variantes del *SOM*

Caminante no hay camino,
se hace camino al andar.

Antonio Machado

4.1. Reentrenamiento parcial

Se puede realizar un **nuevo entrenamiento tomando como base una zona seleccionada previamente**. Mediante esta opción de entrenamiento se genera una nueva muestra a partir de los datos originales asociados a las neuronas seleccionadas por el usuario. Así, el usuario puede elegir una zona sobre la que desee obtener mayor información, a modo de ampliación o *zoom*, y obtener un nuevo cálculo *SOM* con los datos asociados a ésta, dando la posibilidad de ver porciones de *SOM* con diferentes niveles de granularidad. Esta capacidad será de gran utilidad para los modelos presentados en los apartados 4.3 y 4.4.

4.1.1. Operativa del reentrenamiento parcial

El proceso, cuyo esquema se muestra en la figura 4.1, se describe como:

1. El usuario selecciona una zona de las capas de la que desea extraer un mayor nivel de detalle (se recuerda la existencia de distintos modos de selección), y acto seguido lanza la *orden de entrenamiento.*

2. La aplicación extrae, de la muestra original, aquellos datos cuyo mapeo se corresponde con los nodos seleccionados.

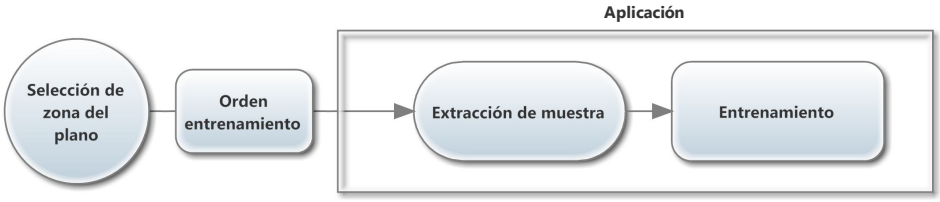


Figura 4.1: Entrenamiento a partir de zona de capa.

3. Se lanza el proceso de entrenamiento, de manera similar al caso original, en el que la muestra se encuentra en un fichero.

Los resultados se almacenan junto al *SOM* original, y el usuario puede acceder al nuevo *SOM* calculado mediante un submenú que aparece como extensión de la entrada *SOM* original.

4.1.2. Ejemplo de reentrenamiento parcial

A continuación se proporciona un ejemplo de reentrenamiento parcial. Para el mismo se ha empleado la misma base de datos que en el ejemplo del apartado 3.3.4, y el *SOM* de la figura 3.5.

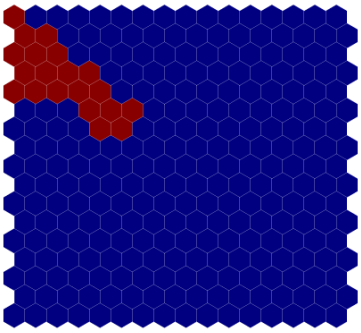


Figura 4.2: Zona seleccionada para reentrenamiento parcial.

Como ejemplo, se pretende indagar con más detalle el colectivo de los ciudadanos que trabajan más horas. Este colectivo tiene relativa importancia (2.280 elementos de la muestra). Se decide realizar un nuevo entrenamiento con las muestras correspondientes a las neuronas representadas en la zona roja de *la figura 4.2*, **para lo que se emplea el mecanismo de selección a medida de la aplicación**.

El resultado de este nuevo entrenamiento se muestra en la figura 4.3. Gracias a este nuevo entrenamiento, se pueden extraer conclusiones dentro del contexto de este subgrupo, en el que el rango de horas de trabajo semanales oscila entre 58 y 89 horas. Se puede observar:

- Dentro de este grupo, las personas con mayor nivel educativo son las que trabajan menos horas, con tipos de ocupaciones muy definidos, y el grupo de ingresos altos coincide mayoritariamente con éstos.

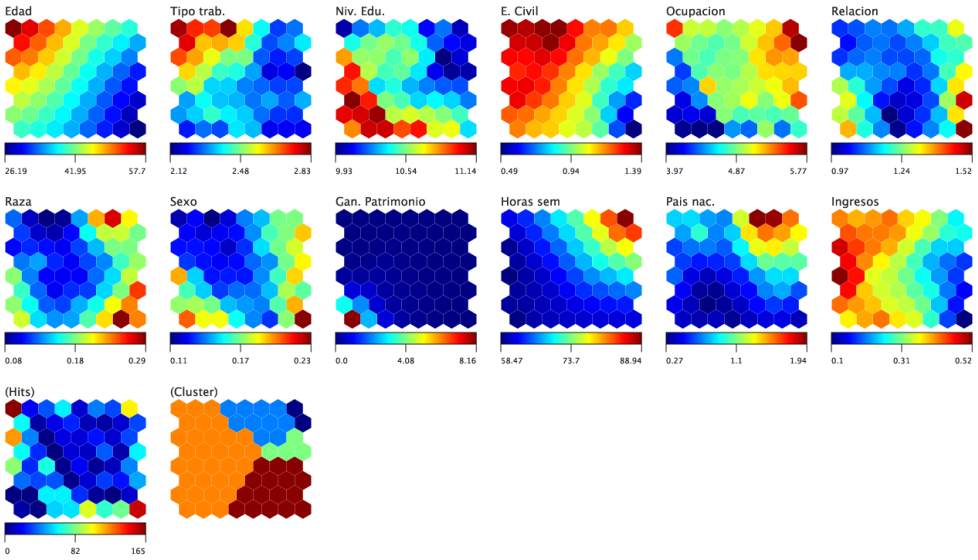


Figura 4.3: Nuevo *SOM* generado a partir de una selección de neuronas de otro SOM.

- El país de nacimiento está muy relacionado con la cantidad de horas trabajadas.
- Las personas con alta edad trabajan en trabajos muy determinados, son hombres de una raza determinada y con una relación familiar específica.

El detalle de cuáles son en concreto los valores determinados de las variables de estudio cuantitativas, como *ocupación*, *relación familiar*, *raza* o *país de nacimiento* **se obtendría con otro tipo de codificación de los datos de entrada (empleando variables dicotómicas)** o incluso con otro tipo de estudios. Esta visualización indica que existen correspondencias, aunque no especifica los valores concretos de las variables de estudio. Se proporciona ejemplo con algunas visualizaciones para percibir el interés de la presentación, notando que un estudio en profundidad proporcionaría una información más completa.

Al igual que, como ejemplo, ha decidido extraerse más información sobre un colectivo en concreto - aquél al que pertenecen los individuos que trabajan más horas -, **podrían estudiarse con mayor detenimiento otros subgrupos discriminados por los valores de los parámetros**, como por ejemplo el de los ciudadanos con más edad o los que presentan mayores pérdidas patrimoniales. Sucesivos estudios de los subgrupos de interés arrojarían información adicional.

El ejemplo presentado sirve como **muestra de cómo pueden realizarse consultas posteriores, a partir de un *SOM* calculado, para ampliar detalle** de zonas o características determinadas descubriendo nueva información de interés no

perceptible en el *SOM* original. **En el apartado 4.3 se presentará un algoritmo que contiene esta utilidad ejecutada de manera automática sobre neuronas que presenten una resolución insuficiente**.

4.2. Modelo de tamaño variable *GSOM*

Según la bibliografía (Kim y To, 2013), y a partir de la propia deducción de los procedimientos heurísticos presentados en los apartados 1.3.4 y 1.3.5, **no existe una teoría formal de descripción de funcionamiento de las dinámicas de aprendizaje SOM que permita generar topologías correctas**, siendo la formulación de ésta un objetivo actual dentro del estudio teórico de los *SOM*. Este hecho no ha impedido que los *SOM*, como se expone en el apartado 1.4, se empleen como sistema de visualización de sistemas complejos de alta dimensionalidad.

Una limitación importante del modelo estático es la determinación de su estructura y tamaño. De hecho, las aplicaciones prácticas de *SOM* requieren de un modelo de prueba y error consistente en la **generación de numerosos mapas basados en criterios subjetivos para determinar de entre éstos el mapa con mayor calidad**, con lo que ello conlleva en costes de ejecución. Este tipo de experimentos cobra su sentido en los estudios empíricos (Tan y George, 2004) que demuestran **que para un mismo conjunto de datos, se obtienen diferentes dinámicas de entrenamiento según los parámetros que se apliquen**, teniendo en cuenta que estos parámetros habitualmente se decrementan a lo largo del tiempo (según expuesto en apartados 1.3.3 y 1.3.4). De las características de los datos depende el que unos parámetros y dimensiones determinadas lleven a un buen *SOM*, y no puedan ser decididos de una manera genérica e independientemente de los datos.

El modelo de tamaño variable (*Growing Self Organizing Map*, GSOM) es una extensión del *SOM* que ayuda a superar esta limitación. Con su estructura dinámica, permite añadir y según topología, eliminar neuronas de la red durante el entrenamiento en las zonas de mayor interés según criterios de densidad y exactitud, que se expondrán a continuación.

4.2.1. Algoritmo y características del *GSOM*

El proceso se muestra en la figura 4.4, y constituye un algoritmo iterativo, que se compone de los siguientes pasos:

1. **Bucle de entrenamiento.** Tanto en las modalidades *batch* como *online*, se realiza *entrenamiento SOM* tal y como se describió en el apartado 1.3.2 empleando una porción de las muestras disponibles para el entrenamiento.
2. **Control de finalización.** En caso de que se haya procesado el total de muestras estipuladas, el algoritmo finaliza. En caso contrario, el algoritmo continúa.

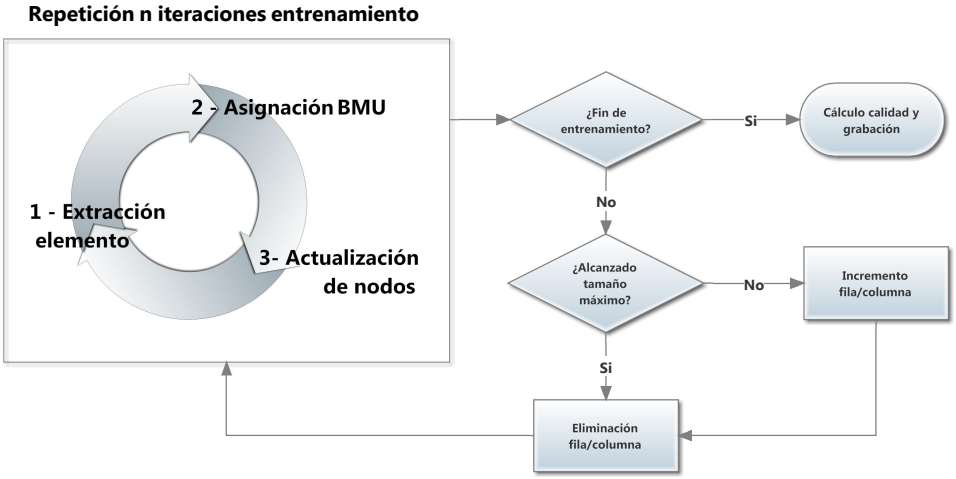


Figura 4.4: Modelo de entrenamiento de SOM de tamaño variable.

3. **Crecimiento de las capas.** En caso de que no se haya alcanzado el tamaño máximo (en el presente estudio se ha tomado como referencia el doble del tamaño calculado automáticamente a partir de la muestra, ver apartado 1.3.5)), se incrementa una fila o una columna. Se selecciona la unidad que tenga peor error de cuantización, y se introduce una fila o columna, según corresponda, entre ésta y la unidad colindante con la que presente mayor disimilitud.

El criterio de crecimiento está inspirado en el algoritmo *batch* de Yu y Alahakoon (Yu y Alahakoon, 2006). **Se añaden nuevas neuronas durante el entrenamiento en búsqueda de mayor precisión en la descripción de la distribución con una menor cantidad de neuronas**. Un criterio diferente de crecimiento es el propuesto por Kim y To (Kim y To, 2013) en el que la inclusión de filas y columnas se realiza en las zonas que contienen mayor densidad de *BMU's*, proporcionando mayor resolución en las zonas donde la distribución de la muestra tiene mayor densidad, de manera independiente a la calidad de la cuantización en estas zonas. Se deduce que ambos criterios alcanzarán resultados similares en los conjuntos de datos para los cuales las zonas del espacio de la muestra con mayor número de elementos sean más difíciles de explicar con pocas neuronas. Estas técnicas facilitan la consecución de una distribución más uniforme de BMU's, con la consiguiente mejora en la lectura e interpretación de datos (Araujo y Rego, 2013; Kim y To, 2013; Qiang *et al.*, 2010). Otras ventajas de los *SOM* de tamaño variable son:

- **Procesado de altos volúmenes de datos** (Yu y Alahakoon, 2006) destacando la agrupación de texto proveniente de Internet (Nathawitharana *et al.*, 2013;

Matharage *et al.*, 2013; Gunasinghe *et al.*, 2012).

- Posibilidad de, una vez realizado el entrenamiento, **efectuar un aprendizaje adicional válido aún en el caso de que las distribuciones de los datos de entrada varíen** (Kuremoto *et al.*, 2010) permitiendo detectar cambios en series de datos a lo largo del tiempo (Daswin Pasantha De Silva y Alahakoon, 2006; Kang, 2012). El parámetro de aprendizaje α es constante en el modelo de tamaño variable, por lo que los SOM de este tipo varían del mismo modo según la entrada de datos, durante todo el proceso de aprendizaje. Gracias a esta característica, **se pueden detectar cambios incluso someros en la distribución de datos de entrada**. No obstante, existen autores (Alahakoon *et al.*, 2000) que emplean un parámetro α variable, según aplicación.

Como resumen, los *SOM* de tamaño variable tienen la **habilidad de añadir nuevas neuronas para conseguir la precisión suficiente en determinadas zonas del espacio**. Por otro lado, se observan diversas limitaciones el modelo en estudio, de tipo matriz:

- **Sobre la selección de las dimensiones iniciales:** Una matriz inicial demasiado reducida requerirá una cantidad mínima de muestras y/o ajustes de crecimiento para aumentar lo suficiente como para que se muestre información de manera útil. Por el contrario, una matriz demasiado grande contendrá numerosas neuronas poco representativas (es decir, que contendrán pocas o ninguna muestra mapeada), por lo que se pierde el sentido de crecimiento en las zonas más necesitadas.
- **Sobre el crecimiento:** Para mantener la estructura de tipo matriz, se añaden filas o columnas enteras, por lo que con alta probabilidad se añaden nuevas neuronas en zonas que no requieren mayor precisión o son poco representativas, lo que supone la aparición de zonas de poco interés en el mapa, además de incurrir en mayores costes de cálculo.
- **Sobre la eliminación de neuronas inservibles:** De manera análoga al caso anterior, en el caso de topologías de tipo matriz, si deciden eliminarse las neuronas menos representativas, también debería eliminarse la fila o columna continente, lo que llevaría a riesgos de eliminar otras neuronas que sí resultan interesantes.

En el apartado 5.2.3 se proponen, como ampliaciones, variantes para soslayar dichas limitaciones. En el apartado 4.2.2 se presenta un ejemplo de entrenamiento *GSOM*, destacando las diferencias obtenidas con respecto al ejemplo *SOM* obtenido en el apartado 4.1.2.

4.2.2. Ejemplo de entrenamiento *GSOM*

Se ha realizado un entrenamiento *GSOM* empleando la misma muestra y parámetros de entrenamiento que el entrenamiento mostrado en el apartado 4.1.2.

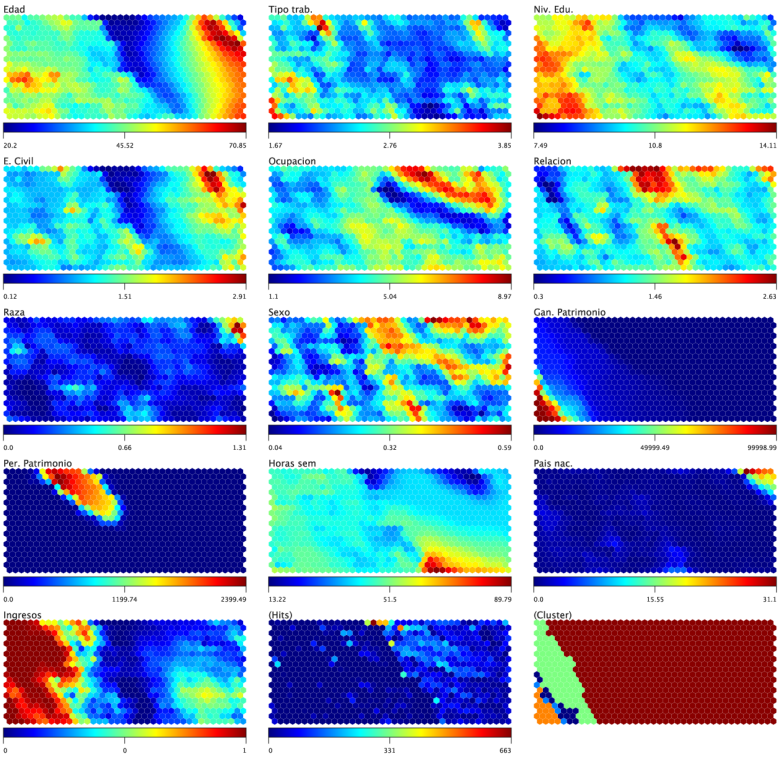


Figura 4.5: Ejemplo de GSOM, ingresos económicos USA 1994.

La figura 4.5 contiene un ejemplo de entrenamiento de *GSOM*, sobre la misma muestra empleada para el apartado 4.1.2. Las diferencias notables con respecto al entrenamiento original vienen dadas por una mayor definición en los dibujos de aquellas capas cuya información es más compleja de discernir, como *estado civil*, *tipo de trabajo* o *relación familiar*. En este caso, especialmente la capa *sexo* presenta zonas diferenciadas a modo de *islas* que permiten separar diferentes segmentos en la información general en base al sexo de los individuos de la muestra.

4.3. Modelo creciente y jerárquico *GHSOM*

Combinando las ampliaciones presentadas (reentrenamiento parcial y modelo creciente), se ha implementado el modelo de *SOM Creciente Jerárquico* (*Growing Hierarquical SOM)*, en adelante *GHSOM* (Rauber *et al.*, 2002; Dittenbach *et al.*, 2005).

El propósito del entrenamiento *GHSOM* es doble: además de responder, al igual que *GSOM*, **a las dificultades en el dimensionamiento de las capas**, **establece de manera automática relaciones jerárquicas entre los datos**. La identificación de estas relaciones es una tarea relevante dentro de la *DM*, que no puede resolverse convenientemente mediante SOM.

4.3.1. Algoritmo y características del *GHSOM*

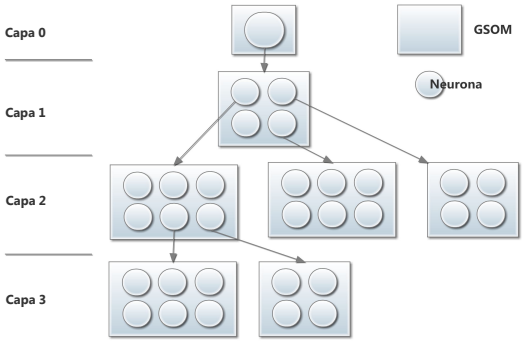


Figura 4.6: Estructura ejemplo de jerarquía GHSOM.

El modelo *GHSOM* (Dittenbach *et al.*, 2005) se compone de **una estructura jerárquica**, donde varias redes *GSOM* constituyen diversas capas de la jerarquía. **El tamaño** de dichas *GSOM* y la cantidad de éstas que constituye cada capa, **así como la profundidad** de la jerarquía, **se determina durante el proceso constructivo** según las dimensiones y distribución de la muestra de aprendizaje. **Partiendo de un mapa de un primer nivel**, compuesto de una única neurona, **cada *GSOM* representa una submuestra** a un nivel de detalle específico. **Después de alcanzar un determinado nivel de precisión en la representación de cada *GSOM*, se analizan las neuronas** para seleccionar aquellas que incumplen individualmente un criterio de precisión, **para expandirlas** mediante respectivos *GSOM*. Por tanto, las neuronas que representan un conjunto homogéneo de datos no requerirán ser expandidas en nuevas capas. El proceso se repite iterativamente y termina cuando todas las neuronas de la capa inferior cumplen los criterios de calidad, o cuando es imposible expandirlas debido a los pocos datos que representan. **El *GHSOM* resultante refleja mediante su arquitectura la estructura jerárquica implícita en los datos**, empleando más espacio para la representación de áreas no homogéneas en el espacio de la muestra.

La figura 4.6 muestra un ejemplo de estructura *GHSOM*. **La capa 0 se compone de una única neurona**, que se emplea para calcular un error de precisión R de referencia. La capa siguiente se calculará para mejorar la precisión R en una proporción τ. **La muestra de entrenamiento de cada mapa es el subconjunto de datos que ha sido mapeado en la neurona de la capa anterior** de la que procede. Según el ejemplo, 3 unidades de la capa 1 han sido expandidas en nuevos

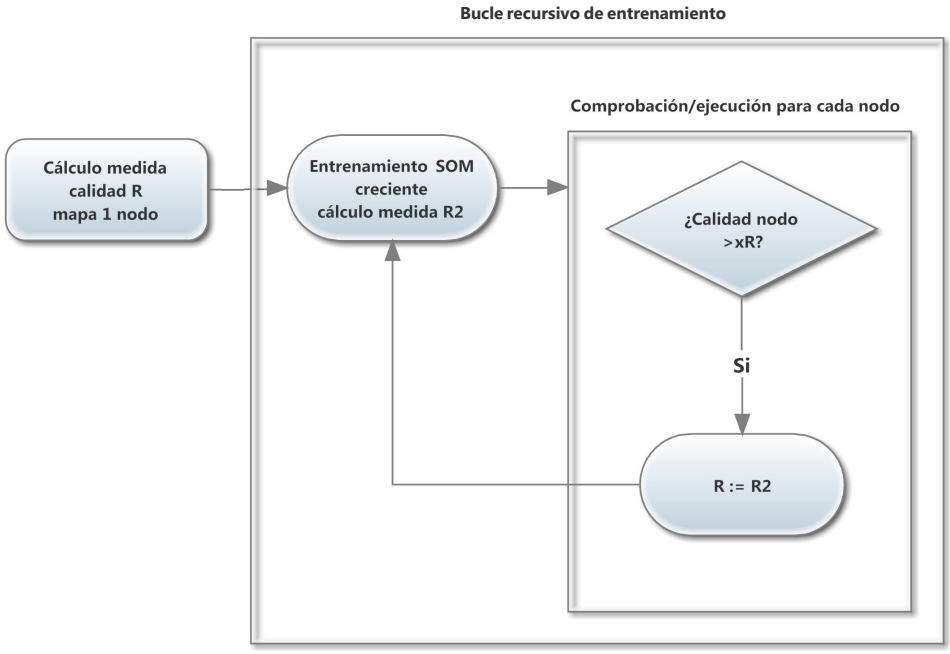


Figura 4.7: Entrenamiento de GHSOM.

GHSOM en la capa 2. La capa 3 provendría de 2 neuronas de un mapa de la capa 2. Cada capa proporcionaría mayor detalle sobre los datos mapeados en la neurona de la que procede. Cabe notar que **los mapas pueden tener diferentes estructuras**, debido a la distribución de los datos con los que se entrenaron.

El esquema de entrenamiento se presenta en la figura 4.7, y se compone de los siguientes pasos:

- **Cálculo de referencia.** Se entrena un mapa de una única neurona con toda la muestra, obteniendo de éste un error de cuantización R que servirá posteriormente como referencia de parada del algoritmo. Este mapa comprende la capa 0.

- **Bucle iterativo de entrenamiento.** El bucle iterativo comprende la generación de la estructura jerárquica en sí, mediante las acciones:

    1. **Entrenamiento de la red con las muestras cuyo nodo seleccionado en el entrenamiento anterior es el BMU**, también llamadas *muestras*

mapeadas por el nodo. Obtención del error de cuantización R_2. Esta red es de tipo *GSOM*, con la siguiente particularidad: partiendo de una red 2x2, se entrena hasta que cumple error de cuantización inferior a τR, siendo τ un parámetro fraccionario. En el caso de la capa 1, el nodo seleccionado es el única neurona que conforma el mapa.

2. **Para cada nodo n_i, se comprueba si su error de cuantización R_i es inferior a $\tau_2 R$**, siendo τ_2 un parámetro fraccionario. Si es así, no se realiza ninguna acción, considerándose ese nodo como terminal. **En caso contrario, se selecciona nueva referencia $R := R_2$ y se repite el bucle recursivo de entrenamiento para este nodo n_i.**

Como se observa, los parámetros τ y τ_2 influyen sobre amplitud y profundidad de la estructura jerárquica generada:

Valores bajos de τ suponen exigencia de mayor calidad en las subredes *GSOM* generadas, por lo que se requerirá mayor entrenamiento y crecimiento de éstas. Valores altos de τ producirán el efecto contrario.

El parámetro τ_2 influye sobre la profundidad de la estructura, dado que influye en la decisión de expandir una neurona en una nueva subred en la siguiente capa. Un valor alto de τ_2 resulta en mayor clasificación de neuronas como terminales, acotando el crecimiento de la estructura en profundidad. **Valores bajos de τ_2 resultan en mayor profundidad de capas**.

Como resultado, **se obtiene una estructura jerárquica navegable de *GSOM*, donde cada neurona con baja precisión sirve de enlace a un nuevo *GSOM*** entrenado con los datos que ésta tiene mapeados. El entrenamiento es específico de *GHSOM*; cada capa deja de crecer en el momento en que alcanza un error de cuantización de referencia. Este tipo de entrenamiento diferencia notablemente la estructura de cada capa con respecto a las SOM y *GSOM* habituales, dado que las capas del *GHSOM* en general tienen pocas neuronas (tómese como referencia el ejemplo de la figura 4.6). **El usuario, sirviéndose de las capas superiores como clasificadores, navega hasta las capas inferiores para extraer de éstas las conclusiones relativas a los datos clasificados por las capas superiores**.

Dado que se dispone de un mecanismo de realización de entrenamientos múltiples y selección, se emplea éste para la implementación del *GHSOM*, seleccionando cada una de las capas, según sus índices de error, de entre una serie de capas calculadas, según los parámetros introducidos por el usuario.

El modelo de entrenamiento *GHSOM* intensifica las ventajas propias de los modelos de tamaño variable: permite dedicar menos neuronas a las zonas del espacio de entrada que no lo precisan, proporcionando mayor resolución a las zonas que así lo requieren por complejidad y/o por volumen de datos. De manera adicional, el *GHSOM* permite al usuario navegar de manera ordenada entre diferentes capas, en una estructura ordenada de tal modo que cada *GSOM* de cada capa explica una zona del espacio de entrada. Sin embargo, existe una serie de inconvenientes en el campo de la visualización que son abordados mediante el modelo propuesto en la sección 4.4.

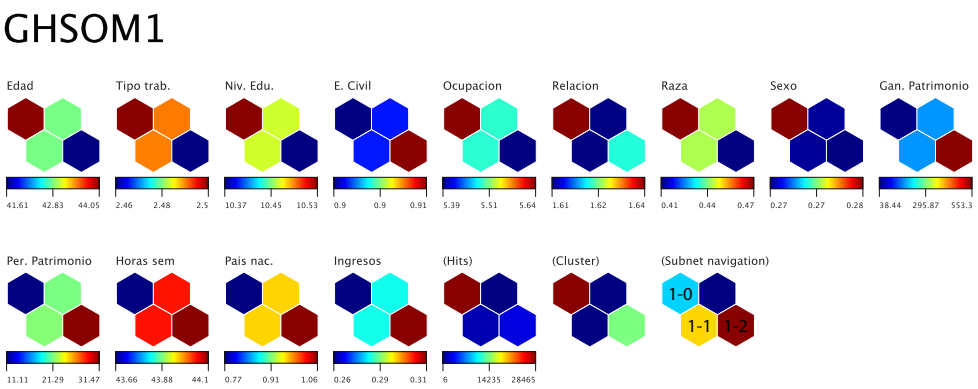


Figura 4.8: Capa inicial GHSOM.

4.3.2. Ejemplo de entrenamiento de modelo de *GHSOM*

Empleando los mismos parámetros de entrenamiento, y con valores $\tau = 0{,}001$ y $\tau_2 = 0{,}1$, se ha generado un *GHSOM*, cuya capa inicial se presenta en la figura 4.8. El mapa *(Subnet navigation)* permite seleccionar diferentes neuronas y navegar hacia las subredes generadas a partir de éstas, si las hubiere.

La capa inicial (capa 0) del *GHSOM*, titulada *GHSOM1* y presentada en la figura 4.8, **realiza una primera clasificación entre individuos**. Cada una de las cuatro neuronas que la conforman representa un segmento de los datos de entrada. Existen tres neuronas que incumplen el criterio de calidad expuesto en el apartado 4.3 (neuronas azul claro, rojo y amarillo del mapa *(Subnet navigation)*), por lo que a partir de los datos mapeados en éstas **se han generado respectivas subredes, que pertenecen a la capa 1**. Estas subredes desarrollan la presentación de los datos contenidos en la capa 0 y se muestran en las figuras 4.9, 4.10 y 4.11.

Sucesivamente y del modo descrito, **a partir de las subredes de la capa 1 se generan nuevas subredes**, estas últimas pertenecientes a la capa 2, presentadas en las figuras 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19 y 4.20.

Se puede observar cómo *cada una de las subredes representa diferentes rangos en las variables*, lo que indica que trata un segmento distinto de los datos de entrada.

Por resumir y dado que el objetivo no es un análisis minucioso de los datos del ejemplo, no se muestran más capas, aunque el experimento generó nuevas capas.

Si bien cada mapa por separado explica y clasifica un segmento de los datos, per-

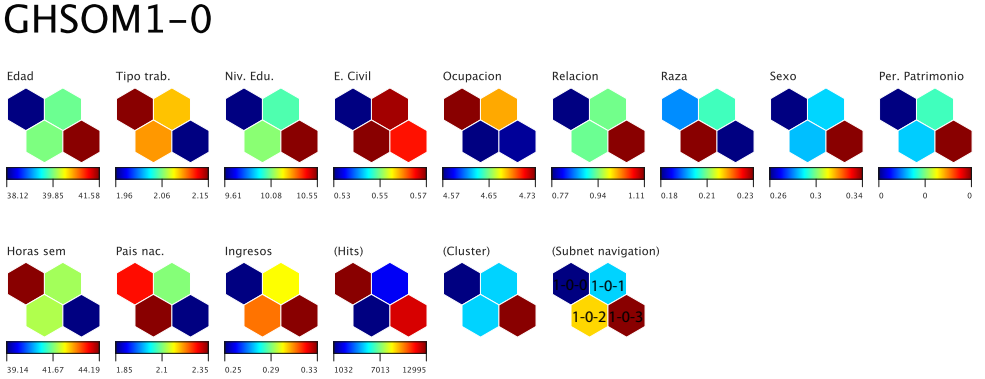


Figura 4.9: Capa 1, red GHSOM1-0.

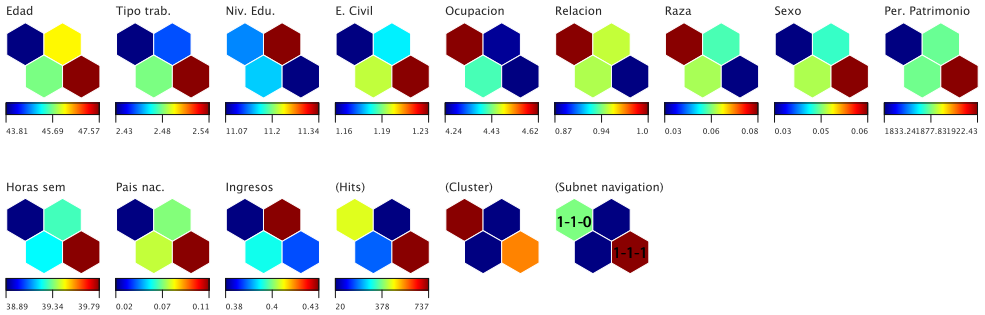


Figura 4.10: Capa 1, red GHSOM1-1.

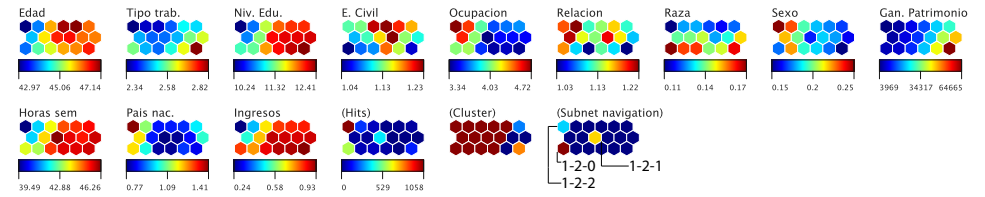


Figura 4.11: Capa 1, red GHSOM1-2.

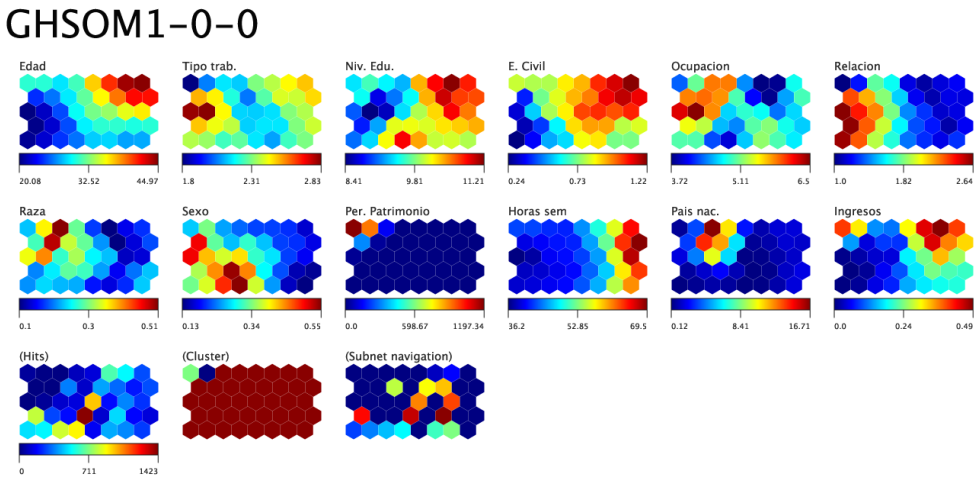


Figura 4.12: Capa 2, red GHSOM1-0-0.

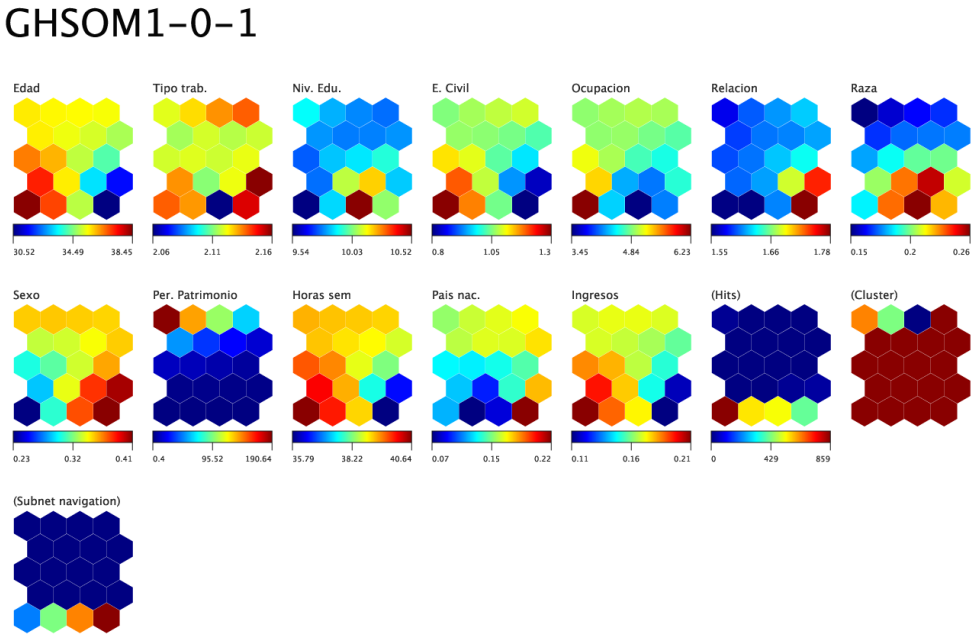


Figura 4.13: Capa 2, red GHSOM1-0-1.

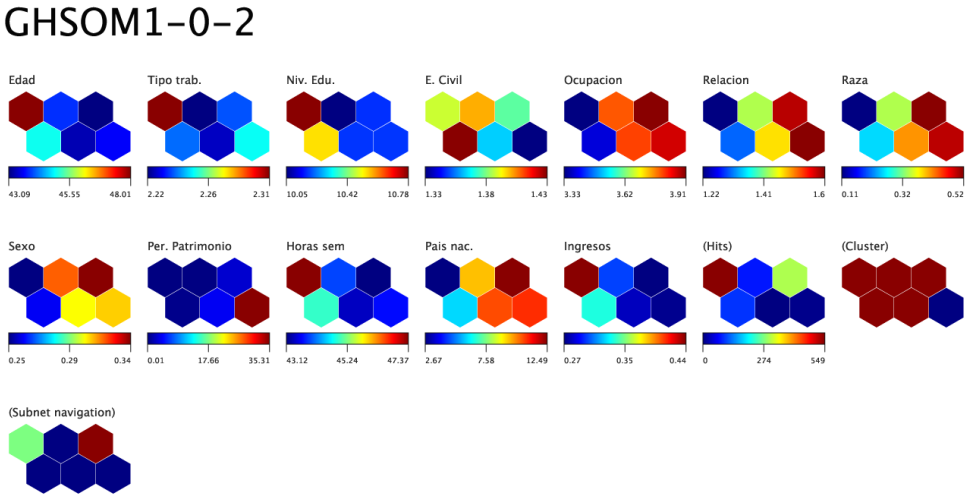


Figura 4.14: Capa 2, red GHSOM1-0-2.

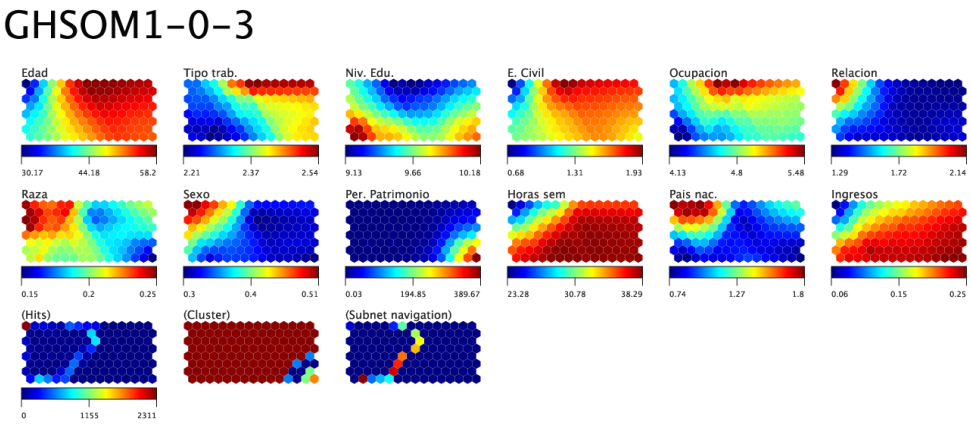


Figura 4.15: Capa 2, red GHSOM1-0-3.

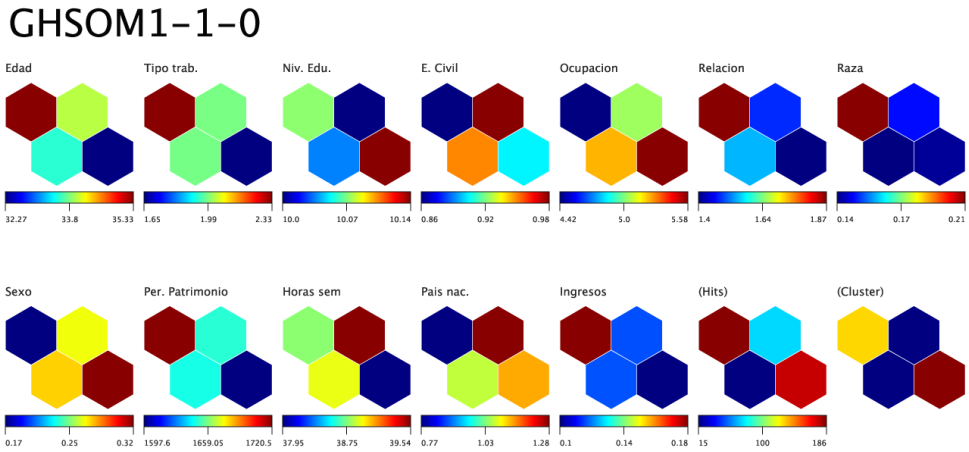


Figura 4.16: Capa 2, red GHSOM1-1-0.

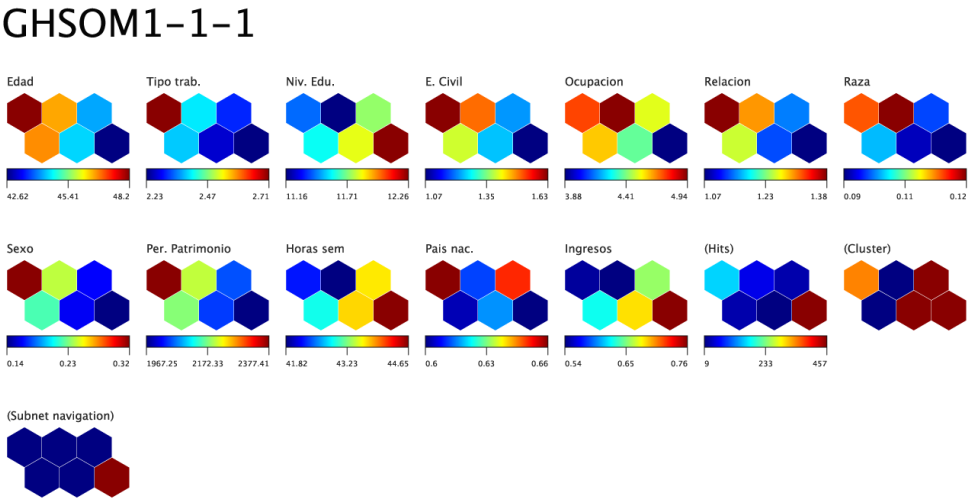


Figura 4.17: Capa 2, red GHSOM1-1-1.

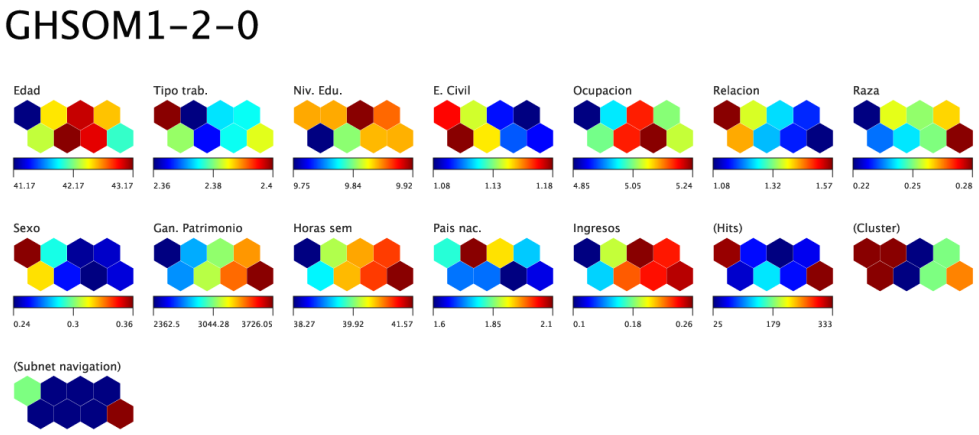


Figura 4.18: Capa 2, red GHSOM1-2-0.

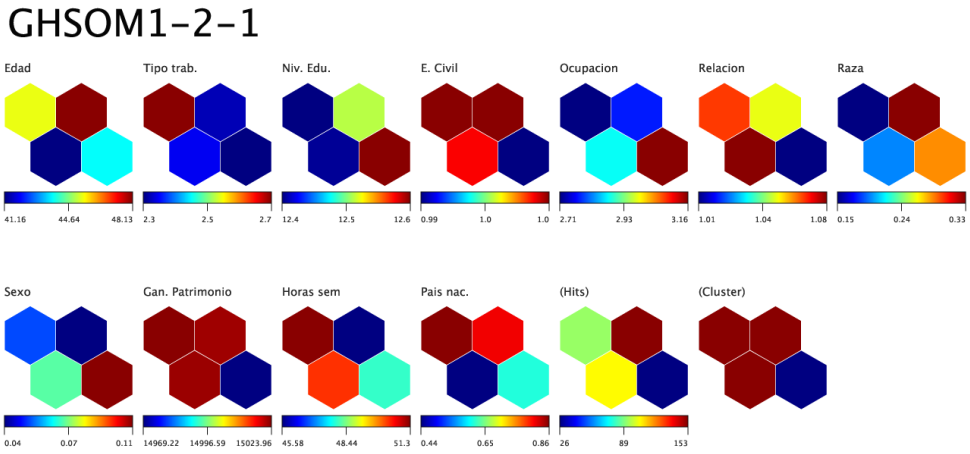


Figura 4.19: Capa 2, red GHSOM1-2-1.

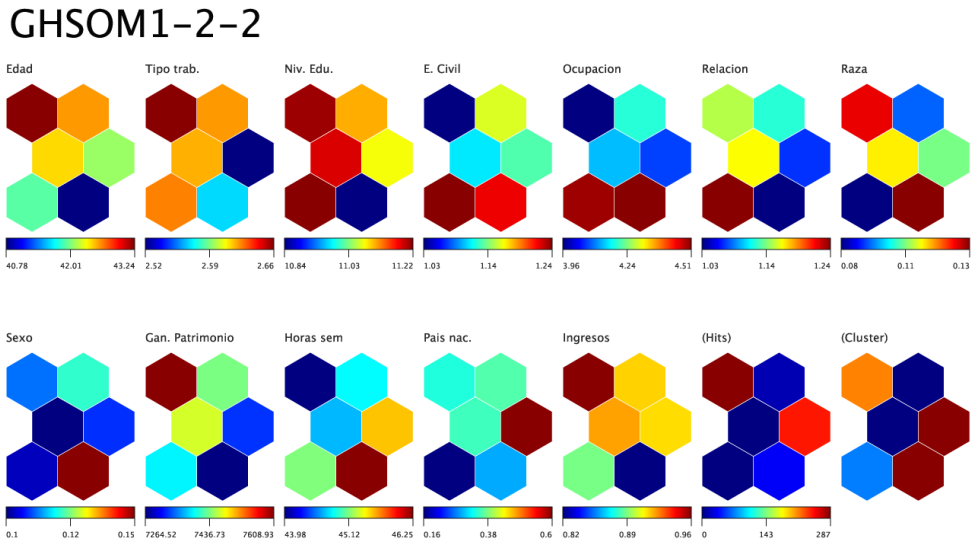


Figura 4.20: Capa 2, red GHSOM1-2-2.

mitiendo navegar por segmentos cada vez más precisos, cabe reseñar la complejidad creciente en la interpretación del contexto y jerarquía de las nuevas capas. Particularmente, las redes con más de cuatro neuronas, como es el caso de *GHSOM1-0-0* (figura 4.12) generan multitud de subredes a partir de neuronas adyacentes, lo que puede causar que los segmentos no estén suficientemente definidos. Por ello, **se propone un nuevo modelo, variante de *GHSOM*, en el apartado 4.4**.

4.4. *Growing Cluster-Hierarchical SOM*. Un modelo novel de aplicación de *GHSOM* para la visualización de datos

El *GHSOM* enriquece con sus características la *VDE*, aunque se pueden hacer las siguientes observaciones éste:

a) **Las capas se componen habitualmente de pocas neuronas**, que sirven como clasificadores para navegar hacia capas más profundas. Esta característica **dificulta obtener una visión de conjunto**, debiendo para ello consultar las diferentes capas por separado.

b) **Las nuevas capas se obtienen a partir de neuronas simples**, desechando las relaciones entre éstas en lo que refiere a la generación de capas posteriores.

Este hecho tiene su sentido teniendo en cuenta el punto a).

Estas características hacen del ***GHSOM* un modelo apropiado para la clasificación de datos de texto** (Rauber *et al.*, 2002; Dittenbach *et al.*, 2005; Herbert y Yao, 2007; Yang *et al.*, 2007), para detección de elementos anómalos dentro de una muestra (Ge *et al.*, 2003; Salem y Buehler, 2013) o incluso para compresión de imágenes mediante cuantización del color (Palomo y Domínguez, 2011), aunque **cabe realizar modificaciones sobre éste para poder explotar sus ventajas para la visualización de datos numéricos** dentro del contexto del presente trabajo. Por tanto, para la visualización de datos numéricos, se propone un modelo basado en *GHSOM* donde:

- **Las capas**, además de servir para clasificar posteriores consultas, **permiten obtener una visión global de la distribución de datos** de entrada. Para ello no se empleará el criterio de finalización de crecimiento de *GHSOM*, que puede resultar prematuro para el objeto de estudio.
- **La generación de nuevas capas**, en lugar de realizarse a partir de neuronas simples, **se realiza a partir de grupos de neuronas con valores próximos**, de tal modo que se facilita la comprensión visual de los elementos de capas posteriores.

Dicho modelo se describe a continuación.

4.4.1. Algoritmo y características del *GCHSOM*

El *Growing Cluster-Hierarchical SOM* (*GCHSOM*), al igual que el *GHSOM*, clasifica los datos para especializarlos en capas posteriores. Sin embargo, las capas superiores siguen permitiendo ver toda la información en su conjunto.

De manera intuitiva, el proceso se puede describir como el automatismo de un entrenamiento *GSOM* tras el cual el usuario fuera seleccionando los clusters de la red que tuvieran peor calidad y realizando entrenamientos parciales, de manera iterativa hasta alcanzar el nivel de detalle suficiente para explicar los datos. **El proceso de entrenamiento *GCHSOM* guarda mucha similitud con el entrenamiento *GHSOM*. Se compone de los pasos**:

- **Cálculo de la capa 0.** Se entrena un mapa *GSOM* con toda la muestra. A diferencia del *GHSOM*, el mapa no tiene un criterio de parada prematuro, por lo que se asemeja a un *GSOM* del tipo expuesto en el apartado 4.2.
- **Se generan i agrupaciones nc_i del mapa**. Cada agrupación contiene neuronas con valores próximos, distanciándose en lo posible de las neuronas del resto de grupos. **Se generarán nuevos mapas a partir de los grupos** nc_i si se cumple $R_i > \tau R$ siendo R_i el error medio de cuantización del grupo i, R el error medio de cuantización del mapa y τ un parámetro fraccionario.

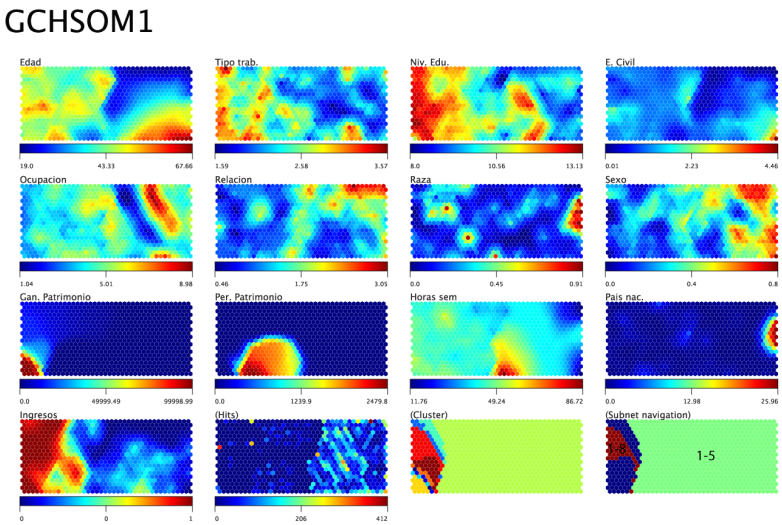


Figura 4.21: Capa inicial GCHSOM.

- **Bucle iterativo de entrenamiento**. El bucle iterativo comprende la generación de la estructura jerárquica en sí. Para cada subred se repite el mismo proceso que el realizado en la capa 0, generando una capa posterior.

A diferencia del *GHSOM*, donde las capas superiores sirven de clasificadores y las capas inferiores presentan el detalle, en el *GCHSOM* las capas superiores, además de clasificar, muestran información global. Las subredes de las capas inferiores disminuyen su tamaño conforme lo hace el volumen de datos de entrada con los que se generan.

A continuación se ofrece un ejemplo de entrenamiento *GCHSOM* donde se pueden apreciar sus características.

4.4.2. Ejemplo de entrenamiento de modelo de *GCHSOM*

Nuevamente se emplea la misma muestra de ejemplo, que sirve para la realización de un entrenamiento *GCHSOM*. La capa inicial aparece en la figura 4.21. A diferencia del entrenamiento *GHSOM*, aparece una visualización general de los datos, sin embargo, **existe la posibilidad de navegar a través de las subredes generadas a partir de aquellos clusters en los cuales no se ha mejorado el error de cuantización con respecto a la referencia**.

Se observa como a lo largo de las capas, dada la distribución asimétrica de la muestra del ejemplo, el grueso de la muestra va definiéndose al quedar en redes aparte los

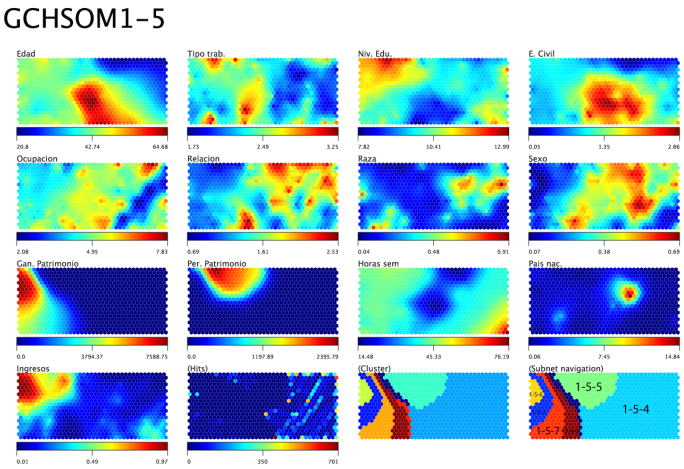


Figura 4.22: Capa 1, GCHSOM1-5.

grupos minoritarios (figuras 4.22 y 4.25). Finalmente, los segmentos definidos por los grupos de neuronas similares, se representan en subredes de unas pocas neuronas.

De manera análoga al *GHSOM*, **las primeras capas permiten realizar una clasificación inicial de los datos**, aunque en este caso, **cada segmento de datos**, en lugar de ser representado por una neurona, **es representado por un cluster**; así se toman en consideración las similitudes entre neuronas para descender en las jerarquías. Como resultado, **el usuario podrá seleccionar segmentos de datos conociendo desde el principio la estructura general de los mismos**.

Según el ejemplo de entrenamiento *GCHSOM* realizado, la figura 4.21 contiene la capa inicial. La matriz *"Subnet navigation"* contiene las referencias a las capas descendientes (4.22 y 4.23). Mediante el mismo procedimiento se puede descender a lo largo de la jerarquía generada, observando:

En contraposición al *GHSOM*, donde las primeras capas contienen pocas neuronas, para ir incrementando en cantidad conforme se profundiza, las primeras capas del *GCHSOM* son más completas, proporcionando mayor información de contexto, cualidad relevante para la visualización de datos. Conforme se desciende en la jerarquía, las capas van teniendo menos neuronas que sus antecesoras, al realizarse sucesivos entrenamientos con muestras de menor volumen. De este modo, se proporciona una importancia relativa - según la cantidad de elementos que contengan - a los segmentos de datos representados en las capas de mayor profundidad.

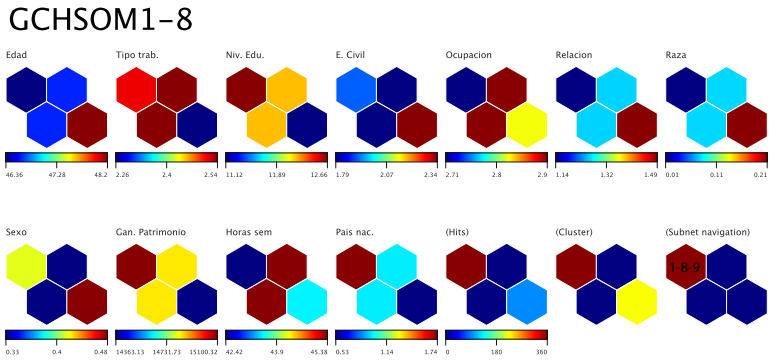


Figura 4.23: Capa 1, GCHSOM1-8.

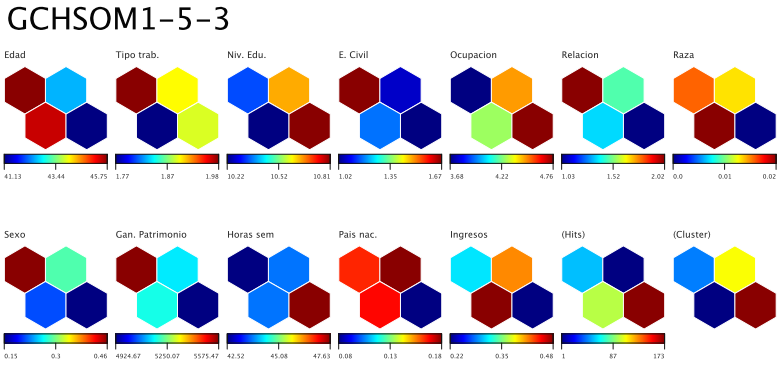


Figura 4.24: Capa 2, GCHSOM1-5-3.

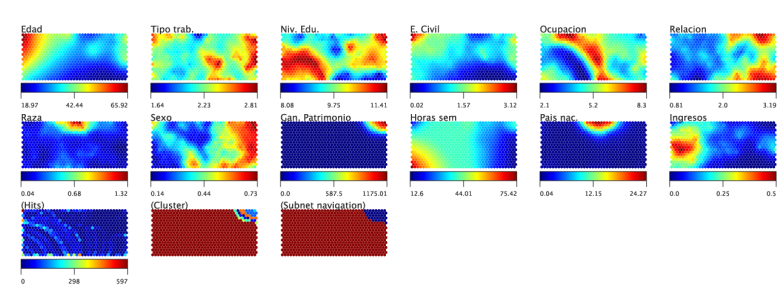


Figura 4.25: Capa 2, GCHSOM1-5-4.

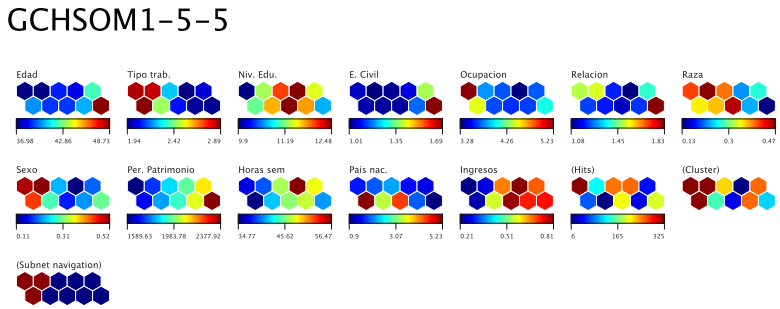


Figura 4.26: Capa 2, GCHSOM1-5-5.

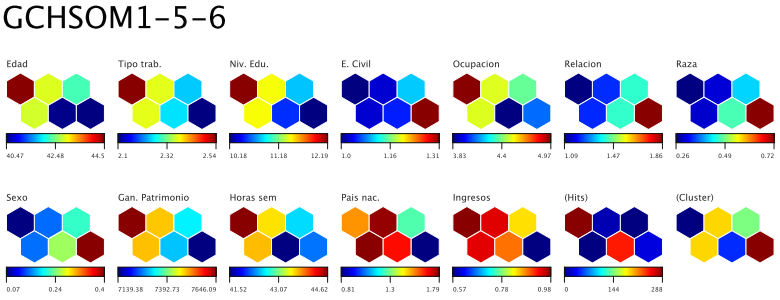


Figura 4.27: Capa 2, GCHSOM1-5-6.

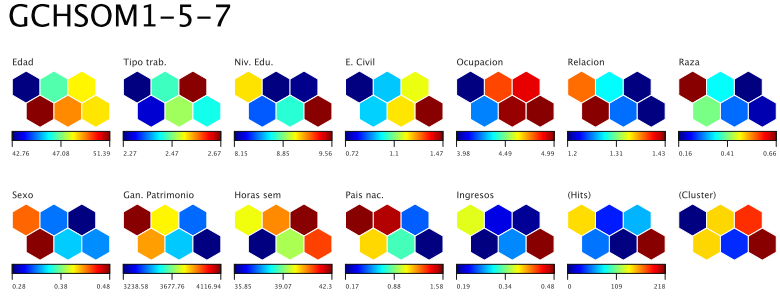


Figura 4.28: Capa 2, GCHSOM1-5-7.

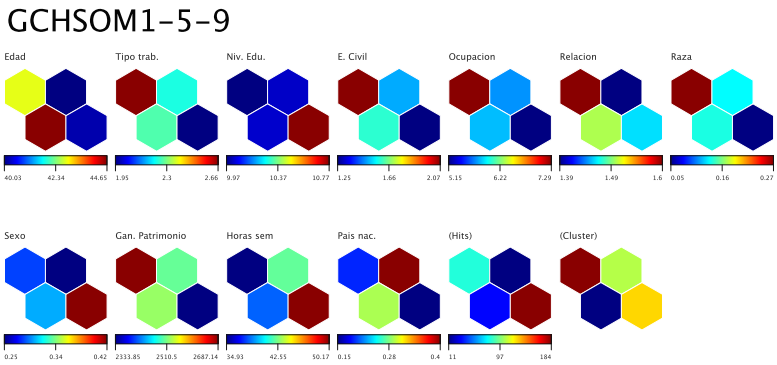


Figura 4.29: Capa 2, GCHSOM1-5-9.

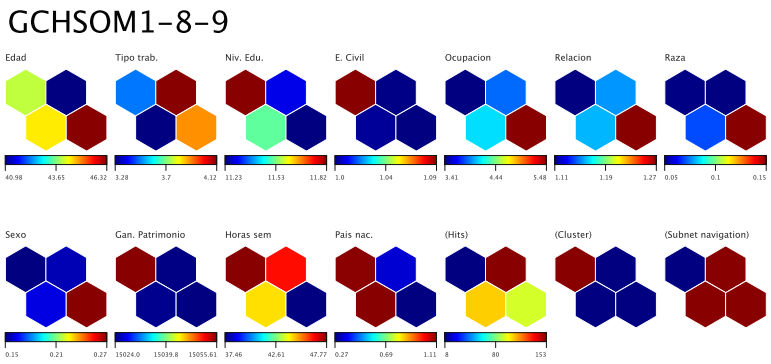


Figura 4.30: Capa 2, GCHSOM1-8-9.

Como cabe esperar, tanto las estructuras de *GHSOM* como de *GCHSOM* son más fáciles de seguir mediante el empleo de la aplicación informática objeto del presente trabajo.

Capítulo 5

Conclusiones y trabajos futuros

Un comienzo no desaparece nunca,
ni siquiera con un final.

Harry Mulisch

5.1. Conclusiones

La presente memoria, una vez expuestas las redes neuronales de tipo *Mapas Autoorganizativos* (SOM) como potentes herramientas para la visualización de complejas fuentes de datos, **describe la estructura y el funcionamiento del entorno de producción desarrollado** para la elaboración de entrenamientos de dicho tipo de redes, proporcionando ejemplos. Dicho *software* es la aportación principal de la presente Tesis Doctoral.

El nuevo sistema *software*, de nombre **LFS** (acrónimo de *Living for SOM*) [1] implementa un experimento que consiste en la realización, a partir de una fuente de datos, de **multitud de entrenamientos SOM realizados mediante combinaciones de valores de los parámetros** de entrada proporcionados por el usuario. De entre los resultados se seleccionarán las mejores redes en base a diferentes criterios de error.

Para la realización de dicho experimento *se han implementado optimizaciones que aceleran los cálculos* con respecto a una ejecución secuencial de los algoritmos de entrenamiento. Estas optimizaciones se resumen en:

- **Cálculos previos de estados iniciales y de kernels de entrenamiento comunes**, de tal modo que costosos cálculos que se realizan frecuentemente se convierten en consultas a tablas guardadas en memoria.

[1] http://www.livingforsom.com

- Utilización de computación paralela, que permite la realización de varios entrenamientos a la vez.

Estas optimizaciones han permitido realizar experimentos relativamente complejos incluso en equipos domésticos y hacen el sistema dimensionable a computación de altas prestaciones.

El software *LFS* se ha provisto de una interfaz que permite al usuario interactuar con los datos mediante diversas acciones que facilitan la visualización de los datos, fundamentalmente: seleccionar zonas y compararlas en diferentes mapas, elegir entre diferentes paletas de colores, ver datos resumen, hacer zoom, cambiar de sitio y ocultar mapas, realizar nuevos entrenamientos a partir de grupos de neuronas de mapas ya calculados, ejecutar agrupamiento automático que ayude a delimitar diferentes zonas y guardar en formato *pdf* los resultados.

Como ampliaciones, se hace revisión y **se exponen dos variantes de SOM**:

1. ***Growing SOM (GSOM)***, un modelo de SOM de tamaño variable que permite obtener mayor resolución en los espacios que por criterio así lo precisen.
2. ***Growing Hierarchical SOM (GHSOM)***. Variante del modelo *GSOM* que incorpora información jerárquica de los datos.

Estos modelos, que incorporan ventajas sobre el entrenamiento base, se implementan dentro de *LFS*.

Finalmente, **conocidas las ventajas del modelo *GHSOM*,** una vez **detectadas diversas limitaciones del mismo en el ámbito de visualización de datos numéricos, se propone una nueva variante de éste: El modelo *Growing Cluster Hierarchical SOM (GCHSOM)***, que incorpora las ventajas de mapas de tamaño variable y relaciones jerárquicas del *GHSOM*. Como diferencias fundamentales entre el modelo propuesto *GCHSOM* y *GHSOM*, **los mapas generados son mayores, lo que permite obtener visualizaciones globales de datos, y se calculan nuevos mapas a partir de grupos de neuronas similares, manteniendo la información de manera jerarquizada y facilitando comprensión y navegación**. Dicho algoritmo se implementa en *LFS* y se muestra un ejemplo de su utilización donde se puede apreciar su interés.

Así pues, **se dispone de una aplicación de *código abierto* que sirve de ayuda en entornos tecnológicos, académicos e industriales**, pudiendo ampliarse tanto en la parte de desarrollo algorítmico de entrenamientos *SOM* como para estudios y mejoras de interfaces interactivos. El entorno desarrollado, por su estructura modular y compatibilidad, abre muchas posibilidades en diferentes ámbitos:

- **Mejoras en interfaz** que faciliten su utilización e introducción.

- **Estudio de nuevos cálculos** facilitados por una conversión a **estructura para los cálculos masivos**.

Resumiendo, **se han presentado los *SOM* como potentes herramientas para la visualización de datos**, y **se ha desarrollado una herramienta de *código abierto* que los implementa**. Dicha herramienta, **dotada de interactividad**, facilita al usuario la **realización de entrenamientos variantes de *SOM***. Además, justificado el interés de repetir diversos entrenamientos para una misma muestra, se han estudiado y se incorporan determinadas **optimizaciones que constituyen una mejora** con respecto al entrenamiento secuencial de *SOM*. **Finalmente, se valoran y proponen variantes del modelo *SOM* para la visualización de datos**.

A continuación se exponen, para las distintas vertientes propuestas, varias líneas de continuidad.

5.2. Proyección futura

5.2.1. Mejoras en la interfaz

5.2.1.1. Inclusión de información multimedia

Uno de los objetivos tecnológicos expuestos en el apartado 1.1 es la *inclusión de redacción de observaciones junto a la red*, ya sean de manera gráfica como auditiva (Segel y Heer, 2010). En este contexto existen dos ampliaciones realizables:

1. Posibilitar la inclusión de comentarios del usuario, asociados a determinadas zonas de las capas, o a la red en general, que se presenten interactivamente. Éstos podrían ser textuales, gráficos o grabaciones auditivas. De este modo, además de poder almacenarse y trasladarse los resultados de entrenamiento, también se almacenarían las observaciones realizadas por los usuarios.

2. De manera más compleja, generar de manera discreta (no continua) observaciones sobre la información presentada. Del mismo modo que se calculan automáticamente agrupaciones de la información, podrían incluirse detecciones de zonas con determinadas características, como gradientes que superasen un determinado umbral o frecuencias similares a las observadas en otros cálculos, que permitieran caracterizar diferentes tipos de *SOM*.

5.2.1.2. Incorporación de jerarquías definidas por el usuario

Se propone como trabajo futuro la incorporación de la posibilidad, por parte del usuario, de generar nuevos entrenamientos a partir de la selección de proyecciones de datos, de tal manera que se puedan realizar, a partir de una muestra dada, visualizaciones parciales para las cuales una variable de entrada determinada adopte un valor o rango de valores.

Este tipo de entrenamiento resulta de gran utilidad cuando el objetivo es obtener diferentes segmentaciones a partir de una muestra dada, tal es el caso clásico de los estudios de marketing, donde cada perfil de cliente se estudia por separado.

5.2.2. Estructura de cálculos masivos

Una ampliación interesante, que permitiría la utilización de potentes máquinas para realizar los cálculos, haciendo los resultados de éstos visibles desde computadoras de trabajo, es la separación de la aplicación en dos elementos diferenciados: Cliente y Servidor.

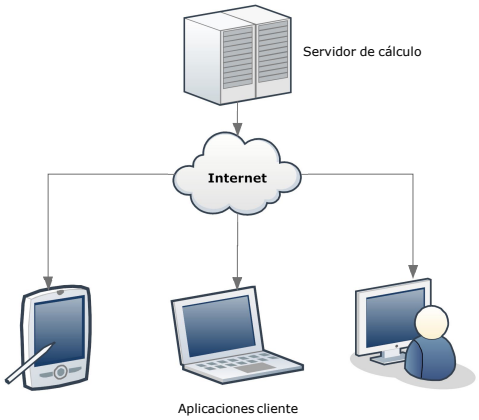


Figura 5.1: Esquema cliente/servidor

El cliente se encargaría de las tareas de visualización, navegación y comunicaciones con los aplicativos de cálculo, que se encontrarían instalados en un servidor dedicado a tal efecto (se muestra esquema en la figura 5.1).

La programación actual, en la cual los módulos pertenecientes al cliente se encuentran separados del software de cálculo, con las modificaciones pertinentes (comunicaciones y sincronización entre máquinas) puede llevar a la conversión a aplicativos cliente-servidor, lo cual posibilitaría la ejecución de un sistema con gran poder de cálculo y la creación de interfaces y compatibilidad con diversos soportes:

- Aplicaciones de escritorio.
- Plataformas móviles.
- Páginas web.

Además, la aplicación cliente-servidor podría favorecer el trabajo colaborativo, al permitir compartir de manera fácil la información almacenada en el servidor entre diferentes clientes, pudiendo ser presentada a varios de éstos simultáneamente.

5.2.3. Nuevos cálculos

La variedad de algoritmos y modelos existente permite realizar una búsqueda e implementación de diferentes métodos de entrenamiento que permitan ajustarse mejor a los diferentes conjuntos de datos de entrada. Algunas posibilidades a considerar son:

- **Implementación y estudio de diferentes topologías** de la red, dado que éstas influyen en las medidas de error de los mapas (Wu *et al.*, 2012a). De este modo se evitaría el *efecto de borde* de los mapas planos, debido a que las neuronas en los extremos de los planos tienen menos neuronas vecinas que el resto, lo que ocasiona que todas las unidades no tienen las mismas posibilidades de verse modificadas. Por tanto, según haya sido inicialización y transcurso del entrenamiento, determinados datos de entrada pueden no modificar el mismo número de neuronas que otros, ocasionando efectos no deseados sobre el resultado.

- **Implementación de modelos de topología flexible**. El crecimiento de los mapas planos requiere la inserción de filas o columnas para preservar la topología, lo que supone la inclusión de neuronas sin necesidad. La implementación de modelos flexibles topológicamente, además de salvar estas dificultades, permitirán implementar otro tipo de mejoras, como por ejemplo:

  - **Construcción de modelos jerárquicos cuyo crecimiento**, en sustitución del parámetro τ, **sea guiado por la distribución de los datos de entrada** (López-Rubio y Palomo, 2011) o dinámicamente según su comportamiento durante el entrenamiento (Alahakoon *et al.*, 2000).
  - **Mejoras en los índices topográficos** mediante prevención de intersecciones de conexiones entre neuronas (Lopez-Rubio, 2013).

  Resultará de interés la valoración de la utilidad de estas técnicas para la visualización de datos.

- **Empleo de diferentes medidas de distancia**. Tanto para el cálculo de error entre un dato de entrada y su correspondiente *BMU* como para las actualizaciones del resto de neuronas, la mayoría de propuestas utilizan como medida de distancia la *Euclídea*, lo que supone la realización de cálculos asumiendo espacios y distribuciones esféricas. Existen propuestas de utilización de diferentes medidas de error (López-Rubio *et al.*, 2014), y de funciones de vecindad asimétricas (Aoki *et al.*, 2007) o adaptativas (Lee y Verleysen, 2002). Por otra parte, las medidas de distancia dependen de la codificación de los datos de entrada (especialmente si se incluyen variables discretas), y para visualizaciones determinadas puede ser conveniente incorporar riesgos a las medidas, si se pretende diferenciar determinadas variables de entrada. Dichas modificaciones se pueden implementar con relativa facilidad dada la estructura modular de la aplicación.

- **Estudio del efecto del muestreo realizado en el experimento compuesto por diferentes entrenamientos**. El experimento realizado genera una cantidad determinada de entrenamientos empleando cada una de las combinaciones de valores de parámetros configurada, realizando implícitamente *remuestreos con repetición* de los datos de entrada. Resulta de interés conocer características y significación del experimento. Existen varios trabajos sobre esta temática (Rousset, 2006; Bodt y Cottrell, 2000).

- **Otros modos de visualización**. Por ejemplo, la *U-matrix (unified distance matrix)* permite visualizar las fronteras entre agrupamientos del *SOM* (Yamaguchi e Ichimura, 2011).

5.2.4. Cálculo automático de los mejores parámetros

Dado que, para cada experimento realizado, se almacenan las relaciones entre parámetros de entrada y los índices de error obtenidos, se posibilita la realización de **estudios estadísticos para determinar qué parámetros funcionan mejor con determinados conjuntos de datos**.

Un posible estudio de funcionamiento podría estar basado en aplicación de combinatoria, por ejemplo algoritmos genéticos, para encontrar los parámetros que mejor funcionamiento ofrecen para cada contexto.

Al final, sólo se tiene lo que se ha dado.

Isabel Allende

Apéndice A

Formato *XML* de Propiedades de Experimento

Cada experimento genérico comprende el entrenamiento de un conjunto de SOM según una serie de valores de parámetros. La configuración de cada experimento se guarda en un fichero en formato *XML*, de nombre *ExpProps.xml* en la carpeta donde se almacenan los resultados de entrenamiento. A continuación se proporciona un fichero ejemplo junto con explicación del sentido de sus apartados correspondientes.

```
<xml version="1.0" encoding="UTF-8" standalone="no">
<expprops>
  <atrib>
    <!-- Parámetros de aprendizaje a emplear -->
    <funcion>setBucleLearnRate</funcion>
    <valor>0.2,0.5</valor>
  </atrib>
  <atrib>
    <!-- Utilización de batch, en este ejemplo se
         entrenaría on-line (false) y batch (true) -->
    <funcion>setBucleUseBatch</funcion>
    <valor>false,true</valor>
  </atrib>
  <atrib>
    <!-- Valores de sigma a emplear -->
    <funcion>setSigma</funcion>
    <valor>3,5</valor>
  </atrib>
  <atrib>
    <!-- Tamaño inicial del plano. 0,0 cuando
         se va a calcular automáticamente -->
```

```
  <funcion>setXYSOM</funcion>
  <valor>0,0</valor>
</atrib>
<atrib>
  <!-- Número de CPS para que trabajen
       en paralelo -->
  <funcion>setNumCPUs</funcion>
  <valor>8</valor>
</atrib>
<atrib>
  <!-- Número de agrupamientos por defecto -->
  <funcion>setnClusters</funcion>
  <valor>5</valor>
</atrib>
<atrib>
  <!-- Fichero con los datos de entrada.
       Se guarda junto con los resultados -->
  <funcion>setFicheroEntrada</funcion>
  <valor>G:\selforganizing\data\nets\n1\data.csv
  </valor>
</atrib>
<atrib>
  <!-- Número de entrenamientos para cada
       combinación de parámetros -->
  <funcion>setNumRepe</funcion>
  <valor>50</valor>
</atrib>
<atrib>
  <!-- Modos de inicialización a emplear:
       10 = Aleatoria
       20 = Intervalos
       30 = Vector
       40 = PCA -->
  <funcion>setBucleInitializationMode</funcion>
  <valor>40,20,10,30</valor>
</atrib>
<atrib>
  <!-- Funciones de vecindad a emplear:
       10 = Gaussiana
       20 = Burbuja
       30 = Gaussiana con sesgo -->
  <funcion>setBucleNeighFunc</funcion>
  <valor>10,20,30</valor>
</atrib>
```

```
<atrib>
  <!-- Tamaño automático -->
  <funcion>setSizeAut</funcion>
  <valor>true</valor>
</atrib>
<atrib>
  <!-- Nombre del experimento -->
  <funcion>setExpName</funcion>
  <valor>GHSOM1-1</valor>
</atrib>
<atrib>
  <!-- Nombres de las redes resultantes -->
  <funcion>setNetNames</funcion>
  <valor>Best Kaski - Lagus Index,
         Best Quantization Error,
         Best Topographic Error</valor>
</atrib>
<atrib>
  <!-- Ficheros que contienen las redes -->
  <funcion>setNetFiles</funcion>
  <valor>kaski.xml,quan.xml,topo.xml</valor>
</atrib>
<atrib>
  <!-- Indicación de si es GSOM -->
  <funcion>setGrowing</funcion>
  <valor>false</valor>
</atrib>
<atrib>
  <!-- Indicación de si es subred generada a
       partir de otro mapa-->
  <funcion>setIsSubred</funcion>
  <valor>true</valor>
</atrib>
<atrib>
  <!-- Índices de las neuronas de la red del
       mapa anterior a partir de las cuales
       se genera el presente experimento -->
  <funcion>setSubredOrigen</funcion>
  <valor>2</valor>
</atrib>
<atrib>
  <!-- Fichero que contiene la red que contiene
       las neuronas a partir de las cuales se realiza
       el presente experimento -->
```

```
    <funcion>setFPadre</funcion>
    <valor>kaski.xml</valor>
  </atrib>
  <atrib>
    <!-- Ruta general donde se encuentran los datos -->
    <funcion>setRootPath</funcion>
    <valor>G:\Processing\selforganizing\data</valor>
  </atrib>
  <atrib>
    <!-- Indicación de si es GHSOM -->
    <funcion>setHier</funcion>
    <valor>true</valor>
  </atrib>
  <atrib>
    <!-- Indicación de si es GCHSOM -->
    <funcion>setGCHSOM</funcion>
    <valor>false</valor>
  </atrib>
  <atrib>
     <!-- Tamaños iniciales del sesgo -->
    <funcion>setBuclePcNeighWidth</funcion>
    <valor>0.5,0.7</valor>
  </atrib>
</expprops>
```

Apéndice B

Formato *XML* de Mapa Autoorganizativo

Se proporciona ejemplo resumido de almacenamiento de red en formato *XML*. Cada mapa se almacena en dos ficheros, un primer fichero con los datos, y otro con los valores de configuración empleados en su entrenamiento.

A continuación se expone un resumen de fichero en formato XML con los datos del SOM:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<som_xml>
  <struct KError="{Indice de Kaski y Lagus}"
            QError="{Error de cuantizacion}"
            TError="{Error topografico}"
            num_layers="{Numero de capas}"
            topolx="{Numero de columnas}"
            topoly="{Numero de filas}"/>
  <layer name="Nombre capa 1" show_cuad="true">
  {valores denormalizados de la capa, separados por espacios}
  </layer>
  <layer name="Nombre capa 2" show_cuad="true">
  {valores denormalizados de la capa, separados por espacios}
  </layer>
  <layer name="Nombre capa 3" show_cuad="true">
  {valores denormalizados de la capa, separados por espacios}
  </layer>
  .
  .
  .
  <layer name="Nombre capa n" show_cuad="true">
```

```
  {valores denormalizados de la capa, separados por espacios}
  </layer>

  <layer name="(Hits)" show_cuad="true">
  {Numero de datos mapeados en cada neurona}
  </layer>
  <layer name="(Cluster)" show_cuad="false">
  {Indicador de numero de grupo al que pertenece cada neurona}
  </layer>
  <atrib>
    <funcion>setLabelAtrib</funcion>
    <valor>{Nombre capa1,
                Nombre capa2,
                Nombre capa n}</valor>
  </atrib>
  <atrib>
    <funcion>setTopolXYZ</funcion>
    <valor>{Numero de columnas},{Numero de filas},1</valor>
  </atrib>
  <atrib>
      <funcion>setExpName</funcion>
    <valor>{Nombre del experimento al que pertenece}</valor>
  </atrib>
  <atrib>
    <funcion>setIsSubnet</funcion>
    <valor>{"true" si proviene de otra red,
                "false" en caso contrario}</valor>
  </atrib>
</som_xml>
```

El fichero en formato *XML* que contiene las opciones de configuración del mapa en cuestión, como se observa, contiene los valores concretos de los parámetros con los que se entrenó el SOM. A continuación se muestra un ejemplo:

```
ï»¿<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<expprops>
  <atrib>
    <!--  Dimensiones de la red -->
    <funcion>setXYZSize</funcion>
    <valor>40,20,1</valor>
  </atrib>
  <atrib>
      <!--  Topologia de la red -->
    <funcion>setGridTopology</funcion>
    <valor>planar</valor>
```

```
</atrib>
<atrib>
  <funcion>setGridLayout</funcion>
  <valor>rectangular</valor>
</atrib>
<atrib>
    <!--  Modo de inicializacion -->
  <funcion>setInitializationMode</funcion>
  <valor>30</valor>
</atrib>
<atrib>
    <!--  Si es tipo Batch -->
  <funcion>setBatchSom</funcion>
  <valor>true</valor>
</atrib>
  <atrib>
      <!--  Funcion de vecindad -->
  <funcion>setNeighbourFunc</funcion>
  <valor>20</valor>
</atrib>
<atrib>
    <!--  Parametro de aprendizaje inicial-->
  <funcion>setLearnrate</funcion>
  <valor>0.5</valor>
</atrib>
<atrib>
    <!--  Metrica empleada  -->
  <funcion>setMetricName</funcion>
  <valor>at.tuwien.ifs.somtoolbox.layers.metrics.L2Metric</valor
</atrib>
<atrib>
    <!--  Numero de iteraciones -->
  <funcion>setNumIterations</funcion>
  <valor>32561</valor>
</atrib>
<atrib>
    <!--  Constante de atenuacion de la Gaussiana -->
  <funcion>setSigma</funcion>
  <valor>5.0</valor>
</atrib>
<atrib>
    <!--  Parametro de crecimiento de red  -->
  <funcion>setTau</funcion>
  <valor>0.001</valor>
```

```
  </atrib>
  <atrib>
      <!--  Nombre del experimento al que pertenece-->
    <funcion>setExpName</funcion>
    <valor>GSOM2</valor>
  </atrib>
  <atrib>
      <!--  Error de cuantizacion de referencia, a emplear
             cuando la red procede de otra  -->
    <funcion>setQRef</funcion>
    <valor>Infinity</valor>
  </atrib>
  <atrib>
      <!--  Indicador de si es GSOM-->
    <funcion>setGrowing</funcion>
    <valor>true</valor>
  </atrib>
  <atrib>
      <!--  Indicador de si es GHSOM-->
    <funcion>setHier</funcion>
    <valor>false</valor>
  </atrib>
  <atrib>
      <!--  Indicador de si es subred -->
    <funcion>setIsSubred</funcion>
    <valor>false</valor>
  </atrib>
  <atrib>
      <!--  Indices de neuronas de origen en caso de que
             sea una red procedente de otra -->
    <funcion>setSubredOrigen</funcion>
    <valor>-1</valor>
  </atrib>
  <atrib>
      <!--  Indicador de si es GCHSOM -->
    <funcion>setGCHSOM</funcion>
    <valor>false</valor>
  </atrib>
  <atrib>
      <!--  Sesgo de vecindad inicial -->
    <funcion>setPcNeighbourWidth</funcion>
    <valor>0.7</valor>
  </atrib>
</expprops>
```

Apéndice C

Resumen de funcionalidades

C.1. Opciones de entrenamiento

Se presenta resumen de las opciones de entrenamiento implementadas.

1) Opción de selección entre las **inicializaciones** implementadas, ya expuestas anteriormente: a) *Inicialización aleatoria*, b) *inicialización mediante interpolación de intervalos de datos*, c) *inicialización directa por vectores de entrada* y d) *inicialización empleando el Análisis de Componentes Principales.*

2) Opción de selección entre las **funciones de vecindad** empleadas: a) *Gaussiana*, b) *gaussiana recortada* y c) *burbuja*. El tipo de rejilla es hexagonal, para evitar favorecer las direcciones vertical y horizontal, como sucede con la rejilla rectangular.

3) El **tamaño** que tendrá la red resultante. También se puede seleccionar la opción automática, en cuyo caso el número de celdas y la altura y anchura de la red se calcula mediante el heurístico siguiente:

- Número de celdas $n = t^{0,54321}$, donde t es el número de vectores de entrada.
- La altura $a = \frac{n}{r}$ donde $r = \sqrt{\frac{I1}{I2}}$ donde $I1$ e $I2$ son respectivamente los índices de los dos mayores valores propios del *Análisis de Componentes Principales (PCA).*

4) El **raso de vecindad** inicial para las funciones de vecindad *gaussiana recortada* y *burbuja*. El cálculo se basa en distancia en unidades.

5) El **parámetro de aprendizaje α inicial**.

6) El **parámetro σ que define la amplitud del kernel** de las *vecindades gaussianas.*

7) El empleo opcional de modos de entrenamiento ***batch* y/o *on line***.

8) El **número de ejecuciones** por cada combinación de parámetros.

En caso de seleccionar diferentes opciones alternativas para los parámetros 1), 2), 4), 5), 6) y 7), la aplicación genera por combinatoria los diferentes parámetros de ejecución del algoritmo, a repetir cada una de estas combinaciones tantas veces como indicadas en el parámetro 8).

C.2. Funciones de visualización

Resumen de funciones de visualización e interactivas:

1) **Visualización del histórico** de los experimentos calculados.

2) **Elección**, para cada experimento calculado, **del mapa con mejor índice** según los criterios: a) mínimo error topográfico, b) mínimo error de cuantización, c) mejor índice de Kaski & Lagus.

3) **Clusterización por *K-Means*** como ayuda a la detección de zonas con similitudes. Elección del número de grupos.

4) **Selección del mapa de colores**, de entre los posibles: *Jet* (por defecto), *escala de grises*, *HSV*, *tonos de rosa*, *colores fríos*, *colores calientes*, *tonos de cobre* y *tonos de hueso*.

5) **Posibilidad de elegir tamaño** de los mapas y de arrastrarlos para cambiarlos de orden.

6) **Posibilidad de arrastrar mapas a un panel inferior**, donde se muestran minimizados.

7) **Preselección visual de nodos**, mostrando interactivamente el valor promedio de éstos y el número de vectores de entrada para los cuales fueron *mejor opción*.

8) **Posibilidad de selección interactiva de grupos de nodos** por los criterios:

- Cercanía al cursor.
- Pertenencia al cluster en el que se encuentra el cursor.
- Dibujo sobre el mapa.

9) **Posibilidad de realizar un nuevo experimento a partir de una zona seleccionada**, haciendo un efecto *zoom*.

10) **Exportación a formato PDF** del contenido en pantalla.

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el cerebro de manera que vino a perder el juicio.

Miguel de Cervantes Saavedra

Akinduko, A. A. y Mirkes, E. M. (2012). Initialization of Self-Organizing Maps: Principal Components Versus Random Initialization. A Case Study. *The Computing Research Repository*, vol. abs/1210.5873.
Disponible en `http://dblp.uni-trier.de/db/journals/corr/corr1210.html#abs-1210-5873`.

Alahakoon, D., Halgamuge, S., y Srinivasan, B. (2000). Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614.

Aoki, T., Ota, K., Kurata, K., y Aoyagi, T. (2007). Ordering Process of Self-Organizing Maps Improved by Asymmetric Neighborhood Function. En Ishikawa, M., Doya, K., Miyamoto, H., y Yamakawa, T., editores, *ICONIP (1)*, volumen 4984 de *Lecture Notes in Computer Science*, pp. 426–435. Springer.

Araujo, A. F. R. y Rego, R. L. M. E. (2013). Self-organizing Maps with a Time-varying Structure. *ACM Computing Surveys*, 46(1):7:1–7:38.

Ascioglu, S. (2008). Open processing community: http://www.openprocessing.org. `http://www.openprocessing.org`.

Bezerianos, A. e Isenberg, P. (2012). Perception of Visual Variables on Tiled Wall-Sized Displays for Information Visualization Applications. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2516–2525.

Bodt, E. y Cottrell, M. (2000). Bootstrapping self-organising maps to assess the statistical significance of local proximity. En *ESANN 2000 proceedings*, pp. 245–254.

Bourennani, F., Pu, K. Q., y Zhu, Y. (2009a). Unified Vectorization of Numerical and Textual Data using Self-Organizing Map. *International Journal On Advances in Systems and Measurements*, 2(2):142–155.

Bourennani, F., Pu, K. Q., y Zhu, Y. (2009b). Visual Integration Tool for Heterogeneous Data Type by Unified Vectorization. En *Proceedings of the IEEE International Conference on Information Reuse and Integration, IRI 2009, 10-12 August 2009, Las Vegas, Nevada, USA*, pp. 132–137.

Cao, N., Lin, Y.-R., Sun, X., Lazer, D., Liu, S., y Qu, H. (2012). Whisper: Tracing the spatiotemporal process of information diffusion in real time. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2649–2658.

Chen, T., Lu, A., y Hu, S.-M. (2012). Visual storylines: Semantic visualization of movie sequence. *Computers and Graphics*, 36(4):241 – 249.

Crnovrsanin, T., Liao, I., Wuy, Y., y Ma, K.-L. (2011). Visual Recommendations for Network Navigation. En *Proceedings of the 13th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'11, pp. 1081–1090. Eurographics Association.

Cui, W., Liu, S., Tan, L., Shi, C., Song, Y., Gao, Z., Qu, H., y Tong, X. (2011). Textflow: Towards Better Understanding of Evolving Topics in Text. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2412–2421.

Daswin Pasantha De Silva, L. y Alahakoon, D. (2006). Analysis of Seismic Activity using the Growing SOM for the Identification of Time Dependent Patterns. En *International Conference on Information and Automation, 2006. ICIA 2006.*, pp. 155–159.

Demartines, P. y Herault, J. (1997). Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154.

Dittenbach, M., Rauber, A., y Polzlbauer, G. (2005). Investigation of alternative strategies and quality measures for controlling the growth process of the growing hierarchical self-organizing map. En *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. IJCNN '05.*, volumen 5, pp. 2954–2959.

Erwin, E., Obermayer, K., y Schulten, K. (1992). Self-Organizing Maps: Stationary States, Metastability and Convergence Rate. *Biological Cybernetics*, 67:35–45.

Fry, B. (2008). *Visualizing Data*. O'Reilly, primera edición. ISBN 978-0-596-51455-6.

Ge, M., Du, R., y Xu, Y. (2003). Fault detection using hierarchical self-organizing map. En *Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003 IEEE International Conference on*, volumen 1, pp. 565–570.

Gomez, S. R., Jianu, R., Ziemkiewicz, C., Guo, H., y Laidlaw, D. H. (2012). Different Strokes for Different Folks: Visual Presentation Design between Disciplines. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2411–2420.

Gunasinghe, U., Matharage, S., y Alahakoon, D. (2012). A sequence based dynamic SOM model for text clustering. En *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.

Hadlak, S., Schulz, H.-J., y Schumann, H. (2011). In Situ Exploration of Large Dynamic Networks. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2334–2343.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., y Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explorations*, 11(1).

Haroz, S. y Whitney, D. (2012). How Capacity Limits of Attention Influence Information Visualization Effectiveness. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2402–2410.

Herbert, J. y Yao, J. (2007). Growing Hierarchical Self-Organizing Maps for Web Mining. En *IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 299–302.

Hofmann, H., Follett, L., Majumder, M., y Cook, D. (2012). Graphical Tests for Power Comparison of Competing Designs. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2441–2448.

Hu, H., Wen, Y., Chua, T.-S., y Li, X. (2014). Toward Scalable Systems for Big Data Analytics: A Technology Tutorial. *Access, IEEE*, 2:652–687.

Jankun-Kelly, T., Ma, K.-L., y Gertz, M. (2007). A Model and Framework for Visualization Exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369.

Kang, Y. (2012). Real-time change detection in time series based on growing feature quantization. En *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6.

Kaski, S. y Lagus, K. (1996). Comparing Self-Organizing Maps. pp. 809–814. Springer. ISBN 3-540-61510-5.

Keim, D., Mansmann, F., Schneidewind, J., y Ziegler, H. (2006). Challenges in Visual Data Analysis. En *Tenth International Conference on Information Visualization.*, pp. 9–16.

Keim, D. A. (2002). Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8.

Kim, S.-H., Dong, Z., Xian, H., Upatising, B., y Yi, J. S. (2012). Does an Eye Tracker Tell the Truth about Visualizations?: Findings while Investigating Visualizations for Decision Making. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2421–2430.

Kim, S. W. y To, T. V. (2013). A self-growing and Self-Organizing Batch Map with automatic stopping condition. En *2013 5th International Conference on Knowledge and Smart Technology (KST)*, pp. 21–26.

Ko, S., Maciejewski, R., Jang, Y., y Ebert, D. S. (2012). Marketanalyzer: An Interactive Visual Analytics System for Analyzing Competitive Advantage Using Point of Sale Data. *Computer Graphics Forum*, 31(3pt3):1245–1254.

Kohonen, T., Schroeder, M. R., y Huang, T. S., editores (2001). *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edición. ISBN 3-540679-21-9.

Kuremoto, T., Komoto, T., Kobayashi, K., y Obayashi, M. (2010). Parameterless-Growing-SOM and Its Application to a Voice Instruction Learning System. *Journal of Robotics*, 2010, Article ID 307293.

Lam, H., Bertini, E., Isenberg, P., Plaisant, C., y Carpendale, S. (2012). Empirical Studies in Information Visualization: Seven Scenarios. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1520–1536.

Lampinen, J. y Oja, E. (1992). Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 2(2-3):261–272.

Lee, J. A. y Verleysen, M. (2002). Self-organizing maps with recursive neighborhood adaptation. *Neural Networks*, 15:993 – 1003.

Li, W.-S. y Clifton, C. (2000). SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks. *Data and Knowledge Engineering*, 33(1):49–84.

Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`. University of California, Irvine, School of Information and Computer Sciences.

Linde, Y., Buzo, A., y Gray, R. M. (2003). An Algorithm for Vector Quantizer Design Communications. *IEEE Transactions on In Communications*, 28(1):84 – 95.

Liu, S., Cao, N., y Lv, H. (2008). Interactive Visual Analysis of the NSF Funding Information. En *Visualization Symposium, 2008. IEEE Pacific*, pp. 183–190.

Liu, S., Cui, W., Wu, Y., y Liu, M. (2014). A survey on information visualization: recent advances and challenges. *The Visual Computer*, 30(12):1373–1393.

Liu, S., Zhou, M. X., Pan, S., Song, Y., Qian, W., Cai, W., y Lian, X. (2012). Tiara: Interactive, Topic-Based Visual Text Summarization and Analysis. *ACM Transactons on Intelligent Systems and Technology (TIST)*, 3(2):25:1–25:28.

Lopez-Rubio, E. (2013). Improving the Quality of Self-Organizing Maps by Self-Intersection Avoidance. *IEEE Transactions on Neural Networks and Learning Systems*, 24(8):1253–1265.

López-Rubio, E. y Palomo, E. (2011). Growing Hierarchical Probabilistic Self-Organizing Graphs. *IEEE Transactions on Neural Networks*, 22(7):997–1008.

López-Rubio, E., Palomo, E., y Domíguez, E. (2014). Bregman divergences for growing Hierarchical Self-Organizing Networks. *International Journal of Neural Systems*, 24(04):1450016. PMID: 24694171.

Ma, J., Liao, I., Ma, K.-L., y Frazier, J. (2012). Living Liquid: Design and Evaluation of an Exploratory Visualization Tool for Museum Visitors. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2799–2808.

MacEachren, A., Roth, R., O'Brien, J., Li, B., Swingley, D., y Gahegan, M. (2012). Visual Semiotics and Uncertainty visualization: An empirical study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2496–2505.

Makhoul, J., Roucos, S., y Gish, H. (1985). Vector quantization in speech coding. *Proceedings of the IEEE*, 73:1551–1588.

Matharage, S., Ganegedara, H., y Alahakoon, D. (2013). A scalable and dynamic self-organizing map for clustering large volumes of text data. En *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.

Mayer, R., Dittenbach, M., y Frank, J. (2011). Java somtoolbox research prototype. `http://www.ifs.tuwien.ac.at/dm/somtoolbox/index.html`.

McKay, E. N. (2013). *UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edición. ISBN 978-0123969804.

Mishra, B. K., Rath, A., Nayak, N. R., y Swain, S. (2012). Far Efficient K-means Clustering Algorithm. En *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pp. 106–110, New York, NY, USA. ACM. ISBN 978-1-4503-1196-0.

Nathawitharana, N., Matharage, S., y Alahakoon, D. (2013). Feature overlap-based dynamic self organizing model for hierarchical text clustering. En *Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on*, pp. 393–398.

Palomo, E. y Domínguez, E. (2011). Image compression based on growing hierarchical self-organizing maps. En *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pp. 1624–1628.

Pérez, D., Zhang, L., Schaefer, M., Schreck, T., Keim, D., y Díaz, I. (2013). Interactive Visualization and Feature Transformation for Multidimensional Data Projection. En Aupetit, M. y van der Maaten, L., editores, *EuroVis Workshop on Visual Analytics using Multidimensional Projections*. The Eurographics Association. ISBN 978-3-905674-53-8.

Qiang, X., Cheng, G., y Li, Z. (2010). A survey of some classic self-organizing maps with incremental learning. En *2010 2nd International Conference on Signal Processing Systems (ICSPS)*, volumen 1, pp. V1–804–V1–809.

Rauber, A., Merkl, D., y Dittenbach, M. (2002). The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341.

Reas, C. y Fry, B. (2005). Processing.org: A networked context for learning computer programming. En *ACM SIGGRAPH 2005 Web Program*, SIGGRAPH '05, New York, NY, USA. ACM.

Resig, J. (2008). Processing js framework. `http://www.processingjs.com`.

Rousset, P. (2006). Self organizing maps: understanding, measuring and reducing variability. En Rizzi, A. y Vichi, M., editores, *Compstat 2006 - Proceedings in Computational Statistics*, pp. 371–382. Physica-Verlag HD.

Salem, M. y Buehler, U. (2013). An Enhanced GHSOM for IDS. En *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1138–1143.

Sarlin, P. y Rönnqvist, S. (2013). Cluster Coloring of the Self-Organizing map: An Information Visualization Perspective. En *17th International Conference on Information Visualisation, IV 2013, London, United Kingdom, July 16-18, 2013*, pp. 532–538.

Schweitzer, D., Boleng, J., y Graham, P. (2010). Teaching Introductory Computer Graphics with the Processing Language. *Journal of Computing Sciences in Colleges*, 26(2):73–79.

Segel, E. y Heer, J. (2010). Narrative Visualization: Telling Stories with Data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1139–1148.

Shi, L., Wei, F., Liu, S., Tan, L., Lian, X., y Zhou, M. (2010). Understanding text corpora with multiple facets. En *2010 IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pp. 99–106.

Sisneros, R., Johnson, C. R., y Huang, J. (2008). Concurrent viewing of multiple attribute-specific subspaces. *Computer Graphics Forum*, 27(3):783–790.

Stefanovic, P. y Kurasova, O. (2011). Influence of Learning Rates and Neighboring Functions on Self-Organizing Maps. En Laaksonen, J. y Honkela, T., editores, *Advances in Self-Organizing Maps*, volumen 6731 de *Lecture Notes in Computer Science*, pp. 141–150. Springer Berlin Heidelberg.

Sun, G.-D., Wu, Y.-C., Liang, R.-H., y Liu, S.-X. (2013). A Survey of Visual Analytics Techniques and Applications: State-of-the-Art Research and Future Challenges. *Journal of Computer Science and Technology*, 28(5):852–867.

Tan, H. S. y George, S. E. (2004). Investigating Learning Parameters in a Standard 2-D SOM Model to Select Good Maps and Avoid Poor Ones. En *AI 2004: Advances in Artificial Intelligence, 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, December 4-6, 2004, Proceedings*, pp. 425–437.

Tory, M. y Moller, T. (2005). Evaluating visualizations: do expert reviews work? *Computer Graphics and Applications, IEEE*, 25(5):8–11.

Vesanto, J. y Alhoniemi, E. (2000). Clustering of the Self-Organizing Map. *IEEE Transactions on Neural Networks*, 11(3):586–600.

Vesanto, J., Himberg, J., Alhoniemi, E., y Parhankangas, J. (2000). Self-Organizing Map in Matlab: the SOM Toolbox. En *In Proceedings of the Matlab DSP Conference*, pp. 35–40.

Villmann, T., Der, R., Herrmann, M., y Martinetz, T. (1997). Topology preservation in self-organizing feature maps: exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266.

Walny, J., Carpendale, S., Riche, N., Venolia, G., y Fawcett, P. (2011). Visual Thinking in Action: Visualizations As Used On Whiteboards. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2508–2517.

Wegman, E. J. (2003). Visual data mining. *Statistics in Medicine*, 22(9):1383–1397.

Wu, H., Gedeon, T., y Zhu, D. (2012a). Investigating the quality of different Self-Organizing Map topologies for complex data. En *2012 4th IEEE International Symposium on Logistics and Industrial Informatics (LINDI)*, pp. 221–226.

Wu, S. y Chow, T. W. (2004). Clustering of the self-organizing map using a clustering validity index based on inter-cluster and intra-cluster density. *Pattern Recognition*, 37(2):175 – 188.

Wu, Y., Yuan, G.-X., y Ma, K.-L. (2012b). Visualizing Flow of Uncertainty through Analytical Processes. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2526–2535.

Yamaguchi, T. e Ichimura, T. (2011). Visualization using multi-layered U-Matrix in growing Tree-Structured self-organizing feature map. En *2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3580–3585.

Yang, H.-C., Chen, D.-W., y Lee, C.-H. (2007). Mining Multilingual Texts using Growing Hierarchical Self-Organizing Maps. En *2007 International Conference on Machine Learning and Cybernetics*, volumen 4, pp. 2263–2268.

Yu, Y. y Alahakoon, D. (2006). Batch implementation of Growing Self-Organizing map. En *2006 International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA 2006), International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2006), 29 November - 1 December 2006, Sydney, Australia*, p. 162. IEEE Computer Society.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes