PROYECTO FIN DE MÁSTER:

Herramientas de análisis de validez de procesos de clustering

Alumno:

Manuel Joaquín Márquez García

Directoras:

Raquel Dormido Canto

Natividad Duro Carralero

Máster en ingeniería de sistemas y de control

Curso: 2012-2013

Convocatoria: septiembre 2013

PROYECTO FIN DE MÁSTER:

Herramientas de análisis de validez de procesos de clustering

Alumno:

Manuel Joaquín Márquez García

Directoras:

Raquel Dormido Canto

Natividad Duro Carralero

Máster en ingeniería de sistemas y de control

Proyecto de Investigación

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado: Manuel Joaquín Márquez García

Resumen

En el presente trabajo se ha desarrollado una herramienta de clustering en Matlab para automatizar el proceso de clasificación y posterior validación en procesos de clasificación.

La estructura de la memoria es la siguiente, hay una primera parte de introducción teórica donde se exponen los distintos métodos y algoritmos que se van a emplear, y a continuación se explica la herramienta de Matlab que se ha creado para realizar el análisis de validez del proceso de clustering.

Dentro de esta herramienta se encuentran las distintas etapas o fases:

1. Selección de datos. Donde se le ofrecerá al usuario la opción de trabajar con dos grupos de datos diferentes y el número de patrones que el desee.
2. Preprocesado de datos, esta etapa se encuentra dentro del proceso de análisis de tendencia, dentro de este paso se han incluido los métodos SOM, VAT e iVAT.
3. Reducción de la dimensionalidad, donde se incluyen las técnicas del Mapeo de Sammon y el Análisis de Componentes Principales. La realización de esta etapa es muy importante, ya que permite al usuario realizar una representación de los datos y el ahorro computacional correspondiente, al trabajar con un menor número de datos.
4. Aplicación de diversos algoritmos de clustering, entre los que se encuentran K-means, Adaptativo y Jerárquico. Se han seleccionado estos algoritmos dentro de la gran variedad existente, en la aplicación se permitirá al usuario introducir un rango de los distintos datos de entrada necesarios para ejecutarlos.
5. Selección del número de clusters óptimo a partir de una serie de índices de validación como son los índices de Dunn, Davies-Bouldin, Hubert, Silhouette, Compacidad, R-Squared y Calinski-Harabasz. En esta sección se validarán todos los agrupamientos creados y se seleccionará el que de el resultado más óptimo según el índice de validación que se haya seleccionado.

El objetivo de dicha herramienta es permitir al usuario escoger el mejor agrupamiento posible a partir de unos datos dados, realizando todo el proceso de validación de clusters, dentro de dicho proceso se encuentran comprendidas las etapas de tendencia, determinación, estabilidad y solución consensuada.

Palabras clave

Adaptativo
Agrupamiento
Algoritmo
Análisis
Autoorganizativo
Calinski-Harabasz
Clasificación
Cluster
Clustering
Compacidad
Correlación
Covarianza
Davies-Bouldin
Dimensionalidad
Disimilitud
Dunn
Euclídea
Hubert
iVAT
Jerárquico
K-means
Matlab
Malla
Mapa
Neurona
Partición
R-Squared
Sammon
Silhouette
SOM
Tendencia
Topología
VAT
Variable

Índice

Índice figuras

Índice funciones

1. Introducción

Las técnicas de clustering se aplican para clasificar los objetos en aquellos que son similiares entre si, separándolos de los que no son semejantes. La aplicación de técnicas de clustering buscan lo siguiente:

- Cada grupo o cluster sea homogéneo o compacto con respecto a ciertas características. Es decir las observaciones dentro de cada grupo han de ser similares entre sí.
- Cada grupo debe diferenciarse de los otros grupos respecto a las mismas características, es decir las observaciones de un grupo deben diferenciarse de las observaciones de los otros grupos.

En un proceso de análisis de clustering se tienen las siguientes fases:

1) Se establecen los objetivos y se seleccionan las variables con las que se van a caracterizar los objetos a agrupar.
2) Se estandarizan las variables y se seleccionan las medidas de similitud.
3) Selección de algoritmos.
4) Obtención de una agrupación o varias.
5) Proceso de validación, dentro de este proceso se incluye el análisis de tendencia previo, la interpretación de las agrupaciones creadas y la validación de los resultados.

Cuando se realiza un sistema de clasificación supervisada se tiene una gran variedad de medidas para evaluar como de bueno es nuestro modelo, como pueden ser la exactitud y la precisión. Cuando se realiza un análisis de cluster, la pregunta sería cuál es la bondad de los resultados. Las técnicas de validación de clusters abarcan diferentes aspectos:

- Determinación de la tendencia del agrupamiento en un conjunto de datos, es decir, distinguir si existe una tendencia en los datos y no es algo aleatorio.
- Comparar los resultados de un análisis de clustering con otros resultados externos ya conocidos.
- Evaluar en qué medida los resultados de un análisis de clustering encajan con los datos sin tener en cuenta la información externa.
- Comparar los resultados obtenidos a través de dos técnicas de clustering diferentes para evaluar cuál es mejor.
- Determinar el número correcto de cluster a hallar.

Diversas medidas numéricas se pueden aplicar para juzgar la validez del análisis de cluster, la clasificación se puede hacer en función de los siguientes tres tipos:

- Índice externo: Se usa para medir sobre el grado en que medida coinciden las etiquetas dadas al análisis de clustering con las proporcionadas exteriormente (entropía).

- Índice interno: Se emplea para medir la bondad de una estructura de clustering sin relación con información externa (suma de errores al cuadrado, SSE).
- Índice relativo: Se utiliza para comparar dos clustering o clusters diferentes.

2. Objetivos

En este trabajo se pretende desarrollar una herramienta de clustering para automatizar el proceso de clasificación y posterior validación en procesos de clasificación. Esta herramienta abarcará todo el proceso de clustering:

i. Selección de los datos a clasificar.
ii. Selección del preprocesado de los datos.
iii. Selección de distintos métodos de reducción de dimensionalidad.
iv. Selección y ejecución de distintos algoritmos de clustering.
v. Selección de diversos índices de validación que permitan estimar el número óptimo de clusters en los diferentes algoritmos, permitiendo visualizar los valores del índice de validación seleccionado en función del número de clusters.

En este trabajo se van a describir los métodos más representativos de cada etapa y se desarrollará una interfaz gráfica en la que aparecerán todos integrados, se ejecutarán por orden y se permitirá al usuario elegir los datos con los que trabajar, los algoritmos o métodos a emplear y la repetición de los mismos hasta obtener el mejor resultado posible.

3. Análisis tendencia de clustering

Uno de los problemas de los algoritmos de clustering [3] es que generan una partición de los datos de entrada aunque estos no tengan ningún tipo de estructura interna. Dada la complejidad de la validación de clustering es recomendable evitar este tipo de problemas realizando un análisis previo con el objeto de establecer si los datos tienen algún tipo de estructura. Además, este análisis previo evitaría realizar un trabajo de clustering potencialmente costoso sobre unos datos imposibles de particionar de forma coherente. Este análisis previo para determinar si los datos tienen cierta predisposición a agruparse en grupos naturales se conoce como tendencia al clustering (cluster tendency) o agrupabilidad (clusterability).

Aunque el análisis de la tendencia al clustering se considera parte de la validación de clustering es un proceso que se realiza antes de la fase de aprendizaje. En este sentido, es un proceso que se sale de la ubicación habitual de la validación en el proceso completo de la minería de datos, que es posterior a la fase de aprendizaje. Por lo tanto, se puede afirmar que el análisis de la tendencia al clustering es un proceso de análisis previo.

Ackerman y Ben-David [26] realizan un estudio interesante sobre los distintos conceptos de agrupabilidad. Las técnicas principales de análisis de la tendencia al clustering se basan en el concepto de aleatoriedad. Básicamente, tratan de determinar si los casos de la muestra de entrenamiento se han generado en base a un mismo proceso aleatorio. Algunas técnicas se centran en analizar la aleatoriedad espacial, es decir, la aleatoriedad en las posiciones que ocupan los distintos casos. Otras técnicas, en cambio, tratan de medir la posible aleatoriedad de la matriz de distancias de los casos de la muestra [27].

Existen otras aproximaciones que se basan en representaciones gráficas de la matriz de distancias. Bezdek y Hathaway [17] propusieron un sistema (VAT) para ordenar los casos de manera tal que la representación gráfica de la matriz de distancias sugiera el nivel de agrupabilidad de la muestra.

Existen diversas variantes del mismo sistema para adaptarlo a bases de datos de gran tamaño como el método reVAT [28], bigVAT [29] o sVAT [30].

Otro método posible es el propuesto por Massey [31] que utiliza técnicas de teoría de resonancia adaptativa, un modelo de red neuronal artificial.

En resumen, se podría decir que la tarea de preprocesado [7] de datos se realiza para que el proceso de minería de datos sea más fácil y efectivo. Algunas de las tareas de las que se compone el preprocesado de datos son:

- Data cleaning o limpieza de datos, es el proceso orientado a eliminar datos con ruido o incorrectos.

- Data integration, trata de integrar diferentes fuentes de datos en un almacén coherente y homogéneo como un data warehouse o un data cube.
- Data transformation, o transformación de los datos como, por ejemplo, una normalización.
- Data reduction, o reducción de los datos, orientado a reducir el tamaño de los datos mediante la agregación y/o eliminación de características redundantes, o clustering.

3.1. Self Organizing Maps (SOM)

Los mapas auto-organizativos (SOM – Self-Organizing Map) [8] fueron inventados por Teuvo Kohonen en Finlandia, en la década de los 80. Un mapa autoorganizativo consiste en un grupo de neuronas organizadas en una malla de baja dimensión. Cada neurona está representada por un vector de pesos de *m* dimensiones (vector prototipo), donde *m* es igual a la dimensión del vector de entrada. Las neuronas se encuentran conectadas a neuronas adyacentes por una relación de vecindad, la cual dicta la topología o estructura del mapa. Un ejemplo de estructuras de vecindad de mapas autoorganizativos se muestra en la siguiente figura:

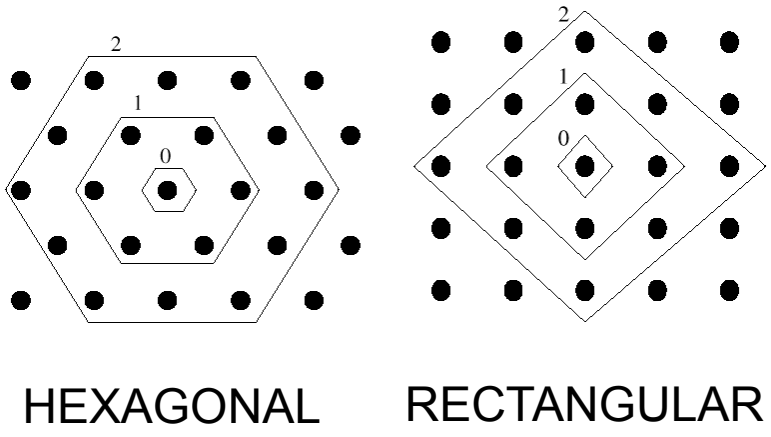

Figura 1.- Estructuras mapa autoorganizativo.

El método SOM tiene la peculiaridad, y de ahí surge su fortaleza, de servir al mismo tiempo:

- Reducir el número de observaciones de los datos.
- Reducir la dimensionalidad del problema preservando la topología, es decir, preservando las relaciones de vecindad o proximidad entre los elementos.

La arquitectura de los mapas Auto-Organizados se podría definir de la siguiente manera:

a) Una primera capa de entrada, en la que se coloca el vector de entrada a la red. Las neuronas de esta capa no realizan ningún procesamiento, sólo resciben el vector de entrada y lo distribuyen a las neuronas de la capa de salida.
b) Una capa de salida que se conoce con el nombre de mapa.

Esta arquitectura se muestra en la siguiente figura:

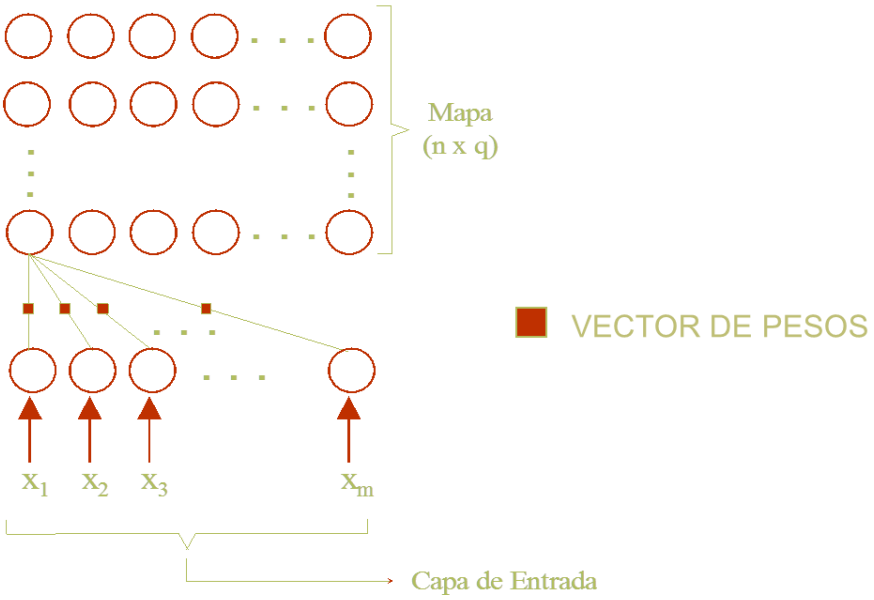

Figura 2.- Arquitectura mapa Auto-Organizativo

El algoritmo de entrenamiento es el siguiente:

1. Se identifica la dimensión *m* del algoritmo del espacio de entrada, que se corresponde con el número de neuronas de la capa de entrada. Los *s* vectores de entrenamiento constituyen el conjunto de datos a clasificar.
2. Se define el tamaño del mapa.
3. Se inicializan los pesos w_j con valores aleatorios.
4. Se selecciona un vector x_j del conjunto de vectores de entrenamiento.
5. Para cada neurona N_i, del mapa, se calcula la distancia euclidiana entre su vector de pesos (w_i) y el vector de entrenamiento seleccionado.
6. Se determina la neurona ganadora, que es aquella cuyo vector de pesos esté a la menor distancia del vector de entrenamiento, esto se conoce como unidad de mejor ajuste (BMU-Best-Matching unit).
7. Se actualiza el vector de pesos asociado a la neurona ganadora, de manera tal que se mueva hacia el vector de entrenamiento en el espacio de entrada.

$$w_i(t+1) = w_i(t) + \alpha(t)[x^j - w_i(t)] \qquad (1)$$

8. Las neuronas N_k que se encuentran en la vecindad de tamaño pre-establecido, *r*, actualizan sus pesos junto a la neurona ganadora. El valor de *r* disminuye a

partir de un valor inicial a medida que avanza el proceso de entrenamiento, la fórmula de actualización de pesos es:

$$w_k(t+1) = w_k(t) + \alpha(t)[x^j - w_k(t)] \quad (2)$$

9. Posteriormente se retorna al paso 4, repitiéndose este proceso durante un número de iteraciones previamente establecido.

Un ejemplo de la actualización del mapa Auto-Organizativo, realizando un movimiento de los pesos de la neurona ganadora (BMU) y su vecindad de radio 1, se muestra a continuación:

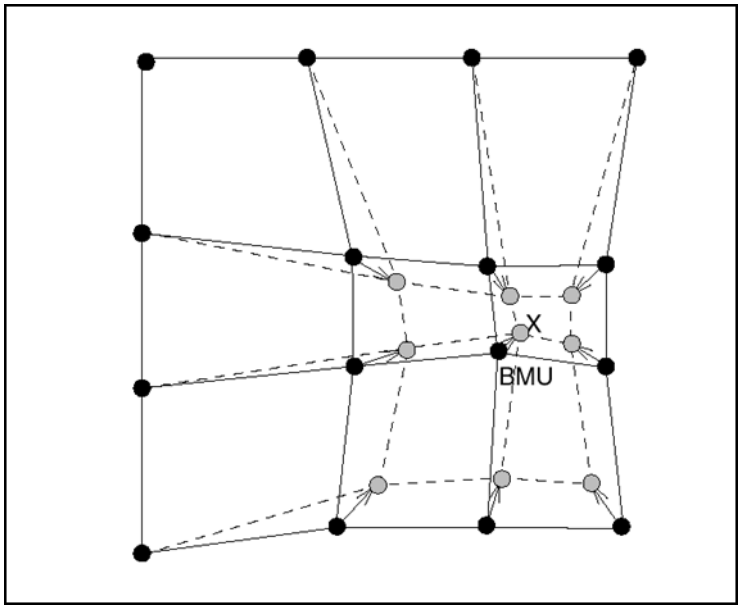

Figura 3.- Actualización del mapa Auto-Organizativo.

Algunas de las características principales que ha de tener el mapa son: la disposición alargada para que refleje direcciones prioritarias y que sea de un tamaño razonable para que no se concentren excesivos elementos en una misma celda.

3.2. Análisis de componentes principales

El Análisis de Componentes Principales (ACP) es una técnica estadística de síntesis de la información, o reducción de la dimensión (número de variables). Es decir, el objetivo es reducir el número de variables perdiendo la menor cantidad de información posible. Los nuevos componentes principales o factores serán una combinación lineal de las variables originales, y además serán independientes entre sí.

Estas técnicas fueron inicialmente desarrolladas por Pearson a finales del siglo XIX y posteriormente fueron estudiadas por Hotelling en los años 30 del siglo XX. Para estudiar las relaciones que se presentan entre *p* variables correlacionadas (que miden

información común) se puede transformar el conjunto original de variables en otro conjunto de nuevas variables incorreladas entre sí (que no tenga repetición o redundancia en la información) llamado conjunto de componentes principales.

Las nuevas variables son combinaciones lineales de las anteriores y se van construyendo según el orden de importancia en cuanto a la variabilidad total que recogen de la muestra.

De modo ideal, se buscan $m < p$ variables que sean combinaciones lineales de las p originales y que estén incorreladas, recogiendo la mayor parte de la información o variabilidad de los datos.

Si las variables originales están incorreladas de partida, entonces no tiene sentido realizar un análisis de componentes principales.

El análisis de componentes principales es una técnica matemática que no requiere la suposición de normalidad multivariante de los datos, aunque si esto último se cumple se puede dar una interpretación más profunda de dichos componentes.

Sea $X = [X_1, \dots, X_p]$ una matriz de datos multivariantes. Lo que sigue también vale si X es un vector formado por p variables observables. Las componentes principales son unas variables compuestas incorrelacionadas tales que unas pocas explican la mayor parte de la variabilidad de X. Las componentes principales son las variables compuestas:

$$Y_1 = Xt_1, Y_2 = Xt_2, \dots, Y_p = Xt_p \tag{3}$$

tales que:

1. $var(Y_1)$ es máxima condicionado a $t_1't_1 = 1$.
2. Entre todas las variables compuestas Y tales que $cov(Y_1, Y) = 0$, la variable Y_2 es tal que $var(Y_2)$ es máxima condicionada a $t_2't_2 = 1$.
3. Si $p \geq 3$, la componente Y_3 es una variable incorrelacionada con Y_1, Y_2 con varianza máxima.
4. Análogamente se definen las demás componentes principales si $p > 3$.

Si $T = [t_1, t_2, \dots, t_p]$ es la matriz $p \times p$ cuyas columnas son los vectores que definen las componentes principales, entonces la transformación lineal $X \to Y$.

$$Y = XT \tag{4}$$

se llama transformación por componentes principales.

Siendo $t_1, t_2, \dots, t_p$ los p vectores propios normalizados de la matriz de covarianza S.

$$St_i = \lambda_i t_i\,,\ \ t_1't_1 = 1,\ \ i = 1, \dots, p \tag{5}$$

Entonces:

1. Las variables compuestas $Y_1 = Xt_1, o = 1, \dots, p,$ son las componentes principales.
2. Las varianzas son los valores propios de *S*.

$$var(Y_1) = \lambda_i\,, \quad i = 1, \dots, p \tag{6}$$

3. Las componentes principales son variables incorrelacionadas:

$$cov(Y_1, Y) = 0,\ i \neq j = 1, \dots, p \tag{7}$$

3.3. Métodos VAT e iVAT

Los métodos VAT e iVAT [17] son técnicas de análisis preliminar de datos relacionados con el problema de análisis de datos de agrupamiento o clustering, donde se realiza la partición de un conjunto de objetos $O = \{o_1, o_2, \dots, o_n\}$ en *c* subconjuntos de características similares. El tipo de agrupamientos formados está fuertemente relacionado por el modelo matemático del agrupamiento, ya que los agrupamientos presentan ciertas características geométricas. Todos los algoritmos de clustering $(1 \leq c \leq n)$ encuentran un número arbitrario de clustering, incluso si no existe un "cluster" actual.

Estas técnicas son el paso previo antes de aplicar técnicas de agrupamiento de saber cuál es el número de clusters ideal o una aproximación se denomina "assessing of clustering tendency". Para realizar este proceso existen varias técnicas basadas en la estadística y algún método más informal [27] [32]. Pero ninguna de estas aproximaciones llega a ser totalmente satisfactoria. El objetivo de estos métodos es proporcionar un sencillo e intuitivo método de aproximación visual a las herramientas de evaluación de tendencias.

Los métodos de aproximaciones visuales para los problemas de análisis de datos han sido ampliamente estudiados en los últimos 25 años. Estas técnicas son ampliamente utilizadas en todos aquellos casos en los que se trabaja con datos numéricos. El método VAT representa la información de disimilitud sobre un conjunto de objetos $O = \{o_1, o_2, \dots, o_n\}$, como una imagen digital cuadrada con n^2 píxeles, con los objetos adecuadamente reordenados de manera que en la imagen se resalte la estructura de grupos potencial.

En análisis de cluster convencional [18] (datos-objeto), los objetos se separan en grupos de acuerdo a sus características $\overrightarrow{x_i} \epsilon \mathbb{R}^p$, donde $\overrightarrow{x_i}$ es el vector de característica p-dimensional del i-ésimo objeto y cada elemento de $\overrightarrow{x_i}$ es una característica numérica del

objeto. Una forma alternativa de representar los datos son los datos relacionales, donde sólo la relación entre pares de objetos se conoce. Los datos relacionales generalmente constan de los valores de N^2 de una matriz de disimilitud *D*, donde $D_{i,j} = d(o_i, o_j)$, es el par de disimilitud o distancia entre los objetos o_i y o_j.

Aunque el proceso de clustering es entendido típicamente en cómo solamente el acto de separar los objetos en los grupos adecuados, el análisis de cluster en realidad consta de tres preguntas concisas:

i) Tendencia del clustering: ¿el número de grupos que hay?
ii) La partición: ¿Qué objetos pertenecen a cada clustering y en qué medida?
iii) Validación del clustering: ¿Las particiones que se han realizado son correctas?

Teniendo en cuenta que en muchos de los algoritmos de clustering el número de grupos que se van a crear es un elemento de entrada, el análisis de tendencia es una parte muy importante en el análisis de cluster. El método VAT se ocupa de esta cuestión reordenando la matriz de disimilitud *D* de modo que, idealmente, el número de grupos se muestra como el número de "bloques oscuros" a lo largo la diagonal. Recientemente, un algoritmo mejorado de este método VAT (iVAT) ha sido propuesto donde se realiza una transformación de la matriz *D*, empleando una transformación gráfico teórica de la distancia.

Se considera que Petrie, en el año 1899, fue el primero en utilizar la matriz de permutación para estudiar las tendencias en la medición de los datos. Sin embargo, en 1909, Czekanowski presentó el primer método para la agrupación de datos de disimilitud utilizando una aproximación visual.

El algoritmo VAT muestra una imagen reordenada de los datos de la matriz de disimilitud a escala. Cada píxel de la escala de grises de la imagen VAT muestra el valor de disimilitud de escala entre dos objetos. Los píxeles blancos destacan por su gran desemejanza, mientras que los negros representan una baja disimilitud. Siendo cada objeto exactamente similar con sí mismo, lo que se traduce en un valor cero (color negro), que son los elementos de la diagonal en *I (D *)*. Los elementos fuera de la diagonal de *I (D *)* son escalados al rango *[0, 1]*. Un cuadrado oscuro a lo largo de la diagonal de *I (D *)* es una sub-matriz de disimilitud de valores "igualmente reducido", por lo que un bloque oscuro representa un grupo de objetos que son relativamente similares entre sí. Por lo tanto, la tendencia del clúster se muestra por el número de bloques oscuros a lo largo de la diagonal de la imagen de VAT.

El algoritmo de VAT está basado en el algoritmo de Prim, que trata de encontrar el árbol de expansión mínimo (mínimum spanning tree, MST) de los pesos conectados gráficamente. El objetivo de encontrar la MST en *D* es para obtener la secuencia de los

índices en el que los bordes se añaden al árbol. Posteriormente, los índices son utilizados para realizar la reordenación de *D*, en particular, el MST no se corta, como en un solo vínculo agrupación, para encontrar particiones de los datos. La matriz de disimilitud de VAT reordenada resultante *D* * puede ser normalizado y asignado a una imagen en escala de grises con negro que representa la disimilitud mínimo y blanco el máximo. Los pasos del algoritmo se pueden resumir como:

1. Seleccionar $(i,j)\epsilon \arg\max_{p\in K, q\in K} D_{pq}$.
2. Establecer $P(1) = i$; I $= \{i\}$; yJ $=$ K $- \{i\}$.
3. Realizar un bucle donde $r = 2,\ldots,N$ y se selecciona $(i,j)\epsilon \arg\min_{p\in J, q\in J} D_{pq}$ y se establece $P(\mathrm{r}) = j$; Reemplazando I $\leftarrow \mathbf{I} \cup \{\mathbf{j}\} = \{i\}$ y J $\leftarrow \mathbf{J} - \{\mathbf{j}\}$.
4. Se obtiene la matriz de disimilitud *D** ordenando el array *P* como: $D^*_{pq} = D_{P_{(p)},P_{(q)}}$ para $1 \le p,q \le N$. La salida es la matriz reordenada *D**.

El algoritmo iVAT (improve VAT), emplea una medida de la distancia basada en rutas. Considerando *D* una matriz que representa los pesos de los bordes de una matriz gráfica totalmente conectada. La distancia basada en rutas se define como:

$$D'_{ij} = min_{p\in P\_ij} max_{1\le h<|p|} D_{p[h]p[h+1]} \quad (8)$$

Donde $P \in P_{ij}$ es un camino acíclico en el conjunto de todos los caminos acíclicos entre el vértice $i(o_i)$ y el vértice $j(o_j)$, $p[h]$ es el índice del h-ésimo vértice del camino *p*, y $|p|$ es el número de vértices a lo largo del camino. Por lo tanto, $D_{p[h]p[h+1]}$ es el peso del borde h-iésimo a lo largo de la trayectoria *p*. En esencia, el costo de cada trayectoria *P*, para la distancia en la ecuación (8), es el peso máximo de sus $|p|$ bordes. La distancia entre *i* y *j* es el mínimo coste del camino en P_{ij}.

Las imágenes obtenidas a través del método iVAT muestran una considerable mejoría con respecto a las imágenes obtenidas con el método VAT para casos "difíciles".

Para realizar el algoritmo iVAT se empieza por aplicar el algoritmo VAT a la matriz de disimilitud de entrada y a continuación se reordenan los datos de la matriz VAT de salida en una matriz de iVAT a través de un algoritmo recursivo. Los pasos del algoritmo se pueden resumir como:

1. Datos de entrada: *D** Matriz VAT reordenada de disimilitud.
2. Realizar un bucle donde $r = 2,\ldots,N$ y se selecciona $j = \arg\min_{k=1,\ldots,r-1} {D^*}_{rk}$, ${D'^*}_{rk} = {D^*}_{rk}, c = j$ y ${D'^*}_{rc} = max\left({D'^*}_{rj}, {D'^*}_{jc}\right), c = 1,\ldots,r-1, c \ne j$.

Siendo la matriz D'^* simétrica.

El algoritmo anterior muestra que la imagen iVAT puede obtenerse aplicando un algoritmo recursivo sobre la imagen VAT.

3.4. Mapeo de Sammon

A menudo es necesario reducir la dimensionalidad de un conjunto de datos, con el fin de poder realizar un análisis computacional lógico, o para facilitar la visualización. En el proceso de reducción de la dimensionalidad es aconsejable mantener, la geometría de las relaciones entre los puntos de datos originales se deben dejar intacta en la mayor medida posible.

La técnica más simple para la reducción de la dimensionalidad es una sencilla proyección lineal, como en el análisis de componentes principales (PCA). Mientras que esto maximiza el valor de la varianza original presente en el conjunto de datos transformado, pero no mantendrá la estructura compleja de los datos. Con el mapeo de Sammon se quiere representar el grado en el cual una estructura compleja se conserva por la transformación de reducción de la dimensionalidad. Recibe el nombre por John Sammon, Jr.

La medida empleada por el mapeo de Sammon está diseñada para reducir al mínimo las diferencias entre las distancias entre puntos correspondientes a los dos espacios. Una transformación se considera como preferente si se conserva, en la mayor medida de lo posible, la distancia entre cada par de puntos. Además, se trata de asegurar que el mapeo no afecta a la topología, es decir, se acentúa la importancia de la preservación de las relaciones entre los lugares.

A diferencia de las técnicas de reducción de dimensionalidad lineales tradicionales (como ACP), el mapeo Sammon no representa explícitamente la función de transformación. En su lugar, simplemente proporciona una medida de qué tan bien el resultado de una transformación refleja la estructura presente en el conjunto de datos original. En otras palabras, no se trata de obtener una asignación óptima de aplicar a los datos originales, sino más bien de construir un nuevo conjunto de datos de menor dimensión, que tiene una estructura lo más similar posible al primer conjunto de datos.

El conjunto de datos original se representa como N vectores en un espacio de L-dimensiones, dados por $X_i,\ i = 1,2,\dots,N$. Estos datos se tratan de asignar en el espacio d-dimensional (con $d<L$), para los vectores dado $Y_i,\ i = 1,2,\dots,N$. Por simplicidad, se escribe d_{ij}, como la distancia entre pares Y_i e Y_j, y del mismo modo $d^*{}_{ij}$ como la distancia entre X_i y X_j. Se asume que la medida de la distancia es euclídea, pero se podrían emplear otros métodos.

Se realiza una medida del error de la cantidad de estructura presente originalmente, pero que se perdió al realizar la transformación en el conjunto de datos, este error (*E*), se define como:

$$E = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j}^{n} \frac{\left(d^*_{ij} - d_{ij}\right)^2}{d^*_{ij}} \tag{9}$$

Esencialmente el error se obtiene mediante la suma de las diferencias al cuadrado (antes y después de la transformación) en las distancias entre pares de puntos. Cada diferencia entre puntos sólo es contado una vez de ahí la presencia del término $i<j$ en el sumatorio. La tendencia a preservar la topología es debida al término d^*_{ij} presente en el denominador del sumatorio principal, lo que garantiza que si la distancia original entre los puntos es pequeña, la ponderación dada a su diferencia al cuadrado es mayor.

El error *E* nos proporciona una medida de la calidad de transformación de cualquier conjunto de datos, sin embargo, hay que determinar el óptimo de tales datos en términos de minimizar *E*. El mapeo de Sammon se define como la transformación óptima. En el documento [19], Sammon describe un método para realizar la optimización. El conjunto de datos transformado Y_i, se inicializa primero mediante la realización de un análisis de componentes principales a los datos originales. Posteriormente se actualiza este valor de Y_i, usando un valor ascendente, teniendo en cuenta el gradiente de *E* con respecto a Y_i, hasta que se logra satisfacer la convergencia.

Este método de optimización presenta una dimensionalidad bastante alta, por lo que pueden aparecer problemas como la presencia de mínimos locales.

4. Algoritmos de clustering

Los algoritmos de clustering se pueden clasificar según el criterio o la filosofía empleada. Un resumen de las distintas clasificaciones sería el siguiente:

1) Basada en el uso o no de la función criterio, y pueden ser:
   a. Directos, constructivos o heurísticos, los cuales no optimizan ninguna función criterio.
   b. Indirectos o por optimización, que usan una función criterio a optimizar.

2) Según la filosofía empleada para la construcción de agrupamientos.
   a. Aglomerativos o incrementales ("bottom-up"). Parten de patrones aislados y tienden a unir agrupamientos de acuerdo a algún umbral fijado.
   b. Divisivos o decrementales ("top-down"). Parten de agrupamientos ya establecidos y tienden a crear agrupamientos más homogénos.
   c. Mixtos. Incorporan procesos de creación y mezcla de nuevos agrupamientos.

3) Los que están basados en si se conoce o no a priori el número de agrupamientos.
   a. Algoritmos que desconocen el número de clusters que deben generar.
   b. Algoritmos que conocen cuantos agrupamientos deben generar.

Una clasificación de los algoritmos de clustering más conocidos es:

A. Algoritmos de clustering con número de clases desconocido. Entre los que se encuentran:
   - Método adaptativo.
   - Algoritmo de Batchelor y Wilkins.
   - Algoritmo jerárquico.

B. Algoritmos de clustering con número de clases conocidos:
   - Algoritmo K-means.
   - Algoritmo de agrupamiento secuencial.

C. Método basado en grafos.
   - Algoritmo basado en la matriz de similitud.

La calidad del resultado del algoritmo depende de varios factores como son:

i. El algoritmo concreto empleado para encontrar los clusters. Existe una gran cantidad de algoritmos de agrupamiento, esta variedad provoca que la elección de una estrategia de agrupamiento, e incluso entre diferentes implementaciones de un mismo algoritmo proporcione resultados diferentes.
ii. El valor de los parámetros del algoritmo, ya que en algunos algoritmos la elección de los valores adecuados resulta difícil. Otros algoritmos presentan

muchos parámetros y en algunos casos el valor de estos depende de las unidades en las que se cuantifican las variables.

iii. Los patrones utilizados y en algunas ocasiones el orden en que se procesan. Muchos algoritmos utilizan solo una parte de los patrones disponibles (caracterización de los agrupamientos). Usualmente se emplea alguna técnica de muestreo para esta selección. Con frecuencia diferentes permutaciones del mismo conjunto de patrones dan lugar a diferentes resultados. Muchos algoritmos están sesgados (determinados) por los primeros patrones que se procesan, lo que hace que para una misma permutación, el orden en que se presentan determina el resultado final.

iv. La medida de similitud adoptada. La medida de similitud requiere especificar alguna medida de distancia y un valor umbral asociado a ésta. Diferentes criterios de distancia proporcionarán diferentes resultados, al igual que diferentes umbrales para un mismo criterio de distancia.

La dificultad inherente a la aplicación práctica de los algoritmos de agrupamiento, se puede resumir en los dos puntos siguientes:

a) Establecer medidas de similitud adecuadas.
b) Establecer buenos criterios de partición.

Como se puede observar, existe una gran diversidad de algoritmos de clustering que permiten determinar los clusters, por lo que en el desarrollo de este proyecto se van a incluir tres de los más representativos: K-means, Adaptativo y Jerárquico.

4.1. Algoritmo K-means

Pese a que el algoritmo K-means es conocido desde hace más de 5 décadas sigue siendo uno de los algoritmos de aprendizaje automático más utilizados y mejor valorados [21]. Dado un conjunto de datos y un valor de *K*, el algoritmo genera una partición de los datos con *K* clusters. El objetivo del algoritmo es minimizar el error cuadrático medio de todos los clusters. El error cuadrático de un cluster se define como la media de los cuadrados de las diferencias de los objetos de un cluster al centroide del cluster. El centroide de un cluster es el punto medio de todos los objetos del cluster.

Este algoritmo presenta el inconveniente de que devuelve un óptimo local. No obstante, estudios recientes demuestran que si los clusters están bien separados el algoritmo K-means obtendría el óptimo global con alta probabilidad.

Este algoritmo también se conoce como algoritmo de las medias móviles porque en cada iteración se recalculan los centros de los agrupamientos, a esta función hay que darle como parámetro el número de agrupamientos (*K*) que debe encontrar. El proceso de funcionamiento del algoritmo es el siguiente:

a) En primer lugar se inicializan arbitrariamente los centros de los k grupos.

b) Posteriormente se calcula la distancia de los centros a las distintas imágenes y se reagrupan estas si encuentran un centro que esté más cercano, y se recalculan los centros.
c) Se repite el algoritmo hasta que no se haya producido cambios en la ubicación de los centros, entonces se considera que se ha conseguido una buena partición y se termina el programa.

Dado que al aumentar el valor de *K* [22] el error cuadrático medio se reduce (siendo 0 para *K* igual al número de patrones (*N*) este puede ser minimizado sólo para un valor prefijado de *K*. Dicho de otro modo, el error cuadrático medio no sirve para comparar la bondad de particiones con distinto número de clusters.

El algoritmo K-means requiere como parámetro de entrada el número de clustering, que es *K*, el número de clusters de la partición. Un método habitual es ejecutar el algoritmo en repetidas ocasiones con diferentes valores de *K* y emplear algún criterio para seleccionar el más adecuado.

Existen multitud de variantes del algoritmo K-means, como por ejemplo el Fuzzy c-means que es una variante del K-means para clustering difuso. Otra variante, llamada X-means, selecciona automáticamente el valor adecuado de *K*. También existen variantes para reducir el coste computacional de K-means o para reducir la cantidad de memoria necesaria.

4.2. Algoritmo Adaptativo

Este método es útil cuando no se conoce de antemano el número de clases del problema (número de clusters desconocido).

Se trata de un método heurístico o incremental, este método requiere dos parámetros de entrada tau (τ) que es el umbral de distancia para crear agrupamiento y theta (θ) que es la fracción de τ que determina total confianza.

El funcionamiento del algoritmo es el siguiente:

a) En el proceso de inicialización se crea el primer agrupamiento que contiene el primer patrón.
b) Creación de una tabla de estados, para ello se crean agrupamiento evaluando para cada patrón la distancia hacia los grupos ya creados.
c) Reasignación hasta estabilización, se vuelven a procesar los distintos patrones, pudiéndose cambiar de estado debido a las actualizaciones de los centros de los agrupamientos. Este proceso se repite hasta que no haya cambios de estado. Al empezar este paso se fija la variable estable a verdad, si se produce alguna modificación se cambia a falso y se fuerza a que se realice una nueva iteración.
d) Los patrones restantes se agrupan según la regla de la mínima distancia.

El algoritmo incluye una clase de rechazo, para aquellos patrones que se encuentran fuera de la zona de asignación. Un ejemplo de las distintas zonas que calcula dicho algoritmo se muestra en la figura siguiente:

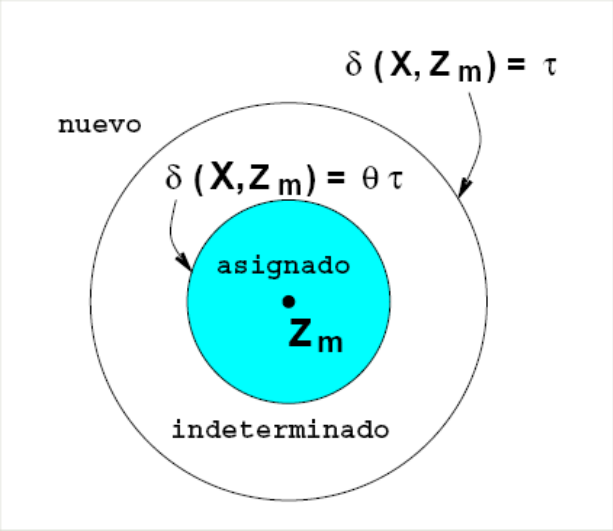

Figura 4.- Explicación algoritmo adaptativo.

Un ejemplo de como funciona el algoritmo se muestra con las siguientes figuras donde se parte de unos patrones iniciales y se termina realizando el agrupamiento.

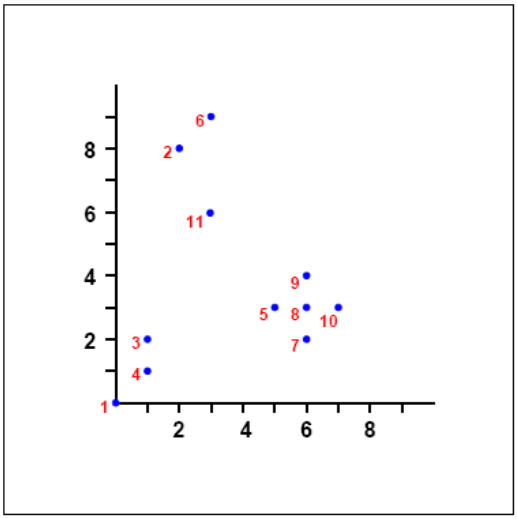

Figura 5.- Ejemplo algoritmo adaptativo datos de partida.

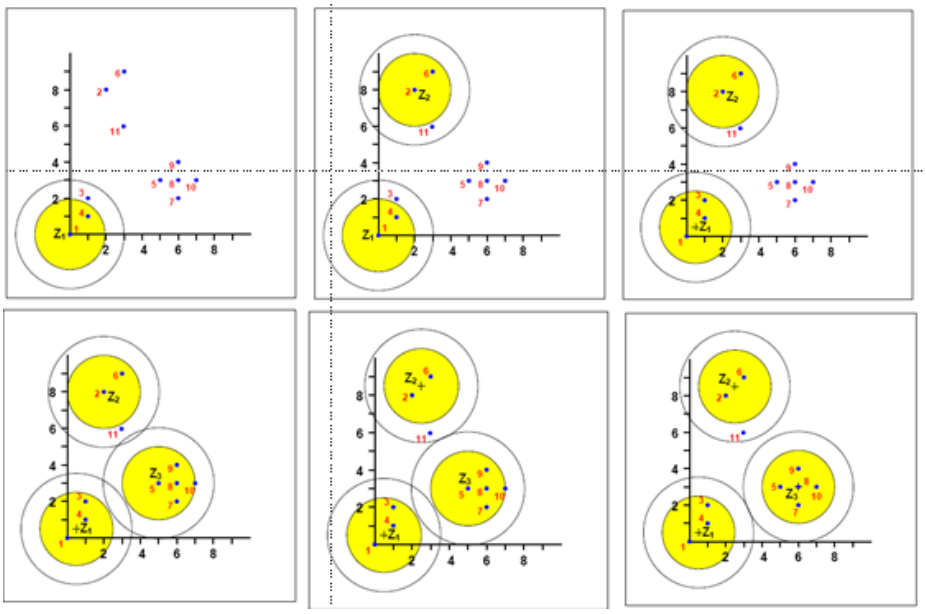

Figura 6.- Ejemplo algoritmo adaptativo desarrollo.

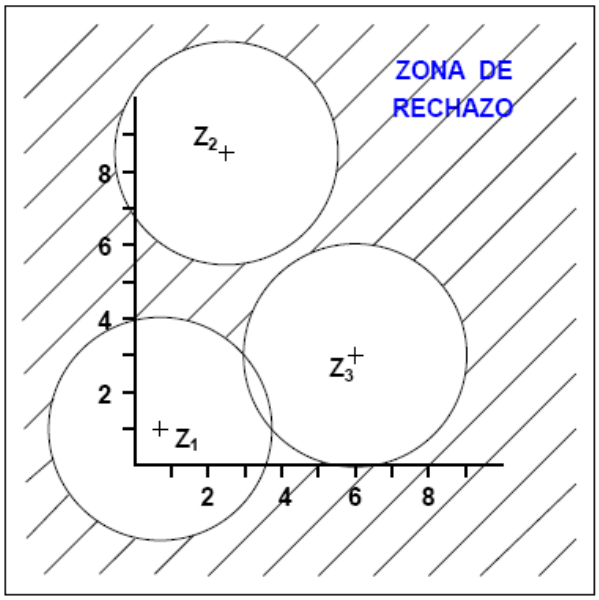

Figura 7.- Ejemplo algoritmo adaptativo final.

Las ventajas de emplear este algoritmo [23] son:

- Simple y eficiente, ya que los cálculos son sencillos.
- Mínimo almacenamiento, ya que se produce un procesamiento secuencial de los patrones.
- No requiere un número de agrupamientos.

Por el contrario presenta los siguientes inconvenientes:

- Hace falta un almacenamiento adicional para registrar el estado de los patrones.
- Supone agrupamientos compactos y separados claramente de los demás.
- Esta sesgado por los primeros patrones.
- Depende del orden de presentación.
- Depende del valor seleccionado para τ y θ.

Este algoritmo también presenta las siguientes desventajas:

- Dependencia del orden de presentación (comportamiento sesgado por el orden de presentación de los patrones).
- Presupone agrupamientos compactos separados claramente de los demás (puede no funcionar adecuadamente en presencia de ruido).

4.3. Algoritmo Jerárquico

Los algoritmos jerárquicos utilizan la matriz de distancia como criterio de clustering, estos algoritmos no necesitan saber el número de cluster que deben usar, pero si una condición de término. Existen dos tipos de algoritmos jerárquicos: aglomerativo y divisivo.

El algoritmo aglomerativo funciona de abajo hacia arriba y mezcla clusters iterativamente, este algoritmo comienza situando cada patrón u objeto por separado, y va mezclando estos clusters atómicos en clusters cada vez mayores, hasta que los objetos están en un único cluster.

Los cluster se van agrupando en función de la similitud existentes entre ellos, para ello se evalúa la distancia entre los distintos clusters y aquellos dos más cercanos se fusionan formando uno sólo, hasta que estén todos agrupados. Cuando se une un cluster con otro atómico o no, se crea un nuevo centro a partir de la media de los dos centros anteriores. Se parte de una matriz con los datos de todos los patrones y en cada iteración dos patrones o centros son sustituidos por un nuevo centro reduciéndose en una unidad el número de filas de dicha matriz.

El algoritmo divisivo va desde arriba hacia abajo, es decir, divide un cluster iterativamente. Este algoritmo realiza el proceso inverso que el algoritmo jerárquico aglomerativo. Comienza con todo el conjunto de datos como si fuese un único cluster y lo va dividiendo sucesivamente hasta que llega a un conjunto de particiones menores.

En los métodos de clustering jerárquicos [24] se crea un árbol o dendrograma. El árbol no es un conjunto de grupos, sino más bien una jerarquía de varios niveles, donde los grupos en un nivel se unen como grupos en el siguiente nivel. Esto le permite decidir el nivel o escala de agrupación que es el más apropiado para cada caso. Una representación de un árbol de clúster se muestra en la siguiente figura:

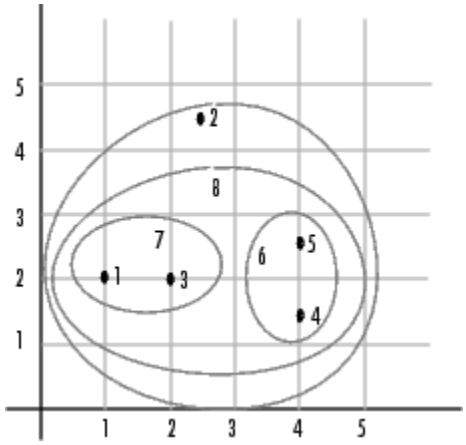

Figura 8.- Representación de patrones

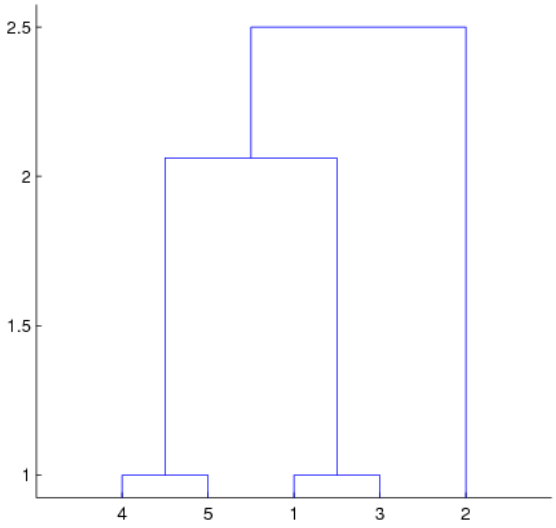

Figura 9.- Ejemplo de dendrograma.

Los patrones se agrupan en función de los links existentes entre objetos, dichos links pueden ser:

- Single Link: Une en cada paso los pares de objetos más similares que no se encuentran todavía en el mismo cluster.
- Complete Link: Utiliza el par menos similar entre cada par de clusters para determinar similitudes entre clusters. Todos los objetos de un cluster se linkan con otros que caigan dentro de un rango de similitud.

5. Índices y validación

Debido a que para un mismo conjunto de objetos, aplicando diferentes algoritmos de agrupamiento se pueden obtener resultados muy diferentes, surge la necesidad de evaluar los agrupamientos realizados, para darle al usuario una idea del grado de eficacia de los resultados obtenidos.

Las medidas de evaluación o índices se espera que sean objetivas y no tengan ninguna preferencia sobre algún algoritmo en particular. Sin embargo, teniendo en cuenta la variedad y la propia ambigüedad que existe en la definición de los distintos algoritmo de agrupamiento, surgen preguntas al respecto como son: ¿Podrá un índice de validación decir cuándo una estructuración de los datos es correcta, o es mejor que otra?, ¿Se podrá confeccionar un índice de validación capaz de trabajar de manera imparcial con cualquier tipo de estructuración de los datos?, ¿Qué es lo que realmente se le puede exigir a un índice de validación?

Existen tres categorías de índices de validación [9]: índices externos, índices relativos e índices internos. Esta clasificación se muestra en la figura siguiente:

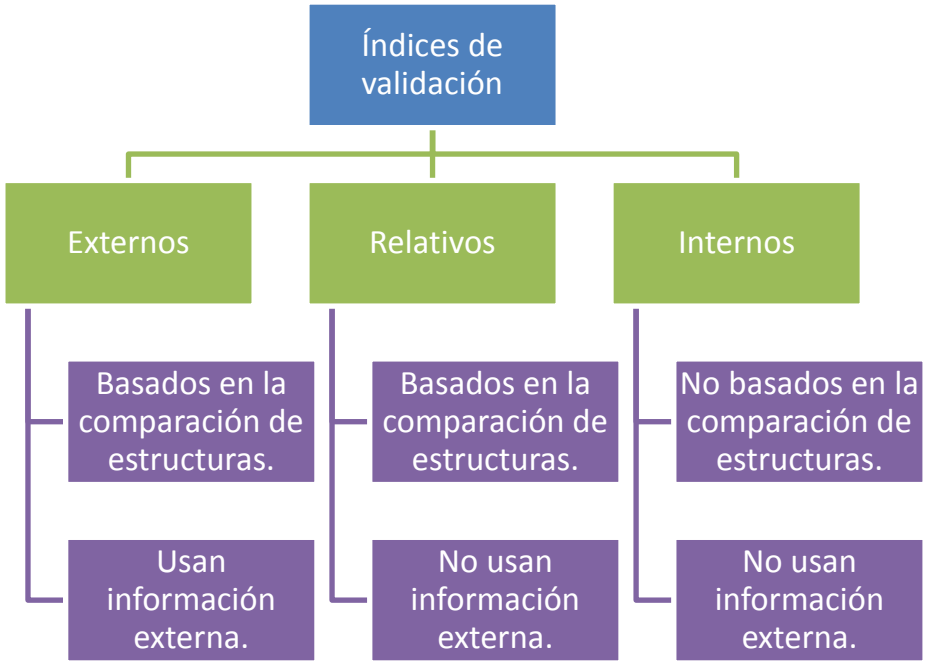

Figura 10.- Clasificación índices.

Los índices externos usan como patrón para compararse, una estructura específica, la cual es obtenida a partir de una información previa acerca de los datos, donde este patrón es visto como la estructura real o verdadera (ground-truth). El agrupamiento obtenido será mejor en la medida que éste se parezca más a dicho patrón. Entre los

índices externos más usados se encuentran la medida F (F-measure), el índice de Rand, el coeficiente de Jaccard y el índice de Minkowski.

Los índices relativos no dependen de información externa, sino que están basados en el cálculo de la consistencia de los clasificadores, comparando las agrupaciones obtenidas a partir de un mismo algoritmo bajo condiciones diferentes, con el objetivo de decidir cuál revela mejor las características reales de los objetos. Entre estas medidas podemos encontrar al llamado Figura de Mérito (Figure of Merit, FOM) y el índice de Estabilidad.

Los índices internos se basan solamente en la información del propio agrupamiento, no requieren de información adicional ni la repetición del proceso de agrupamiento. Un gran número de estos índices asumen que las agrupaciones obtenidas por los algoritmos de clustering deben cumplir que los objetos en un mismo agrupamiento deben estar más cerca que los objetos de clusters diferentes, por tanto evalúan la compacidad u homogeneidad de cada agrupamiento y la separación de cada uno de los agrupamientos entre sí, por ejemplo: el índice de validación SD, los basados en teoría de grafos, el índice de Davies-Bouldin y el Silhouette.

No es posible crear un índice de validación capaz de trabajar de manera imparcial con cualquier algoritmo de clustering. Cada algoritmo impone una estructura sobre los datos, si la propiedad que mide cierto índice no está acorde con la estructura impuesta por él, el índice va a valorar la clasificación como mala, incluso cuando dicha clasificación tenga sentido y brinde información relevante para un problema en particular. Por tanto se podría decir que dado un índice de validación, siempre se puede buscar una agrupación dada para la cual este índice no va a funcionar bien.

Precisamente esta imposibilidad de decidir con toda certeza, a partir de un conjunto de clusters, cuál es la mejor, es uno de los motivos por el cual surge la necesidad de combinar agrupaciones de clusters como una variante de solución al problema de la clasificación no supervisada, en lugar de generar múltiples particiones y evaluarlas utilizando un índice y seleccionar la mejor.

5.1. Índice de Davies-Bouldin

Este índice presenta una medida de la similitud de los grupos que se supone que tienen una densidad de datos, que es una disminución de función de la distancia a partir de un vector característico de la agrupación. Este índice se emplea para deducir la idoneidad de las particiones de datos y por lo tanto puede ser utilizado para comparar la conveniencia relativa de las diversas divisiones de los datos. El valor del índice obtenido no depende del número de grupos analizados ni el método de partición de los datos.

La formulación teórica de cómo se obtiene el índice se muestra a continuación [11], se parte de que una separación de clusters debe poseer los siguientes atributos:

1. Se requiere poca o ninguna interacción con el usuario o la especificación de parámetros.
2. Se aplica a los conjuntos de datos jerárquicos.
3. Debe ser computacionalmente factible para conjunto de datos relativamente grande.
4. Se debe dar resultados significativos para los datos de arbitraria dimensionalidad.

A partir de estos atributos se pueden obtener las siguientes definiciones:

- Definición 1: Una función de valor real se dice que es una distancia métrica si cumplen las siguientes propiedades:

$$\begin{gathered} 1\ d(X_i, X_j) \geq 0\ \forall X_i, X_j \in E_p \\ 2\ d(X_i, X_j) = 0\ si\ X_i = X_j \\ 3\ d(X_i, X_j) = d(X_j, X_i)\ \forall X_i, X_j \in E_p \\ 4\ d(X_i, X_j) \leq d(X_i, X_k) + d(X_k, X_j)\ \forall X_i, X_j, X_j \in E_p \end{gathered} \tag{10}$$

  donde E_p es un espacio Euclídeo de *p*-dimensiones.

- Definición 2: Una función de valor real se dice que es una dispersión media si cumplen las siguientes propiedades: teniendo un cluster *C* con miembros *X1, X2, ... , Xm* ∈ *Ep*.

$$\begin{gathered} 1\ S(X_1, X_2, \dots, X_m) \geq 0 \\ 2\ S(X_1, X_2, \dots, X_m) = 0\ si\ X_i = X_j\ \forall X_i, X_j \in C \end{gathered} \tag{11}$$

  El objetivo es definir una medida general de separación de clúster, $R\ (S_i, S_j, M_{ij})$, lo que permite el cálculo de la similitud promedio de cada grupo con sus grupos más similares.

- Definición 3: Se puede medir el valor real de similitud de un cluster si se cumplen las siguientes propiedades:

$$\begin{gathered} 1\ R(S_i, S_j, M_{ij}) \geq 0 \\ 2\ R(S_i, S_j, M_{ij}) = R(S_j, S_i, M_{ji}) \\ 3\ R(S_i, S_j, M_{ij}) = 0\ si\ S_i = S_j = 0 \\ 4\ Si\ S_j = S_k\ y\ M_{ij} \leq M_{ik} => R(S_i, S_j, M_{ij}) > R(S_i, S_k, M_{ik}) \\ 5\ Si\ M_{ij} = M_{ik}\ y\ S_j \geq S_k => R(S_i, S_j, M_{ij}) > R(S_i, S_k, M_{ik}) \end{gathered} \tag{12}$$

  donde M_{ij} es la distancia entre los vectores característicos de los clusters *i* y *j*, y S_i y S_j son las dispersiones de las agrupaciones *i* y *j*, respectivamente. Esta definición impone ciertas limitaciones en *R*, que son arbitrarias pero heurísticamente significativas. Estas son:

1. La función de similitud *R* es no negativa.
2. Tiene la propiedad de la simetría.
3. La similitud entre distintas agrupaciones es 0 sólo si sus funciones de dispersión se anulan.
4. Si la distancia entre clusters se incrementa mientras, mientras que sus dispersiones permanecen constantes, la similitud entre dos agrupaciones decrece.
5. Si la distancia entre los clusters permanece constante mientras aumenta la dispersión, la similitud aumenta.

Las condiciones de las definiciones 2 y 3 son mínimas necesarias condiciones. Estas propiedades sugieren que *R* se formó con algunas funciones auxiliares $F(S_i, S_j)$ y $G(M_{ij})$ en una relación de reciprocidad. La siguiente función satisface los criterios requeridos y se reduce a cierta similitud familiarizado medidas para las opciones especiales de medidas de dispersión, la distancia medida y los vectores característicos.

- Definición 4: Viene dada por la siguiente ecuación:

$$R_{ij} = \frac{S_i + S_j}{M_{ij}} \tag{13}$$

- Definición 5: El valor de $\bar{R}$ se define como:

$$\bar{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \tag{14}$$

donde R_i es el máximo de R_{ij} para $i \neq j$.

Este índice es pequeño cuando los clusters son compactos y están lejos el uno del otro, por lo que el resultado del índice será pequeño cuando se tenga un buen resultado clustering.

5.2. Índice de Dunn

El índice de Dunn [12] fue introducido por J.C. Dunn en el año 1974 pertenece al grupo de índices de validez que al igual que el índice de Davies-Bouldin, se trata de un esquema de evaluación interna, donde el resultado se basa en los datos en sí agrupados. El objetivo de esta clase de índices es identificar la compacidad de los clusters, estudiando pequeñas variaciones entre los miembros de la agrupación, y bien separados, cuando la media de los diferentes grupos están suficientemente separados, en comparación con la varianza dentro del grupo. Para una asignación dada de grupos, un índice de Dunn mayor indica una mejor agrupación. Uno de los inconvenientes de la utilización de este, es el coste computacional como el número de grupos y la dimensionalidad de los datos de aumento.

Hay muchas maneras de definir el tamaño o diámetro de un clúster. Esto podría ser la distancia entre los dos puntos más alejados dentro de un clúster, ser la media de todas las distancias por pares entre los puntos de datos dentro del clúster, o podría ser la distancia de cada punto de datos hacia el centroide de clúster. Cada una de estas formulaciones se muestran matemáticamente a continuación:

Sea C_i ser un grupo de vectores y sean *x* e *y* dos *n* dimensional vectores cualesquiera de características asignadas al mismo C_i clúster.

$$\Delta_i = \max_{x,y \in C_i} d(x,y) \tag{15}$$

La ecuación anterior calcula la máxima distancia y la ecuación siguiente calcula la distancia media entre todos los pares:

$$\Delta_i = \frac{1}{|C_i||C_i-1|} \sum_{x,y \in C_i, x \neq y} d(x,y) \tag{16}$$

La siguiente ecuación calcula la distancia de todos los puntos hasta el punto medio:

$$\Delta_i = \frac{\sum_{x \in C_i} d(x,\mu)}{|C_i|}, \mu = \frac{\sum_{x \in C_i} x}{|C_i|} \tag{17}$$

Para una partición de clusters, donde C_i representa eli-cluster de la partición, el índice de Dunn, *D*, puede ser calculado por la siguiente formula:

$$D = \min_{1 \leq j \leq n} \left\{ \min_{1 \leq j \leq \binom{n}{*}} \left\{ \frac{d(C_i,C_j)}{\max_{1 \leq k \leq n} d'_{(C_k)}} \right\} \right\} \tag{18}$$

donde $d(c_i, c_j)$ es la distancia entre los clusters c_i y c_j (distancia entre clusters); $d'_{(C_k)}$ es la distancia "intracluster" del cluster C_k, siendo *n* el número de clusters. El principal objetivo de la medida es maximizar las distancias entre cluster mientras que se aumenta la compacidad. Por lo tanto, el número de cluster que maximiza *D* es el que nos indica el número óptimo de clusters.

5.3. Índice R-Squared

Este índice [13] pertenece a una familia de índices de validez que tiene sentido sobre todo aplicar en los clusters que se han creado a través de algoritmos jerárquicos. Estos cuatro métodos se suelen aplicar a cada paso de un algoritmo de agrupamiento jerárquico y son:

- Desviación estándar de la raíz cuadrada media (Root-Mean-Square Standard Deviation, RMSSTD).
- Semi-Partial R-squared (SPR).
- R-Squared (RS).

- Distancia entre dos clusters (CD).

El índice R-squared se obtiene partiendo de sea $D = \{x_1, \dots, x_n\}$ un conjunto de datos/objetos y la suma de los cuadrados de *D* se define como:

$$SS = \sum_{i=1}^{n}(x_i - \bar{y})^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}\left(x_{ij} - \bar{y}_j\right)^2 \quad (19)$$

Donde *y* es la media de los puntos de *D.*

Para la obtención de la ecuación con la que se calcula el índice se parte de sea $C = \{C_1, \dots, C_K\}$ una partición de los datos *D*. Entonces se define:

- SS_w como la suma de los cuadrados dentro de un grupo.
- SS_b como la suma de los cuadrados entre grupos.
- SS_t como la suma total de cuadrados de todos los datos.

Y el índice *RS* se expresa como:

$$RS = \frac{SS_b}{SS_t} = \frac{SS_t - SS_w}{SS_t} =$$

$$= \frac{\sum_{x \in D}\sum_{j=1}^{d}\left(x_j - \bar{y}_j\right)^2 - \sum_{i=1}^{k}\sum_{x \in C_i}\sum_{j=1}^{d}\left(x_j - \mu_{ij}\right)^2}{\sum_{i=1}^{n}\sum_{j=1}^{d}\left(x_{ij} - \bar{y}_j\right)^2} \quad (20)$$

Donde *k* es el número de clusters, *d* es el número de variables (dimensiones de los datos), *n* es el número de datos de *j* dimensiones, x_{ij} es el dato de dimensión *j* que pertenece al patrón *i* e $\bar{y}_j$ es la media de los datos de la dimensión *j*.

Con respecto a este índice añadir que cuantas más diferencias hay entre grupos, más homogéneos son cada uno de los grupos, y al revés, cuantas menos diferencias menor es la homogeneidad de los grupos, por este motivo el índice *RS* puede ser considerado como una medida de similitud entre clusters.

5.4. Estadística de Hubert Modificada

La estadística de Hubert modificada [13] se define como:

$$\Gamma = \frac{1}{M}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} P_{ij}Q_{ij} \quad (21)$$

donde *n*, es el número de puntos objetos a agrupar, *P*, es la matriz de proximidad y *Q* es la matriz definida por:

$$Q_{ij} = d\left(\mu_{Ci}, \mu_{Cj}\right), 1 \le i, j \le n \quad (22)$$

donde *d* es una distancia euclídea, y μ_{Ci}, μ_{Cj}, son los centroides de los clusters a los que pertenecen los puntos *i,j*.

El parámetro *M* se calcula como:

$$M = \frac{n(n-1)}{2} \tag{23}$$

Un alto valor del índice indica que los clusters existentes son compactos, por lo tanto, se busca que en la aplicación de índice aquel que tiene un valor más alto. Añadir que para un número de clusters igual a 1 o *n*, el índice no está definido.

Del mismo modo, se puede definir la estadística de Hubert normalizada que viene dada por la siguiente expresión:

$$\hat{\Gamma} = \frac{1}{M}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(X(i,j) - \mu_x)\big(Y(i,j) - \mu_y\big) \tag{24}$$

Si el valor de $d(\mu_{Ci}, \mu_{Cj})$ es próximo a $d(x_i, x_j)$ para $i, j = 1,2,\dots,N$. *P* y *Q* tendrán un valor similar y los valores de Γ y $\hat{\Gamma}$ serán elevados.

5.5. Índice Calinski-Harabasz

El índice de Calinski-Harabasz [14], está definido utilizando las trazas de las matrices "scatter" entre los clusters y dentro de los clusters. Siendo *n* el número de puntos objeto y *K* el número de clusters, este índice es calculado de la siguiente manera:

$$CH = \frac{[Tr\,(B)/(K-1)]}{[TR\,(w)/(n-k)]} \tag{25}$$

donde $Tr\,(B)$ y $Tr\,(W)$ son las trazas de las matrices *B* y *W* respectivamente. Y *B* y *W* son las matrices "scatter":

$$Tr\,(B) = \sum_{k=1}^{K} n_k |z_k - z|^2 \tag{26}$$

donde n_k es el número de patrones dentro del cluster *k* y *z* es el centroide del conjunto de datos.

La traza de la matriz *W* que mide la dispersión dentro de un cluster se calcula como:

$$Tr\,(W) = \sum_{k=1}^{K}\sum_{i=1}^{n_k} |x_i - z_k|^2 \tag{27}$$

donde z y z_k son la media de todos los puntos y del cluster C_i respectivamente. Por lo que el índice *CH* puede escribirse como:

$$CH = \frac{\sum_{k=1}^{K} n_k |z_k - z|^2}{(K-1)} \Big/ \frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} |x_i - z_k|^2}{(n-k)} \quad (28)$$

Un mayor valor del índice indicará una agrupación más compacta del mismo.

5.6. Índice de Compacidad

De acuerdo con [15], el índice de compacidad (CP) mide la distancia media entre cada par de puntos objeto que pertenecen al mismo cluster. Más precisamente se define como:

$$CP = \frac{1}{N} \sum_{k=1}^{K} n_k \left(\frac{\sum_{x_i, x_j \in C_k} d(x_i, x_j)}{n_k (n_k - 1)/2} \right) \quad (29)$$

donde *K* denota el número de clusters, n_k es el número de puntos objeto que pertenecen al cluster k, $d(x_i, x_j)$es la distancia entre los puntos x_i y x_j, y *N* es el número total de puntos objeto a agrupar. Idealmente, los miembros de cada cluster deberían estar lo más cercanos posible. Esto significa que pequeños valores para *CP* significa una mejor configuración.

Otro criterio para evaluar un algoritmo de clustering es la precisión (accuracy), que mide como de bien se ha realizado el proceso de agrupación en comparación con las verdaderas etiquetas de los puntos de datos:

$$Accuracy\ (\pi) = \frac{\sum_{k=1}^{K} majority(C_k | L_k)}{N} \quad (30)$$

donde $majority(C_k|L_k)$es el número de puntos con la etiqueta de la pluralidad en el clúster C_k (si la etiqueta l apareció en el clúster k más veces que cualquier otra etiqueta, luego $majority(C_k|L_k)$ es el número de puntos en C_k con la etiqueta *l*.

5.7. Índice de Silhouette

El índice de validación de Silhouette [16] calcula la anchura de la silueta para cada punto objeto. Se calculan también la silueta promedio para todos los puntos objeto de un cluster y de todos los clusters.
El empleo de este índice tiene sentido cuando las distancias entre patrones se encuentran en una razón de escala (como en el caso de las distancias euclídeas) y cuando se están buscando grupos compactos y bien separados. De hecho, la definición del índice está basada en el promedio de las proximidades. Para construir las siluetas son necesarias dos cosas: la partición que se ha obtenido por medio de alguna técnica de partición y el

conjunto de todas las proximidades entre objetos. La obtención del índice es la siguiente:

1. Para cada patrón *i* se introduce un cierto valor de *s(i)*, y estos números son representados en un gráfico.
2. Se toma cualquier patrón *i* y se denota por A el cluster que tiene asignado. Como dicho grupo posee otros patrones aparte de *i* y se obtiene *a(i)*, que es la disimilitud promedio de *i* para todos los demás objetos de A.
3. Se considera otro grupo C, distinto de A y se calcula *d(i,C)*, que es la disimilitud promedio de *i* para todos los valores de *C*.
4. Después de haber calculado el valor de *d* pata todos los clusters distintos de A, y se obtiene $b(i) = \min_{C \neq A} d(i, C)$.

Una representación gráfica se muestra en la siguiente figura:

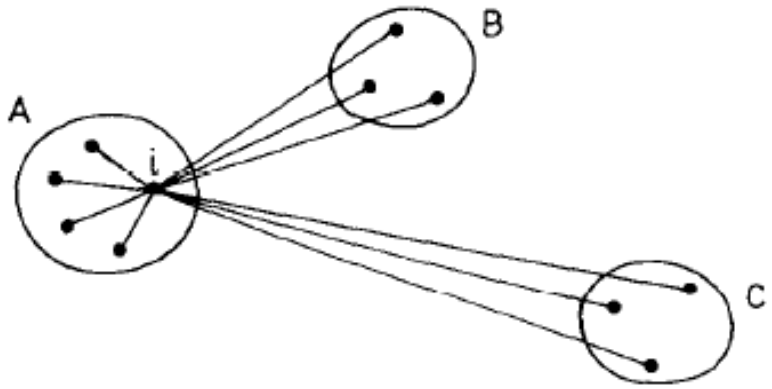

Figura 11.- Explicación índice Silhouette

5. El cluster B para el cual se alcanza este mínimo ($d\,(i, B) = b\,(i)$) que llamamos el vecino, se conoce como el vecino opuesto a *i*, y es la segunda mejor opción para el patrón *i*.
6. Se obtiene el número de *s(i)* mediante la combinación de *a(i)* y *b(i)* de la siguiente manera:

$$s(i) = \begin{cases} 1 - a(i)/b(i) & si\ a(i) < b(i) \\ 0 & si\ a(i) = b(i) \\ b(i)/a(i) - 1 & si\ a(i) > b(i) \end{cases} \quad (31)$$

La silueta promedio puede ser utilizada para evaluar la validez de un resultado clustering y también para decidir el número de clusters. Para construir las siluetas *S(i)* se usa la fórmula:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \quad (32)$$

donde *a(i)* (disimilitud promedio) del punto objeto *i* con todos los demás puntos objeto en el mismo cluster; *b(i)* (mínimo del promedio de disimilitud) del objeto *i* hacia todos los objetos en otro cluster (el más cercano).

De la fórmula anterior se deduce que $-1 \leq S(i) \leq 1$. Si la silueta está próxima de 1, significa que el punto objeto está asignado a su cluster. Con valor 0, el punto objeto podría pertenecer a otro cluster. Si el valor es próximo a -1, el punto objeto está mal clasificado y por lo tanto esta en alguna parte entre los clusters. Para tener un único valor que represente a todos los cluster, se calcula el promedio de la silueta de todos los puntos objeto. Por lo tanto, el mejor resultado clustering será el que proporcione un valor máximo del promedio de Silhouette.

6. Explicación del programa y resultados

El programa se ha realizado en Matlab, para ello se ha creado una interfaz gráfica de usuario (Graphical User Interfaces, "gui"). Una imagen de la interfaz creada es:

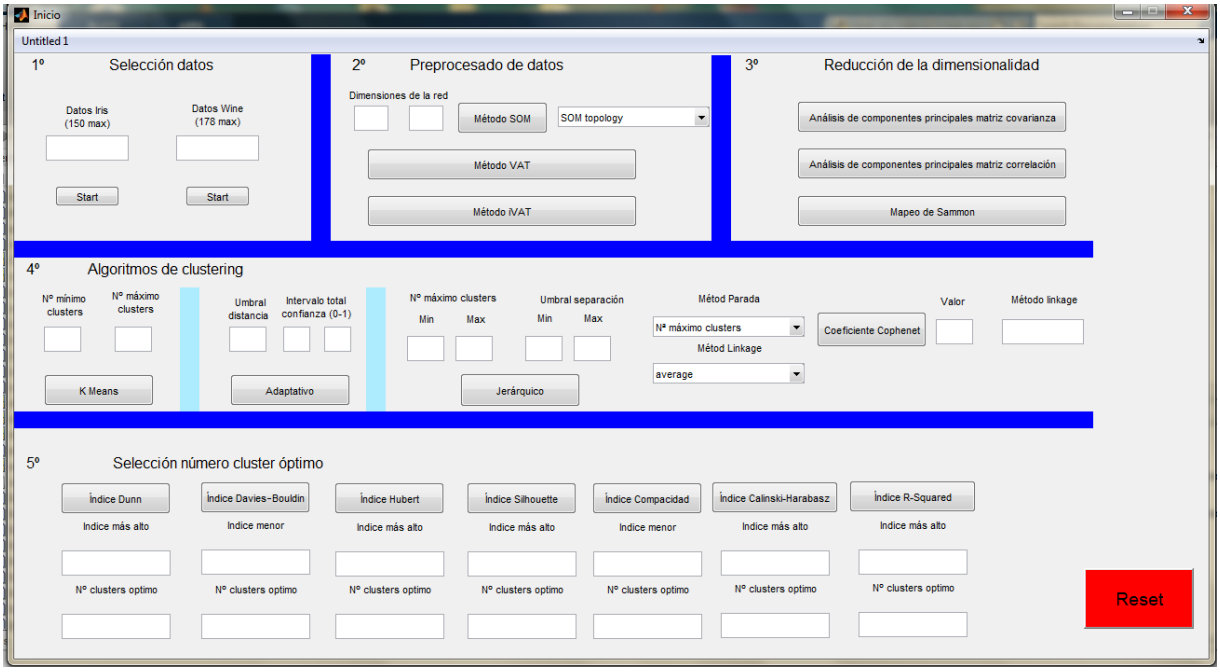

Figura 12.- Interfaz gráfica de usuario

En esta interfaz el usuario puede realizar todas las operaciones del proceso de clustering, desde la selección de datos hasta la selección del número de cluster óptimo, pasando por el proceso de análisis de tendencia (preprocesado de datos y reducción de la dimensionalidad) y el empleo de distintos algoritmos de clustering.

6.1. Selección de datos

Los datos que se han empleado para el análisis de funcionalidad del programa se han obtenido de la página web *"http://www.ics.uci.edu/~mlearn/MLRepository.html"*. Los datos seleccionados son IRIS y WINE. Los datos IRIS son 150 casos con 4 atributos reales numéricos y están relacionados con 3 especies de flores (Iris-Setosa, Iris-Versicolor, y Iris-Virginica) caracterizadas por 4 atributos: longitud y anchura del petalo, longitud y anchura del sepalo. Son un conjunto de 150 observaciones en donde cada clase contiene 50 flores. La clase Iris-Setosa es linealmente separable de las otras 2 clases, pero la Iris-Versicolor y Iris-Virginica no son clases separables linealmente.

Por otro lado los datos WINE son 178 casos con 13 variables, estos atributos son número reales y enteros. Estos datos son los resultados de un análisis químico hecho a vinos que son de una misma región de Italia pero que proceden de tres diferentes cultivos. El análisis ha determinado la cantidad de trece componentes encontrados en cada uno de los tres tipos de vinos. Algunos de estos componentes son: alcohol, ceniza, magnesio, número total de fenoles,intensidad del color, etc.

Se han escogido estas dos variables porque sus atributos son numéricos, con lo que no era necesario hacer ningún paso de convertir variables cualitativas en cuantitativas.

En el programa se le da la opción al usuario de que elija entre las dos clasificaciones de datos, además se le permite que elija el número de patrones o casos con los que desea trabajar. Si se elige un número menor al máximo de datos, la selección de datos se hace aleatoriamente, mientras que si elige un número mayor de los existentes, se mostrará un mensaje de aviso y se seleccionarán todos los patrones disponibles. Una imagen de esta parte del programa se muestra en la siguiente figura:

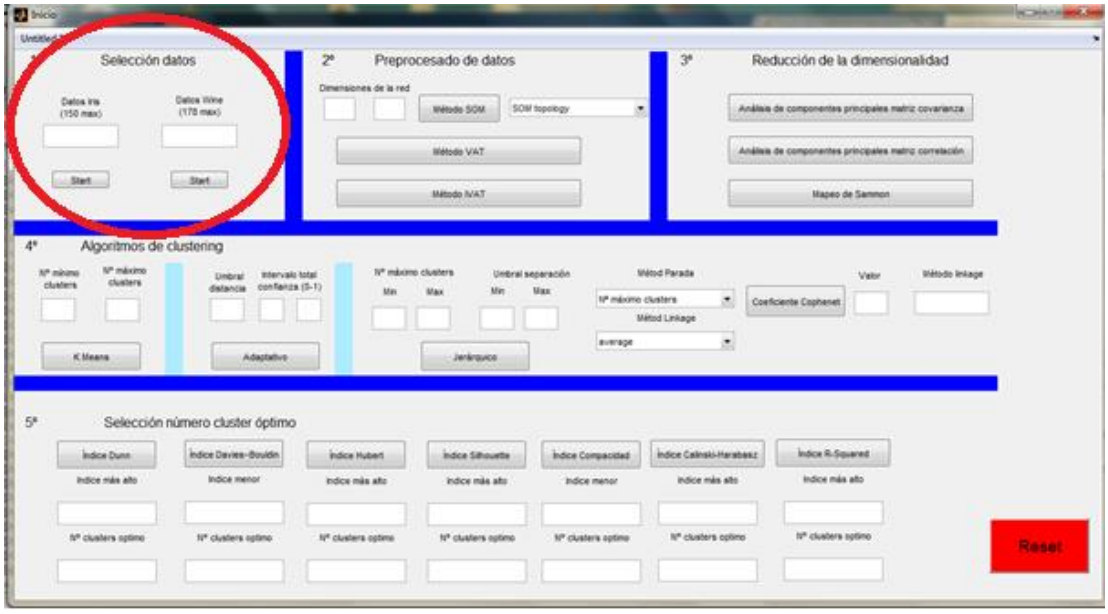

Figura 13.- Selección de datos.

El código del programa donde se seleccionan los datos de IRIS es el siguiente:

```
% --- Executes on button press in Start_Iris.
function Start_Iris_Callback(hObject, eventdata, handles)
% hObject    handle to Start_Iris (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=str2double( get(handles.Num_Iris, 'string') );
%%%Mensaje de error si se elige un número mayor de 150
if n>150
     mensaje_error % Función que muestra un menú con un mensaje de error
     n=150;
end
load('Iris_data.mat');
M=bezdekIris;%Se cargan los datos en una variable
[dato]=seleccion(M,n); %Función que realiza la selección de datos aleatoria
estado=1;
save('datos','dato')%Se guardan los datos en una variable
save('est','estado')
```

Función 1.- Botón Start IRIS.

Y para la selección de datos WINE:

```
% --- Executes on button press in Start_Wine.
function Start_Wine_Callback(hObject, eventdata, handles)
% hObject    handle to Start_Wine (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=str2double( get(handles.Num_Wine, 'string') );
%%%Mensaje de error si se elige un número mayor de 178
if n>178
     mensaje_error;% Función que muestra un menú con un mensaje de error
     n=178;
end
M=load ('wine.data');%Se cargan los datos en una variable
[dato]=seleccion(M,n);%Función que realiza la selección de datos aleatoria
estado=1;
save('datos','dato')%Se guardan los datos en una variable
save('est','estado')
```

Función 2.- Botón Start WINE.

El funcionamiento de ambos botones del *Graphical User Interface* creado en Matlab es el mismo, con la función *str2double*, se recoge el valor del número de patrones con el que se quiere trabajar, una vez se halla pulsado el *pushbotton* de la interfaz gráfica, que para ambos casos se ha llamado *Start*.

Lo primero que se realiza es una comparación para comprobar si el número de patrones que se ha añadido es mayor del máximo, en caso afirmativo aparecerá un *menu* con un mensaje de error y se seleccionará el máximo número de patrones disponibles (150 o 178). La función que muestra el mensaje de error es:

```
function mensaje_error
repetir=1;
while repetir==1
    mensaje=menu('Ha elegido un número mayor del permitido, se
seleccionara el máximo','OK');
    %El mensaje no desaparece hasta que repetir sea igual a 0
    switch mensaje
        case 1
            repetir=0;
    end
end
```

Función 3.- mensaje_error.

El mensaje error que se le muestra al usuario aparece en la siguiente figura:

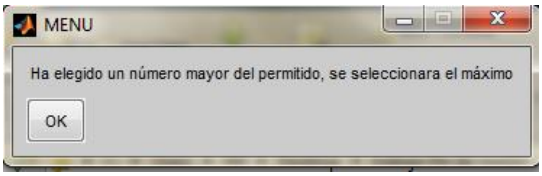

Figura 14.- Mensaje error.

La función donde se hace la selección aleatoria de los patrones se muestra a continuación:

```
%Programa al que se le pasa como entrada la matriz de datos con la
que se trabaja y el número de datos que se quieren seleccionar. La
selección se hará aleatoriamente.

function [dato]=seleccion(M,n)
matriz=M;
inc=1;
while n>=1
    [f,c]=size(matriz);
    r = randi(f,1); %selección de un número aleatorio
    dato(inc,:)=matriz(r,:);
    inc=inc+1;
    [matriz]=reduccion(matriz,r);%Función en la que se extrae la
fila
    n=n-1;
end
```

Función 4.- selección.

En esta función se pasa como parámetros la matriz de datos de donde se va a realizar la selección aleatoria y el número de datos que se quiere seleccionar. En esta función se realiza un bucle iterativo donde en cada iteración se escoge un dato aleatorio a partir de la función *randi*, este dato aleatorio se corresponde con el dato de la matriz de patrones que se va a extraer el dato en cuestión y se añade a la matriz de los datos que se van a seleccionar. Esta iteración se detiene cuando ya se han seleccionado todos los datos

necesarios. La función donde se produce la extracción del patrón de la matriz de datos iniciales se llama reducción, y se ha implementado con el siguiente código:

```
%Función a la que se le entrega una matriz de cluster y una posición
y se extrae ese cluster de la matriz
function [A]=reduccion(M,pos)

[f,c]=size(M);
inc=1;
if f==1
   A(f,:)=M(f,:);
else
    for i=1:f
        if i~=pos
          A(inc,:)=M(i,:);
          inc=inc+1;
        end
    end
end
```

Función 5.- reduccion.

6.2. Preprocesado de datos

El análisis de clustering, comienza con el preprocesado de datos, es decir, una primera visualización de que características tienen los patrones con los que se va a trabajar, se pueden emplear los métodos de SOM, VAT e iVAT. Esto se realiza a través de la siguiente parte del programa, la cual se muestra en la siguiente figura:

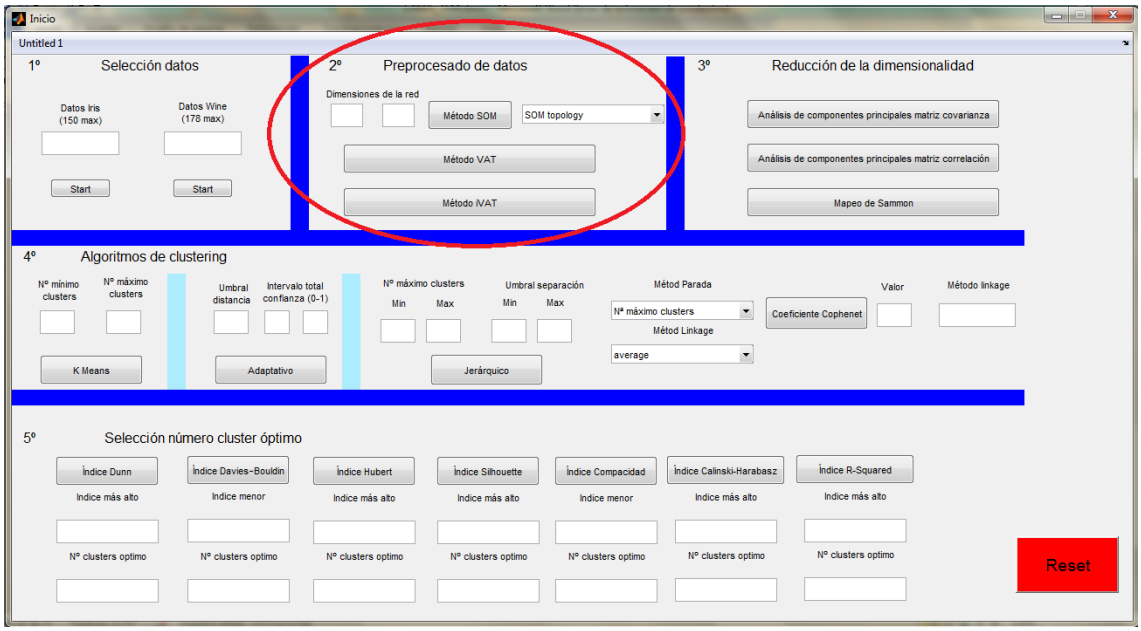

Figura 15.- Preprocesado de datos

6.2.1. Método SOM

Para la operación de preprocesado de datos Matlab incluye la toolbox *clustering*, dentro de dicha toolbox se encuentra la función *selforgmap*. Esta función te crea un Mapa Auto Organizativo (SOM) a partir de unas dimensiones dadas de la matriz.

Al usuario se pide que introduzca las dimensiones de la red a las que va a realizar el preprocesado y se pulse el botón *"Método SOM"*. Cuando se produce esto se ejecuta el siguiente código de programa:

```
% --- Executes on button press in met_SOM.
function met_SOM_Callback(hObject, eventdata, handles)
% hObject    handle to met_SOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<1
    orden_incorrecto;
else
    load('datos');
    estado=1;
    save('est','estado')
    dimension1=str2double( get(handles.dim1, 'string') );
    dimension2=str2double( get(handles.dim2, 'string') );
    [outputs,net]=SOM(dato,dimension1,dimension2);
    save('SOM_output','outputs')
    save('SOM_net','net')
end
```

Función 6.- Botón Método SOM.

El primer paso que se realiza es comprobar el estado en que se encuentra el programa, si se detecta que el estado es menor de 1 (aún no se ha producido la selección de datos), se produce una llamada la función *orden_incorrecto*, en la cual se muestra una ventana emergente, indicándole al usuario que no ha seguido el orden correcto a la hora de ejecutar el programa, el código de dicha función es:

```
function orden_incorrecto
repetir=1;
while repetir==1
mensaje=menu('Ejecute el programa en el orden correcto','OK');

switch mensaje
    case 1
        repetir=0;
end
end
```

Función 7.- orden_incorrecto.

El mensaje contextual que se muestra por pantalla aparece en la siguiente figura:

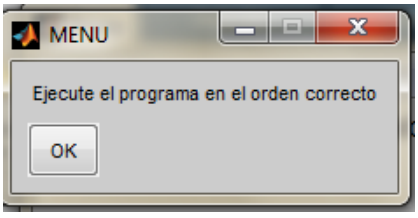

Figura 16.- Mensaje de orden incorrecto.

Si el orden en que se está ejecutando el programa es el correcto, se cogen los valores de las dos dimensiones que son necesarias para crear la red del método de SOM con la función *str2double*, se actualiza el estado actual del programa, se guardan los datos con los que se están trabajando y se ejecuta la función *SOM*, el código de la cual se muestra a continuación:

```
%Función donde se ejecta el algoritmo de SOM
function [outputs,net]=SOM(dato,dimension1,dimension2)
% Creación del Self-Organizing Map
net = selforgmap([dimension1 dimension2]);
% Entrenamiento de la estructura, se le pasa como parámetros la red
y los datos
[net,tr] = train(net,dato);
%Obtención de los datos de salida
outputs = net(dato);
% View the Network
%figure
view(net)
```

Función 8.- SOM.

En esta función se pasan como datos de entrada las dimensiones de la red que se va a crear y los datos que se van a procesar y devuelve como salida la red del mapa autoorganizativo creado y los datos obtenidos tras el entrenamiento realizado a los mismos. Las funciones de Matlab por orden empleadas son:

- *selforgmap*: Esta función aprende de los datos del cluster basándose en la similitud, topología, con una preferencia de asignar el mismo número de casos para cada clase. Con esta función se obtiene la red del mapa autoorganizativo.
- *train*: Con esta función se entrena la red neuronal obtenida con anterioridad.
- *net*: Después de que la red se haya entrenado se obtiene la salida de la red.
- *view*: Se visualiza la red de SOM que se ha creado.

Tras seleccionar 150 patrones de los datos IRIS, e introducir en los campos de dimensiones de la red de SOM los valores de 10 x 10, aparecen por pantalla las siguientes figuras:

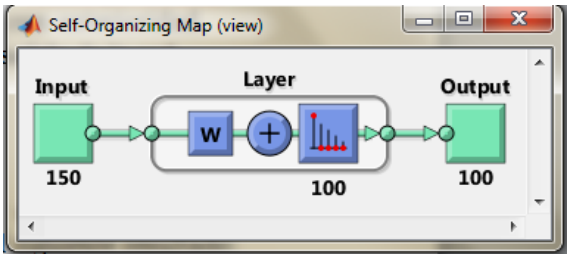

Figura 17.- Vista de la red de SOM.

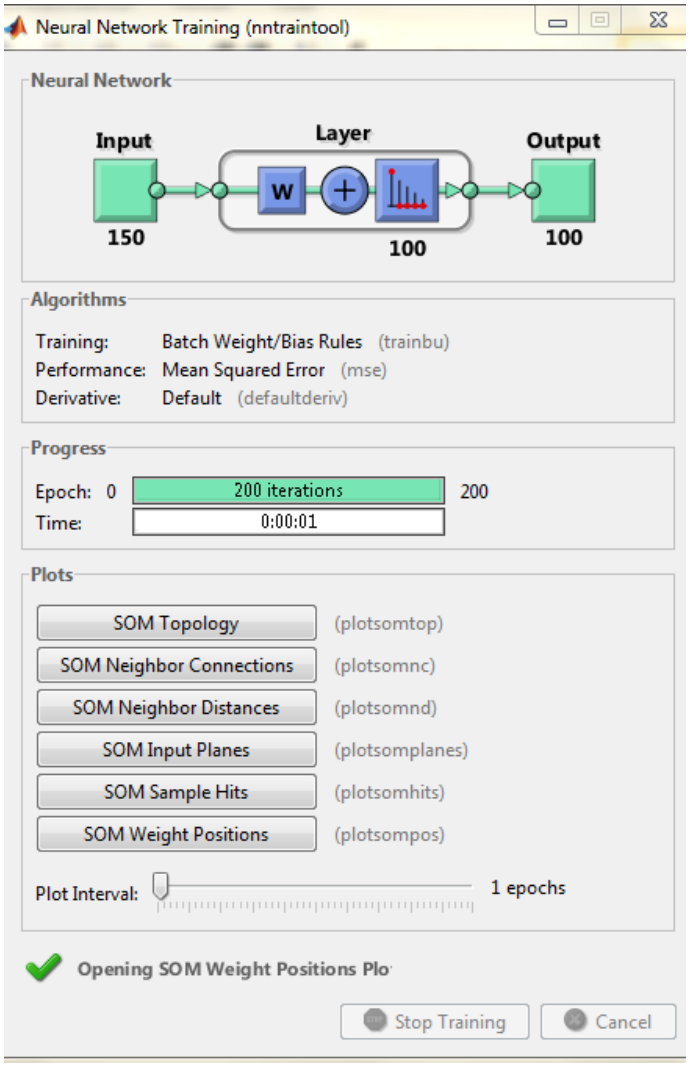

Figura 18.- Resumen del proceso de obtención de la red de SOM.

En la segunda imagen se muestra la toolbox de Matlab *Neural Network Training* (nntraintool) [25], dentro de esta pantalla se pueden representar las diferentes formas de representación de la red de SOM, esto también se puede realizar eligiendo entre las distintas opciones que aparecen en el desplegable a la derecha del botón "Método SOM". Estas opciones junto con algunos ejemplos son:

- SOM Topology: Representación de la topología de SOM.

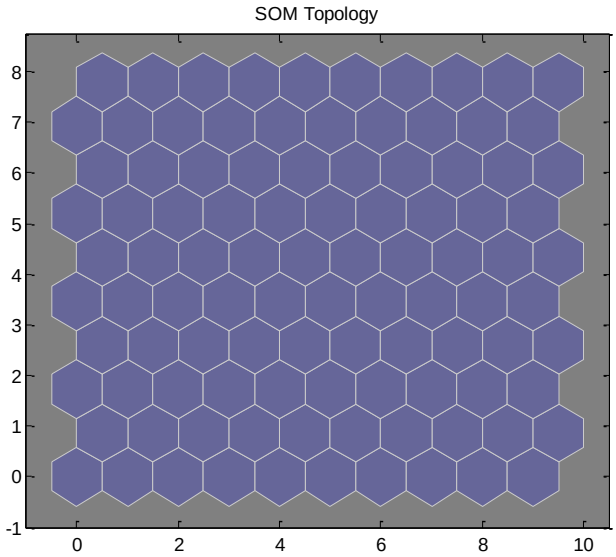

Figura 19.- Representación SOM Topology, red 10x10, 150 datos Iris.

- SOM Neighbor Connections: Representa una capa de neuronas del mapa de SOM mostrando las neuronas como parches de color azul grisáceo y las relaciones de vecindad directa con líneas rojas.

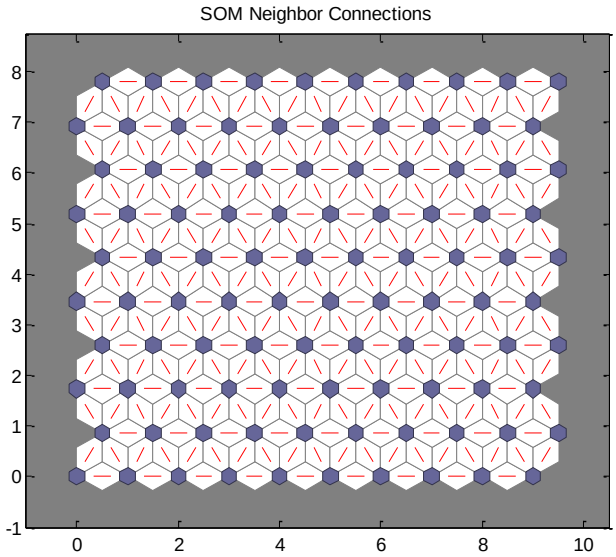

Figura 20.- Representación SOM Neighbor Connections, red 10x10, 150 datos Iris.

- SOM Neighbor Distances: Donde se muestra la distancia entre neuronas vecinas. Esta figura representa en hexágonos azules la neuronas, con líneas rojas las conexiones entre neuronas vecinas, los colores en las regiones que contienen las

líneas rojas indican las distancias entre neuronas, siendo los colores oscuros para grandes distancias y los claros para distancias menores.

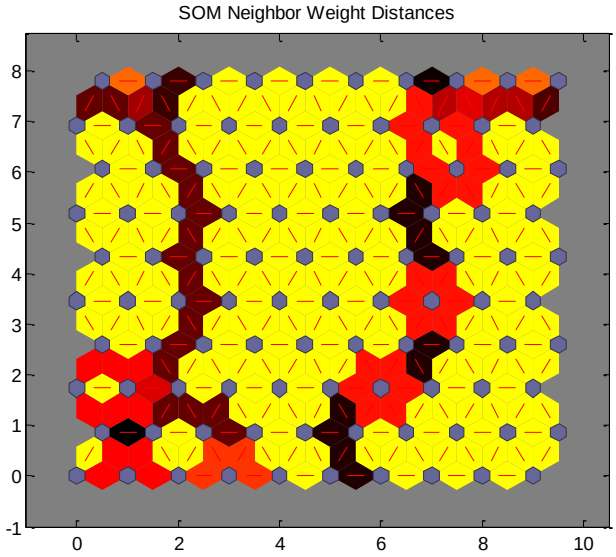

Figura 21.- Representación SOM Neighbor Distances, red 10x10, 150 datos Iris.

- SOM Input Planes: Genera una serie de subgráficas donde cada subgráfica *i* muestra los pesos de cada una de las neuronas de la capa, donde las conexiones más negativas se muestran con el color azul, si hay cero conexiones con el negro y las más positivas con el rojo. La gráfica es mostrada sólo para las capas organizadas en una o dos dimensiones.

Figura 22.- Representación SOM Input Planes, red 10x10, 150 datos Iris.

- SOM Sample Hits: Muestra una capa de SOM, donde cada neurona muestra el número de vectores que clasifica. El número relativo de vectores para cada neurona se muestra a través del tamaño de un parche coloreado.

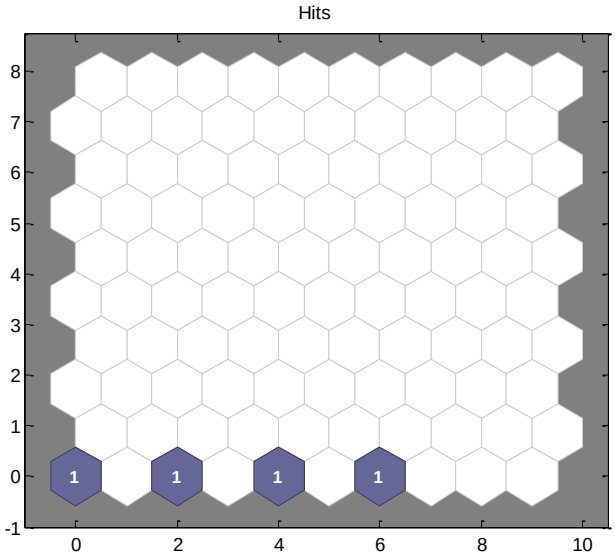

Figura 23.- Representación SOM simple hits, red 10x10, 150 datos Iris.

- SOM Weight Positions: Muestra la ubicación de los patrones y los vectores de peso, dibuja los vectores de entrada como puntos verdes y muestra como el algoritmo de SOM clasifica el espacio de entrada, muestra de color azul grisáceo el vector de pesos de cada neurona y la conexión de neuronas vecinas con líneas rojas.

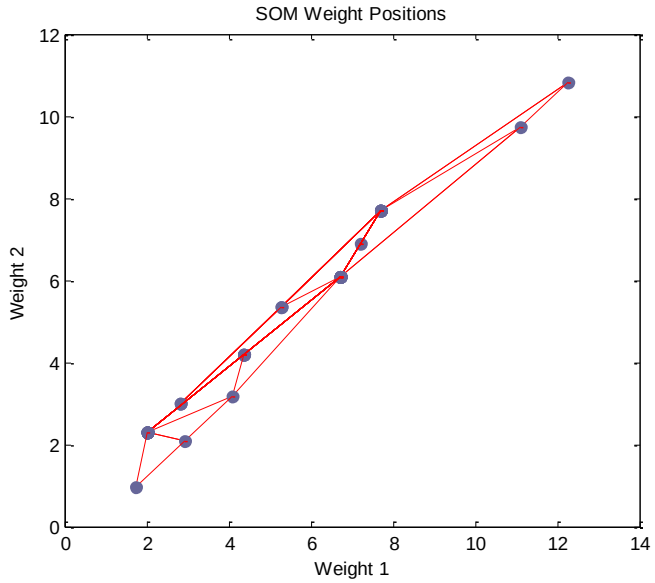

Figura 24.- Representación SOM Weight Positions, red 10x10, 150 datos Iris.

Para los mismos datos de entrada, se selecciona una red de matriz de SOM de dimensión 5x20, obteniéndose los siguientes resultados:

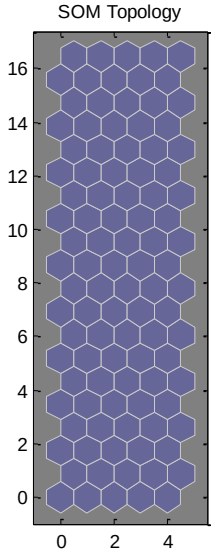

Figura 25.- Representación SOM Topology, red 5x20, 150 datos Iris.

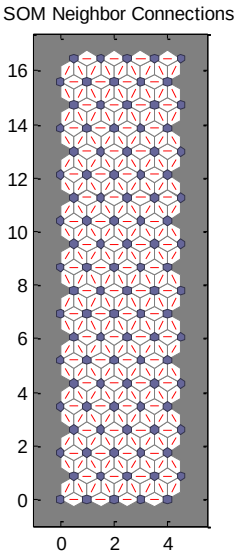

Figura 26.- Representación SOM Neighbor Connections, red 5x20, 150 datos Iris.

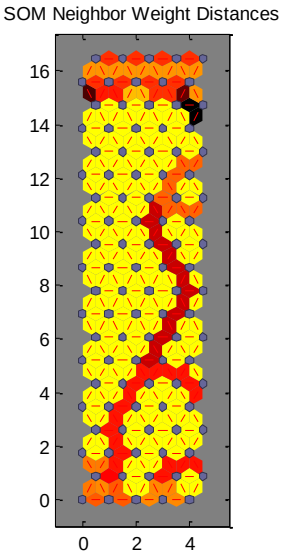

Figura 27.- Representación SOM Neighbor Distances, red 5x20, 150 datos Iris.

Figura 28.- Representación SOM Input Planes, red 5x20, 150 datos Iris.

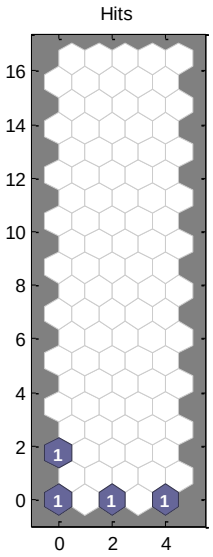

Figura 29.- Representación SOM simple hits, red 5x20, 150 datos Iris.

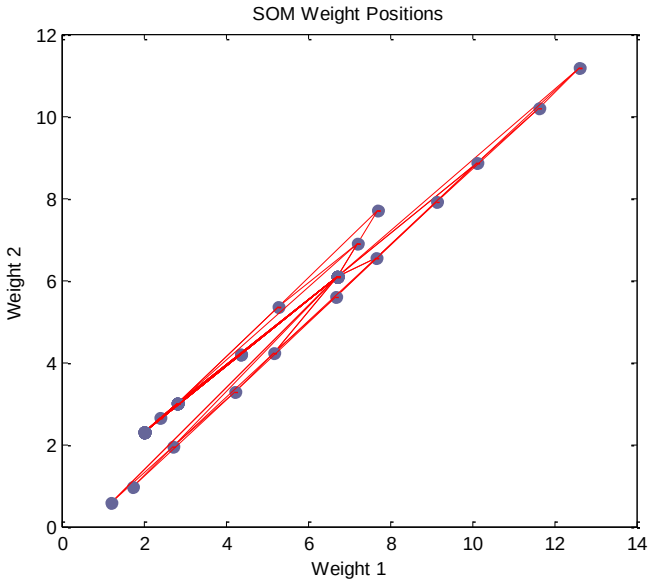

Figura 30.- Representación SOM Weight Positions, red 5x20, 150 datos Iris.

El código del desplegable, a partir del cual se puede obtener las distintas visualizaciones se muestra a continuación. En dicho código se recoge el número de la opción del desplegable seleccionada y se le introduce como parámetro a una función.

```
% --- Executes on selection change in SOM_visualizacion.
function SOM_visualizacion_Callback(hObject, eventdata, handles)
% hObject    handle to SOM_visualizacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns
SOM_visualizacion contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
SOM_visualizacion
v=get(handles.SOM_visualizacion,'Value');
load('SOM_output');
load('SOM_net');
load('datos');
SOM_figure(dato,outputs,net,v);
```

Función 9.- Desplegable visualización SOM.

El código de la función *SOM_figure*, que es la que realiza el proceso de visualización, se muestra en el siguiente recuadro:

```
function SOM_figure(dato,outputs,net,figura)
figure
switch figura
    case 1
        plotsomtop(net)
    case 2
        plotsomnc(net)
    case 3
        plotsomnd(net)
    case 4
        plotsomplanes(net)
    case 5
        plotsomhits(net,dato)
    case 6
        plotsompos(net,dato)
end
```

Función 10.- SOM_figure.

6.2.2. Método VAT

Dentro de la sección de preprocesado de datos, se encuentra también la aplicación del método VAT. Pulsando el botón del *"Método VAT"* se produce una llamada a la siguiente función de la interfaz gráfica de usuario de Matlab, el código que se ejecuta es el siguiente:

```
% --- Executes on button press in met_VAT.
function met_VAT_Callback(hObject, eventdata, handles)
% hObject    handle to met_VAT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<1
    orden_incorrecto;
else
    load('datos');
    estado=1;
    save('est','estado')
    [RV,C]=VAT(dato);
end
```

Función 11.- Botón Método VAT.

En esta llamada se actúa de la misma manera que en la sección del método SOM, en primer lugar se comprueba el estado en que se encuentra el sistema, y si dicho estado es mayor de 1, es decir, ya se ha producido la selección de datos, se produce la llamada a la función *VAT*, el código correspondiente a dicha función es el siguiente:

```
%%Función que aplica el método VAT, para una matriz de patrones
function [RV,C]=VAT(dato);

%Cálculo de la matriz de diferencias
[f,c]=size(dato);%Tamaño de la matriz
for i=1:f
   for j=1:f
       acumulador=0;
       for h=1:c
           acumulador=acumulador+((dato(i,h)-dato(j,h))^2);
           distancia(i,j)=abs(sqrt(acumulador));
       end
   end
end

[N,M]=size(distancia);%Se calcula el tamaño de la matriz de
distancia

K=1:N;
J=K;
```

```
%Valor máximo de la matriz de distancia
[y,i]=max(distancia);
[y,j]=max(y);
%Valor mínimo de la matriz de distancias
I=i(j);
J(I)=[];
[y,j]=min(distancia(I,J));

%Creación del índice de la matriz de disimilitud
I=[I J(j)];
J(J==J(j))=[];
C(1:2)=1;

%Reordenamiento de la matriz de disimilitud
for r=3:N-1,
    [y,i]=min(distancia(I,J));
    [y,j]=min(y);
    I=[I J(j)];
    J(J==J(j))=[];
    C(r)=i(j);
end;
[y,i]=min(distancia(I,J));
I=[I J];
C(N)=i;

for r=1:N,
    RI(I(r))=r;
end;

RV=distancia(I,I);

%Representación gráfica
figure
imagesc(distancia)
title('Representación de la matriz de disimilitud')
figure
imagesc(RV)
title('Representación de la matriz de disimilitud ordenada')
```

Función 12.- VAT.

El funcionamiento del programa es el siguiente, en primer lugar se calcula la matriz de diferencias, para ello se calcula la diferencia de todos los patrones entre sí, en las distintas dimensiones y se obtiene una matriz cuadrada, de tamaño *NxN*, siendo *N* el número de patrones. En el siguiente paso se calcula el valor máximo y mínimo de la matriz de distancia, calculada previamente y se crea el índice de la matriz de disimilitud, y por último se produce el reordenamiento de la matriz de disimilitud. La visualización de los resultados se realiza a través de la función de Matlab *imagesc*, donde los datos son escalados para representarlos usando el mapa de colores en su totalidad.

A continuación se muestran varios ejemplos, en el primero de ellos se han seleccionado los datos de *IRIS* con 150 patrones, en primer lugar se muestra la representación de la

matriz de disimilitud y luego la matriz de disimilitud ordenada, tras aplicarle el método VAT.

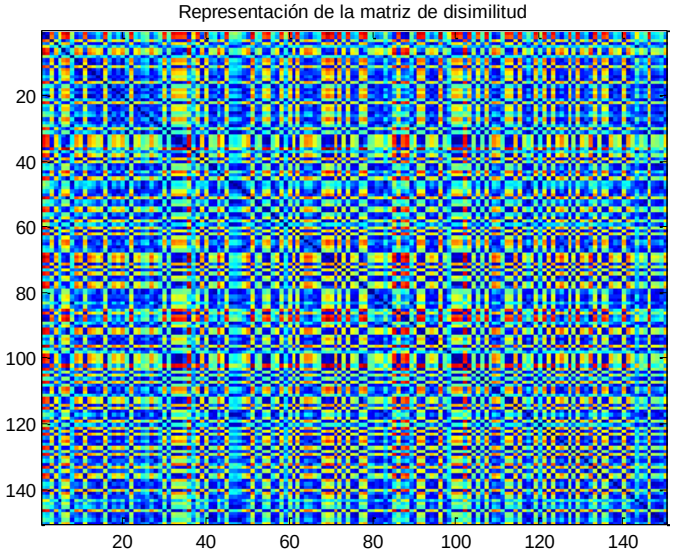

Figura 31.- Matriz de disimilitud, 150 datos IRIS.

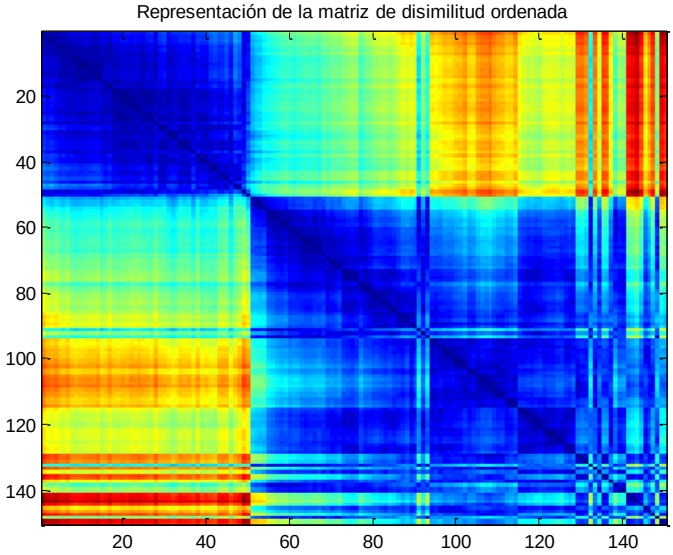

Figura 32.- Matriz de disimilitud ordenada método VAT, 150 datos IRIS.

Visualizando las imágenes obtenidas se puede observar el número de cuadros que hay en la diagonal. Si estos son identificables entonces es signo de que los objetos se pueden agrupar. Ademas si es posible contar el numero de cuadrados, es una forma sencilla de indicar una posible aproximacion del numero de clusters a encontrar. Cuando los cuadros no se forman, es signo de que los datos presentan una estructura aleatoria y no

vale la pena aplicar un algoritmo de analisis de Cluster. De la observacion de la figura 32 se refleja que los datos Iris utilizados pueden ser agrupados y ademas el numero de clusters será al menos igual o superior a 2.
Si en lugar de seleccionar los datos de *IRIS*, se seleccionan todos los patrones de *WINE*, se obtienen las siguientes imágenes:

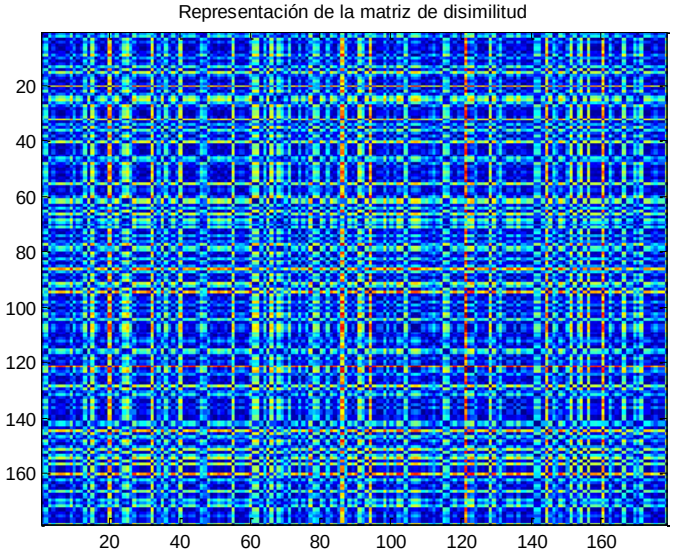

Figura 33.- Matriz de disimilitud, 178 datos WINE.

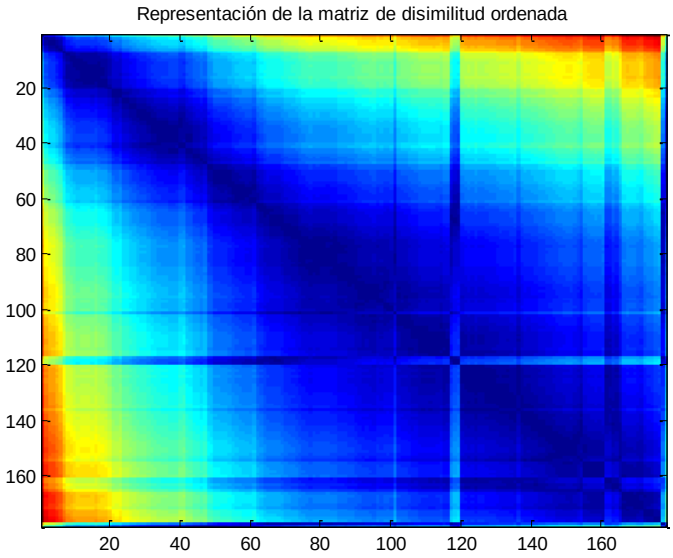

Figura 34.- Matriz de disimilitud ordenada método VAT, 178 datos WINE.

En este caso no se aprecia la formación de cuadrados, con lo que no se puede predecir el número de clusters que sería óptimo que se creara.

Si en lugar de seleccionar los 178 datos de *WINE*, se han seleccionado sólo 50 datos, obteniéndose las siguientes figuras:

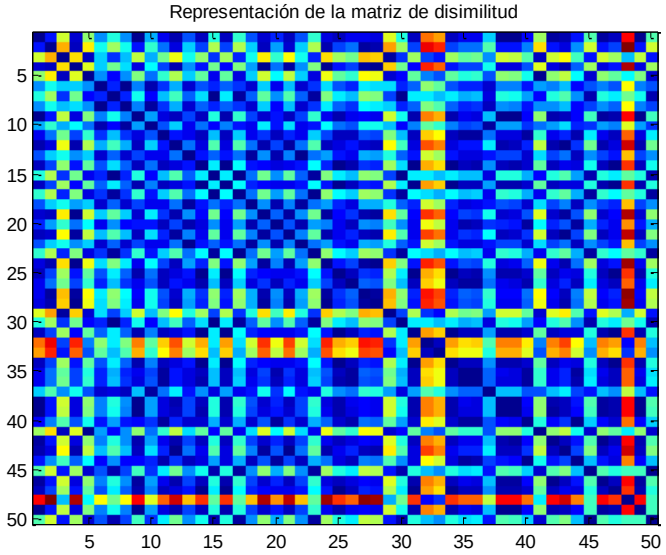

Figura 35.- Matriz de disimilitud, 50 datos WINE.

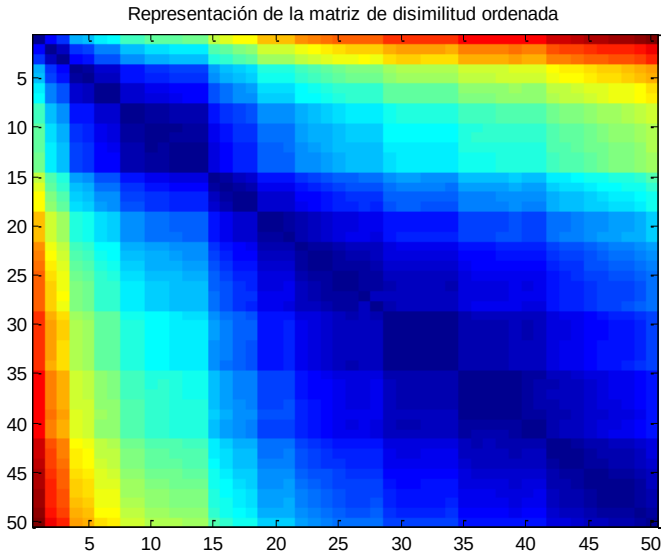

Figura 36.- Matriz de disimilitud ordenada método VAT, 50 datos WINE.

Al aplicar el método VAT, como se puede observar en las figuras anteriores, se produce un reordenamiento de los patrones, de forma que queden agrupados en función de los valores obtenidos por la matriz de disimilitud. La diagonal se representa con color negro, ya que el valor es 0, porque es la diferencia de un patrón consigo mismo, y cuanto más alejado de la diagonal y de forma simétrica aparecen los colores que representan una mayor intensidad. En este caso al igual que en el anterior no se observa la creación de cuadrados, por lo que no se puede predecir el número ideal de clusters a crear en los futuros algoritmos.

6.2.3. Método iVAT

En el apartado de preprocesado de datos, además de los métodos de SOM y VAT, se encuentra el método iVAT, que se trata de una versión mejorada del método VAT, el código que se ejecuta cuando se presiona el botón *"Método iVAT"*, es el siguiente:

```
% --- Executes on button press in met_iVAT.
function met_iVAT_Callback(hObject, eventdata, handles)
% hObject    handle to met_iVAT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<1
    orden_incorrecto;
else
    load('datos');
    estado=1;
    save('est','estado')
    [RiV]=iVAT(dato);
end
```

Función 13.- Botón Método iVAT.

El código que se ejecuta es el siguiente, en primer lugar se comprueba que se esté ejecutando el programa en el orden correcto, y posteriormente se produce la llamada a la función *iVAT*, cuyo código es el siguiente:

```
%%Función que aplica el método iVAT, para una matriz de patrones
function [RiV]=iVAT(dato);

%Cálculo de la matriz de diferencias
[f,c]=size(dato);%Tamaño de la matriz
%%Creación de la matriz de distancias
for i=1:f
   for j=1:f
       acumulador=0;
       for h=1:c
           acumulador=acumulador+((dato(i,h)-dato(j,h))^2);
           distancia(i,j)=abs(sqrt(acumulador));
       end
   end
end

%Cálculo de longitud de la matriz de disimilitud
N=length(distancia);
%Llamada a la función VAT
[RV,C]=VAT_no_image(dato);
%Creación de una matriz de ceros
RiV=zeros(N);
%Reordenamiento de la matriz de disimilitud
 for r=2:N,
     c=1:r-1;
     RiV(r,c)=RV(r,C(r));
     cnei=c(c~=C(r));
     RiV(r,cnei)=max([RiV(r,cnei); RiV(C(r),cnei)]);
     RiV(c,r)=RiV(r,c)';
 end

%Representación gráfica
figure
imagesc(distancia)
title('Representación de la matriz de disimilitud')
figure
imagesc(RiV)
title('Representación de la matriz de disimilitud ordenada')
```

Función 14.- iVAT.

La explicación del código implementado en la función *iVAT* es el siguiente, en primer lugar se crea una matriz de diferencias o disimilitud de tamaño *NxN*, de todos los patrones entre sí. En el siguiente paso se produce la llamada a la función *VAT_no_image*, que es una función igual a la función *VAT*, pero sin la representación gráfica, que devuelve la matriz de disimilitud ordenada y el vector de índices correspondiente. En el siguiente paso, es donde se presenta la diferencia con el método *VAT* anterior, ya que es donde se produce el nuevo reordenamiento de la matriz de disimilitud, aplicando un algoritmo recursivo sobre la matriz VAT, obtenida anteriormente. Por último se produce la representación de los datos, que al igual que en

el método VAT, primero se representa la matriz de disimilitud y posteriormente la matriz de disimilitud ordenada.

Al igual que con el método VAT, se han realizado varias representaciones, en el primer caso se representa la matriz de disimilitud ordenada por el método iVAT, para los 150 datos disponibles. Debido a que la matriz de disimilitud previa, es la misma que en el método VAT, se ha omitido dicha representación.

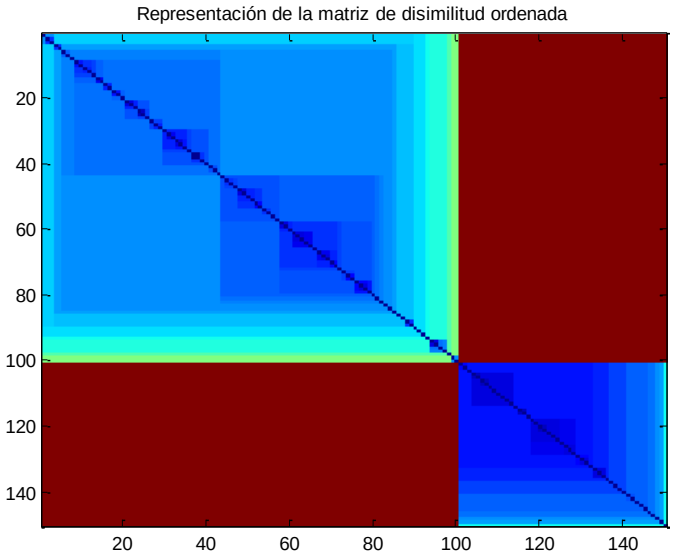

Figura 37.- Matriz de disimilitud ordenada método iVAT, 150 datos IRIS.

Visualizando las imágenes obtenidas por el método iVAT, al igual que se ha hecho con las imágenes obtenidas por el método VAT se puede observar el número de cuadros que hay en la diagonal. Al ser estos identificables entonces es signo de que los objetos se pueden agrupar y contando el número de cuadros, se puede aproximar cuál será el número de clusters a encontrar en los futuros algoritmos. En la figura 37 se refleja que los datos Iris utilizados pueden ser agrupados y ademas el número de clusters será al menos igual o superior a 2.

Si en lugar de seleccionar los 150 patrones del conjunto de datos IRIS, se seleccionan sólo 100 se obtiene la siguiente matriz de disimilitud ordenada.

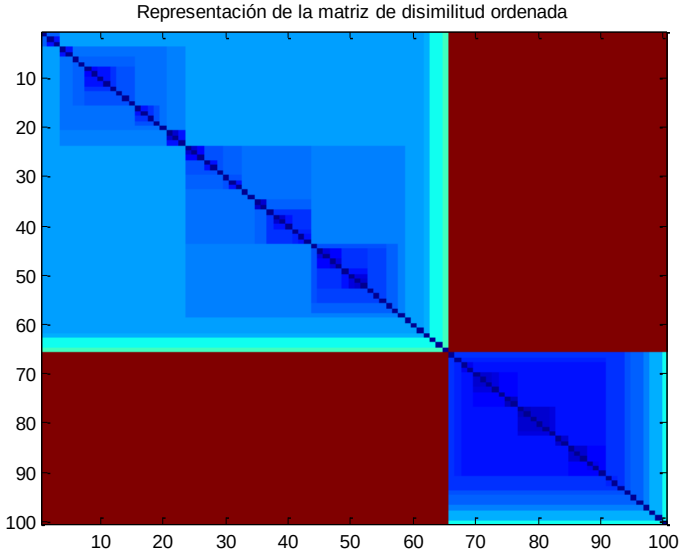

Figura 38.- Matriz de disimilitud ordenada método iVAT, 100 datos IRIS.

En este caso se observa la creación de tres rectángulos claramente diferenciados entre sí, por lo que se podría preveer que el número de clusters a crear será igual o mayor a tres.

Si en lugar de seleccionar los datos de IRIS, se selecciona todo el conjunto de los datos de WINE, se obtiene la siguiente matriz de disimilitud ordenada.

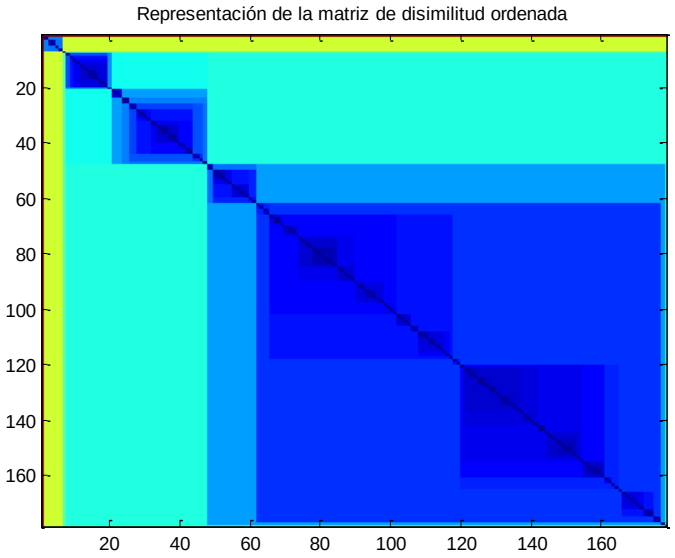

Figura 39.- Matriz de disimilitud ordenada método iVAT, 178 datos WINE

Si en lugar de seleccionar todos los datos de WINE, se seleccionan sólo 50 como en el apartado anterior, se obtiene el siguiente resultado.

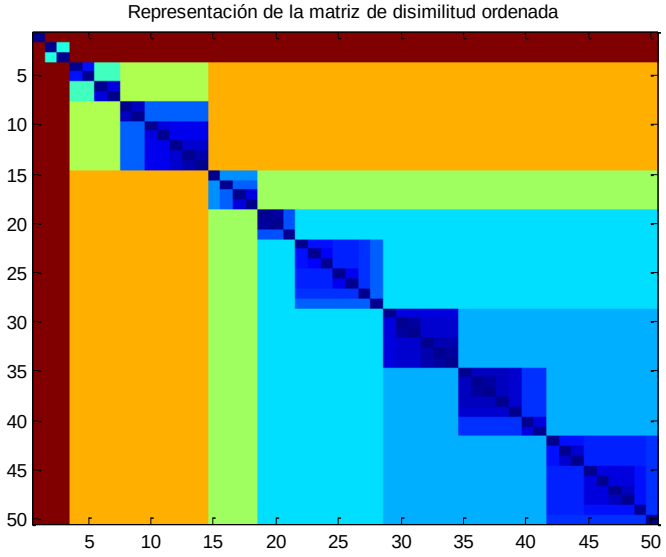

Figura 40.- Matriz de disimilitud ordenada método iVAT, 50 datos IRIS.

En este caso se observan varias zonas diferenciadas que se podrían corresponder con distintos clusters. Como se puede observar, a diferencia de cuando se aplica el método VAT, el método iVAT representa una estructura mucho más cuadrada, con una ordenación de los patrones formando rectángulos, con lo que es más sencillo, obtener una aproximación del número de agrupaciones necesarias.

Al aplicar el método VAT, como se puede observar en las figuras anteriores, se produce un reordenamiento de los patrones, de forma que queden agrupados en función de los valores obtenidos por la matriz de disimilitud. La diagonal se representa con color negro, ya que el valor es 0, porque es la diferencia de un patrón consigo mismo, y cuanto más alejado de la diagonal y de forma simétrica aparecen los colores que representan una mayor intensidad.

6.3. Reducción de la dimensionalidad

Dentro del análisis de tendencia de datos, se encuentra el apartado de reducción de la dimensionalidad, en dicho apartado se presentan las técnicas del Mapeo de Sammon y del Análisis de Componentes Principales, tanto a partir de la matriz de covarianza como de la matriz de correlaciones.

La parte del programa que realiza este apartado se muestra la siguiente figura:

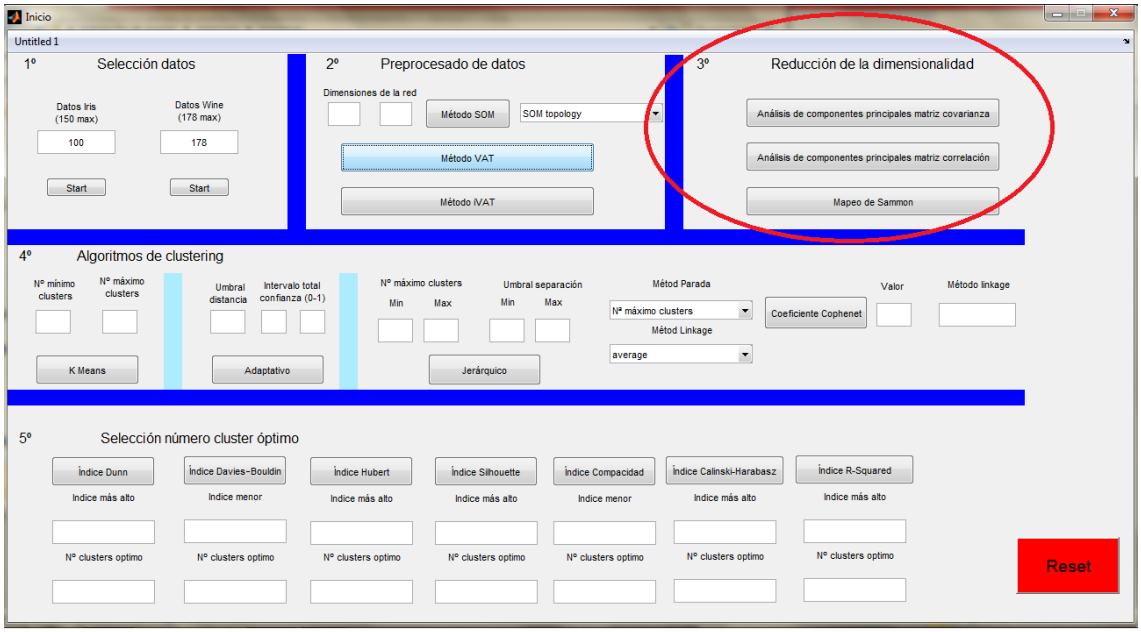

Figura 41.- Reducción de la dimensionalidad.

6.3.1. Análisis de componentes principales matriz de covarianza

Una vez se presiona el botón de selección del *"Análisis de componentes principales matriz covarianza"*, se ejecuta el siguiente código del archivo *GUI* de Matlab, dicho código del programa es el siguiente:

```
% --- Executes on button press in ACP_covarianza.
function ACP_covarianza_Callback(hObject, eventdata, handles)
% hObject    handle to ACP_covarianza (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<1
    orden_incorrecto;
else
    load('datos');
    estado=2;
    save('est','estado')
    [datos_tendencia]=ACP_cov(dato);
    save('dat','datos_tendencia')
end
```

Función 15.- Botón Análisis Componentes principlaes matriz covarianza.

El funcionamiento de dicho programa es el siguiente, en primer lugar se compara el estado actual del programa, si previamente se han seleccionado los datos (IRIS o WINE), se produce un aumento en el estado del programa y se ejecuta la función que realiza el análisis ACP, dicha función es *ACP_cov*, y se guardan el nuevo estado del programa y los nuevos datos obtenidos con los que se trabajará posteriormente. El código de la función *ACP_cov*, es el siguiente:

```
%Función que realiza el análisis de componentes principales a partir
de la matriz de covarianza
function [Y]=ACP_cov(dato)

[f,c]=size(dato);%análisis del número de datos

%1º se tipifican los datos
media=mean(dato);
varianza=var(dato);
for j=1:c
   for i=1:f
        dato_tip(i,j)=(dato(i,j)-media(j)/varianza(j));
   end
end
%2º Obtención de la matriz de covarianza
S=cov(dato_tip,1);

%3º Se hayan los componentes principales a través de la matriz de
covarianza
[T1,D1]=eigsort(S);%Función que ordena los valores y vectores
propios
T1=T1';

%4º Corrección de los signos
if ((sum(sign(T1(:,1)))<0)&(sum(sign(T1(:,2)))<0))
    T1=-T1;
end
%5º Representación de los datos
figure
Y1=dato*T1;
Y=Y1(:,1:2);
plot(Y1(:,1),Y1(:,2),'.b')
xlabel('Primer componente')
ylabel('Segundo componente')
title('Análisis de componentes principales')
end
```

Función 16.- ACP_cov.

El funcionamiento de dicho programa es el siguiente, el primer paso consiste en tipificar los datos, para ello en primer lugar se calcula la media y la varianza de los datos, a través de las funciones de Matlab *mean* y *var*, respectivamente. Una vez se han tipificado los datos se obtiene la matriz de covarianza, a partir de la función de Matlab *cov*. El tercer paso consiste en la obtención de la matriz de covarianza a partir de la función *eigsort*, que ordena los valores y vectores propios. En el cuarto paso se realiza

la corrección de los signos, y por último se realiza la representación de los datos en dos dimensiones, a los que posteriormente se podrán aplicar los algoritmos de clustering.

El código de la función *eigsort*, nombrada con anterioridad se muestra a continuación:

```
%Función que ordena los valores propios y los vectores propios según
el porcentaje de variabilidad.
function [v,d]=eigsort(a)
    [v,d]=eig(a);
    [x,i]=sort(-diag(real(d)));
    d=-x;
    v=v(:,i);
end
```

Función 17.- eigsort.

En la función que se muestra con anterioridad, en primer lugar se calculan los autovalores de la matriz de datos que se pasa como entrada a la función a través de la función *eig*, y con la función *sort*, se ordenan los valores reales de la diagonal de la matriz de autovalores obtenida anteriormente.

A continuación se muestran varios ejemplos de los resultados obtenidos aplicando el método del análisis de componentes principales a partir de la matriz de covarianza. Para una selección de todo el conjunto de los datos de IRIS (150 patrones), se obtiene el siguiente resultado:

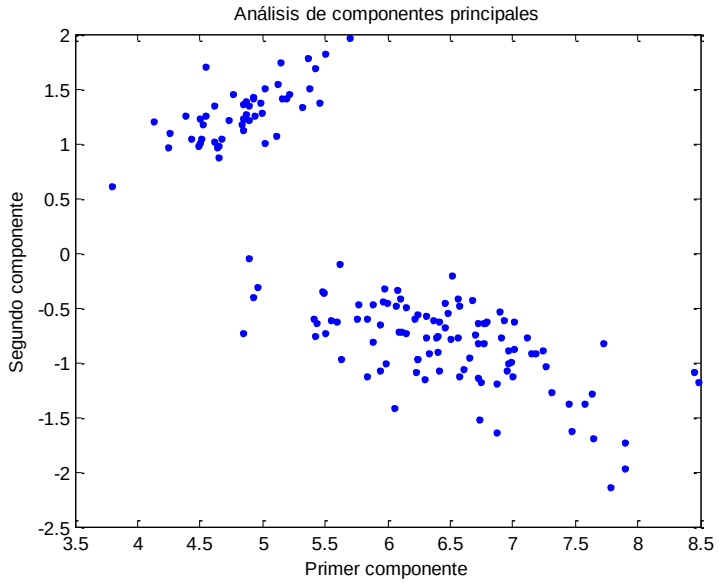

Figura 42.- ACP matriz covarianza, 150 datos IRIS.

Si en lugar de seleccionar los 150 datos, se selecciona solamente 20 se obtendría el siguiente resultado:

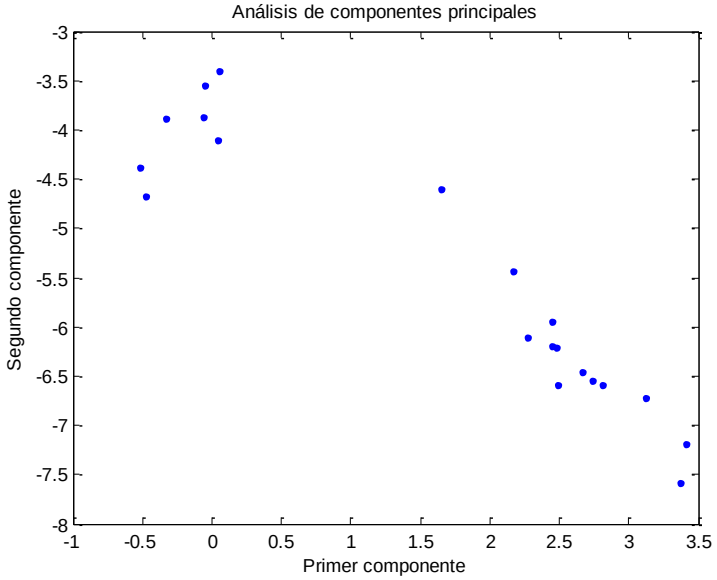

Figura 43.- ACP matriz covarianza, 20 datos IRIS.

Si en lugar de seleccionar los datos de IRIS, se selecciona el total conjunto de datos WINE (178 patrones), se obtiene el siguiente resultado:

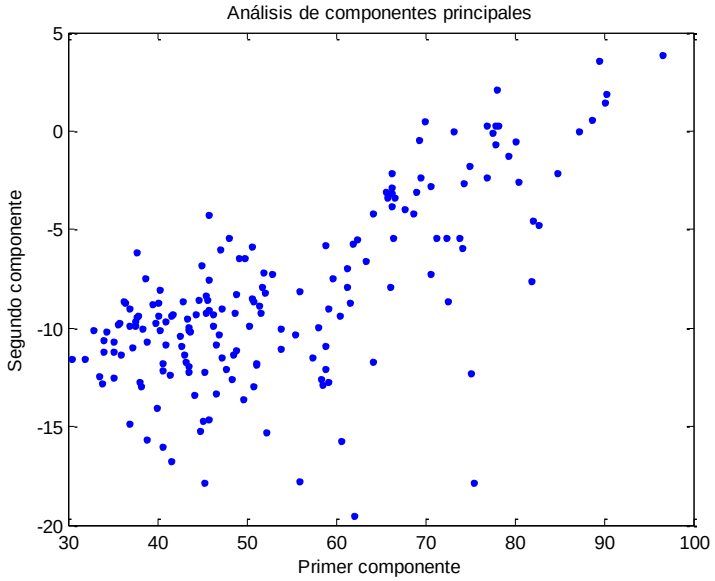

Figura 44.- ACP matriz covarianza, 178 datos WINE.

Y por último se muestra la representación de 30 datos de WINE elegidos aleatoriamente.

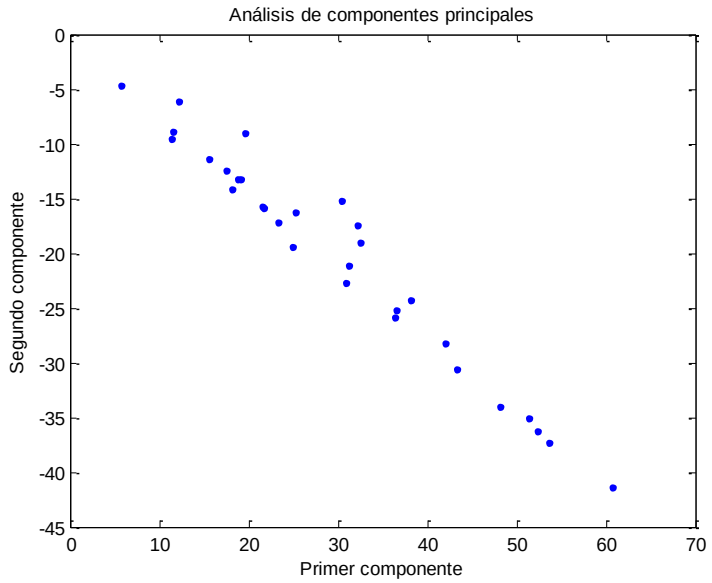

Figura 45.- ACP matriz covarianza, 30 datos WINE.

Como se puede observar en las figuras anteriores, tras aplicar el análisis de componentes principales, se produce una reducción de la dimensionalidad de los patrones, pasando los mismo de tener *N* dimensiones, a sólo dos, manteniendo una estructura similar, con lo que se puede proceder a su representación gráfica, cosa que no sería posible para patrones de más de tres dimensiones. También tras reducir la dimensionalidad, ya se puede observar la distribución de los patrones, con lo que se puede realizar un análisis de la tendencia de clustering.

6.3.2. Análisis de componentes principales matriz de correlación

Si se presiona el botón del análisis de componentes principales de la matriz de correlación, se ejecuta el siguiente código del programa *Inicio* de Matlab:

```
% --- Executes on button press in ACP_correlacion.
function ACP_correlacion_Callback(hObject, eventdata, handles)
% hObject    handle to ACP_correlacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<1
    orden_incorrecto;
else
    load('datos');
    estado=2;
    save('est','estado')
    [datos_tendencia]=ACP_corr(dato);
    save('dat','datos_tendencia')
end
```

Función 18.- Botón Análisis Componentes principlaes matriz correlacion.

La explicación del código es la siguiente, en primer lugar se comprueba que se hayan seleccionado los datos y posteriormente se incrementa el estado y se ejecuta la función *ACP_corr* y los datos resultantes de ejecutar esa función son almacenados, para proceder a efectuar los siguientes apartados del programa. El código de dicha función es el siguiente:

```
%Función que realiza el análisis de componentes principales a partir
de la matriz de correlación
function [Y]=ACP_corr(dato)
[f,c]=size(dato);%análisis del número de datos
%1° se tipifican los datos
media=mean(dato);
varianza=var(dato);
for j=1:c
   for i=1:f
    dato_tip(i,j)=(dato(i,j)-media(j)/varianza(j));
   end
end
%2° Obtención de la matriz de correlación y covarianza
R=corr(dato_tip);
S=cov(dato_tip,1);

%3° Se hayan los componentes principales a través de la matriz de
%correlación
[T1,D1]=eigsort(R);%Función que ordena los valores y vectores
propios
T1=T1';

%4° Corrección de los signos
if ((sum(sign(T1(:,1)))<0)&(sum(sign(T1(:,2)))<0))
    T1=-T1;
end

%5°se estandarizan las variables
s=diag(sqrt(diag(S)));
Y1=dato*inv(s)*T1;

%6° Representación de los datos
Y=Y1(:,1:2);
figure
plot(Y1(:,1),Y1(:,2),'.b')
xlabel('Primer componente')
ylabel('Segundo componente')
title('Análisis de componentes principales')
end
```

Función 19.- ACP_corr.

El funcionamiento de esta función es bastante similar a la función *ACP_cov*, explicada anteriormente, primero se tipifican los datos y en el siguiente paso se hayan las matrices de correlación y covarianza. Luego se hayan los componentes principales a partir de la matriz de correlación, para ello se emplea la función *eigsort*, que ordena los valores y los vectores propios, posteriormente se realiza la corrección de los símbolos y se estandarizan las variables. El último paso consiste en la representación de los datos.

Se van a representar varios resultados obtenidos, al aplicar dicho método de reducción de la dimensionalidad, se han utilizado los mismos datos tanto para el análisis de componentes principales a partir de la matiz de covarianza, como para a partir de la

matriz de correlación. El resultado obtenido tras aplicar el análisis a los 150 patrones del conjunto de datos IRIS es el siguiente:

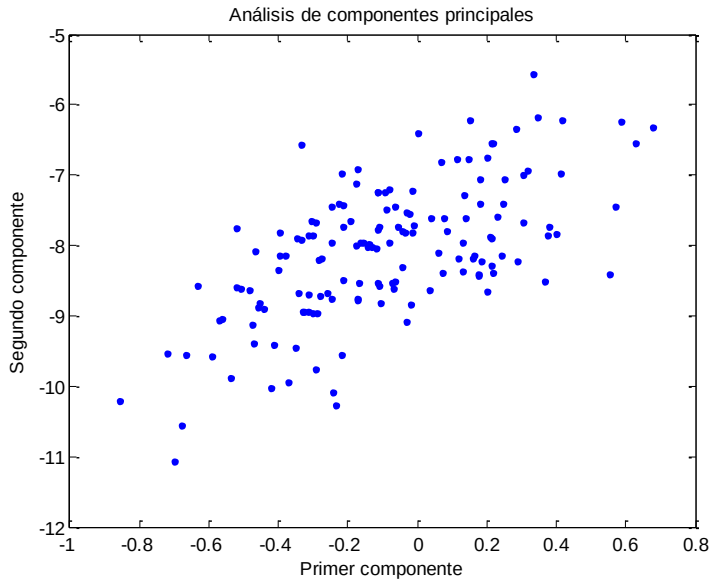

Figura 46.- ACP matriz correlación, 150 datos IRIS.

Si en lugar de seleccionar los 150 patrones del conjunto de datos IRIS, se selecciona sólo 20 el resultado es el siguiente:

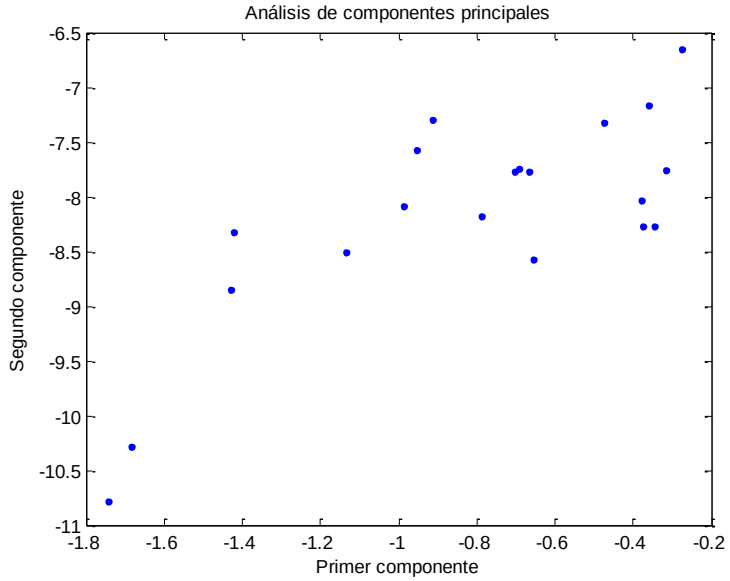

Figura 47.- ACP matriz correlación, 150 datos IRIS.

A continuación se han seleccionado todos los datos del conjunto Wine (178 patrones), obteniéndose el siguiente resultado:

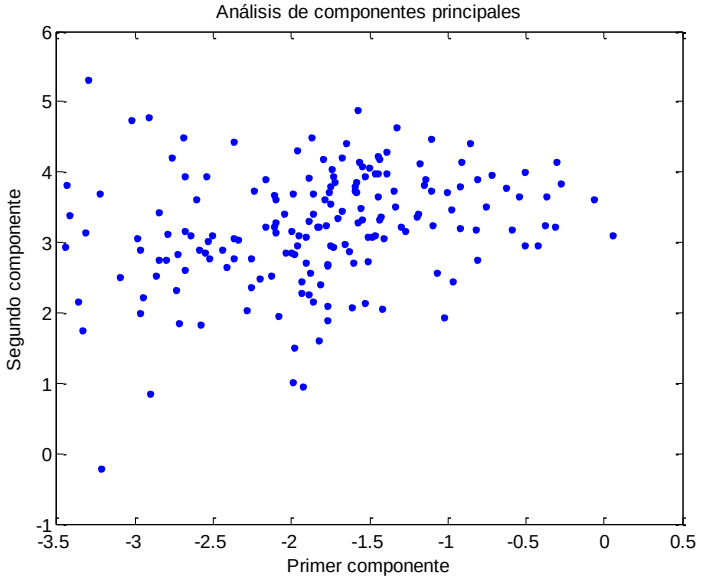

Figura 48.- ACP matriz correlación, 178 datos WINE.

Por último se han seleccionado 30 de los datos de WINE, obteniéndose el siguiente resultado:

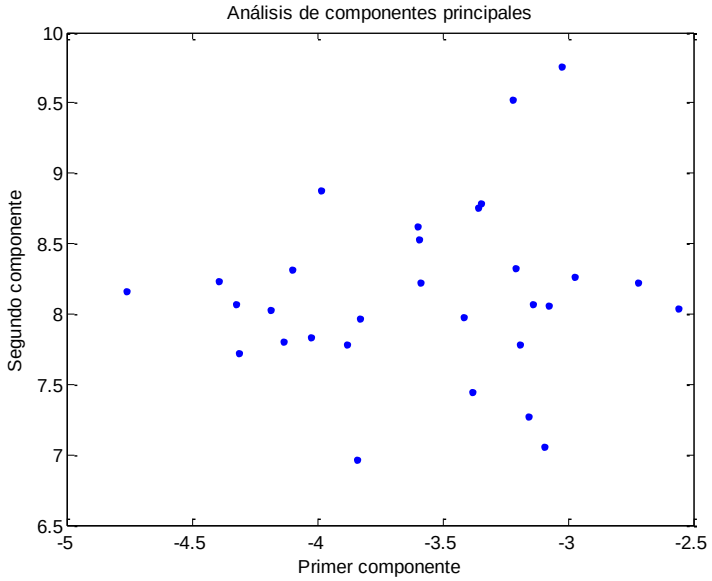

Figura 49.- ACP matriz correlación, 30 datos WINE.

Con ambas técnias de análisis de componentes principales se produce una reducción de la dimensionalidad de los datos, con lo que se permite trabajar más cómodamente con ellos y permite realizar una representación en dos dimensiones. El resultado es diferente según se aplique la matriz de covarianza o de correlación, pero la proximidad entre los datos se mantiene.

6.3.3. Mapeo de Sammon

Cuando se presiona el botón de "*Mapeo de Sammon"*, se ejecuta el código que se muestra a continuación del programa GUI de Matlab donde se produce la llamada a la función *Sammon*.

```
% --- Executes on button press in Mapeo_Sammon.
function Mapeo_Sammon_Callback(hObject, eventdata, handles)
% hObject    handle to Mapeo_Sammon (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<1
    orden_incorrecto;
else
    load('datos');
    estado=2;
    save('est','estado')
    datos_tendencia = sammon(dato)
    save('dat','datos_tendencia')
end
```

Función 20.- Botón Mapeo de Sammon.

El código del programa que se ejecuta cuando se presiona el botón "*Mapeo de Sammon"*, es igual al que se ejecuta cuando se presiona alguno de los botones de análisis de componentes principales, con la salvedad que ahora se produce la llamada a la función *sammon.* El código Matlab de dicha función es el siguiente:

```
%Función que realiza el mapeado de Sammon
function [y, E] = sammon(x)
%Número de dimensiones
n=2;
%Creación de la matriz de distancia
D = euclid(x, x);
%Inicialización
N     = size(x, 1);
scale = 0.5/sum(D(:));
D     = D + eye(N);
Dinv  = 1./D;
%Aleatoriedad y comienzo
y = randn(N, n);
one   = ones(N,n);
d     = euclid(y,y) + eye(N);
dinv  = 1./d;
delta = D - d;
E     = sum(sum((delta.^2).*Dinv));

%Cálculo del gradiente
delta    = dinv - Dinv;
deltaone = delta*one;
g        = delta*y - y.*deltaone;
dinv3    = dinv.^3;
y2       = y.^2;
H        = dinv3*y2 - deltaone -2*y.*(dinv3*y) + y2.*(dinv3*one);
s        = -g(:)./abs(H(:));
y_old    = y;
%Terminación del proceso
y(:) = y_old(:) + s;
d     = euclid(y,y) + eye(N);
dinv  = 1./d;
delta = D - d;
E_new = sum(sum((delta.^2).*Dinv));

%Representación gráfica
figure
plot(y(:,1),y(:,2),'.b')
xlabel('Primer componente')
ylabel('Segundo componente')
title('Mapeo de Sammon')
```

Función 21.- Sammon.

El funcionamiento del programa es el siguiente, en primer lugar se selecciona el número de dimensiones del nuevo mapa, y se calcula la matriz de distancia, la cual se calcula a través de la función *euclid*, posteriormente se realiza la inicialización del programa otorgándole al sistema una componente de aleatoriedad. A continuación se calcula el gradiente y se termina el proceso con la representación gráfica del mismo. El código correspondiente a la función *euclid*, es el siguiente:

```
%Función que calcula la distancia euclídea entre los puntos de dos
matrices
function d = euclid(x,y)
d =
sqrt(sum(x.^2,2)*ones(1,size(y,1))+ones(size(x,1),1)*sum(y.^2,2)'-
2*(x*y'));
```

Función 22.-euclid.

A continuación se muestran algunos ejemplos de aplicar el mapeo de Sammon como técnica de reducción de la dimensionalidad, cabría decir que al ser una técnica que tiene una componente aleatoria y está basada en el gradiente, por lo que para dos conjuntos de datos iguales el resultado puede ser distinto. Empleando el total del conjunto de datos de IRIS, se obtiene el siguiente resultado:

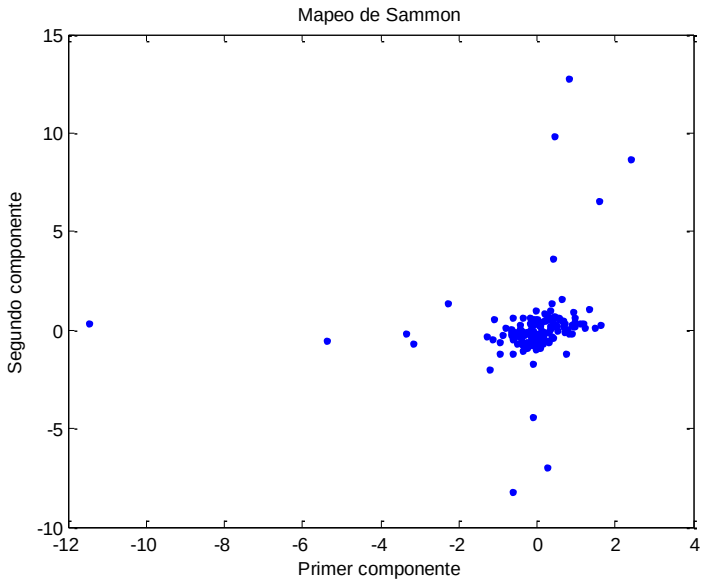

Figura 50.- Mapeo de Sammon, 150 datos IRIS.

Si en lugar de seleccionar todo el conjunto de datos de IRIS, seleccionamos sólo 20 se obtiene la siguiente figura:

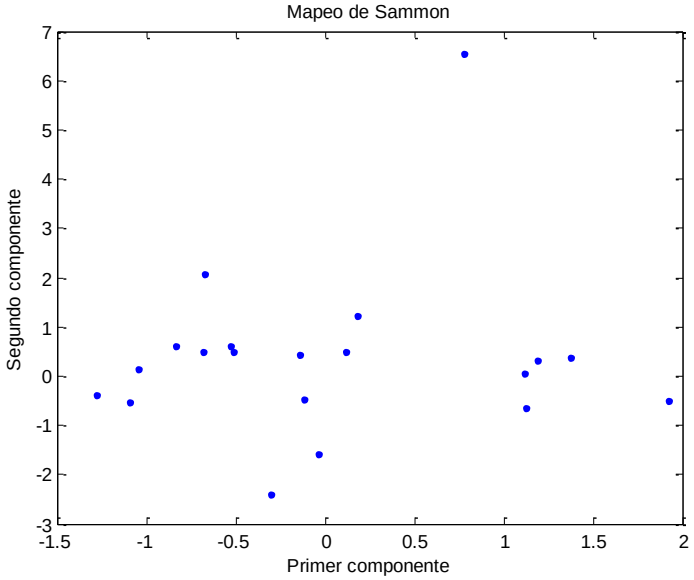

Figura 51.- Mapeo de Sammon, 20 datos IRIS.

Seleccionando el conjunto de datos de WINE, se obtiene la siguiente figura:

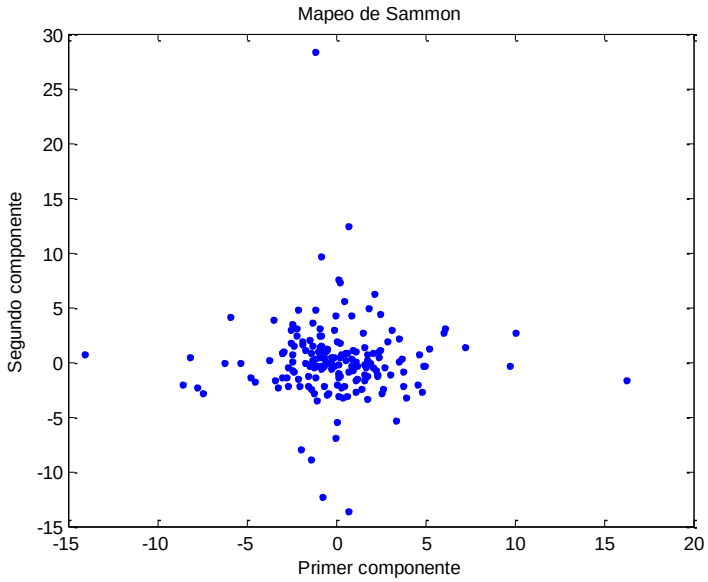

Figura 52.- Mapeo de Sammon, 178 datos WINE.

Y si en lugar de los 178 datos de WINE, se selecciona sólo 30, el resultado es:

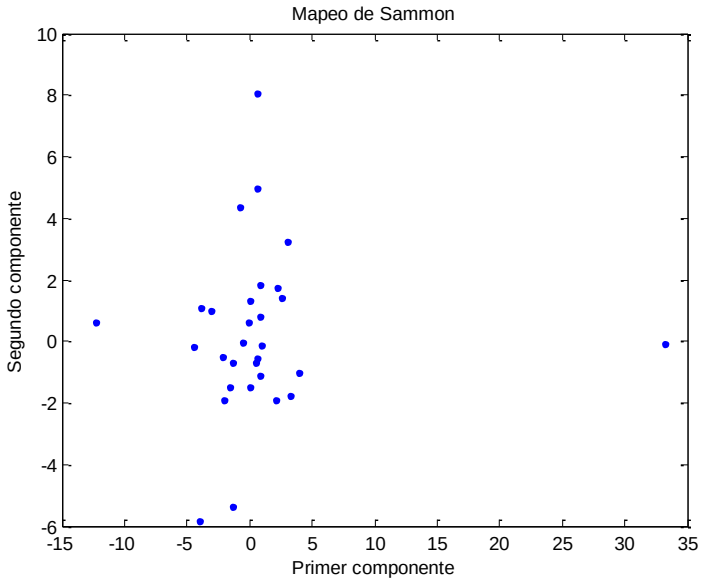

Figura 53.- Mapeo de Sammon, 30 datos WINE.

En las figuras anteriores se observa la distribución de los componentes o patrones del conjunto de datos con el que se está trabajando, observando la distribución de estos, se puede preveer el número de clusters a crear (algoritmo K-means) o la distancia entre los clusters (algoritmo Adaptativo).

6.4. Algoritmos de clustering

En la siguiente sección se da la opción al usuario de elegir entre tres algoritmos de clustering, para poder realizar la clasificación de los datos a los que previamente se les ha realizado un análisis de tendencia y se les ha reducido la dimensionalidad a dos, para poder realizar una representación en dos dimensiones. Esta parte del programa es la siguiente:

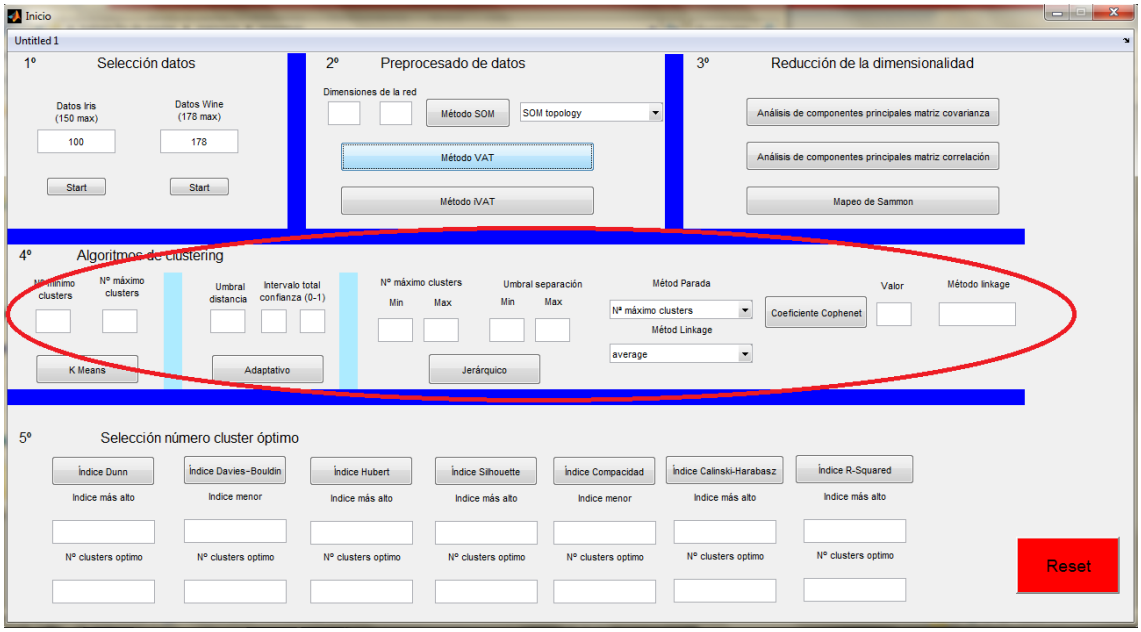

Figura 54.- Algoritmos de clustering.

Los tres tipos de algoritmos de clustering que se han empleado en este trabajo son:

- K-Means.
- Adaptativo.
- Jerárquico.

6.4.1. Algoritmo K-means

El algoritmo K-means se ejecuta cuando se presiona el botón del programa *"K-Means"*, donde previamente se han tenido que introducir el número mínimo de Clusters que se van a crear y el número máximo, en sus correspondientes casilleros, esto es necesario, ya que el algoritmo K-means necesita que se le pase por definición como parámetro de entrada el número de clusters que se quieren crear. Esto se observa mejor en la siguiente figura:

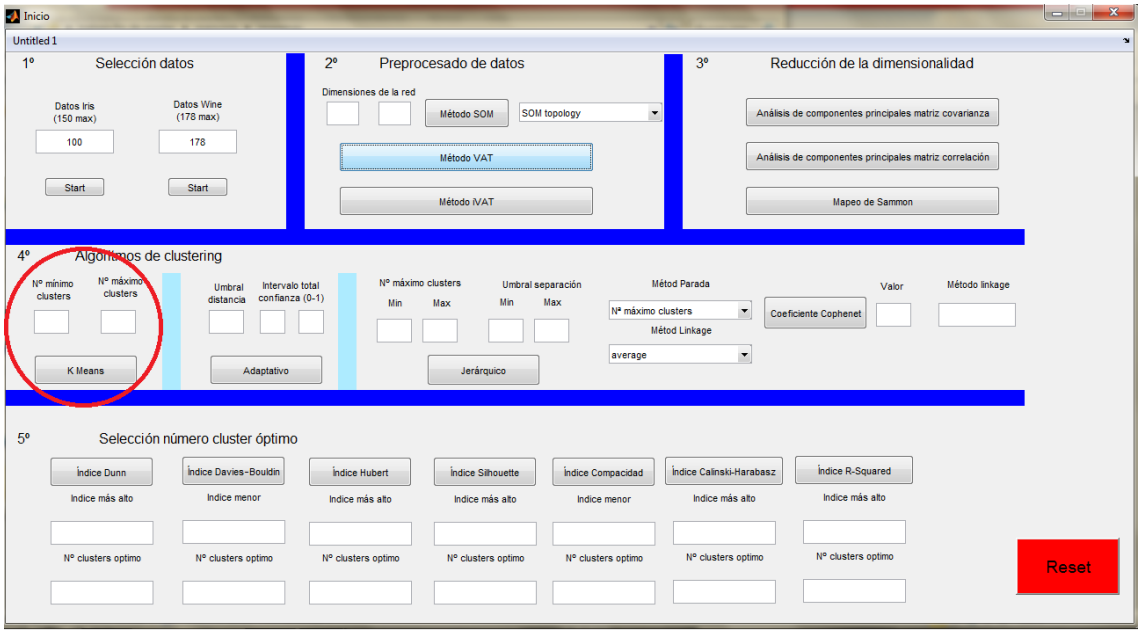

Figura 55.- Algoritmo K-Means.

El código del programa *Inicio.fig* que se ejecuta cuando se pulsa el botón *"K Means"*, es el siguiente:

```
% --- Executes on button press in al_kmeans.
function al_kmeans_Callback(hObject, eventdata, handles)
% hObject    handle to al_kmeans (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
num_min=str2double( get(handles.n_min, 'string') );
num_max=str2double( get(handles.n_max, 'string') );
load('est');
if estado<2
    orden_incorrecto;
else
    load('dat');
    estado=3;
    save('est','estado')
    j=1;
    [f,c]=size(datos_tendencia);
    numero=num_max-num_min+1;
    Pos_patrones=zeros(numero,f,c+1);
    for i=num_min:num_max
        [A,C]=K_means(datos_tendencia,i)
        Pos_patrones(j,:,:)=A;
        j=j+1;
    end
    datos_cluster=Pos_patrones;
    save('dat_cluster','datos_cluster')
end
```

Función 23.- Botón K Means.

El funcionamiento de esta parte del programa es el siguiente, en primer lugar una vez se ha entrado en la función se recogen los valores del número mínimo y máximo de clusters que se van a crear. Esto se realiza a través de las funciones de Matlab *get* y *str2double*, la primera recoge un valor introducido por pantalla y la segunda convierte una cadena de caracteres a un valor numérico.

El siguiente paso consiste en comparar la variable de estado del programa, se supone que el estado mínimo al llegar a esta fase debería ser dos, ya que se deberían de haber seleccionado los datos y haberse realizado el proceso de reducción de la dimensionalidad. A continuación se incrementa la variable *estado* del programa y se guarda su valor, y se calcula el número de veces que se realizará la llamada a la función *K_means*. Por último se realiza un bucle secuencial en el que se producen progresivas llamadas a la función que calcula la distribución de los patrones, pasándole el número de agrupaciones a crear y se almacenan los resultados.

El código de la función *K_means*, que es donde se produce el algoritmo es el siguiente:

```
%%Función que ejecuta el algoritmo K_Means
%D matriz de datos
%n número de agrupamientos
%A conjunto de patrones (agrupamientos) y centroides
%C centro de los agrupamientos
function [A,C]=K_means(D,n)
A=D;
%%1° Inicialización de los centros
C=zeros(n,2);
for i=1:n
    C(i,:)=D(i,:);
end
C_old=zeros(n,2);%Se guarda el actual valor de los centros para
compararlo con el valor anterior en la siguiente iteración

[f,c]=size(D);%Tamaño de la matriz
while C ~=C_old
      C_old=C;
      for i=1:f
          distancia_old=1000000;%Inicialización de la variable
      distancia a un valor muy grande
          for j=1:n
              distancia=sqrt((C(j,1)-D(i,1))^2+(C(j,2)-D(i,2))^2);
              if distancia<distancia_old
                   A(i,3)= j;
                   distancia_old=distancia;
              end
          end
      end
```

```
        %%2° Cálculo de los nuevos centros
        for j=1:n
            numero=0;
            Cx_acum=0;
            Cy_acum=0;
            for i=1:f
               if A(i,3)==j
                   Cx_acum=A(i,1)+Cx_acum;
                   Cy_acum=A(i,2)+Cy_acum;
                   numero=numero+1;
               end
            end
            Cx=Cx_acum/numero;
            Cy=Cy_acum/numero;
            C(j,1)=Cx;
            C(j,2)=Cy;
        end
end
%%3° Representación gráfica
representacion(A);%%Representación gráfica
title('Algoritmo K means')
```

Función 24.- K_means.

El funcionamiento del programa es el siguiente, en primer lugar se realiza una inicialización de los centros, en función del número de agrupaciones que se quiere crear. A continuación se crea una iteración en la cual los distintos patrones se van agrupando progresivamente en los distintos clusters y se van recalculando los nuevos centros de cada cluster. Para comprobar si los patrones deben de pertenecer a un cluster u a otro se calcula la distancia de dicho patrón a todos los centros de los clusters ya creados y se asigna al más cercano, recalculándose los centros nuevamente. La iteración termina cuando tras dos iteraciones consecutivas los centros de los agrupamientos o clusters realizados no se han movido, por lo que se considera que ya se ha hecho el mejor agrupamiento posible. Una vez se ha terminado el algoritmo, se procede a realizar la llamada a la función representación, la cual se encarga de representar la agrupación realizada, el código de dicha función es el siguiente:

```
%Función que representa gráficamente unos clusters dados
function representacion(A)
[f,c]=size(A);%Tamaño de la matriz
%%Representación gráfica
figure
hold on

for i=1:f
    resto=mod(A(i,3),7);%%Resto de la división
    cociente=floor(A(i,3)/7);%Cociente de la división
    if cociente==0
        if resto==0
              plot(A(i,1), A(i,2),'.b')
        elseif resto==1
              plot(A(i,1), A(i,2),'.r')
        elseif resto==2
              plot(A(i,1), A(i,2),'.k')
        elseif resto==3
              plot(A(i,1), A(i,2),'.g')
        elseif resto==4
              plot(A(i,1), A(i,2),'.y')
        elseif resto==5
              plot(A(i,1), A(i,2),'.c')
        elseif resto==6
              plot(A(i,1), A(i,2),'.m')
        end
    end
    if cociente==1
        if resto==0
              plot(A(i,1), A(i,2),'db')
        elseif resto==1
              plot(A(i,1), A(i,2),'dr')
        elseif resto==2
              plot(A(i,1), A(i,2),'dk')
        elseif resto==3
              plot(A(i,1), A(i,2),'dg')
        elseif resto==4
              plot(A(i,1), A(i,2),'dy')
        elseif resto==5
              plot(A(i,1), A(i,2),'dc')
        elseif resto==6
              plot(A(i,1), A(i,2),'dm')
        end
    end
    if cociente==2
        if resto==0
              plot(A(i,1), A(i,2),'ob')
        elseif resto==1
              plot(A(i,1), A(i,2),'or')
        elseif resto==2
              plot(A(i,1), A(i,2),'ok')
        elseif resto==3
              plot(A(i,1), A(i,2),'og')
        elseif resto==4
```

```
            plot(A(i,1), A(i,2),'oy')
        elseif resto==5
            plot(A(i,1), A(i,2),'oc')
        elseif resto==6
            plot(A(i,1), A(i,2),'om')
        end
    end
    if cociente==3
        if resto==0
            plot(A(i,1), A(i,2),'hb')
        elseif resto==1
            plot(A(i,1), A(i,2),'hr')
        elseif resto==2
            plot(A(i,1), A(i,2),'hk')
        elseif resto==3
            plot(A(i,1), A(i,2),'hg')
        elseif resto==4
            plot(A(i,1), A(i,2),'hy')
        elseif resto==5
            plot(A(i,1), A(i,2),'hc')
        elseif resto==6
            plot(A(i,1), A(i,2),'hm')
        end
    end
    if cociente==4
        if resto==0
            plot(A(i,1), A(i,2),'*b')
        elseif resto==1
            plot(A(i,1), A(i,2),'*r')
        elseif resto==2
            plot(A(i,1), A(i,2),'*k')
        elseif resto==3
            plot(A(i,1), A(i,2),'*g')
        elseif resto==4
            plot(A(i,1), A(i,2),'*y')
        elseif resto==5
            plot(A(i,1), A(i,2),'*c')
        elseif resto==6
            plot(A(i,1), A(i,2),'*m')
        end
    end
    if cociente==5
        if resto==0
            plot(A(i,1), A(i,2),'+b')
        elseif resto==1
            plot(A(i,1), A(i,2),'+r')
        elseif resto==2
            plot(A(i,1), A(i,2),'+k')
        elseif resto==3
            plot(A(i,1), A(i,2),'+g')
        elseif resto==4
            plot(A(i,1), A(i,2),'+y')
        elseif resto==5
            plot(A(i,1), A(i,2),'+c')
```

```
        elseif resto==6
              plot(A(i,1), A(i,2),'+m')
        end
    end
    if cociente==6
        if resto==0
              plot(A(i,1), A(i,2),'xb')
        elseif resto==1
              plot(A(i,1), A(i,2),'xr')
        elseif resto==2
              plot(A(i,1), A(i,2),'xk')
        elseif resto==3
              plot(A(i,1), A(i,2),'xg')
        elseif resto==4
              plot(A(i,1), A(i,2),'xy')
        elseif resto==5
              plot(A(i,1), A(i,2),'xc')
        elseif resto==6
              plot(A(i,1), A(i,2),'xm')
        end
    end
    if cociente==7
        if resto==0
              plot(A(i,1), A(i,2),'pb')
        elseif resto==1
              plot(A(i,1), A(i,2),'pr')
        elseif resto==2
              plot(A(i,1), A(i,2),'pk')
        elseif resto==3
              plot(A(i,1), A(i,2),'pg')
        elseif resto==4
              plot(A(i,1), A(i,2),'py')
        elseif resto==5
              plot(A(i,1), A(i,2),'pc')
        elseif resto==6
              plot(A(i,1), A(i,2),'pm')
        end
    end
    if cociente==8
        if resto==0
              plot(A(i,1), A(i,2),'sb')
        elseif resto==1
              plot(A(i,1), A(i,2),'sr')
        elseif resto==2
              plot(A(i,1), A(i,2),'sk')
        elseif resto==3
              plot(A(i,1), A(i,2),'sg')
        elseif resto==4
              plot(A(i,1), A(i,2),'sy')
        elseif resto==5
              plot(A(i,1), A(i,2),'sc')
        elseif resto==6
              plot(A(i,1), A(i,2),'sm')
        end
```

```
    end
    if cociente==9
        if resto==0
              plot(A(i,1), A(i,2),'vb')
        elseif resto==1
              plot(A(i,1), A(i,2),'vr')
        elseif resto==2
              plot(A(i,1), A(i,2),'vk')
        elseif resto==3
              plot(A(i,1), A(i,2),'vg')
        elseif resto==4
              plot(A(i,1), A(i,2),'vy')
        elseif resto==5
              plot(A(i,1), A(i,2),'vc')
        elseif resto==6
              plot(A(i,1), A(i,2),'vm')
        end
    end
    if cociente==10
        if resto==0
              plot(A(i,1), A(i,2),'^b')
        elseif resto==1
              plot(A(i,1), A(i,2),'^r')
        elseif resto==2
              plot(A(i,1), A(i,2),'^k')
        elseif resto==3
              plot(A(i,1), A(i,2),'^g')
        elseif resto==4
              plot(A(i,1), A(i,2),'^y')
        elseif resto==5
              plot(A(i,1), A(i,2),'^c')
        elseif resto==6
              plot(A(i,1), A(i,2),'^m')
        end
    end
    if cociente==11
        if resto==0
              plot(A(i,1), A(i,2),'<b')
        elseif resto==1
              plot(A(i,1), A(i,2),'<r')
        elseif resto==2
              plot(A(i,1), A(i,2),'<k')
        elseif resto==3
              plot(A(i,1), A(i,2),'<g')
        elseif resto==4
              plot(A(i,1), A(i,2),'<y')
        elseif resto==5
              plot(A(i,1), A(i,2),'<c')
        elseif resto==6
              plot(A(i,1), A(i,2),'<m')
        end
    end
    if cociente==12
        if resto==0
```

```
                plot(A(i,1), A(i,2),'>b')
            elseif resto==1
                plot(A(i,1), A(i,2),'>r')
            elseif resto==2
                plot(A(i,1), A(i,2),'>k')
            elseif resto==3
                plot(A(i,1), A(i,2),'>g')
            elseif resto==4
                plot(A(i,1), A(i,2),'>y')
            elseif resto==5
                plot(A(i,1), A(i,2),'>c')
            elseif resto==6
                plot(A(i,1), A(i,2),'>m')
            end
        end
    end
```

Función 25.- representación.

Un ejemplo de cómo se introducen los datos se muestra a continuación, en este caso se han seleccionado la totalidad de los datos de IRIS y se ha realizado una reducción de la dimensionalidad por el análisis de componentes principales matriz de covarianza, se ha seleccionado como número mínimo de clusters 3 y como máximo 5, esto se muestra en la siguiente imagen:

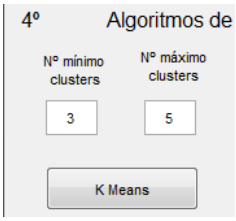

Figura 56.- Ejemplo algoritmo K-Means.

Tras presionar el botón *"K Means"*, el programa nos muestra las tres imágenes siguientes:

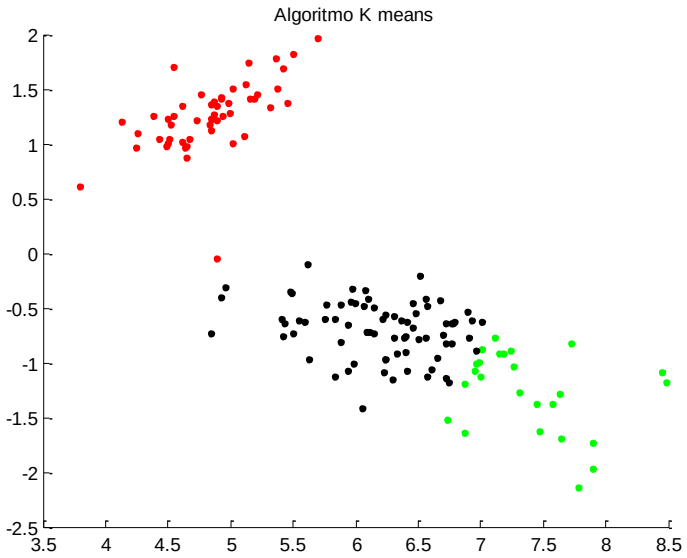

Figura 57.- Algoritmo K-Means tres clusters.

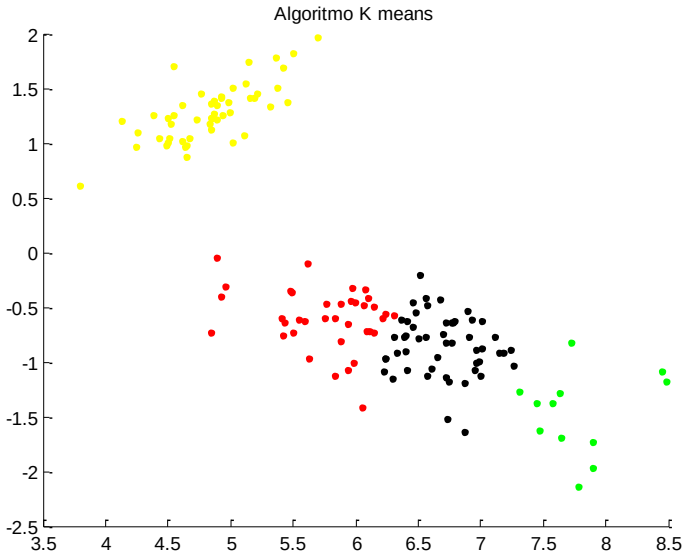

Figura 58.- Algoritmo K-Means cuatro clusters.

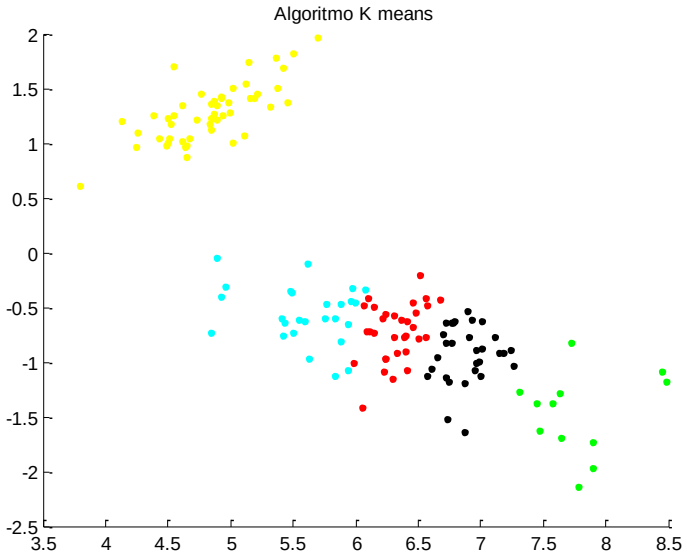

Figura 59.- Algoritmo K-Means cinco clusters.

Y en el Command Window de Matlab, por cada imagen representada se nos muestra la siguiente información:

```
A =

   6.4047   -1.0735    2.0000

   6.5710   -1.1310    2.0000

                .

                .

                .

   5.4887   -0.3563    2.0000

   4.8854    1.2186    1.0000

C =

   4.8342    1.2583

   6.1897   -0.6955

   7.3848   -1.2584
```

Donde cada una de las filas de la matriz resultante *A* corresponde a un patrón, siendo las dos primeras columnas su posición y la tercera el cluster al que pertenece, y la variable *C*, representa la posición de los centros de cada cluster.

Si en lugar de seleccionar los datos de IRIS, se selecciona la totalidad de los dato de WINE, y se les aplica la técnica de reducción de la dimensionalidad del Mapeo de Sammon, se obtiene la siguiente distribución.

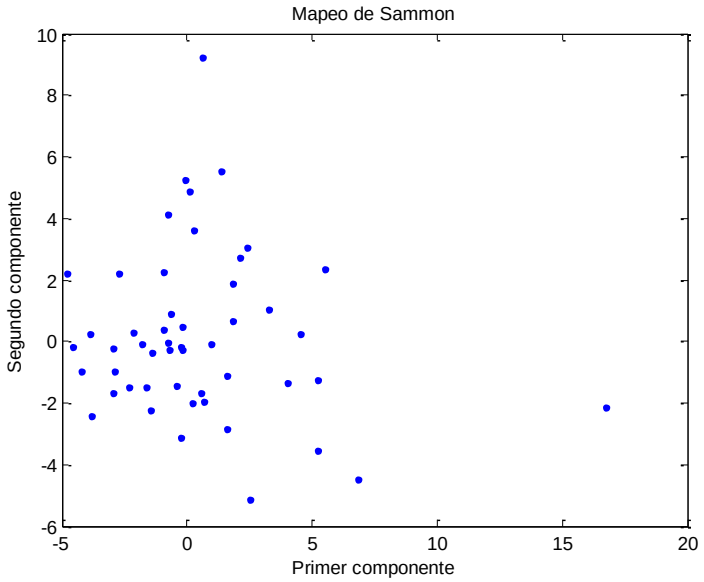

Figura 60.- Mapeo de Sammon, 50 datos WINE.

Realizándole a la distribución anterior un análisis de clustering K-means con número mínimo de clusters 8 y número máximo de clusters 11, se obtienen las siguientes figuras.

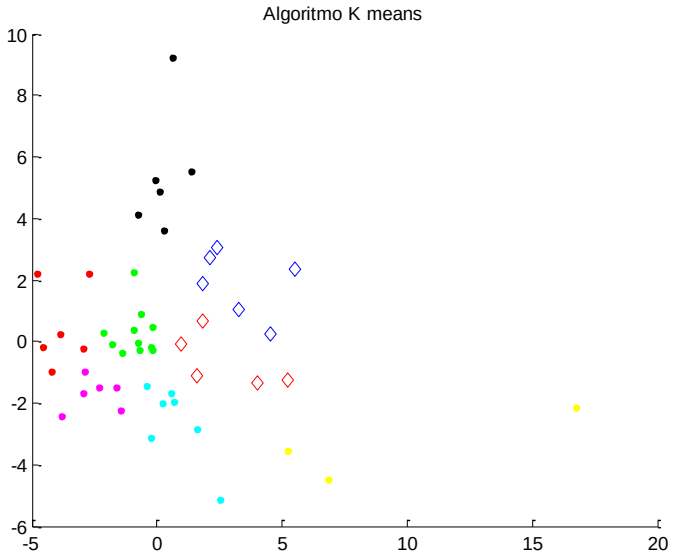

Figura 61.- Algoritmo K-means ocho clusters.

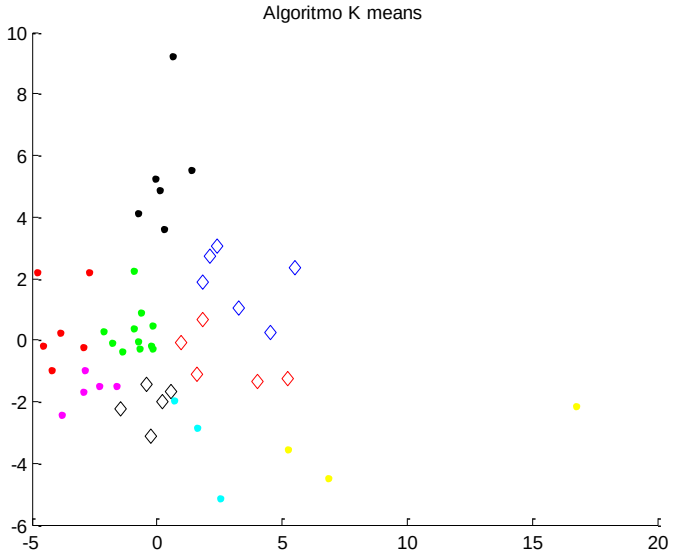

Figura 62.- Algoritmo K-means nueve clusters.

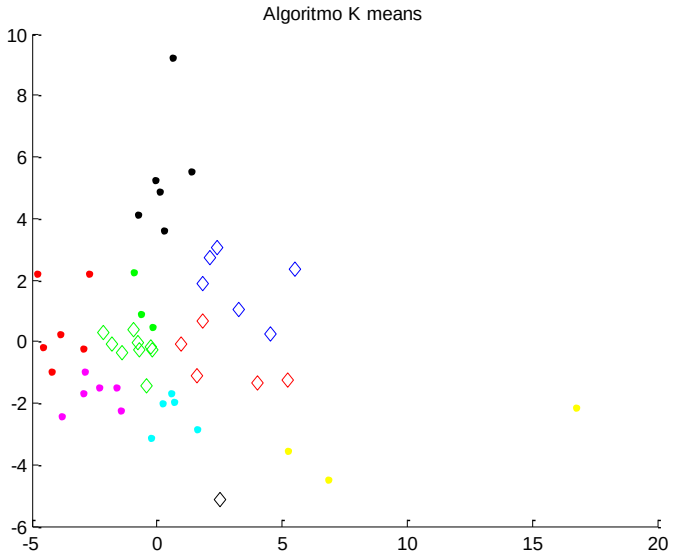

Figura 63.- Algoritmo K-means diez clusters.

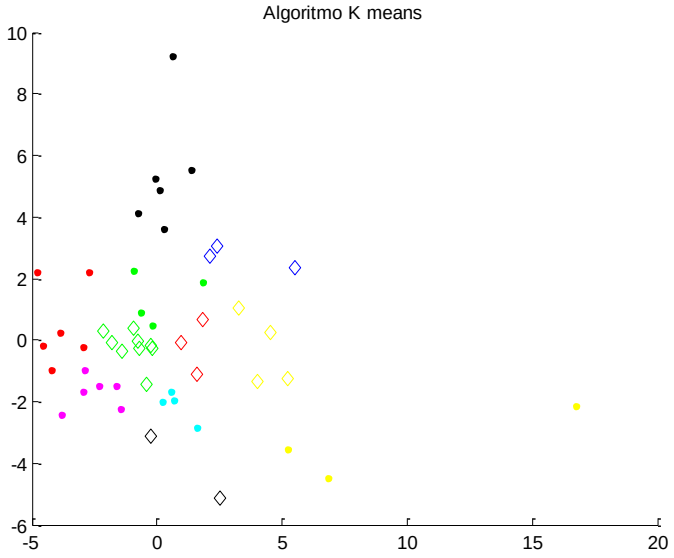

Figura 64.- Algoritmo K-means once clusters.

Como se puede observar las agrupaciones se hacen en función de los patrones que están más cercanos, pero al tratarse de un grupo de asignaciones establecidas y de que el proceso de inicialización se hace por los primeros patrones, puede provocar que la solución esté sesgada y que cambiando el orden de los patrones se obtenga un resultado diferente.

6.4.2. Algoritmo Adaptativo

El algoritmo Adaptativo se ejecuta dentro del programa de Matlab *Inicio*, cuando se presiona el botón del programa Adaptativo, el cual se muestra en la siguiente figura:

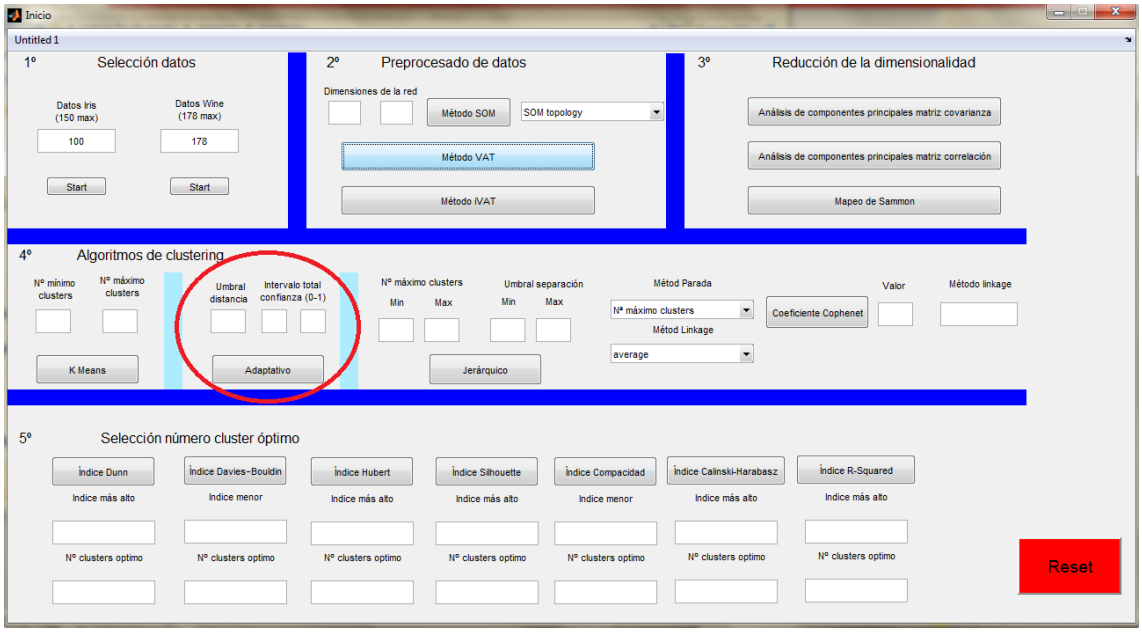

Figura 65.- Algoritmo Adaptativo.

Como bien se ha explicado con anterioridad, en la introducción teórica, el funcionamiento de este algoritmo consiste en la búsqueda de regiones circulares en las que todos los patrones que se encuentren dentro de una misma región de confianza pertenezcan a un mismo cluster, para ello hay que introducir el valor del Umbral de distancia, y dos valores comprendidos entre 0-1, que permitirán hallar para cada caso el intervalo de total confianza. El código del programa *Inicio*, que se ejecuta cuando se pulsa el botón del programa *"Adaptativo"* es el siguiente:

```
% --- Executes on button press in al_adaptativo.
function al_adaptativo_Callback(hObject, eventdata, handles)
% hObject    handle to al_adaptativo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tau=str2double( get(handles.umbral, 'string') );
theta_min=str2double( get(handles.fraccion_min, 'string') );
theta_max=str2double( get(handles.fraccion_max, 'string') );
load('est');
if estado<2
    orden_incorrecto;
else
    load('dat');
    estado=3;
    save('est','estado')
    j=1;
    [f,c]=size(datos_tendencia);
    numero=theta_min:0.05:theta_max;
    long=length(numero);
    if numero(long)~=theta_max;
        numero=[numero theta_max];
    end
    Pos_patrones=zeros(numero,f,c+2);
    for i=numero
        [A,Z,N]=adaptativo(datos_tendencia,tau,i)
        Pos_patrones(j,:,:)=A;
        j=j+1;
    end
    datos_cluster=Pos_patrones;
    save('dat_cluster','datos_cluster')
end
```

Función 26.- Botón Adaptativo.

El funcionamiento de este trozo de código es similar al del resto de los botones del programa, la primera acción que se realiza cuando se pulsa el botón *"Adaptativo"*, es coger los valores de *tau*, *theta_min* y *theta_max*, que son los valores correspondiente al umbral de distancia y a los intervalos de total confianza mínimo y máximo respectivamente. Una vez recogidos los valores introducidos por el usuario, se comprueba que el programa se te ejecutando en el orden correcto, posteriormente se incrementa el estado y se guarda. A continuación se crea un vector con los valores del intervalo de total confianza con los que se ejecutará el programa. Se ha tomado como paso estándar entre el número mínimo y máximo un paso de 0.05. Por último se crea un bucle iterativo de llamadas a la función *Adaptativo*, cada una de ella proporcionará un nuevo agrupamiento, y se mostrará una figura por cada agrupamiento realizado además de mostrar por el Command Windows de Matlab la posición de cada patrón y el cluster al que pertenece. El código de la función adaptativo se muestra en la siguiente figura:

```
%%Algoritmo Adaptativo
%D matriz de datos
%tau umbral de distancia para crear agrupamientos
%theta fracción de tau que determina umbral total confianza
comprendido entre 0 y 1
%A conjunto de patrones (agrupamientos) y centroides
%Z matriz de los centros de los A agrupamientos
%N número de agrupamientos encontrados

function [dato,Z,N]=adaptativo(D,tau,theta)
%Defínción de variables
%A(:,3)=estado; 1=asignado, 2=indeterminado, 3=nuevo
%A(:,4)=agrupamiento asignado
[f,c]=size(D);%Tamaño de la matriz
A=zeros(f,c+2);
A(:,1:2)=D;
N=1;%Inicialmente solo hay un cluster
umbral_asi=tau*theta;%umbral asignado
umbral_ind=tau;%umbral indeterminado
%%1° Inicialización
A(:,3)=3;
A(1,3)=1; %estado asignado
A(1,4)=1;%Agrupamiento asignado 1
Z(1,:)=A(1,1:2);

%%2° Asignación Inicial
for i=2:f
   [fz,cz]=size(Z);
   for j=1:fz
       distancia=abs(sqrt((A(i,1)-Z(j,1))^2+(A(i,2)-Z(j,2))^2));
       if distancia<=umbral_asi
           A(i,3)=1;
           A(i,4)=j;
           %Recalcular centro
              numero=0;
              Cx_acum=0;
              Cy_acum=0;
               for h=1:i
                  if A(h,4)==j
                       Cx_acum=A(h,1)+Cx_acum;
                       Cy_acum=A(h,2)+Cy_acum;
                       numero=numero+1;
                   end
                end
                Cx=Cx_acum/numero;
                Cy=Cy_acum/numero;
                Z(j,1)=Cx;
                Z(j,2)=Cy;
       elseif distancia<=umbral_ind
           A(i,3)=2;%Estado indeterminado
           A(i,4)=0;%No está asignada a ningún centro
       end
   end
   if A(i,3)==3 %Creación de un nuevo centro si no se ha actualizado
el estado
```

```
            Z(fz+1,1)=A(i,1);
            Z(fz+1,2)=A(i,2);
            A(i,3)=1;
            A(i,4)=fz+1;
            N=fz+1;
    end
end
%%3° Reasignar hasta estabilización
estable=false;
contador=0;
while estable==false
    Estado_old=A(:,3);
    grupo_old=A(:,4);
    for i=1:f
        [fz,cz]=size(Z);
        for j=1:fz
            distancia=abs(sqrt((A(i,1)-Z(j,1))^2+(A(i,2)-
Z(j,2))^2));
            if distancia<=umbral_asi
                A(i,3)=1;
                A(i,4)=j;
                %Recalcular centro
                numero=0;
                Cx_acum=0;
                Cy_acum=0;
                    for h=1:f
                        if A(h,4)==j
                            Cx_acum=A(h,1)+Cx_acum;
                            Cy_acum=A(h,2)+Cy_acum;
                            numero=numero+1;
                        end
                    end
                    Cx=Cx_acum/numero;
                    Cy=Cy_acum/numero;
                    Z(j,1)=Cx;
                    Z(j,2)=Cy;
            elseif distancia<=umbral_ind
                    A(i,3)=2;%Estado indeterminado
                    A(i,4)=0;%No está asignada a ningún centro
            end
        end
       if A(i,3)==3 %Creación de un nuevo centro si no se ha
actualizado el estado
                Z(fz+1,1)=A(i,1);
                Z(fz+1,2)=A(i,2);
                A(i,3)=1;
                A(i,4)=fz+1;
                N=fz+1;
       end
    end

    %Comparación de si se ha producido algún cambio o no
    if Estado_old==A(:,3) & grupo_old==A(:,4)
        contador=contador+1;
    end
```

```
    if contador>=2
        estable=true;
    end
end

%%4° Agrupar libremente
%%Los patrones que no están asignado a nngún cluster se asignan a
aquel que están más cercano a tarvés del criterio de mínima
distancia
   for i=1:f
      if   A(i,3)==2
        [fz,cz]=size(Z);
        distancia_minima=10000000;
        for j=1:fz
            distancia=abs(sqrt((A(i,1)-Z(j,1))^2+(A(i,2)-
Z(j,2))^2));
            if distancia<=distancia_minima
                distancia_minima=distancia;
                A(i,3)=1;%Estado determinado
                A(i,4)=j;%asignación al centro más próximo
            end
        end
      end
   end

%5°Eliminación de los clusters que estén vacíos
%%Se realiza porque en el proceso de reasignacón puede ser que un
cluster que esté originalmente compuesto por algún patrón
posteriormente se quede vacío
N=max(A(:,4));%Se mira si hay algún cluster vacío
for i=1:N
  busqueda(i)=false;
end

for i=1:N%Si un cluster está vacío se mantiene como false
    for j=1:f
      if A(j,4)==i
        busqueda(i)=true;
      end
    end
end

contador=0;
for i=1:N%%Se crea una nueva matriz de centros
    if busqueda(i)==true
        contador=contador+1;
        New_Z(contador,:)=Z(i,:);
   end
end
N=contador;
Z=New_Z;
```

```
%%5° Representación gráfica
   figure
   hold on
    [fz,cz]=size(Z);
%se reordenan los datos de salida para que haya concordancia con los
demás algoritmos de clustering
dato=[A(:,1),A(:,2),A(:,4),A(:,3)];

  representacion(dato);
   title('Algoritmo Adaptativo')
```

Función 27.- adaptativo.

El funcionamiento del programa es el siguiente, en primer lugar se calculan los valores de los círculos o regiones de asignamiento e indeterminación, y se produce la inicialización del algoritmo, donde se establece el primer centro. Posteriormente se van haciendo comprobaciones con los demás patrones, incorporándolos a algún cluster ya creado o creando uno nuevo, si el patrón no está dentro de la zona de total confianza, su estado se considera indeterminado. Este proceso se realiza iterativamente hasta que los centros de los cluster no se han actualizado con respecto a la iteración anterior. Cuando esto se produce, se procede a asignar los patrones que se encuentran en zonas indeterminadas a los cluster más cercanos, entendiéndose como más cercano a aquel cuyo centro del cluster está a una menor distancia euclídea del patrón en cuestión. El último paso que se realiza consiste en eliminar los clusters que se encuentren vacíos, debido a que el programa parte de unos clusters iniciales, puede ser que con el paso de las iteraciones, los patrones que inicialmente se encontraban incorporados a ese cluster pasen a otro dejando ese cluster vacío, por lo que el centro asociado a dicho cluster no contenga ningún patrón. Por último se produce la representación gráfica del resultado de clustering realizado. La representación gráfica se ha realizado al igual que con el algoritmo K-Means con la función representación.

Para comprobar el correcto funcionamiento del programa se procede a la representación del mismo, se han seleccionado 50 de los datos de IRIS, y se les ha realizado una reducción de la dimensionalidad a partir del análisis de componentes principales, la imagen resultante es la siguiente:

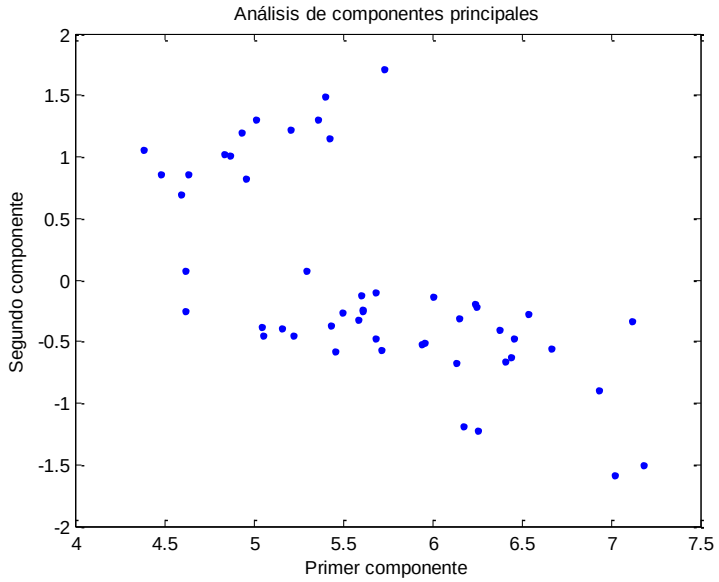

Figura 66.- Análisis de componentes principales, 50 datos IRIS.

Aplicándole al resultado obtenido el algoritmo adaptativo, con un valor de *tau* de 0,8 y valores de *theta* mínima y máxima de 0.7 y 0.8 respectivamente se obtienen las siguientes figuras:

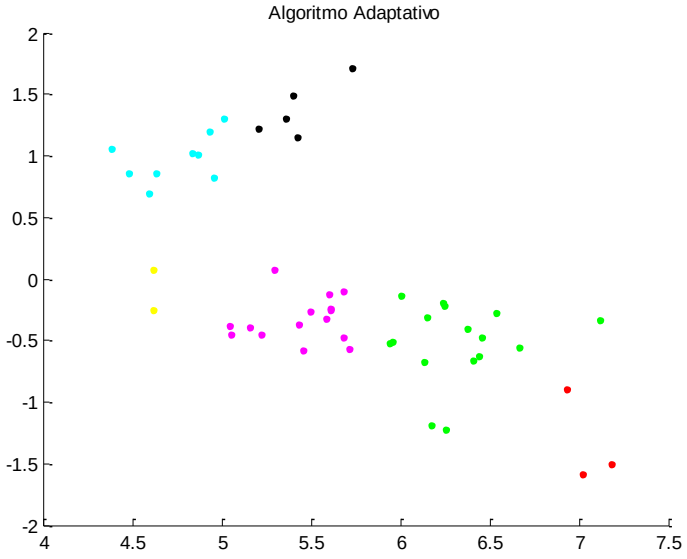

Figura 67.- Algoritmo Adaptativo, τ=0.8 y θ=0.6.

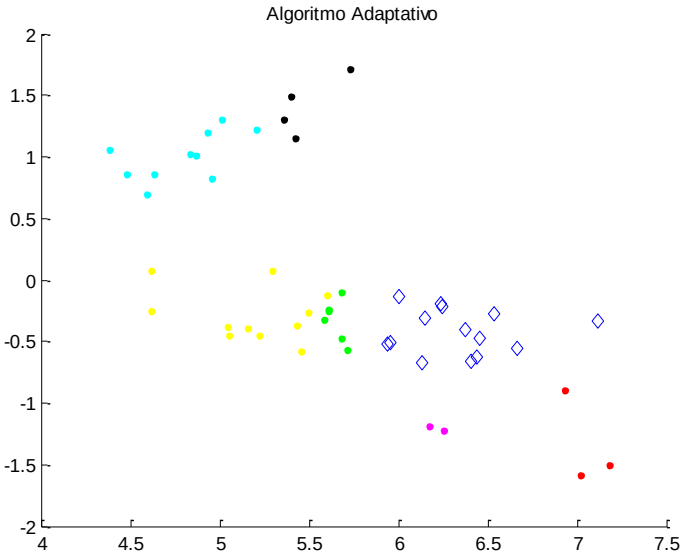

Figura 68.- Algoritmo Adaptativo, τ=0.8 y θ=0.65.

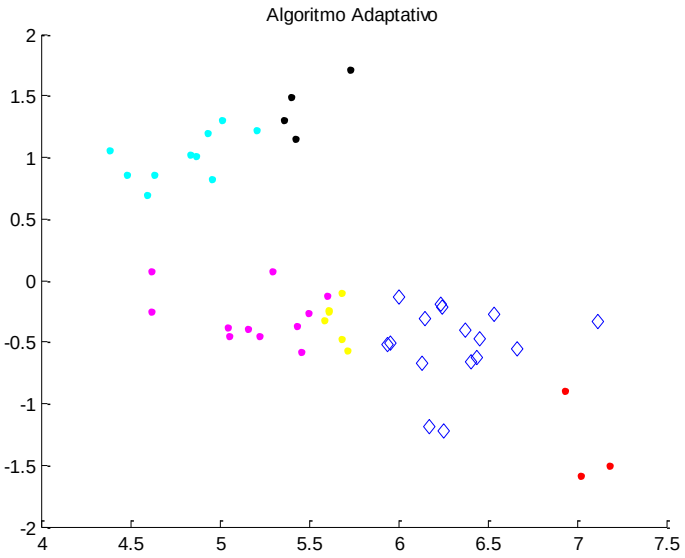

Figura 69.- Algoritmo Adaptativo, τ=0.8 y θ=0.7.

En la primera y en la tercer figura se han creado seis cluster, mientras que en la segunda siete clusters.

Si se eligiera un valor de *tau* o *umbral de distancia* muy grande, como por ejemplo podría ser 4, todos los patrones pertenecerían a un mismo cluster, esto se puede mostrar en la siguiente figura:

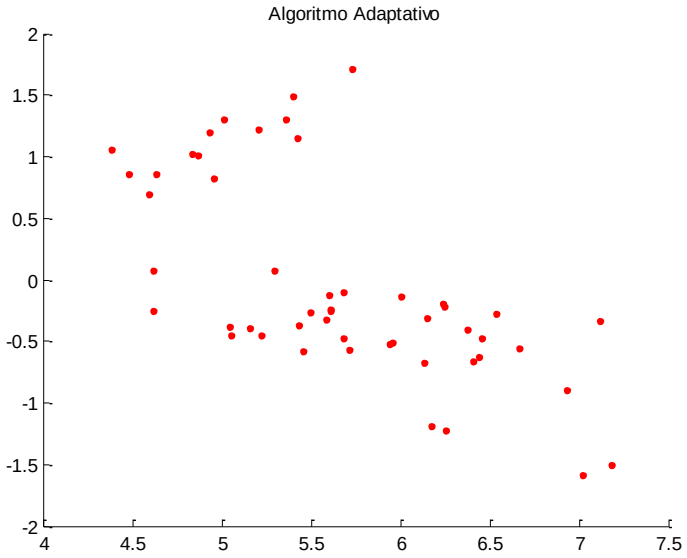

Figura 70.- Algoritmo Adaptativo, τ=4 y θ=0.7.

Por el contrario si se eligiera un valor de *tau* muy pequeño, prácticamente se crearía un patrón por cada cluster, esto se muestra en la figura siguiente:

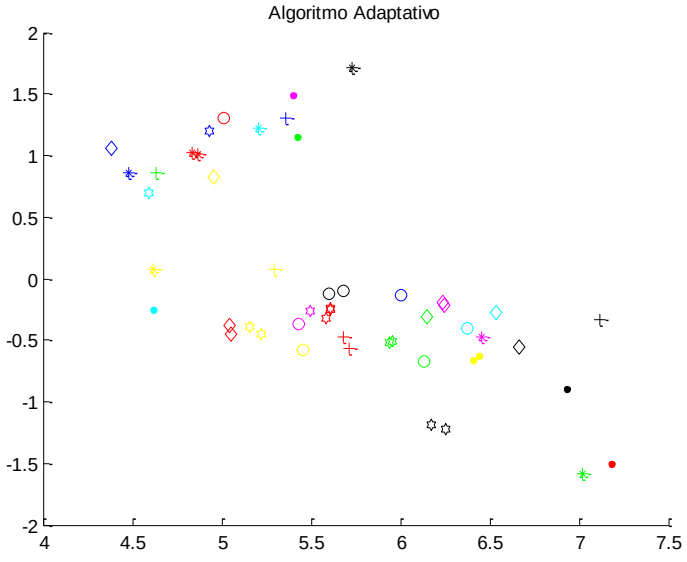

Figura 71.- Algoritmo Adaptativo, τ=0.1 y θ=0.6.

6.4.3. Algoritmo Jerárquico

El último algoritmo de clustering que se presenta en este trabajo, es el algoritmo Jerárquico, que se encuentra en la siguiente parte del programa *Inicio*.

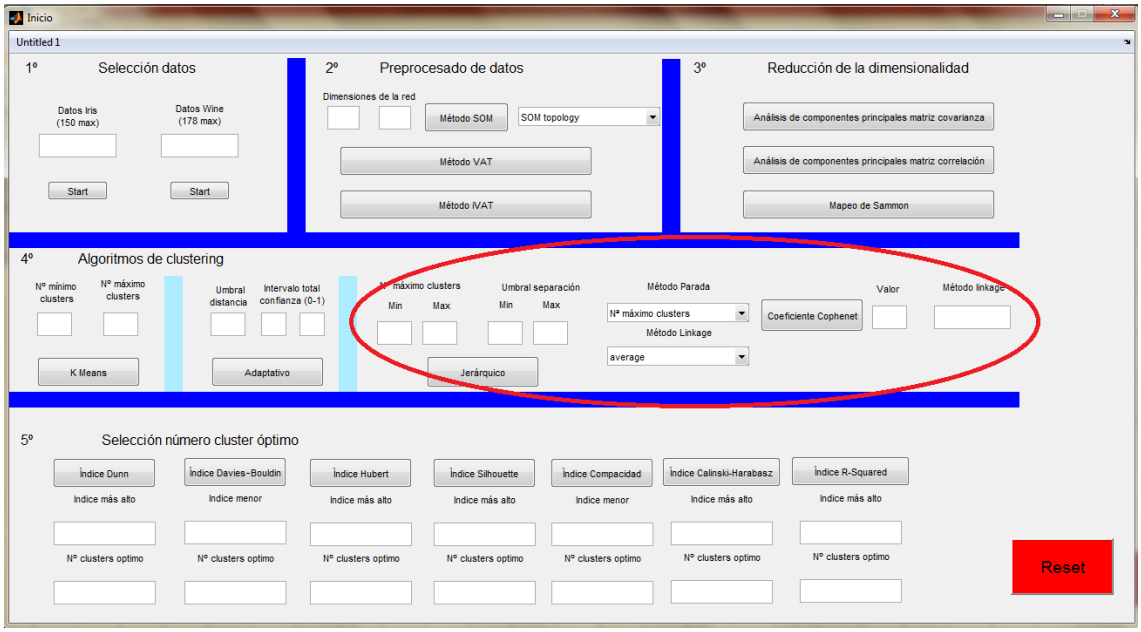

Figura 72.- Algoritmo jerárquico.

El algoritmo Jerárquico como ya se ha explicado con anterioridad consiste en la creación de un dendrograma, en el cual en cada paso se van uniendo los clusters más cercanos entre sí o separándose los más lejanos.

Para la correcta realización del algoritmo hay que introducir el número máximo de clúster o el umbral de separación, en función del método de parada que se seleccione. También hay que elegir el método de linkage (criterio de agrupación de los patrones en un cluster, distancia más cercana, distancia del centro del cluster, etc.). Una vez introducidos estos parámetros y pulsado el botón *"Jerárquico"*, se ejecuta la siguiente parte del programa:

```
% --- Executes on button press in al_jerarquico.
function al_jerarquico_Callback(hObject, eventdata, handles)
% hObject    handle to al_jerarquico (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
numero_min=str2double( get(handles.num_clusters_min, 'string') );
numero_max=str2double( get(handles.num_clusters_max, 'string') );
coef_min=str2double( get(handles.umbral_jerarquico_min, 'string') );
coef_max=str2double( get(handles.umbral_jerarquico_max, 'string') );
load('est');
if estado<2
    orden_incorrecto;
else
    load('dat');
    estado=3;
    save('est','estado')
    [f,c]=size(datos_tendencia);
    v=get(handles.sel_criterio,'Value');
    h=get(handles.sel_linkage,'Value');
    switch h
        case 1
            met_linkage='average';
        case 2
             met_linkage='centroid';
        case 3
            met_linkage='complete';
        case 4
             met_linkage='median';
        case 5
             met_linkage='single';
        case 6
            met_linkage='ward';
        case 7
            met_linkage='weighted';
    end

    if v==1
        numero=numero_min:numero_max;
        var=length(numero);
        Pos_patrones=zeros(var,f,c+1);
        j=1;
        visualizar=true;

[datos_clustering,coef_cophenet]=jerarquico_numero(datos_tendencia,n
umero(1),visualizar,met_linkage);
        visualizar=false;
        for i=numero

[datos_clustering,c]=jerarquico_numero(datos_tendencia,i,visualizar,
met_linkage);
            Pos_patrones(j,:,:)=datos_clustering;
            j=j+1;
         end
```

```
    elseif v==2
        numero=coef_min:0.05:coef_max;
        var=length(numero);
        j=1;
        visualizar=true;

[datos_clustering,c]=jerarquico_coeficiente(datos_tendencia,numero(1
),visualizar,met_linkage);
        visualizar=false;
         for i=numero

[datos_clustering,c]=jerarquico_coeficiente(datos_tendencia,i,visual
izar,met_linkage);
            Pos_patrones(j,:,:)=datos_clustering;
            j=j+1;
         end
    end
    Pos_patrones(1,:,:)=datos_clustering;
    datos_cluster=Pos_patrones;
    save('dat_cluster','datos_cluster')
end
```

Función 28.- Botón Jerárquico.

El funcionamiento del programa es el siguiente, en primer lugar se obtienen los valores de la condición de parada, ya sea el número máximo de cluster o el umbral de separación. Una vez realizado esto se comprueba que el programa se esté ejecutando en el orden correcto, se actualiza el estado del programa y se recoge la información seleccionada a partir de los desplegables *"Método Parada"* y *"Método Linkage"*. El siguiente paso consiste en guardar el método de linkage seleccionado en una variable y en función del método de parada seleccionado, se ejecuta una parte del programa u otra, siendo ambas partes iguales, diferenciándose únicamente en la existencia de dos funciones de realización del algoritmo jerárquico distintas en función del criterio de parada. En ambos casos se crea un vector con los valores de criterio de parada, y se produce una primera llamada a la función correspondiente donde se representa dendrograma, una vez que ya se han realizado todas las llamadas a la función se guardan los resultados obtenidos en una variable, para el proceso de validación de clustering.

La primera función que aparece es *jerarquico_numero*, que tiene como criterio de parada el número máximo de clusters a crear, el código correspondiente a dicha función es el siguiente:

```
%Función que realiza un algoritmo de clustering jerárquico,
representa un dendrograma
%El criterio de parada de la agrupación es por el número máximo de
clusters a crear
function
[datos_clustering,c]=jerarquico_numero(datos,numero,visualizar,met_l
inkage)
Y = pdist(datos);%función que retorna un vector con la información
sobre las distancias entre patrones
Z = linkage(Y,met_linkage);%función que toma información del vector
creado con anterioridad y los va linkando por pares para crear el
árbol.
if visualizar==true
    figure
    dendrogram(Z)%Representación del dendrograma
    title('Dendrograma')
end
%Verificación del algoritmo
%Para ello se obtiene el coeficiente de cophenet.
c = cophenet(Z,Y);
%Creación del cluster
T = cluster(Z,'maxclust',numero);
datos_clustering=[datos,T];
%%Representación gráfica
if visualizar==false
    figure
    hold on
    representacion(datos_clustering);
    title('Algoritmo Jerárquico')
end
```

Función 29.- jerarquico_numero.

La explicación del código es la siguiente, en primer lugar se ejecuta la función *pdist* a los patrones existentes, esta función de Matlab retorna un vector que contiene la distancia euclídea entre cada par de datos de una matriz de datos previa. A continuación junto con el vector resultante anteriormente y el método de linkage seleccionado con anterioridad, se procede al proceso de unión de los distintos pares para crear el árbol o dendrograma. A continuación se produce la visualización del dendrograma, se ha realizado dentro de un bucle *if*, para que sólo se muestre la primera vez que se llame a la función. A continuación se calcula el coeficiente de cophenet (correlación cofenética), y se realiza la creación de los clusters, a partir de la función de Matlab *cluster*, a la que se le pasa como variables, los datos del árbol creado anteriormente, el criterio de parada y el número de cluster a crear. Esta función devuelve un vector donde cada índice del vector corresponde al patrón correspondiente del mismo índice introducido previamente y almacena el número del cluster al que pertenece. Por último se unen en una misma matriz los patrones introducidos en la función y el vector con los clusters creados, y se procede a su representación gráfica.

La otra función que calcula el algoritmo jerárquico para el criterio de parada el umbral de separación es la función *jerarquico_coeficiente*, el código de la cual se muestra a continuación:

```
%Función que realiza un algoritmo de clustering jerárquico,
representa un dendrograma
%El criterio de parada de la agrupación es por el valor de
consistencia
function
[datos_clustering,c]=jerarquico_coeficiente(datos,coef,visualizar,me
t_linkage)
Y = pdist(datos);%funciona que retorna un vector con la información
sobre las distancias entre patrones
Z = linkage(Y,met_linkage);%función que toma información del vector
creado con anterioridad y los va linkando por pares ara crear el
árbol.
if visualizar==true
    figure
    dendrogram(Z)%Representación del dendrograma
    title('Dendrograma')
end
%Verificación del algoritmo
%Para ello se obtiene el coeficiente de cophenet.
c = cophenet(Z,Y);
%Creación del cluster
T = cluster(Z,'cutoff',coef);
datos_clustering=[datos,T];
%%Representación gráfica
if visualizar==false
    figure
    hold on
    representacion(datos_clustering);
    title('Algoritmo Jerárquico')
end
```

Función 30.- jerarquico_coeficiente.

La diferencia de esta función con respecto a la función *jerarquico_numero*, reside en la forma en la que se calculan los cluster. En este caso a la función *cluster*, se le introduce la instrucción *'cutoff'*, que indica que el modo de parada del algoritmo es el umbral de separación entre cluster y el valor umbral.

Para comprobar el correcto funcionamiento del algoritmo se han seleccionado 80 datos IRIS, y se ha producido la reducción de la dimensionalidad a partir del mapeo de Sammon, obteniéndose los siguientes patrones:

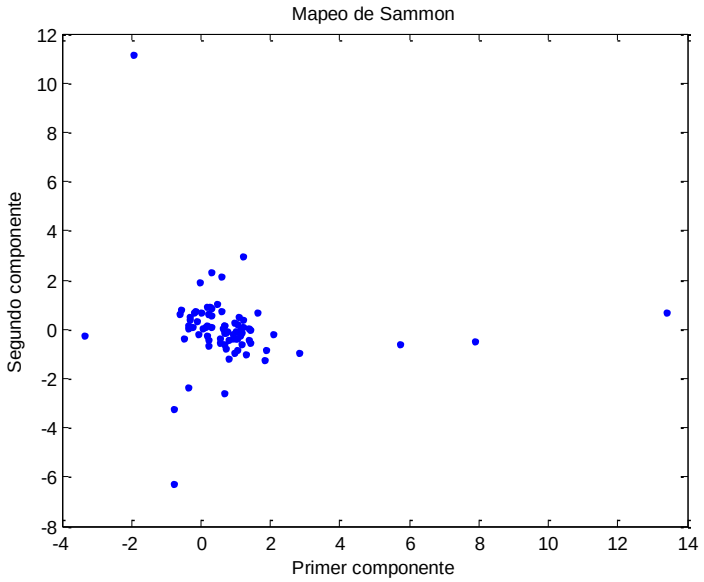

Figura 73.- Mapeo de Sammon, 80 datos IRIS.

Seleccionando la opción de realizar un algoritmo jerárquico con método de parada número máximo de cluster y método linkage "median", siendo el número máximo de cluster 10 y el mínimo 8, como se muestra en la siguiente imagen:

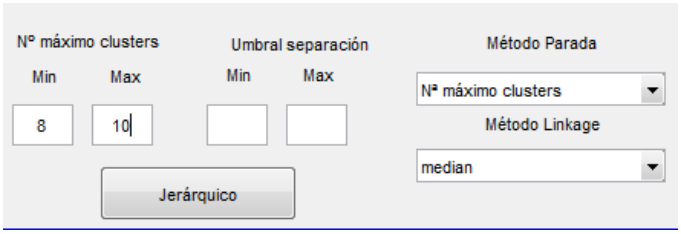

Figura 74.- Algoritmo Jerárquico.

Se obtienen las imágenes siguientes tras pulsar el botón *"Jerárquico"*, donde en primer lugar se muestra el dendrograma.

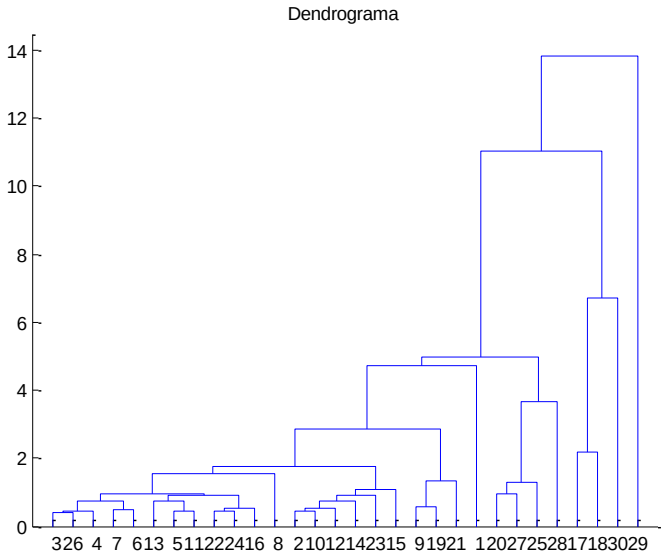

Figura 75.- Dendrograma algoritmo Jerárquico.

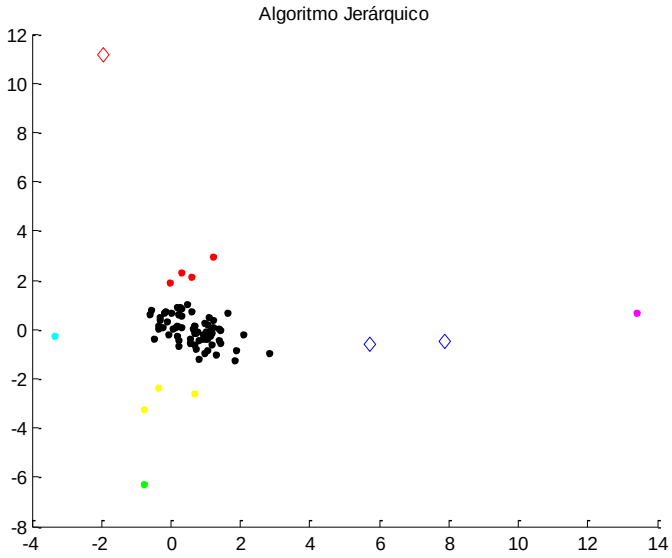

Figura 76.- Algoritmo Jerárquico 8 clusters.

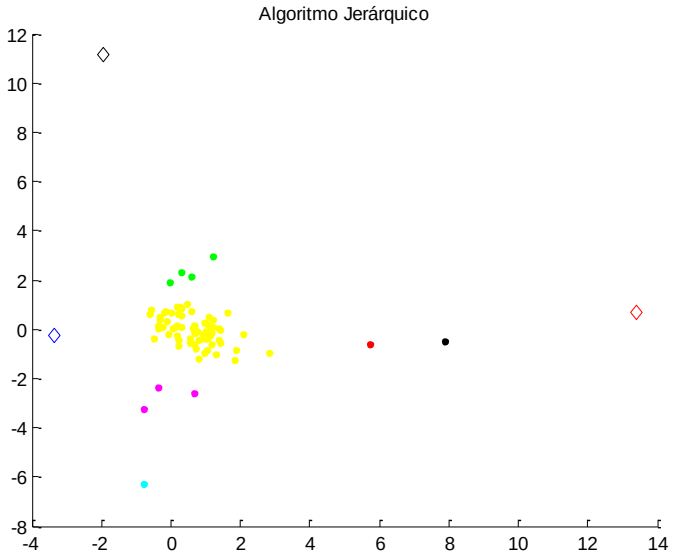

Figura 77.- Algoritmo Jerárquico 9 clusters.

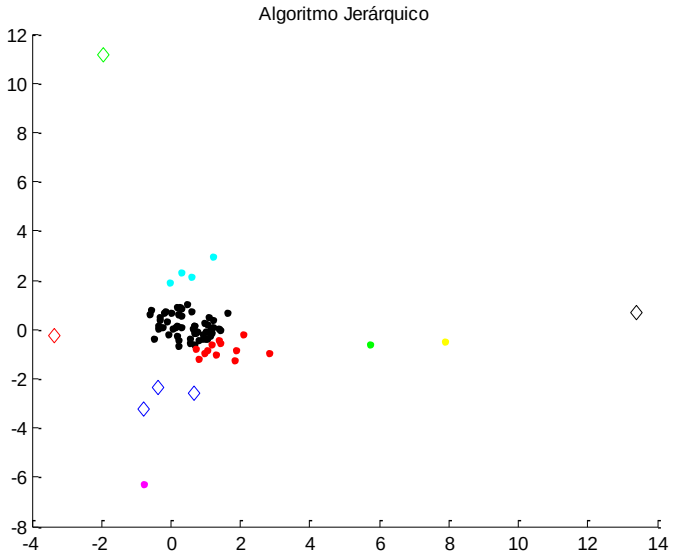

Figura 78.- Algoritmo Jerárquico 10 clusters.

Como se puede observar, según se van creando nuevos clusters, los clusters se van dividiendo progresivamente en clusters con menos número de patrones. Esto tiene sentido debido a que en el proceso de creación del árbol, se tiende a agrupar los patrones en una misma rama, con lo que progresivamente se va reduciendo el número de cluster, a uno sólo.

Si en lugar de seleccionar 80 datos de IRIS, se seleccionan 100 datos WINE, y se produce la reducción de la dimensionalidad a partir de un análisis de componentes principales a partir de la matriz de covarianza, se obtiene la siguiente imagen:

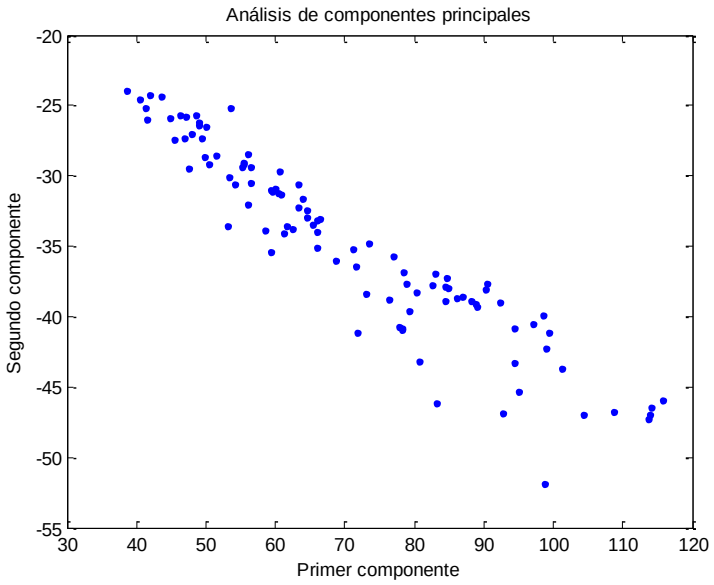

Figura 79.- Análisis de componentes principales, 100 datos WINE.

Seleccionando ahora para realizar el algoritmo jerárquico, como método de parada umbral de separación y método linkage "centroid", siendo el mínimo umbral de separación mínimo 1 y máximo 1.1, como se muestra en la imagen que se muestra a continuación:

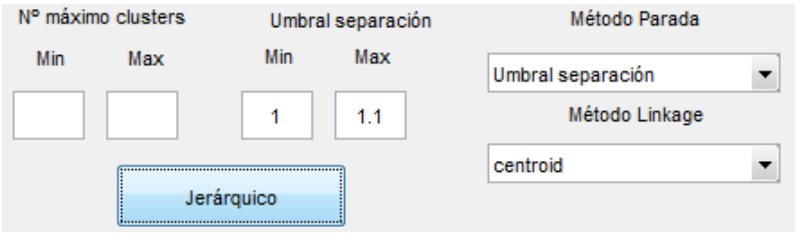

Figura 80.- Algoritmo Jerárquico.

Tras pulsar el botón *"Jerárquico"*, se obtienen las imágenes siguientes:

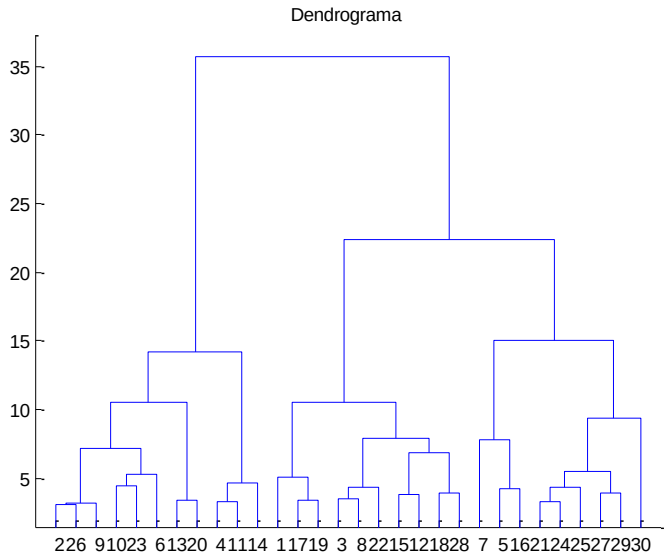

Figura 81.- Dendrograma algoritmo Jerárquico.

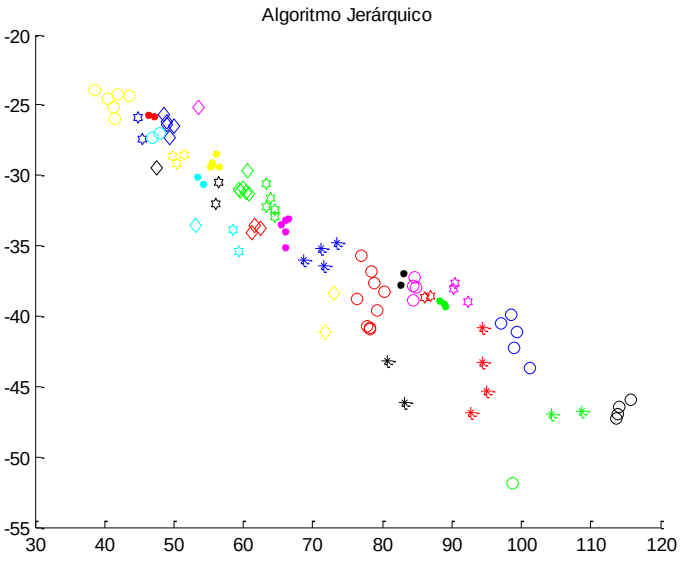

Figura 82.- Algoritmo Jerárquico umbral separación 1.

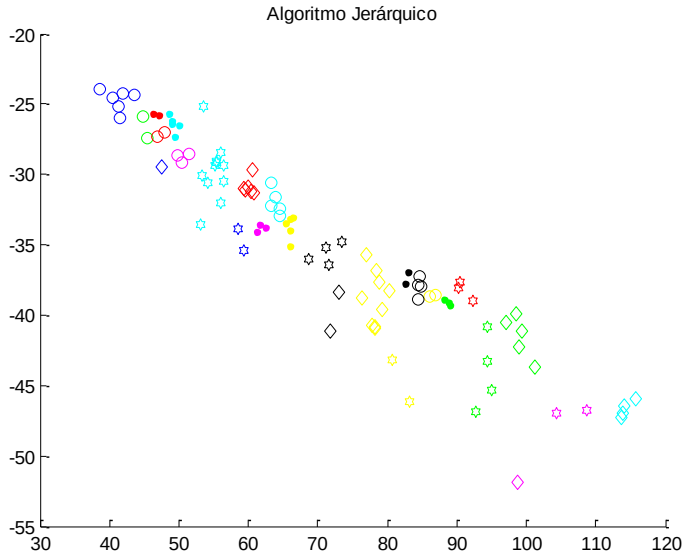

Figura 83.- Algoritmo Jerárquico umbral separación 1,05.

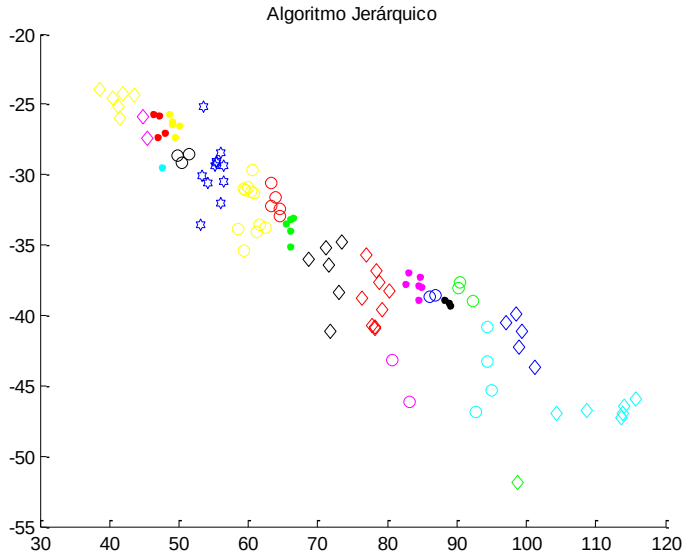

Figura 84.- Algoritmo Jerárquico umbral separación 1,1.

Dentro del algoritmo Jerárquico, es muy importante el valor del coeficiente de cophenet, para lo cual se ha introducido un botón mediante el cual calcula el valor del coeficiente de cophenet a partir de un conjunto de patrones, a los que aún no se les ha aplicado ningún algoritmo de clustering e indica cual es el método de linkage para el cual cuando se aplique el algoritmo Jerárquico se obtendrá el mejor resultado por esta parte del programa se muestra en la siguiente figura:

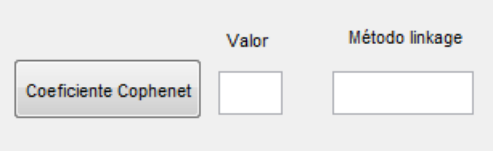

Figura 85.- Coeficiente cophenet.

Cuando se pulsa el botón *"Coeficiente Cophenet"*, el código del programa *Inicio* que se ejecuta es el siguiente:

```
% --- Executes on button press in sel_jerarquico_cophenet.
function sel_jerarquico_cophenet_Callback(hObject, eventdata,
handles)
% hObject    handle to sel_jerarquico_cophenet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('dat');
[c,metodo]=valor_cophenet(datos_tendencia);
set( handles.mejor_cophenet, 'string',num2str(c) );
    switch metodo
        case 1
            met='average';
        case 2
            met='centroid';
        case 3
            met='complete';
        case 4
            met='median';
        case 5
            met='single';
        case 6
            met='ward';
        case 7
            met='weighted';
    end
set( handles.met_linkage, 'string',met );
```

Función 31.- Botón Coeficiente Cophenet.

El funcionamiento de este programa es el siguiente, en primer lugar se cargan los datos que se han almacenado después de realizar una reducción de la dimensionalidad y se produce una llamada a la función *valor_cophenet*, esta función calcula el valor del coeficiente cofenético, para todos los tipos de linkages y devuelve el mayor valor del coeficiente de todos los calculados y el método de linkage correspondiente, posteriormente se procede a mostrar por pantalla ambos valores. El método será mejor cuanto más próximo esté el valor del coeficiente a uno.

El código de la función *valor_cophenet*, se muestra a continuación:

```
%Función que te calcula el mejor valor de cophenet y el método de
linkage para el cual se obtiene dicho valor.
%El valor de dicho coeficiente cuando más próximo esté a uno mayor
calidad tendrá.
function [c,metodo]=valor_cophenet(datos)
Y = pdist(datos);
N=7;%Número de métodos disponibles
maximo=0;
for i=1:N
    switch i
        case 1
             Z = linkage(Y,'average');
        case 2
             Z = linkage(Y,'centroid');
        case 3
             Z = linkage(Y,'complete');
        case 4
             Z = linkage(Y,'median');
        case 5
             Z = linkage(Y,'single');
        case 6
            Z = linkage(Y,'ward');
        case 7
            Z = linkage(Y,'weighted');
    end
    c = cophenet(Z,Y);
    if c>maximo
        maximo=c;
        metodo=i;
    end
end
```

Función 32.- valor_cophenet.

En esta función progresivamente se evalúan los distintos métodos de linkage, y se calcula con cuál de ellos se obtiene un valor más próximo a 1. Seleccionando los 150 datos de IRIS, y realizando una reducción de dimensionalidad con un análisis de componentes principales a partir de la matriz de correlación se obtiene la siguiente figura:

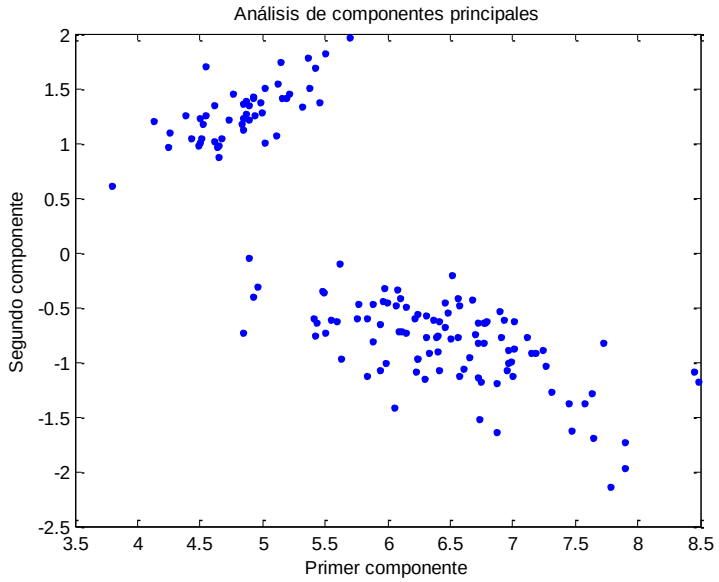

Figura 86.- Análisis de componentes principales, 150 datos IRIS.

Y posteriormente si se pulsa el botón de *"Coeficiente Cophenet"*, se obtiene el siguiente resultado:

Figura 87.- Coeficiente cophenet, 150 datos IRIS.

Donde se muestra el valor del coeficiente de Cophenet para el conjunto de datos con el que se está trabajando y el método de linkage con el que se obtiene el mejor resultado, que en este caso es "average".

Si en lugar de seleccionar los 150 datos de IRIS, se seleccionan los 178 datos de WINE, se obtiene el siguiente resultado:

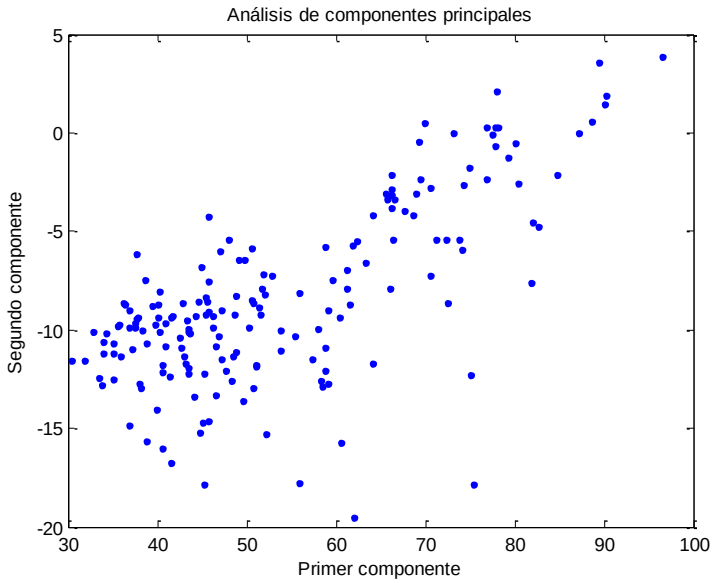

Figura 88.- Análisis de componentes principales, 178 datos WINE.

Figura 89.- Coeficiente cophenet, 178 datos WINE.

En la figura anterior se observa que el valor del coeficiente de cophenet es menor que en el caso anterior y que el método de linkage más adecuado para este conjunto de patrones es “complete”.

6.5. Selección número cluster óptimo

En esta parte del programa se selecciona el número de cluster óptimo, a partir de varios algoritmos de validación. Se han añadido siete índices de validación a los cuales se les pasa las diferentes agrupaciones de clusters creadas y se evalúan cada una, eligiendo en cada caso la mejor de ellas, ya sea porque el valor del índice sea el menor o el mayor correspondiente. Esta parte del programa se muestra en la siguiente figura:

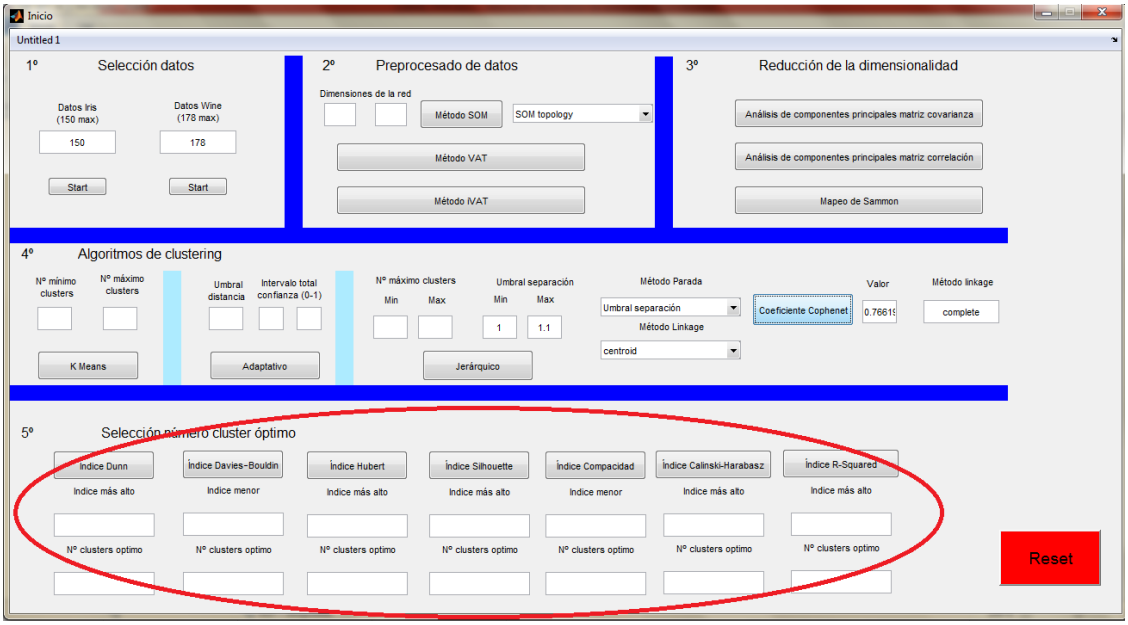

Figura 90.- Selección número cluster óptimo.

6.5.1. Índice de Dunn

El primer índice que se emplea es el índice de Dunn, este índice de validación se ejecuta cuando se pulsa el botón *"Índice Dunn"*, el código de programa que se ejecuta cuando se pulsa el botón es el siguiente:

```
% --- Executes on button press in Ind_dunn.
function Ind_dunn_Callback(hObject, eventdata, handles)
% hObject    handle to Ind_dunn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
maximo,num_clusters]=comprobar_indice_Dunn(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_dunn, 'string',num2str(maximo) );
    set( handles.numero_clusters, 'string',num2str(num_clusters) );
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 33.- Botón Índice Dunn.

Cuando se pulsa el botón, lo primero que se realiza es una comprobación de que el programa se está ejecutando correctamente, es decir, que ya se ha realizado el análisis de cluster correspondiente, una vez realizado esto se cargan los datos del análisis de cluster anterior y produce la llamada a la función *comprobar_indice_Dunn*, se guardan los resultados obtenidos y se visualiza por pantalla el número de clusters óptimo y valor del índice con la ayuda de las funciones de Matlab *set* y *num2str*. Por último se representa el cluster mejor. El código de la función *comprobar_indice_Dunn* es el siguiente:

```
function [Cluster_mejor,
maximo,numero_clusters]=comprobar_indice_Dunn(datos_cluster)
[f,c,w]=size(datos_cluster);
maximo=0;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_Dunn(Clusters);
    if indices(i)>maximo
       maximo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice Dunn')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 34.- comprobar_indice_Dunn.

El código de esta función es el siguiente, en él se producen sucesivas llamadas a la función *Indice_Dunn*, la cual devuelve el número de cluster que componen cada agrupación y el valor del índice, y se comparan los valores obtenidos hasta obtener el máximo. Por último se realiza una representación de la evolución de los valores del índice en función del número de clustering. El código de la función *Indice_Dunn* es el siguiente:

```
%Función que calcula el valor del índice de Dunn
%Un valor más alto del índice indica un mejor resultado
function [Value,num_clusters]=Indice_Dunn(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%Se averigua el número de clusters
actual
maximo_den=0;%Búsqueda del máximo denominador
%%Máxima distancia entre los elementos de un cluster
h=1;
for i=1:num_clusters
    distancia_maxima=0;
    for j=1:f
       if Clusters(j,3)==i;
           cluster_actual(h,1)=Clusters(j,1);
           cluster_actual(h,2)=Clusters(j,2);
           h=h+1;
       end
    end
    %%Calcular la máxima distancia entre los elementos de un cluster
    [fc,cc]=size(cluster_actual);
    for k=1:fc
        for l=1:fc
           if k~=l
                distancia_cluster=((cluster_actual(l,1)-
cluster_actual(k,1))^2+(cluster_actual(l,2)-cluster_actual(k,2))^2);
                if distancia_cluster>distancia_maxima
                    distancia_maxima=distancia_cluster;
                end
            end
        end
    end
    if distancia_maxima>maximo_den
        maximo_den=distancia_maxima;
    end

end
%Calcular los distintos centros
Centros=zeros(num_clusters,2);
for i=1:num_clusters
    acumuladorx=0;
    acumuladory=0;
    h=1;
```

```
    for j=1:f
       if Clusters(j,3)==i;
           acumuladorx=Clusters(j,1);
           acumuladory=Clusters(j,2);
           h=h+1;
       end
    end
    clusterx=acumuladorx/h;
    clustery=acumuladory/h;
    Centros(i,1)=clusterx;
    Centros(i,2)=clustery;
end
%Crear una matriz de distancias con los centros
matriz_distancia=zeros(num_clusters,num_clusters);
for i=1:num_clusters
   for j=1:num_clusters
       if i~=j
           matriz_distancia(i,j)=abs(sqrt((Centros(i,1)-
Centros(j,1))^2+(Centros(i,2)-Centros(j,2))^2));
       else
           matriz_distancia(i,j)=inf;
       end
   end
end
Value=(min(min(matriz_distancia)))/maximo_den;
```

Función 35.- Indice_Dunn.

En esta función, en primer lugar se calcula el número de patrones y de cluster que tienen el conjunto de datos, posteriormente se calcula la máxima distancia entre los elementos de un cluster y los centros respectivos de cada cluster, posteriormente se crea una matriz de distancia de los centros y se procede a calcular el valor del índice con los valores previamente obtenidos.

Seleccionando los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice Dunn"*, se obtiene el siguiente resultado.

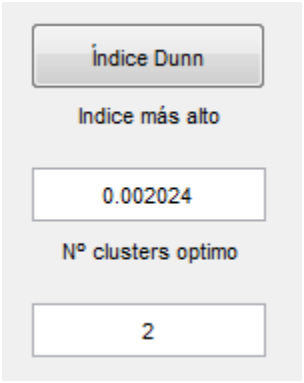

Figura 91.- Índice Dunn.

La gráfica en la que se muestra la evolución del índice en función del número de cluster es:

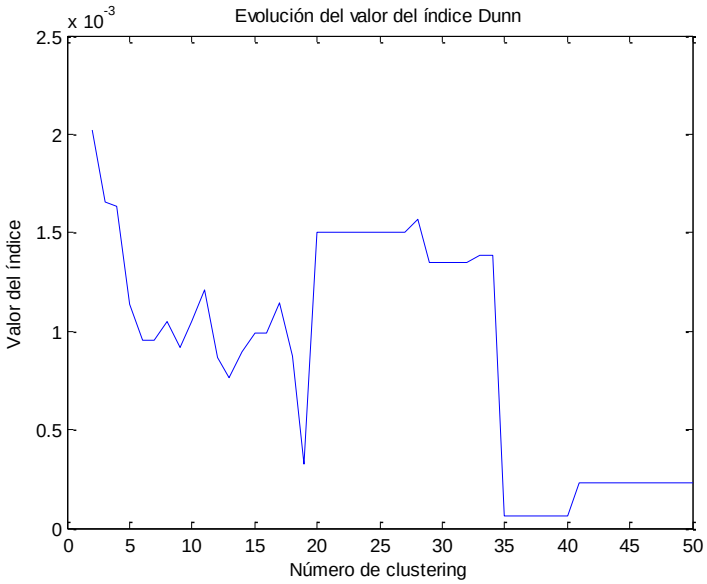

Figura 92.- Evolución del valor del Índice de Dunn.

Y la mejor agrupación es:

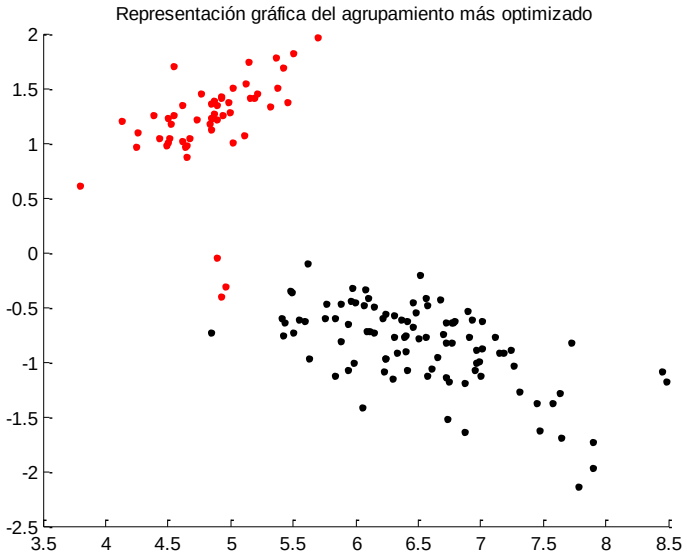

Figura 93.- Mejor agrupación índice de Dunn

6.5.2. Índice de Davies-Bouldin

Otro índice de validación es el de Davies-Bouldin, dicho índice al contrario del de Dunn, un menor valor del índice índica una mejor agrupación. Este índice se ejecuta cuando se pulsa el botón del programa *Inicio "Índice Davis-Bouldin"* y el código correspondiente al botón es el siguiente:

```
% --- Executes on button press in ind_DB.
function ind_DB_Callback(hObject, eventdata, handles)
% hObject    handle to ind_DB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
minimo,num_clusters]=comprobar_indice_DB(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_DB, 'string',num2str(minimo) );
    set( handles.numero_clusters_DB, 'string',num2str(num_clusters)
);
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 36.- Botón Índice Davvis-Bouldin.

En este trozo de código, se produce la llamada a la función *comprobar_indice_DB*, que es en la que se calcula el valor del índice para las diferentes agrupaciones, y luego muestra por pantalla la mejor agrupación según el índice dado. El código correspondiente a dicha función es el siguiente:

```
function [Cluster_mejor,
minimo,numero_clusters]=comprobar_indice_DB(datos_cluster)
[f,c,w]=size(datos_cluster);
minimo=inf;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_Davies_Bouldin(Clusters);
    if indices(i)<minimo
       minimo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice Davies-Bouldin')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 37.- comprobar_indice_DB.

El código correspondiente a esta función es el siguiente, en él se producen sucesivas llamadas a la función *Indice_Davies_Bouldin*, la cual devuelve el número de cluster que componen cada agrupación y el valor del índice, y se comparan los valores obtenidos hasta obtener el mínimo. Por último se realiza una representación de la evolución de los valores del índice en función del número de clustering. El código de la función *Indice_Davies_Bouldin* es el siguiente:

```
%Función que calcula el valor del índice de Davies-Bouldin
%Un valor menor indica una mayor calidad del índice
function [Value,num_clusters]=Indice_Davies_Bouldin(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%Se averigua el número de clusters
actual
%Se calculan los centros de los clusters
Centros=zeros(num_clusters,2);
%Calcular los distintos centros
for i=1:num_clusters
    acumuladorx=0;
    acumuladory=0;
    h=0;
    for j=1:f
       if Clusters(j,3)==i;
           acumuladorx=Clusters(j,1);
           acumuladory=Clusters(j,2);
           h=h+1;
       end
    end
    clusterx=acumuladorx/h;
    clustery=acumuladory/h;
    Centros(i,1)=clusterx;
    Centros(i,2)=clustery;
end
%Número de patrones que pertenecen a un cluster
T=zeros(num_clusters);
for i=1:num_clusters
    acumulador=1;
    for j=1:f
        if Clusters(j,3)==i
            acumulador=1+acumulador;
        end
    end
    T(i)=acumulador;
end
%Medida de la dispersion de los patrones dentro del cluster
S=zeros(num_clusters);
for j=1:num_clusters
    distancia=0;
    for i=1:f
        if Clusters(f,3)==j
            distancia=((Clusters(f,1)-
Centros(j,1))^2+(Clusters(f,2)-Centros(j,2))^2)+distancia;
        end
    end
   S(j)=abs(sqrt(distancia/T(j)));
end
```

```
%Medida de lo bueno que es un clustering R
matriz_R=zeros(num_clusters,num_clusters);
for i=1:num_clusters
   for j=1:num_clusters
       if i~=j
           M=abs(sqrt((Centros(i,1)-Centros(j,1))^2+(Centros(i,2)-
Centros(j,2))^2));%Separación entre los centros
           matriz_R(i,j)=(S(i)+S(j))/M;
       else
           matriz_R(i,j)=0;
       end
   end
end
D=zeros(num_clusters);
for i=1:num_clusters
    D(i)=max(matriz_R(i,1));
end
DB=0;
for i=1:num_clusters
    DB=D(i)+DB;
end
Value=DB/num_clusters;
```

Función 38.- Indice_Davies_Bouldin.

La explicación de cómo se haya este índice es la siguiente, en primer lugar se averigua el número de clusters y se calculan los centros de los clusters, luego se calcula el número de patrones que hay dentro de cada cluster y se mide la dispersión de los patrones dentro del cluster, a continuación se mide la separación entre los centros y se procede a calcular el valor del índice. Este índice mide la separación entre los clusters y la fortaleza dentro de ellos, por lo que un menor valor del índice nos proporcionaría un mejor resultado.

Para evaluar el correcto funcionamiento del índice, se han seleccionado los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice Davies-Bouldin"*, se obtiene el siguiente resultado.

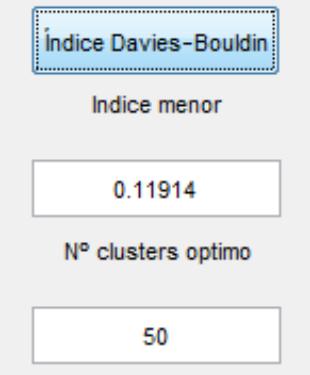

Figura 94.- Índice Davies-Bouldin.

El programa también nos muestra las figuras siguientes:

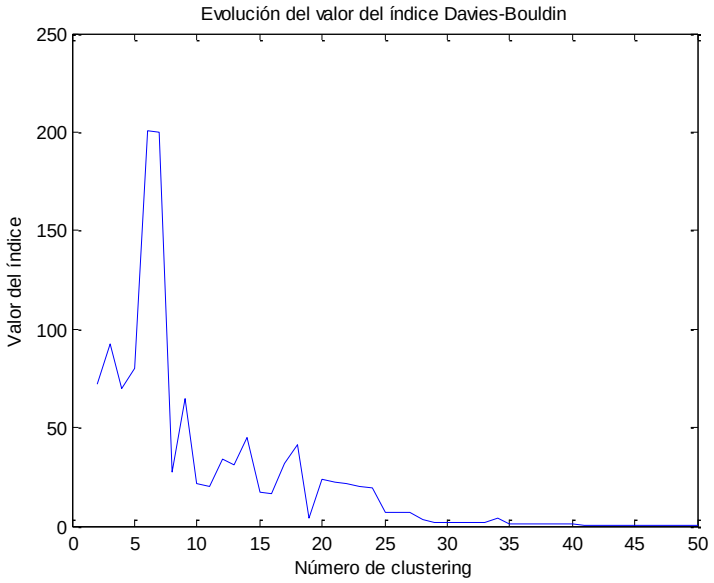

Figura 95.- Evolución del valor del Índice de Davies-Bouldin.

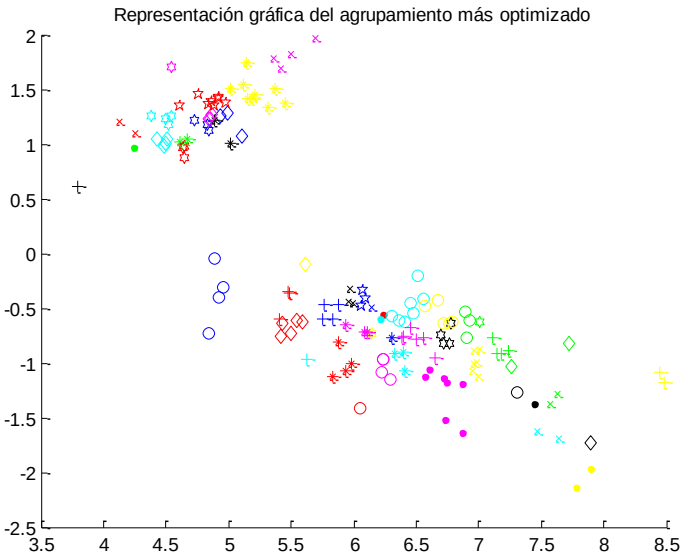

Figura 96.- Mejor agrupación índice de Davies-Bouldin.

Como se puede observar según se incrementa el número de clusters, aumenta la robustez interna de los clusters, siendo en este caso ese parámetro más importante que la distancia entre los mismos, ya que los clusters se encuentran bastante próximos entre sí.

6.5.3. Índice de Hubert

El índice de Hubert, es otro índice de validación que se ha empleado, para un valor mayor del índice se supone una mejor agrupación de los clusters. Para proceder a calcular dicho índice se pulsa el botón *"Índice Hubert"*, ejecutándose el código que se muestra a continuación:

```
% --- Executes on button press in Ind_hub.
function Ind_hub_Callback(hObject, eventdata, handles)
% hObject    handle to Ind_hub (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
maximo,num_clusters]=comprobar_indice_hubert(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_hub, 'string',num2str(maximo) );
    set( handles.numero_clusters_hub, 'string',num2str(num_clusters)
);
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 39.- Botón Índice Hubert.

La explicación de esta parte del programa es la siguiente, por primer lugar se comprueba el estado del programa, es decir, que antes se hayan aplicado a los datos técnicas de agrupamiento de clustering, en cuyo caso el programa mostraría un mensaje de error, en caso contrario el programa carga los datos previamente obtenidos y llama a la función *comprobar_indice_hubert*, esta función devuelve los valores del mejor agrupamiento realizado, el número de cluster que tiene dicho agrupamiento y el valor del índice para dicho caso. Por último se muestra la figura del mejor agrupamiento realizado. El código de la función *comprobar_indice_hubert*, es el siguiente:

```
function [Cluster_mejor,
maximo,numero_clusters]=comprobar_indice_hubert(datos_cluster)
[f,c,w]=size(datos_cluster);
maximo=0;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_hubert(Clusters);
    if indices(i)>maximo
       maximo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice Hubert')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 40.- comprobar_indice_hubert.

La explicación de esta parte del programa es la siguiente siguiente, en él se producen sucesivas llamadas a la función *Indice_hubert*, tantas como agrupamientos se hayan creado y la cual devuelve el número de cluster que componen cada agrupación y el valor del índice, y se comparan los valores obtenidos hasta obtener el máximo de todos los clusters creados. Por último se realiza una representación de la evolución de los valores del índice en función del número de clustering. El código de la función *Indice_hubert* es el siguiente:

```
%Función que calcula el valor estadístico de Hubert Modificada
%Un valor más alto del índice indica un mejor resultado
function [Value,num_clusters]=Indice_hubert(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%Se averigua el número de clusters
actual
n=f;%Número de puntos objetos a agrupar
M=n*(n-1)/2;
%Creación de la matriz de similitud
P=zeros(f,f);
for i=1:f
   for j=1:f
            P(i,j)=abs(sqrt((Clusters(i,1)-
Clusters(j,1))^2+(Clusters(i,2)-Clusters(j,2))^2));
   end
end
%Creación de la matriz Q (Matriz de los centros)
Centros=zeros(num_clusters,2);
for i=1:num_clusters%Calculo de los centros
    acumuladorx=0;
    acumuladory=0;
    h=1;
    for j=1:f
       if Clusters(j,3)==i;
           acumuladorx=Clusters(j,1);
           acumuladory=Clusters(j,2);
           h=h+1;
       end
    end
    clusterx=acumuladorx/h;
    clustery=acumuladory/h;
    Centros(i,1)=clusterx;
    Centros(i,2)=clustery;
end
Q=zeros(f,f);
for i=1:f
   for j=1:f
       centro_patron1=Clusters(i,3);
       centro_patron2=Clusters(j,3);
       centro1=Centros(centro_patron1,:);
       centro2=Centros(centro_patron2,:);
       Q(i,j)=abs(sqrt((centro1(1,1)-centro2(1,1))^2+(centro1(1,2)-
centro2(1,2))^2));
   end
end
sumatorio=0;
for i=1:f-1
   for j=i+1:f
       sumatorio=P(i,j)*Q(i,j)+sumatorio;
   end
end
Value=1/M*sumatorio;
```

Función 41.- Indice_hubert.

El proceso para calcular este índice es el siguiente, en primer lugar se calcula el número de patrones y de clusters que hay, luego se crea la matriz de similitud *P* y la matriz de los centros *Q*, y a partir de las dos matrices se procede al cálculo del índice. Decir que un mayor valor de este índice indica una mejor agrupación.

Al igual que con los índices anteriores, para evaluar el correcto funcionamiento del índice, se han seleccionado los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice Hubert"*, se obtiene el siguiente resultado.

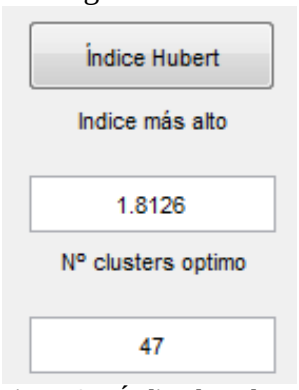

Figura 97.- Índice de Hubert.

A su vez el programa nos muestra las figuras siguientes en la que se ve la evolución del índice en función del número de cluster y el mejor agrupamiento posible según el índice:

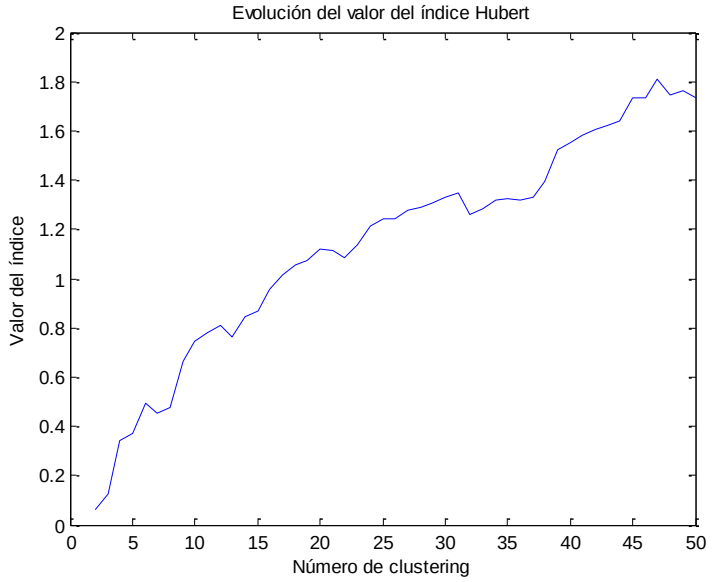

Figura 98.- Evolución del valor del índice de Hubert.

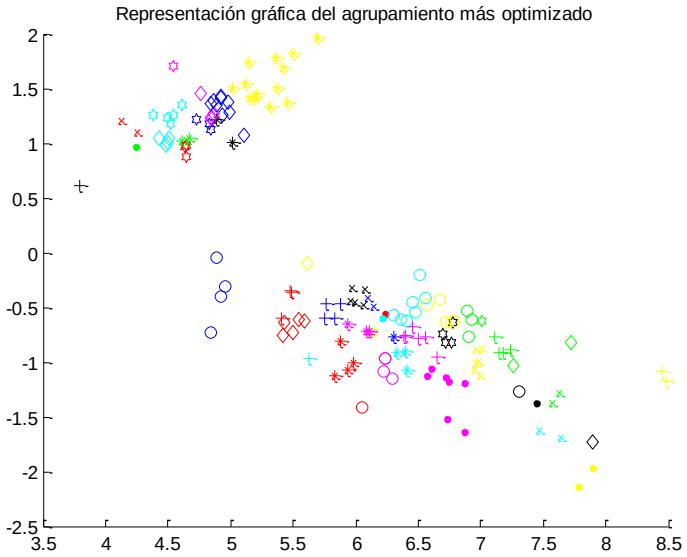

Figura 99.- Mejor agrupación índice de Hubert.

Como se puede observar en la gráfica de la evolución del índice, para un mayor número de cluster el índice suele tener un valor mayor, con pequeños cambios que pueden ser debidos a los distintos centros creados.

6.5.4. Índice de Silhouette

El índice de validación de Silhouette se calcula dentro del programa *Inicio*. El valor de este índice se mueve entre -1 y 1, si el valor se encuentra próximo a -1, quiere decir que sea ha realizado un mal agrupamiento y si se encuentra próximo a 1 es que el agrupamiento realizado es bastante bueno, por el contrario si el valor es próximo a 0, el resultado del índice es indiferente. Cuando se pulsa el botón de *"Índice Silhouette"*, ejecutándose el siguiente código del programa:

```
% --- Executes on button press in Ind_sil.
function Ind_sil_Callback(hObject, eventdata, handles)
% hObject    handle to Ind_sil (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
maximo,num_clusters]=comprobar_indice_silhouette(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_sil, 'string',num2str(maximo) );
    set( handles.numero_clusters_sil, 'string',num2str(num_clusters)
);
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 42.- Botón Índice Silhouette.

El código que se ejecuta al pulsar el botón es el siguiente, como primera acción se comprueba el estado del programa, y si todo es correcto se cargan los datos obtenidos del proceso de análisis de clustering y se llama a la función *comprobar_indice_silhouette*, esta función devuelve los valores del mejor agrupamiento realizado, el número de cluster que tiene dicho agrupamiento y el valor del índice para dicho caso. Por último se muestra la figura del mejor agrupamiento realizado. El código correspondiente a esta función, es el siguiente:

```
function [Cluster_mejor,
maximo,numero_clusters]=comprobar_indice_silhouette(datos_cluster)
[f,c,w]=size(datos_cluster);
maximo=0;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_Silhouette(Clusters);
    if indices(i)>maximo
       maximo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice Silhouette')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 43.- comprobar_indice_silhouette.

La explicación del código que aparece anteriormente es el siguiente, en él se producen sucesivas llamadas a la función *Indice_Silhouette*, tantas como agrupamientos se hayan creado y la cual devuelve el número de cluster que componen cada agrupación y el valor del índice, y se comparan los valores del índice obtenidos hasta obtener el máximo de todos los clusters creados. Por último se realiza una representación de la evolución de los valores del índice en función del número de clustering. El código de dicha función se muestra a continuación:

```
%Un valor del índice próximo a uno indica un buen resultado
%El valor del índice oscila entre -1 y 1
%-1 malo
%0 indiferente
%1 bueno
function [Value,num_clusters]=Indice_Silhouette(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%SE averigua el número de clusters
actual
%a(i) (disimilitud promedio) del punto objeto i con todos los demás
puntos objeto en el mismo cluster
%Se mide la distancia del patrón en estudio a los demás patrones del
mismo cluster y se divide por el número de patrones dentro del mismo
cluster
a=zeros(f);
for i=1:f
    cluster_actual=Clusters(i,3);
    N=0;
    acumulador=0;
```

```
    for j=1:f
        if Clusters(j,3)==cluster_actual
            acumulador=abs(sqrt((Clusters(i,1)-
Clusters(j,1))^2+(Clusters(i,2)-Clusters(j,2))^2))+acumulador;
            N=N+1;
        end
     a(i)= acumulador/N;
    end
end
%b(i) (mínimo del promedio de disimilitud) del objeto i hacia todos
los objetos en otro cluster (el más cercano).
b=zeros(f);
for i=1:f
    cluster_actual=Clusters(i,3);
    minimo=inf;
    for h=1:num_clusters
        N=1;
        acumulador=0;
        for j=1:f
            if Clusters(j,3)~=cluster_actual && Clusters(j,3)==h
                acumulador=abs(sqrt((Clusters(i,1)-
Clusters(j,1))^2+(Clusters(i,2)-Clusters(j,2))^2))+acumulador;
                N=N+1;
            end
         temporal= acumulador/N;
        end
        if temporal<minimo
            minimo=temporal;
        end
    end
    b(i)=temporal;
end
s=zeros(f);
for i=1:f
    maximo=max(a(i),b(i));
    s(i)=(b(i)-a(i))/maximo;
end
SC=0;
for i=1:f
    SC=s(i)+SC;
end
SC=SC/f;
Value=SC;
```

Función 44.- Indice_Silhouette.

El proceso para calcular este índice es el siguiente, en primer lugar se calcula el número de patrones y de clusters que hay, luego se calcula la disimilitud promedio *(a(i))* de cada punto con todos los demás puntos del cluster y se divide por el número de patrones dentro del mismo cluster. A continuación se calcula el mínimo del promedio de disimilitud *(b(i))*, que es la distancia de un objeto hacia todos los patrones que se encuentran en otro cluster. Una vez calcula estos valores se haya el índice de Silhouette por cada patrón y se hace la media de todos.

Como se ha hecho con los índices anteriores, para evaluar el correcto funcionamiento del índice, se han seleccionado los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice Silhouette"*, obteniéndose el siguiente resultado:

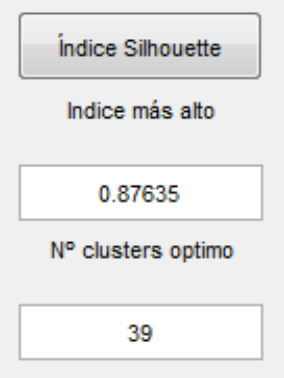

Figura 100.- Índice de Silhouette.

El programa también nos muestra las siguientes figuras:

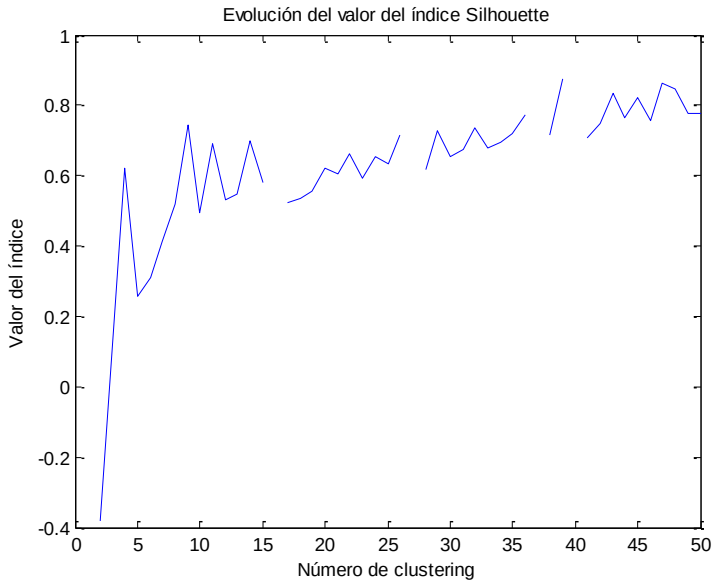

Figura 101.- Evolución del valor del índice de Silhouette.

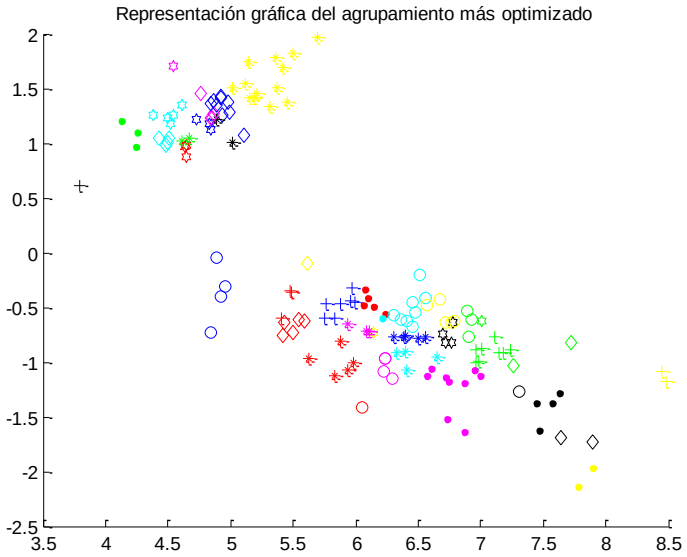

Figura 102.- Mejor agrupamiento índice de Silhouette.

Como se puede observar en la gráfica que muestra la evolución del índice, el valor del mismo oscila entre -1 y 1, siendo mejor para un mayor número de clusters, ya que el resultado del índice se aproxima más a 1.

6.5.5. Índice de Compacidad

Este índice nos proporciona una idea de los bueno o malo que es un agrupamiento, para un menor valor del índice el agrupamiento realizado será más compacto. Este índice incorpora la deficiencia de que si el agrupamiento presenta un cluster con un único patrón, el índice no se encuentra definido para dicho valor por su definición. El código del programa *Inicio* que se ejecuta cuando se pulsa el botón *"Índice Compacidad"*, es el siguiente:

```
% --- Executes on button press in Ind_CP.
function Ind_CP_Callback(hObject, eventdata, handles)
% hObject    handle to Ind_CP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
minimo,num_clusters]=comprobar_indice_CP(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_CP, 'string',num2str(minimo) );
    set( handles.numero_clusters_CP, 'string',num2str(num_clusters)
);
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 45.- Botón Índice Compacidad.

El funcionamiento de este botón del programa es el siguiente, en primer lugar se comprueba que el programa se esté ejecutando en el orden correcto y se cargan los datos almacenados de un análisis de cluster anterior. A continuación se procede a realizar la llamada a la función *comprobar_indice_CP* y se representa el mejor agrupamiento creado cuya información ha sido devuelta por la función anterior. El código correspondiente a dicha función es el siguiente:

```
function [Cluster_mejor,
minimo,numero_clusters]=comprobar_indice_CP(datos_cluster)
[f,c,w]=size(datos_cluster);
minimo=inf;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_Compacidad(Clusters);
    if indices(i)<minimo
       minimo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice Compacidad')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 46.- comprobar_indice_CP.

La explicación de cómo funciona esta función es similar a las demás funciones que calculan los valores de los demás índices, con la salvedad que en esta se produce la llamada a la función *Indice_Compacidad* y se busca aquella agrupación que proporcione un valor del índice menor. El código correspondiente a dicha función es el siguiente:

```
%Función que calcula el valor estadístico del Índice de Compacidad
%Un valor del índice pequeño dará un mejor resultado
%%Si un cluster tiene un solo patrón, el índice no esá definido
function [Value,num_clusters]=Indice_Compacidad(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%Se averigua el número de clusters
actual
N=f;%Número de patrones
K=num_clusters;%Número de clusters que se han creado
%Cálculo del número de patrones que pertenece a cada cluster
for j=1:K
   acumulador=0;
   for i=1:f
        if Clusters(i,3)==j
            acumulador=acumulador+1;
        end
   end
   n(j)=acumulador;
end
%Variable donde se va a guardar la distancia entre todos los
patrones que pertenecen a cada cluster
for num=1:num_clusters
    acumulador=0;
    cluster_actual=num;
    for i=1:f
        if Clusters(i,3)==cluster_actual
            for j=i+1:f
                if Clusters(j,3)==cluster_actual
                    acumulador=abs(sqrt((Clusters(i,1)-
Clusters(j,1))^2+(Clusters(i,2)-Clusters(j,2))^2))+acumulador;
                end
            end
        end
    end
distancia(num)=acumulador;
end
temporal=0;
for i=1:num_clusters
   temporal=n(i)*distancia(i)/(n(i)*((n(i)-1)/2))+temporal;
end
CP=1/N*temporal;
Value=CP;
```

Función 47.- Indice_Compacidad.

El proceso que se sigue para calcular dicho índice es el siguiente, primero se averigua el número de patrones y de clusters que hay, y se calcula la distancia entre todos los

patrones que pertenecen a un mismo cluster y una vez obtenido ese valor se procede a la obtención del índice.

Como se ha hecho con los índices anteriores, para evaluar el correcto funcionamiento del índice, se han seleccionado los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice Compacidad"*, obteniéndose el siguiente resultado:

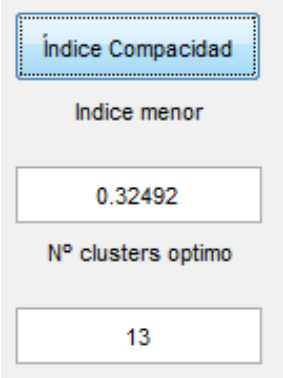

Figura 103.- Índice de compacidad.

El programa también nos devuelve las dos siguientes figuras:

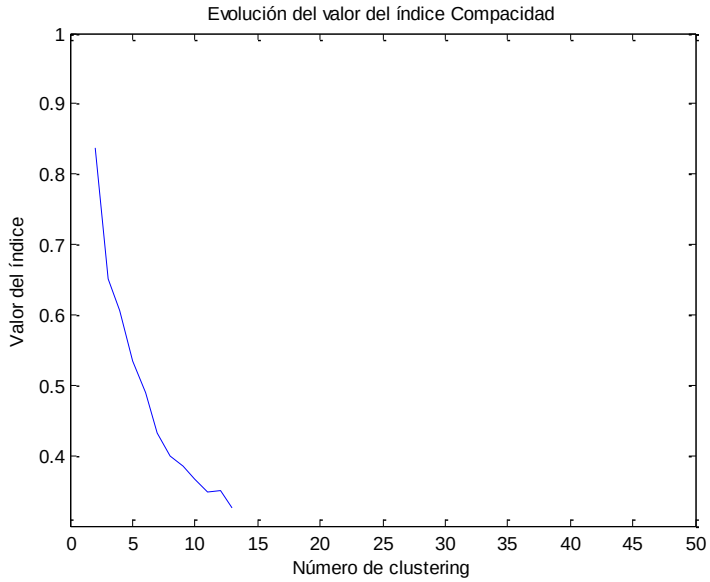

Figura 104.- Evolución del índice de compacidad.

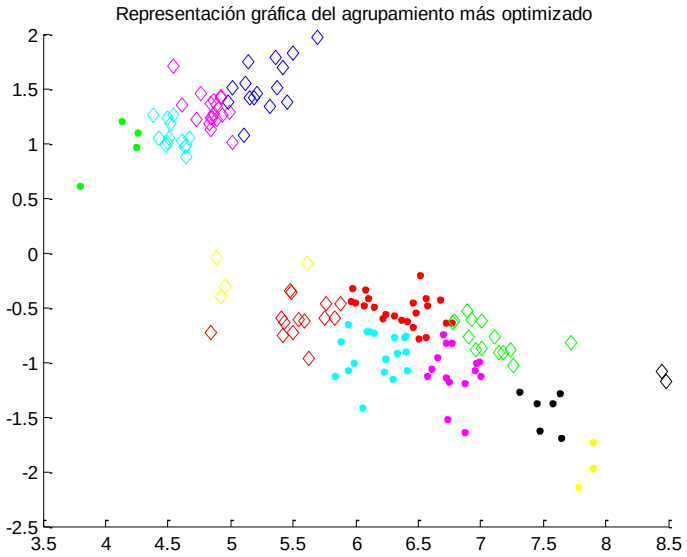

Figura 105.- Mejor agrupamiento índice de compacidad.

Como ya se ha comentado anteriormente, el índice de compacidad no está definido para agrupaciones donde aparecen clusters con un solo miembro, ese es el motivo, por lo que el índice sólo está definido para las agrupaciones que contienen entre dos y trece clusters. Teniendo un valor cada vez menor, según aumenta el número de clusters.

6.5.6. Índice de Calinski-Harabasz

Este índice se haya pulsando el botón de *"Índice Calinski-Harabasz"*, se trata de un índice que mide la calidad de un agrupamiento, siendo mejor el agrupamiento cuanto mayor sea el valor del índice.

```
% --- Executes on button press in Ind_C_H.
function Ind_C_H_Callback(hObject, eventdata, handles)
% hObject    handle to Ind_C_H (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
maximo,num_clusters]=comprobar_indice_C_H(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_C_H, 'string',num2str(maximo) );
    set( handles.numero_clusters_C_H, 'string',num2str(num_clusters)
);
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 48.- Botón Índice Calinski-Harabasz.

El funcionamiento de esta parte del programa es la siguiente, primero se comprueba que el programa se esté ejecutando en el orden correcto y se cargan los datos almacenados de un análisis de cluster anterior. A continuación se procede a realizar la llamada a la función *comprobar_indice_C_H* y se representa el mejor agrupamiento creado cuya información ha sido devuelta por la función anterior. El código correspondiente a dicha función es el siguiente:

```
function [Cluster_mejor,
maximo,numero_clusters]=comprobar_indice_C_H(datos_cluster)
[f,c,w]=size(datos_cluster);
maximo=0;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_Calinski_Harabasz(Clusters);
    if indices(i)>maximo
       maximo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice Calinski-Harabasz')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 49.- comprobar_indice_C_H.

El código de esta parte del programa es el siguiente, en él se realizan sucesivas llamadas a la función *Indice_Calinski_Harabasz*, tantas como agrupamientos se hayan creado anteriormente. Esta función devuelve el valor del índice y el número de clusters que tiene cada agrupación. Se evalúan todos los resultados obtenidos, hasta quedarse con el máximo de todos. Por último se realiza una gráfica que muestra el valor del índice en función del número de clusters. El código de la función *Indice_Calinski_Harabasz* que calcula el valor del índice en cuestión es el siguiente:

```
%Un valor mayor del índice indica una mayor calidad del agrupamiento
de datos
function [Value,num_clusters]=Indice_Calinski_Harabasz(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%Se averigua el número de clusters
actual
K=num_clusters;
n=f;%Número de patrones
%%Se calcula el valor de Z que es la media de todos los puntos
acumuladorx=0;
acumuladory=0;
for i=1:f
    acumuladorx=Clusters(i,1)+acumuladorx;
    acumuladory=Clusters(i,2)+acumuladory;
end
Z(1,1)=acumuladorx/f;
Z(1,2)=acumuladory/f;
%z media de todos los puntos dentro de un cluster
z=zeros(num_clusters,2);
C=zeros(num_clusters);%%Número de patrones que pertenecen al cluster
for i=1:num_clusters
    acumuladorx=0;
    acumuladory=0;
    h=0;
    for j=1:f
       if Clusters(j,3)==i;
           acumuladorx=Clusters(j,1);
           acumuladory=Clusters(j,2);
           h=h+1;
       end
    end
    clusterx=acumuladorx/h;
    clustery=acumuladory/h;
    z(i,1)=clusterx;
    z(i,2)=clustery;
    C(i)=h;
end
%%Tr_B=trazas de las matrices B
acumulador=0;
for i=1:num_clusters
    distancia=abs(sqrt((z(i,1)-Z(1,1))^2+(z(i,2)-Z(1,2))^2));
    acumulador=C(i)*distancia^2+acumulador;
end
Tr B=acumulador;
```

```
%%Tr_W=trazas de las matrices W
acumulador=0;
for i=1:num_clusters
    for j=1:f
        if Clusters(j,3)==i
            distancia=abs(sqrt((z(i,1)-Clusters(j,1))^2+(z(i,2)-
Clusters(j,2))^2));
            acumulador=distancia^2+acumulador;
        end
    end
end
Tr_W=acumulador;
C_H=((n-K)*Tr_B)/((K-1)*Tr_W);
Value=C_H;
```

Función 50.- Indice_Calinski_Harabasz.

La forma de como se calcula el valor del índice es la siguiente, en primer lugar se calcula el número de patrones y de cluster que hay, y luego se calcula la media de todos los puntos (*Z*) y de los patrones dentro de cada cluster (*z*). A continuación se haya el valor de las trazas de las matrices *B* y *W*, y se procede a hallar el valor del índice.

Al igual que se ha realizado con los índices anteriores, para evaluar el correcto funcionamiento del índice, se han seleccionado los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice Calinski-Harabasz"*, obteniéndose el siguiente resultado:

Figura 106.- Índice Calinski-Harabasz.

A su vez el programa nos devuelve las dos siguientes figuras:

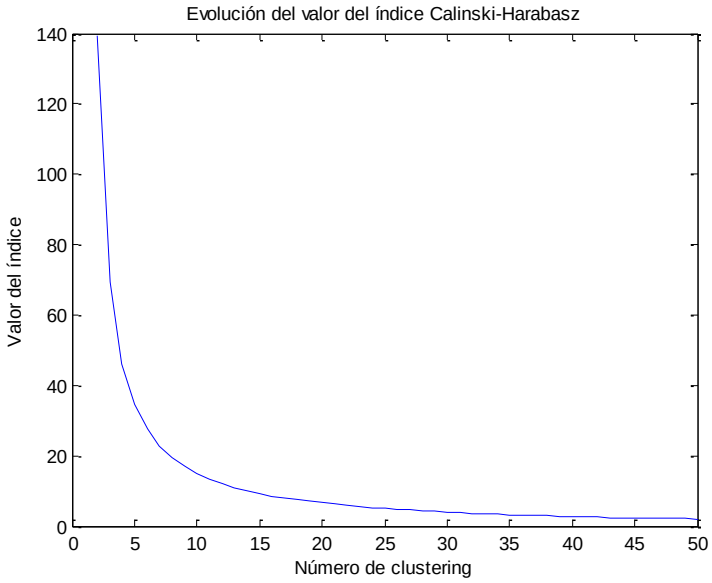

Figura 107.- Evolución del índice Calinski-Harabasz.

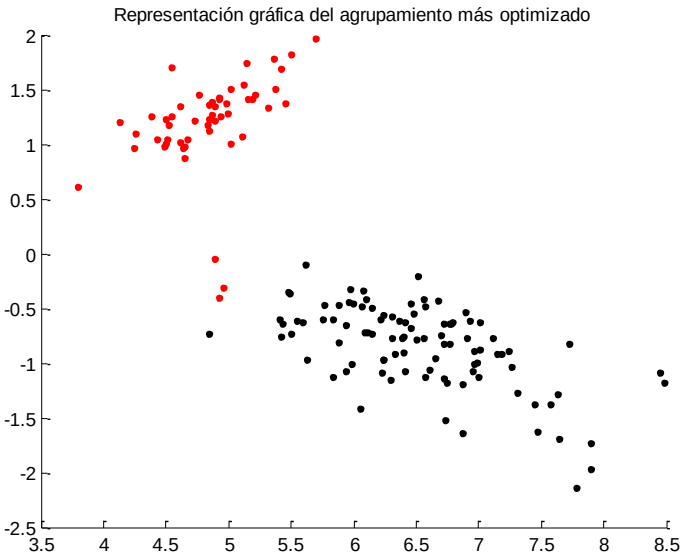

Figura 108.- Mejor agrupamiento índice Calinski-Harabasz.

Como se ha comentado con anterioridad este índice ofrece un mejor resultado, para agrupaciones que proporcionen un valor de índice mayor, debido a la forma con la que se calcula el índice, se presupone que cuanto menos clusters haya mejor será el resultado que se obtenga.

6.5.7. Índice R-Squared

El último índice que se ha implementado es el índice R-Squared, el cual se obtiene cuando se pulsa en el botón *"Índice R-Squared"*, un valor mayor del índice indica una mayor calidad del agrupamiento creado. El código del programa que se ejecuta es el siguiente:

```
% --- Executes on button press in Ind_R_S.
function Ind_R_S_Callback(hObject, eventdata, handles)
% hObject    handle to Ind_R_S (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('est');
if estado<3
    orden_incorrecto;
else
    load('dat_cluster');
    [Cluster_mejor,
maximo,num_clusters]=comprobar_indice_R_Squared(datos_cluster);
    save('Cluster_def','Cluster_mejor');
    set( handles.value_R_S, 'string',num2str(maximo) );
    set( handles.numero_clusters_R_S, 'string',num2str(num_clusters)
);
    figure
    hold on
    representacion(Cluster_mejor);
end
```

Función 51.- Botón Índice R-Squared.

En este código del programa al igual que en los anteriores, se comprueba que el programa se esté ejecutando correctamente y se produce la llamada a la función *"comprobar_indice_R_Squared"* y por último se representa el mejor agrupamiento que se ha creado según el valor del índice R-Squared. El código de la función *"comprobar_indice_R_Squared"* es el siguiente:

```
function [Cluster_mejor,
maximo,numero_clusters]=comprobar_indice_R_Squared(datos_cluster)
[f,c,w]=size(datos_cluster);
maximo=0;
for i=1:f
    a=datos_cluster(i,:,:);
    Clusters(:,:)=a;
    [indices(i),num_clusters(i)]=Indice_R_Squared(Clusters);
    if indices(i)>maximo
       maximo=indices(i);
       b=datos_cluster(i,:,:);
       Cluster_mejor(:,:)=b;
       numero_clusters=num_clusters(i);
    end
end
figure
plot(num_clusters,indices)
title('Evolución del valor del índice R-Squared')
xlabel('Número de clustering')
ylabel('Valor del índice')
```

Función 52.- comprobar_indice_R_Squared.

El funcionamiento de esta función es análogo a las demás funciones de comprobación de los demás índices, en ella se producen sucesivas llamadas a la función *"Indice_R_Squared"*, tantas como agrupamientos se hayan creado previamente. Esta función devuelve el valor del índice y se elige aquella que da un valor mayor, como la que presenta el mejor agrupamiento. Por último se muestra una gráfica donde se puede observar la evolución del índice en función del número de clusters. El código de la función *"Indice_R_Squared"* es el siguiente:

```
%Función que calcula el valor del índice de R-Squared
%Un valor más alto del índice indica un mejor resultado
function [Value,num_clusters]=Indice_R_Squared(Clusters)
[f,c]=size(Clusters);%Función que calcula el número de patrones que
hay
num_clusters=max(Clusters(:,3));%%Se averigua el número de clusters
actual
%%Cálculo de SSt: la suma total de cuadrados de todos los datos
y=zeros(c-1,1);%Se va a calcular la media de las dimensiones
for j=1:(c-1)
    acumulador=0;
    for i=1:f
        acumulador=acumulador+Clusters(i,j);
    end
    y(j)=acumulador/f;
end
acumulador=0;
for j=1:(c-1)
    for i=1:f
        temp=(Clusters(i,j)-y(j))^2;
        acumulador=acumulador+temp;
    end
end
SSt=acumulador;
%%Cálculo de SSw: la suma de los cuadrados dentro de un grupo
u=zeros(num_clusters,c-1);
for k=1:num_clusters
    for j=1:(c-1)
        ac=0;
        numero=0;
        for i=1:f
           if Clusters(i,3)==k
               ac=ac+Clusters(i,j);
               numero=numero+1;
           end
        end
        u(k,j)=ac/numero;
    end
end
acumulador=0;
for k=1:num_clusters
    for j=1:(c-1)
        for i=1:f
           if Clusters(i,3)==k
               temp=(Clusters(i,j)-u(k,j))^2;
               acumulador=acumulador+temp;
           end
        end
    end
end
SSw=acumulador;
R_Squared=(SSt-SSw)/SSt;
Value=R_Squared;
```

Función 53.- Indice_R-Squared.

El funcionamiento de esta parte del programa es la siguiente, en primer lugar se averigua el número de patrones y de clusters existentes, y luego se calcula la suma de los cuadrados de todos los datos y de los datos que pertenecen a un mismo cluster, y con estos datos se procede a calcular el valor del índice.

Al igual que se ha realizado con los índices anteriores, para evaluar el correcto funcionamiento del índice, se han seleccionado los 150 datos de IRIS y realizando un análisis de componentes principales matriz de covarianza, y posteriormente se le aplica el algoritmo K-Means con número de clusters comprendido entre 2 y 50, y se pulsa el botón *"Índice R-Squared"*, obteniéndose el siguiente resultado:

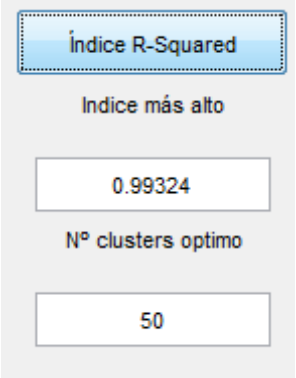

Figura 109.- Índice R-Squared.

Y el programa también nos devuelve las dos siguientes gráficas:

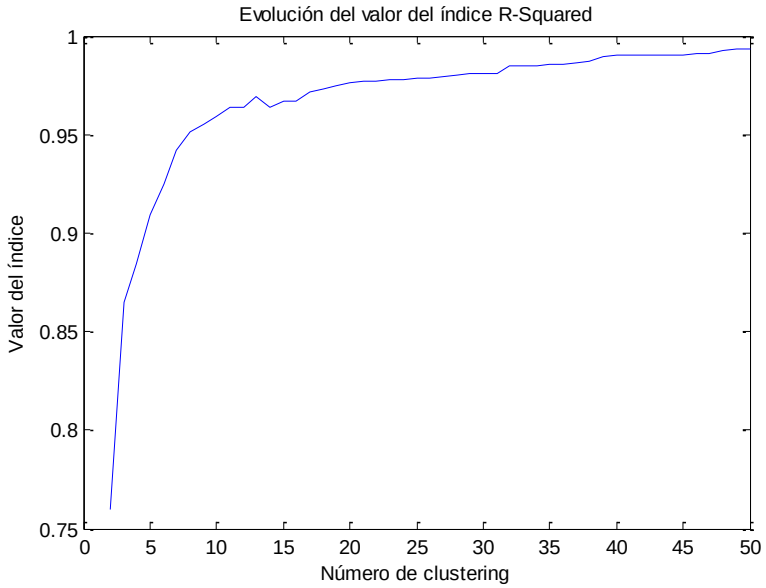

Figura 110.- Evolución del valor del índice R-Squared.

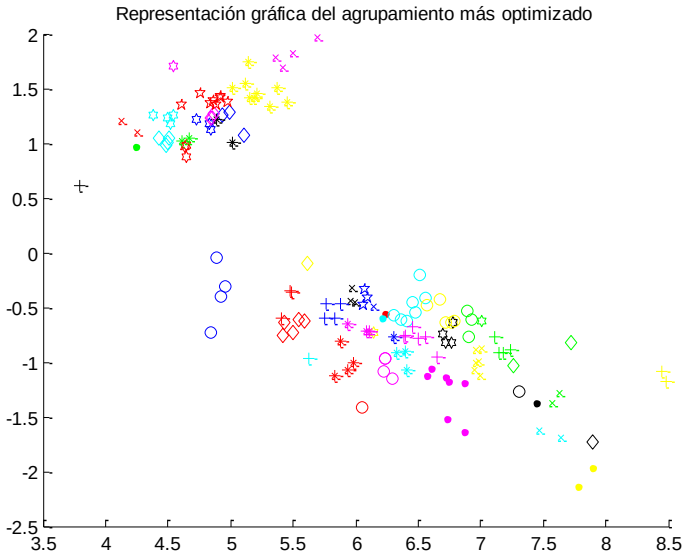

Figura 111.- Mejor agrupamiento índice R-Squared.

Como se puede observar para este índice, cuanto mayor sea el número de cluster mejor es el resultado que proporciona.

6.6. Reset

Se ha introducido en el programa el botón de *"Reset"*, cuando se pulsa dicho botón se ejecuta el siguiente código:

```
% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
estado=0;
save('est','estado')
clc;
close all;
clear all;
```

Función 54.- Botón Reset.

Cuando se pulsa el botón se pasa la variable *estado* del programa al valor 0, por lo que si se quisiera volver a trabajar con el programa habría que empezar desde el principio, a continuación se limpia la pantalla del Command Windows, se borran todas las variables de la zona de trabajo y se cierran todas las ventanas emergentes y figuras abiertas, incluida la del programa *Inicio.*

Este botón se encuentra situado en la siguiente posición del programa.

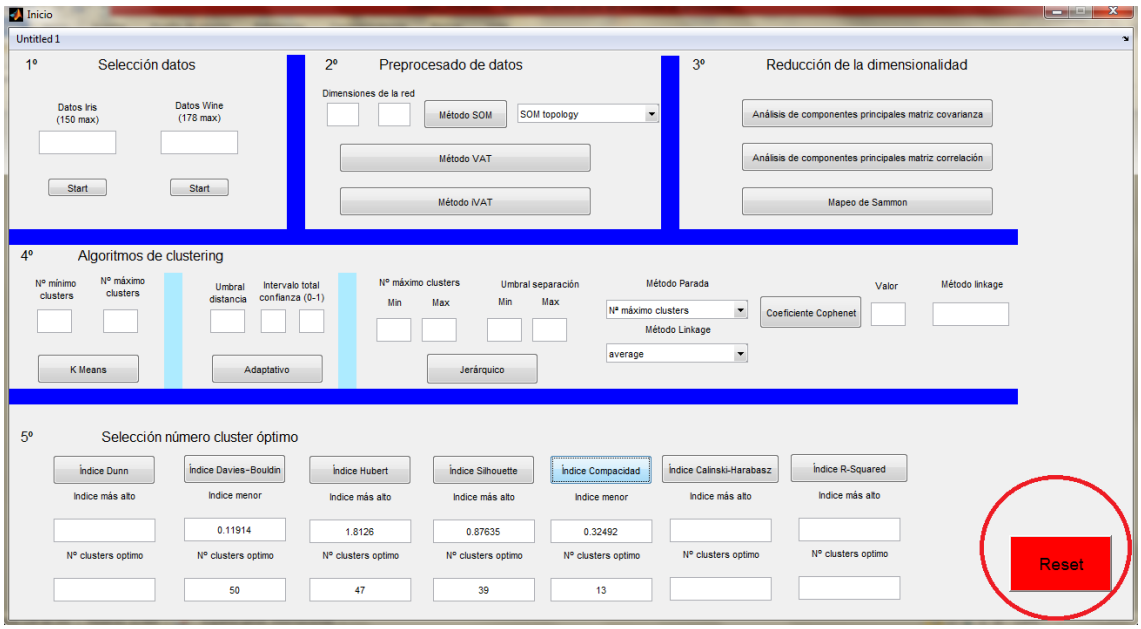

Figura 112.- Botón Reset.

7. Conclusiones

Las herramienta de análisis de validez del proceso de clustering, son muy útiles ya que permiten analizar los datos obtenidos, compararlos con otros resultados y obtener el mejor, también permiten encaminar el análisis con la ayuda de las técnicas de análisis de tendencia y de reducir las dimensiones de los datos. También permiten evaluar la calidad del resultado a través de los índices de validación, saber si el resultado es estable y no es fruto del azar, es decir, que pequeñas variaciones en los datos producirían pequeños o nulos cambios en los clusters creados, y que el resultado esté consensuado, ya que para cada caso se podría seleccionar el mejor resultado creado.

Realizar un análisis de tendencia previo a la aplicación de los algoritmos de clustering es importante por dos motivos: reducción de la dimensionalidad y técnicas de información visual. El primero es la reducción de las dimensiones de los datos pasando de tener patrones que tienen varios datos, por lo que no pueden ser representados, a patrones con dos o tres datos que permiten su representación, manteniendo la estructura y relación entre los datos y con el consiguiente ahorro computacional, esto se consigue con las técnicas de Mapeo de Sammon o Análisis de Componentes Principales, que producen una reducción de las dimensiones de los datos siendo más fácil trabajar con ellos y realizar una representación gráfica de los mismos.

Luego están las técnicas como los mapas autoorganizativos (SOM) o los métodos VAT y sus distintas variantes ofrecen una información visual de la estructura de los datos, la cual puede ser muy útil para predecir el número de clusters a crear y el tamaño de los mismos. Estas técnicas realizan un análisis de la tendencia de los datos previas, que puede ser muy útil, ya que para aplicar ciertos algoritmos de clustering es necesario pasarle valores como número de clusters a crear (algoritmo K-means), proximidad entre los patrones que van a formar un cluster (algoritmo Adaptativo) o el umbral de separación y el número de clusters como criterios de parada en los algoritmos Jerárquicos.

La calidad del resultado de aplicar un algoritmo de clustering depende de varios factores como pueden ser: el algoritmo empleado, ya que la elección de la estrategia de agrupamiento, e incluso diferentes implementaciones de un algoritmo puede proporcionar resultados distintos; el valor de los parámetros del algoritmo, que no es una tarea sencilla sino se ha realizado un análisis de tendencia previo; los patrones que se emplean y el orden en que se procesan, ya que en función del patrón por el que se comience se pueden llegar a obtener agrupaciones distintas; y la medida de similitud adoptada, ya que diferentes criterios de distancia pueden proporcionar diferentes resultados.

La mayor dificultad inherente a la aplicación de técnicas de agrupamiento son: el establecer medidas de similitud adecuadas y el establecer buenos criterios de partición. Algunos algoritmos presentan los siguientes problema o debilidades, como el algoritmo K- means que presenta el inconveniente de que devuelve un mínimo local, que puede no

ser el mínimo global, o el método Adaptativo que es útil cuando no se conoce de antemano el número de clusters a crear, pero tiene el inconveniente de que depende en exceso del orden de presentación de los datos. En ambos algoritmos debido a que las agrupaciones se hacen en función de los patrones que están más cercanos, y que al tratarse de un grupo de asignaciones establecidas y de que el proceso de inicialización se hace por los primeros patrones, puede provocar que la solución este sesgada y que cambiando el orden de los patrones se obtenga un resultado diferente. En el algoritmo Adaptativo valores no adecuados del umbral de selección pueden provocar que todos los patrones pertenezcan a un mismo cluster o que haya un cluster por cada patrón, cono lo vuelve a tomar importancia el análisis de tendencia previo.

En los algoritmos Jerárquicos, o bien se parte de un único cluster y se llega a una división con un cluster por cada patrón o se parte de un cluster por cada patrón y se llega a un único cluster que agrupe todos los patrones, en ambos casos se presenta el inconveniente de saber en que momento parar el proceso de agrupación.

Respecto a los índices de validación decir que no es posible crear un índice de validación capaz de trabajar de manera imparcial con cualquier algoritmo de clustering. Cada algoritmo impone una estructura sobre los datos, que si la propiedad que mide cierto índice no está acorde con la estructura impuesta por el algoritmo, el índice va a valorar la clasificación como mala, incluso cuando dicha clasificación tenga sentido y brinde información relevante para un problema en particular. Por lo que se hace necesario aplicar varios índices de validación al resultado obtenido y llegar a un consenso, de si la partición realizada es correcta o no. También añadir que para algunos índices, como por ejemplo Hubert modificado, el valor del índice no está definido si un cluster corresponde a un solo patrón o todos los patrones pertenecen al mismo cluster.

8. Trabajos futuros

El trabajo realizado se podría continuar o ampliar con algunas de las posibles opciones de trabajo que se describen a continuación:

a) Implementación de una mayor cantidad de algoritmos de clustering, como podrían ser los algoritmos de Batchelor y Wilkins o el algoritmo de agrupamiento secuencial, entre otros.
b) Implementando más técnicas de análisis de tendencia, dentro de esta sección se podrían incluir por ejemplo las distintas versiones del método VAT (reVAT, bigVAT o sVAT).
c) Desarrollando una mayor cantidad de índices de validación. También debido a que no todos los índices de validación son adecuados para todos los agrupamientos creados por los distintos algoritmos de clustering, se podría hacer una selección de índices por algoritmos.
d) Añadir a la herramienta una solución de clustering consensuada. Muchos algoritmos de clustering requieren de una inicialización aleatoria, que en muchos casos, condiciona el resultado, por lo que sería interesesante ejecutar el algoritmo varias veces con distintas inicializaciones y comparar los resultados para obtener el mejor posible. También se podrían aplicar diversos algoritmos al mismo conjunto de datos y comparar los resultados para así obtener la mejor solución. Esta solucion aumentaría la robustez y la calidad de los resultados del proceso de clustering.
e) Desarrollo de algoritmos genéticos, que permitirían hallar una mejor solución a partir de la evolución del algoritmo en función de los resultados obtenidos. Como por ejemplo puede ser el algoritmo IEKA (Intelligent Evolutionary K-means Algorithm), que es un algortimo genético basado en el algoritmo K-means que se ha realizado en este trabajo.

9. Referencias

[1] Pattern Recognition. Concepts, methods and applications. J.P. Marques de Sá. Springer.

[2] Algorithms for clustering data. Jains and Dubes. Prentice Hall.

[3] Aportaciones a la clasificación no supervisada y a su validación. Aplicación a la seguridad informática. Ibai Gurrutxaga Goikoetxea.

[4] Validación de Clusters usando IEKA y SL-SOM. Alessandro Bokan Garay. Universidad Católica San Pablo

[5] Data Preparation for Data Mining. Dorian Pyle. Morgan Kauffman Publisher, 1999.

[6] Data Mining. Concepts and Techniques. Jiawei Han and Micheline Kamber. Morgan Kauffman Publisher, 2001.

[7] Preprocesado de Datos. Juan A. Botía. Departamento de Ingeniería de la Información y las Comunicaciones. Universidad de Murcia.

[8] Fundamentos de data mining y sus aplicaciones. N. Queipo, S. Pintos

[9] Combinación de agrupamientos: un estado del arte. Lic. Sandro Vega-Pons, Dr. C. José Ruiz-Schulcloper. RT_029 CENATAV. Enero 2010.

[10] Dunn JC (1974). Well Separated Clusters and Optimal Fuzzy Partitions." Cybernetics and Systems, 4(1), 95-104.

[11] Davies DL, Bouldin DW (1979). “A Cluster Separation Measure." IEEE Transactions on Pattern Analysis and Machine Intelligence, 1(2), 224-227.

[12] Dunn JC (1974). Well Separated Clusters and Optimal Fuzzy Partitions." Cybernetics and Systems, 4(1), 95-104.

[13] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, " Clustering Validity Checking Methods: Part II" ACM SIGMOD Record, vol. 31, no. 3, September 2002.

[14] U. Maulik and S. Bandyopadhyay, "Performance evaluation of some clustering algorithms and validity indices," IEEE Transaction on Pattern Analysis and Machine Learning, vol. 24, no. 12, pp. 1650-1654, December 2002.

[15] Nguyen N, Caruana R (2007). "Consensus Clusterings." In Proceedings of IEEE International Conference on Data Mining, pp. 607-612. IEEE Computer Society, Washington, DC.

[16] P.J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of 160

cluster analysis. 1987. Journal of Computational and Applied Mathematics. 20. 53-65.

[17] J. C. Bezdek, R. J. Hathaway. VAT: a tool for visual assessment of (cluster) tendency. International Symposium on Neural Networks - ISNN , vol. 3, pp. 2225-2230, 2002.

[18] Timothy C. Havens, James C. Bezdek. An Efficient Formulation of the Improved Visual Assessment of Cluster Tendency (iVAT) Algorithm.

[19] J. W. Sammon, Jr, "A nonlinear mapping for data structure analysis," IEEE Transactions on Computers, vol. C-18, no. 5, pp. 401–409, 1969.

[20] Carles M. Cuadras. Nuevos Métodos de Análisis Multivariantes. 21 de diciembre de 2012.

[21] Xindong Wu, Kumar Vipin, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Georey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, y Dan Steinberg. Top 10 algorithms in data mining. Knowledge and Information Systems, 14:1-37, 2008.

[22] Ibai Gurrutxaga Goikoetxea, Aportaciones a la clasificación no supervisada y a su validación. Aplicación a la seguridad informática. Facultad de Informática. Universidad País Vasco. 2010.

[23] Jiawei Han: "Data Mining: Concepts and Techniques", capítulo 7, 2006.

[24] http://www.mathworks.es/es/help/stats/hierarchical-clustering.html.

[25] http://www.mathworks.es/es/help/nnet/ug/self-organizing-feature-maps.html.

[26] Margareta Ackerman y Shai Ben-David. Clusterability: A theoretical study. En Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 5, pág. 1-8, 2009.

[27] Anil K. Jain y Richard C. Dubes. Algorithms for Clustering Data. Prentice- Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[28] J.M. Huband, J.C. Bezdek, y R.J. Hathaway. Revised visual assessment of (cluster) tendency (reVAT). En Proceedings of the North American Fuzzy Information Processing Society (NAFIPS), pág. 101-104, 2004.

[29] J.M. Huband, J.C. Bezdek, y R.J. Hathaway. bigVAT: Visual assessment of cluster tendency for large data dets. Pattern Recognition, 38:1875-1886, 2005.

[30] Richard J. Hathaway, James C. Bezdek, y Jacalyn M. Huband. Scalable visual assessment of cluster tendency for large data sets. Pattern Recognition, 39:1315-1324, 2006.

[31] Louis Massey. Determination of clustering tendency with art neural networks. En Proceedings of 4th International Conference on Recent Advances in Soft Computing, 2002.

[32] B.S. Everitt. Graphical Techniques for Mirltivuriute Datu. New York, NY: North Holland, 1978.