Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Licenciatura en Ciencias de la Computación

Tesina de Grado

Algoritmo divisivo de clustering con determinación automática de componentes

Erica Vidal
ericavidal@gmail.com
Director: Dr. Ariel Bayá
baya@cifasis-conicet.gov.ar
Co-Director: Dr. Pablo Granitto
granitto@cifasis-conicet.gov.ar

Rosario - Santa Fe - Argentina

2014

Resumen

Cluster analysis es el estudio de algoritmos y métodos cuyo objetivo es encontrar una forma conveniente y válida de organizar un conjunto de datos en grupos. Entre sus múltiples aplicaciones se encuentran la segmentación de imágenes, la clasificación automática de documentos o archivos multimedia, la detección de comunidades en redes sociales y la identificación de genes con funciones similares, por nombrar algunas.

Un problema todavía abierto en el área de cluster analysis es la determinación automática del número de clusters o grupos en un conjunto de datos. Además, dicho número depende de la escala o resolución con la que se ven los datos, lo que incrementa la complejidad del problema, en especial cuando hay clusters a diferentes escalas en el conjunto de datos. Otro problema común es la presencia de parámetros libres que regulan la performance de los métodos de clustering, ya que la determinación del valor óptimo de los parámetros es un problema de difícil solución para el usuario no experto, sobre todo si los métodos son muy sensibles a la variación de los valores de dichos parámetros. Como regla general, la utilidad práctica de un método de clustering depende de la posibilidad de optimizar dichos parámetros de forma simple y eficaz.

En este trabajo exploramos soluciones a estos problemas y como resultado desarrollamos un nuevo algoritmo de clustering, DHclus, que descubre clusters con formas arbitrarias en los datos, determina automáticamente la cantidad de componentes presentes en los datos (incluso cuando hay clusters a diferentes escalas en un mismo problema) y además selecciona los parámetros que afectan su performance. Además implementamos el algoritmo como un paquete de software libre para el entorno de programación R y se encuentra disponible para toda la comunidad.

Palabras clave: clustering divisivo, spectral clustering, métricas , test de hipótesis.

Agradecimientos

Quiero agradecer a mis directores, Ariel y Pablo, ellos me dieron la oportunidad de realizar este trabajo y me guiaron y apoyaron hasta que pude completarlo. También deseo agradecerles a Gabriela Argiroffo y Guillermo Grinblat que con sus comentarios y correcciones ayudaron a mejorarlo.

Mi total gratitud a la gente del Grupo de Sistemas Inteligentes del CIFASIS por generar un excelente ambiente de trabajo, en especial a Mónica por sus buenos consejos. También a la Institución por brindarme el espacio y las herramientas necesarias para realizar mi investigación. En el mismo sentido, gracias al grupo de Materia condensada del IFIR, en particular a Claudio Gazza, por permitirme usar sus equipos de cálculo del cluster de alta performance del Centro Científico y Tecnológico.

También gracias a mis profesores, por su enseñanza y dedicación, y a todos mis compañeros de carrera. Gracias por las horas de estudio a Regina, Mariano, Guille y Silvia.

Agradezco a mis amigas y a mi familia, la natural y la política, por su apoyo incondicional y por su contención. Y muy especialmente quiero agradecer a mi amor, mi amigo y mi esposo Matías, por su paciencia, su amor, su apoyo y colaboración en este trabajo y en todo lo que emprendo.

Muchas gracias a todos!!!

Índice general

1 Introducción

Cluster analysis es el estudio de algoritmos y métodos cuyo objetivo es encontrar una forma conveniente y válida de organizar un conjunto de datos en grupos. Entre sus múltiples aplicaciones se encuentran la segmentación de imágenes, la clasificación automática de documentos o archivos multimedia, la detección de comunidades en redes sociales y la identificación de genes con funciones similares, por nombrar algunas.

El desarrollo de la metodología de clustering ha sido interdisciplinario. Investigadores de muchas áreas han contribuido: taxonomistas, psicólogos, biólogos, estadísticos, cientistas sociales, ingenieros e informáticos [1]. Por ello es que clustering se puede encontrar bajo diferentes nombres en diferentes contextos, como aprendizaje no supervisado (en Machine Learning y Pattern Recognition), taxonomía numérica (en Biología, Ecología), tipología (en Ciencias Sociales) y partición (en Teoría de Grafos) [2]. Además de diferentes terminologías, estas comunidades tienen diferentes supuestos acerca de los componentes del proceso de clustering y el contexto en el que se usa [3].

Este trabajo sigue la terminología de Pattern Recognition y Machine Learning. En estas disciplinas se hace una distinción entre los problemas de aprendizaje automatizado en I) supervisados (clasificación) y II) no supervisados (clustering). Los primeros involucran datos etiquetados que se usan para entrenar un learner para luego predecir el comportamiento de un conjunto de datos no vistos, mientras que los segundos involucran datos sin etiquetas [4].

Aunque se han realizado importantes desarrollos en clustering en los últimos tiempos [5, 6], un problema todavía abierto es la determinación automática del número de clusters, grupos o componentes en un conjunto de datos, además el número de clusters depende de la escala o resolución con la que se ven los datos lo que incrementa la complejidad del problema. Otro problema común es la presencia de parámetros libres que regulan la performance de los métodos de clustering, ya que la determinación del valor óptimo de

los parámetros es un problema de difícil solución para el usuario no experto, sobre todo si los métodos son muy sensibles a la variación de los valores de dichos parámetros.

En este trabajo presentamos el desarrollo de un algoritmo de clustering que descubre clusters con formas arbitrarias en los datos, determina automáticamente la cantidad de componentes presentes en los datos (incluso cuando hay clusters a diferentes escalas en un mismo problema) y además selecciona los parámetros que afectan su performance.

El capítulo 2 introduce algunas nociones básicas: conceptos de medidas de proximidad, métricas y conceptos de la teoría espectral de grafos. El capítulo 3 introduce algunos algoritmos de clustering populares, y profundiza sobre los algoritmos de spectral clusterig. En el capítulo 4 presentamos una métrica denominada `pknng`, necesaria para el desarrollo del algoritmo. El capítulo 5 contiene medidas que sirven para evaluar los resultados de un algoritmo y métodos para determinar el número de clusters presentes en los datos. El capítulo 6 contiene el desarrollo del algoritmo `dhclus`, el pseudocódigo del algoritmo desarrollado, una explicación del funcionamiento del algoritmo a través de un ejemplo y el análisis de la complejidad temporal. En el capítulo 7 se encuentran los resultados de la ejecución del algoritmo `dhclus` sobre datasets artificiales y luego sobre datos de expresión de genes de dominio público. En cada caso describimos los dataset utilizados, el detallamos los experimentos realizados, mostramos los resultados obtenidos y analizamos dichos resultados. Finalmente el capítulo 8 contiene las conclusiones del trabajo y posibles trabajos futuros.

2 Conceptos generales

Es difícil tener una definición matemática universal de clustering que sirva para cualquier posible aplicación. Intuitivamente, se tiene un conjunto de objetos (imágenes, expresión de genes, textos, etc.) y se quiere encontrar alguna agrupación significativa entre ellos. Por lo general la cantidad de grupos no es conocida de antemano y los grupos se crean en función de la naturaleza de los datos.

Los objetos a agrupar suelen estar representados por una matriz de patrones. Un **patrón** es un vector $x = [x_1, x_2, \ldots, x_d]$ compuesto por d mediciones que se denominan **atributos** (también conocidos como features) del patrón y d es la dimensión del patrón o del espacio de patrones. Los valores de los atributos pueden ser:

- cuantitativos, es decir valores reales.
- categóricos o nominales.
- ordinales (existe una noción de orden entre los valores).

Si los atributos están a escalas muy diferentes será necesario estandarizarlos. También se deben observar los valores atípicos (outliers) o faltantes de los atributos, y tener en cuenta si ciertos atributos tienen más relevancia que otros para el clustering. Estas tareas tienen el objetivo de elegir apropiadamente los atributos con los cuales trabajará el algoritmo de clustering, constituyen la primer etapa del proceso de cluster analysis y se las suele denominar **feature selection**. El resto del proceso comprende la selección de la **medida de proximidad** entre pares de objetos, el **clustering**, la **validación** y la interpretación de los resultados [2],[3].

El problema crucial de identificar clusters es especificar qué es proximidad y cómo medirla.

2.1. Medidas de proximidad

A menudo los datos están representados en términos de proximidad entre pares de objetos. Una **matriz de proximidad** es una matriz en la que cada columna y fila representa un patrón y que contiene en sus elementos los índices de proximidad entre los pares de patrones. Se supone que es simétrica. Un índice de proximidad es una medida de similaridad o de disimilaridad. La medida más común de disimilaridad es la distancia.

2.1.1. Distancia

Dado un conjunto de datos X, una **distancia** es una función $d : X \times X \to \mathbb{R}$ que satisface:

- **No negatividad**: $d(x, y) \geq 0$
- **Simetría**: $d(x, y) = d(y, x)$
- **Reflexividad**: $d(x, x) = 0$

para cualquier par de puntos x e y del conjunto.

Para la tarea de clustering, cuando sólo se dispone de la matriz de patrones, primero es necesario definir una disimilaridad entre los pares de puntos. El tipo de disimilaridad utilizada dependerá del tipo de los atributos. En [7] y [8] se repasan medidas de similaridades y disimilaridades para diferentes tipos de variables.

2.1.2. Métrica

Una **métrica** es una función de distancia que además satisface las siguientes propiedades

- **Identidad de los indiscernibles**: $d(x, y) = 0 \Longleftrightarrow x = y$
- **Desigualdad triangular**: $d(x, y) \leq d(x, z) + d(z, y)$

para cualquier punto x, y y z del conjunto.

La métrica más común es la **métrica de Minkowski** que se define:

$$d(x, y) = (\sum_{i}^{d} |x_i - y_i|^r)^{1/r} \tag{2.1.1}$$

donde x e y son vectores d-dimensionales y $r \geq 1$. Si $r = 1$, $d(x, y)$ es la distancia de Manhattan, para $r = 2$ se trata de la distancia euclídea.

2.1.3. Función de similaridad

La similitud o similaridad es una medida de significado opuesto a la disimilaridad. Cada medida de disimilaridad puede transformarse en una medida de similaridad y viceversa. Dado un conjunto de datos X, la **similaridad** es una función $s : X \times X \to \mathbb{R}$ que satisface:

- $0 \leq s(x, y) \leq 1$
- $s(x, x) = 1$
- $s(x, y) = s(y, x)$

donde x e y son dos puntos del conjunto.

2.2. Cluster

Dado un conjunto de datos $X = \{x_1, \ldots, x_n\}$ y alguna noción de similaridad $s_{ij} > 0$ entre los pares de puntos x_i y x_j, se define un $k-$cluster como una partición de X en k conjuntos no vacíos, tal que las similaridades entre los datos dentro del mismo conjunto sean altas mientras que la de los datos entre diferentes grupos sean bajas [2].

Dado que un $k-$cluster se define como una partición, no se permite que un objeto pertenezca a más de un cluster. Este tipo de clusters se denomina crips o hard en contraposición a clusters fuzzy que permiten que un objeto pertenezca a más de un cluster con diferentes grados de pertenencia. El resto del documento se refiere a clusters de tipo crips.

2.3. Teoría espectral de grafos

La teoría espectral de grafos estudia las propiedades de los grafos a través de los autovalores y autovectores de sus matrices asociadas: la matriz de adyacencia y Laplaciana. Ambas matrices han sido muy bien estudiadas desde un punto de vista algebraico por Fan Chung en [9]. Los siguientes conceptos son necesarios para describir el funcionamiento del algoritmo de spectral clustering que presentaremos en el siguiente capítulo.

Grafos Pesados

Un grafo simple[1], no dirigido y pesado $G = (V, E, w)$, es un grafo con vértices $V = \{v_1, \ldots, v_n\}$, y una función de peso w, que satisface que cada arista $e_{ij} \in E$ entre los vértices v_i y v_j posea un peso $w(v_i, v_j) \geq 0$ y $w(v_i, v_j) = w(v_j, v_i)$.

Un grafo G es **conexo** si dados dos vértices v_i, v_j cualesquiera en G, existe un camino de v_i a v_j, que es una sucesión de aristas $e_{i1}, \ldots, e_{nj} \in E$.

Dos vértices v_i y v_j de un grafo G son **adyacentes**, si G contiene una arista (v_i, v_j).

La **vecindad** de un vértice $v_i \in V$, $\Gamma(v_i) \subseteq V$, es el conjunto de todos los vértices que son adyacentes a v_i. $\Gamma(v_i) = \{v_j | (v_i, v_j) \in E\}$.

El **grado de un vértice** v_i, denotado d_i en un grafo pesado es la suma de los pesos de las aristas entre él y los nodos de su vecindario, es decir:

$$d_i = \sum_{v_j \in \Gamma(v_i)} w(v_i, v_j). \tag{2.3.1}$$

La **matriz de grados** de un grafo pesado, D, es una matriz diagonal con $D(i, i) = d_i$.

El **volumen** $vol(V_1)$ de un subconjunto de vértices V_1 de un grafo $G(V, E, w)$ es la suma de los grados de los vértices de V_1.

$$vol(V_1) = \sum_{v_i \in V_1} d_i \tag{2.3.2}$$

Matriz de adyacencia

La **matriz de adyacencia** de un grafo pesado $G = (V, E, w)$, se define como una matriz W a coeficientes reales de dimensión $n \times n$ donde el elemento $W_{(i,j)} = w(v_i, v_j)$.

Matriz Laplaciana

La **matriz Laplaciana** L de un grafo pesado G, se define como $L = D - W$. La **matriz Laplaciana normalizada simétrica** de un grafo pesado, Lsym, es una matriz simétrica $n \times n$ con los elementos definidos de la siguiente manera:

[1] Grafo simple significa que para cada par de vértices se permite a lo sumo una arista y no se permiten lazos.

$$L\text{sym}_{(i,j)} = \begin{cases} 1 - \frac{w(v_i,v_i)}{d_i} & \text{si } i = j \wedge d_i \neq 0 \\ -\frac{w(v_i,v_j)}{\sqrt{d_i d_j}} & \text{si } i \neq j \wedge w(v_i,v_j) > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Si $d_i > 0 \; \forall i \in \{1,\ldots,n\}$ entonces $L\text{sym} = I - D^{-1/2} W D^{-1/2}$.

Se llama **matriz Laplaciana normalizada de random walk** a la matriz cuyas componentes se definen como:

$$L\text{rw}_{(i,j)} = \begin{cases} 1 - \frac{w(v_i,v_i)}{d_i} & \text{si } i = j \wedge d_i \neq 0 \\ -\frac{w(v_i,v_j)}{d_i} & \text{si } i \neq j \wedge w(v_i,v_j) > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Si $d_i > 0 \; \forall i \in \{1,\ldots,n\}$ entonces $L\text{rw} = I - D^{-1} W$.

Propiedades de las matrices $L, L\text{rw}, L\text{sym}$

- Para cualquier vector $f \in \mathbb{R}^n$ se tiene:

$$f' L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2$$

$$f' L\text{sym} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{(}d_i)} - \frac{f_j}{\sqrt{(}d_j)} \right)^2$$

  donde $w_{ij} = W_{(i,j)}$.

- El menor autovalor de L y $L\text{rw}$ es 0 correspondiente al autovector constante $\mathbf{1} = (1,\ldots,1)$. El menor autovalor de $L\text{sym}$ es 0 correspondiente al autovector $D^{1/2}\,\mathbf{1}$.

- L, $L\text{rw}$, y $L\text{sym}$ son matrices semidefinidas positivas y tienen n autovalores a valores reales $0 = \lambda_1 \leq \cdots \leq \lambda_n$.

- λ es un autovalor de $L\text{rw}$ con autovector u si y solo si λ y u resuelven el problema generalizado $Lu = \lambda D u$.

- Si λ es un autovalor de $D^{-1}W$ con autovector x entonces $1 - \lambda$ es autovalor de $I - D^{-1}W$ con autovector x.

3 Algoritmos de clustering

Existen varias categorías de algoritmos de clustering. Basados en cómo quedan definidos los clusters, se pueden clasificar en jerárquicos y particionales [10].

3.1. Algoritmos jerárquicos

Los algoritmos jerárquicos producen una secuencia de particiones anidadas de los datos. Es decir, sea $X = \{x_1, \ldots, x_n\}$ un conjunto de patrones, una partición C de X parte a X en subconjuntos $\{C_1, \ldots, C_m\}$, la partición B está anidada en C si cada componente de B es un subconjunto de una única componente en C, o sea que C se forma mezclando componentes de B. Una **partición jerárquica** es una secuencia de particiones en la cuál cada partición está anidada en la siguiente en secuencia [1].

Los algoritmos jerárquicos proceden de manera divisiva o bien de manera aglomerativa. La estrategia aglomerativa, también llamada bottom-up, comienza considerando a cada dato como un cluster y en cada iteración mezcla un conjunto de clusters pequeños en un cluster más grande hasta que todos los datos sean considerados un único cluster.

Los métodos divisivos, también llamados top-down, comienzan considerando a todo el conjunto de datos como un único cluster y en cada nivel dividen cada cluster en dos hasta terminar con cada objeto en un único cluster, a menos que se indique algún criterio para dejar de dividir los datos. Los algoritmos divisivos pueden aplicar recursivamente cualquier método particional como k-means, que se verá en detalle más adelante (3.2.1), con $k = 2$, en cada iteración [8].

Una característica positiva de los métodos jerárquicos es que son capaces de capturar la posible estructura jerárquica de los datos. Por lo general una jerarquía de clusters se representa mediante un árbol denominado dendrograma. El dendrograma muestra cómo los

clusters están relacionados y cada nivel de la jerarquía representa una partición particular de los datos.

3.2. Algoritmos particionales

La característica de los métodos particionales es que dividen a los datos en k conjuntos disjuntos de manera simultánea y como resultado de eso producen una clasificación en k clases sin relación entre ellas. Usualmente calculan los clusters optimizando algún criterio definido en forma local (usando parte de los patrones) o global (usando la totalidad de los datos). El criterio más comúnmente usado es el error cuadrático, y el algoritmo más popular y simple de este género es el algoritmo k-means que se describe a continuación.

3.2.1. Algoritmo k-means

Sea $X = \{x_1, \ldots, x_n\}$ un conjunto de n puntos d-dimensionales para ser agrupados en k clusters, $C = \{C_i \,|\, i = 1, \ldots, k\}$. El algoritmo k-means (MacQueen, 1967 [11], Lloyd, 1957/1982 [12]) busca una partición tal que se minimice el error cuadrático entre la media de cada cluster y sus respectivos puntos.

El objetivo de k-means es la optimización de la siguiente función objetivo

$$J(C) = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - m_i||^2. \tag{3.2.1}$$

donde

$$m_i = \frac{1}{n_{C_i}} \sum_{x \in C_i} x_i$$

es el centro del cluster C_i. Minimizar $J(C)$ es un problema NP-hard, incluso para $k = 2$. Es por eso que el algoritmo, que es greedy[1] , sólo puede converger a mínimos locales [13].

Inicialmente el algoritmo comienza con una elección aleatoria de k centros y luego repite las siguientes operaciones hasta que los centros no cambien o hasta que se alcance un número máximo de iteraciones.

1. Asignar el resto de los puntos al cluster del centro más cercano de modo de reducir el error cuadrático.

[1] greedy: sigue una heurística consistente en elegir la opción óptima en cada paso.

2. Recalcular los centros m_i de los clusters encontrados.

El tiempo de complejidad de k-means es $O(Nkdm)$, done k es la cantidad de clusters, N el tamaño del dataset, d la dimensión de los datos y m la cantidad de iteraciones. Las principales desventajas de k-means son que es sensible al ruido y los outliers[2], y no puede ser usado para encontrar clusters con formas arbitrarias. Un algoritmo que es menos sensible a la presencia de outliers es PAM.

3.2.2. Partitioning Around Medoids

El algoritmo PAM (a veces denominado K-medoids) usa la misma función objetivo que k-means con la restricción de que los centros de los clusters deben pertenecer al dataset, su objetivo es determinar el mejor representante del centro de cada cluster (medoide) y toma como entrada la matriz de disimilitudes, lo que le permite trabajar con diferentes métricas.

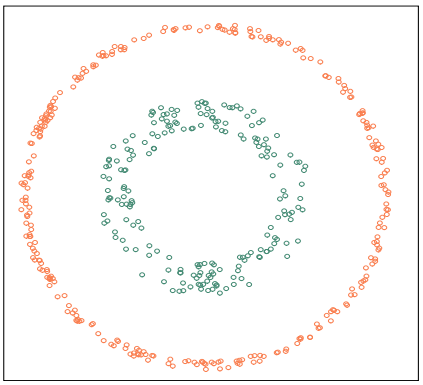


Figura 3.1
Ejemplo de dos clusters no linealmente separables.

El algoritmo comienza eligiendo k medoides, y luego cada punto que no es un centro se agrupa a su medoide más cercano. PAM intercambia los medoides con otros puntos candidatos, de forma de obtener una configuración que minimice la función objetivo. El proceso continúa hasta que no se produzcan más intercambios de medoides.

3.2.3. Spectral Clustering

Los métodos como k-means, que minimizan el error cuadrático, son incapaces de separar correctamente clusters con formas arbitrarias como por ejemplo los clusters de la Figura 3.1, mientras que los algoritmos de spectral clustering responden bien a este tipo de situaciones.

[2]Un outlier es un punto que cae muy lejos del centro del cluster más cercano.

Spectral clustering es una técnica popular que se remonta a principios de los años 70, con los trabajos de Donath, Hoffman y Fiedler [14, 15, 16]. Donath y Hoffman fueron quienes sugirieron usar los autovectores de la matriz de adyacencia de los grafos para encontrar particiones y Fiedler asoció la conectividad de un grafo al segundo autovalor de su Laplaciano y sugirió cómo usar los autovectores para particionarlo. En su forma más simple spectral clustering utiliza el segundo autovector de la matriz Laplaciana construida a partir del grafo de similaridad, para obtener una partición de los datos en dos grupos. La principal diferencia entre los algoritmos de spectral clustering es si utilizan el Laplaciano normalizado o sin normalizar [17].

3.2.3.1. Grafo de similaridad

Una forma de representar la similitud entre los patrones de un conjunto de datos es mediante un **grafo de similaridad**, que es un grafo pesado $G = (V, E, w)$, donde cada vértice v_i en el grafo representa a un dato x_i, cuyas aristas se obtienen a partir de las reglas mencionadas más abajo y el peso w_{ij} de la arista e_{ij} representa la similitud s_{ij} entre los datos x_i y x_j. Entonces el problema de clustering se puede expresar como encontrar una partición del grafo tal que las aristas entre los distintos grupos tengan pesos muy bajos y las aristas entre vértices del mismo grupo tengan pesos altos.

Siguiendo a Von Luxburg [18], a continuación enunciamos las formas más regularmente usadas en spectral clustering para transformar un conjunto de puntos $\{x_1, \ldots, x_n\}$, sus similaridades s_{ij} o distancias d_{ij} en un grafo de similaridad. El objetivo es siempre modelar la relación local de vecindad entre los puntos.

La **regla** ε: Se conectan todos los puntos cuyas distancias sean menores que cierto valor $\varepsilon \in \mathbb{R}$. Una de las desventajas con esta técnica es que el grafo resultante (grafo ε) suele tener problemas con datos con diferentes escalas, es decir, cuando las distancias entre puntos son diferentes en diferentes regiones del espacio.

La **regla** k **vecinos mutuos**: Se conectan el punto x_i al punto x_j si el punto x_j es uno de los primeros k vecinos del punto x_i y x_i es uno de los k primeros vecinos del punto x_j. El grafo resultante (k-nn mutuo) tiende a conectar puntos en regiones de densidad constante, pero no conecta regiones de diferentes densidades.

La **regla k vecinos**: Se conectan el punto x_i al punto x_j si el punto x_j es uno de los primeros k vecinos del punto x_i o x_i es uno de los k primeros vecinos del punto x_j. El grafo (k-nn) resultante es más adecuado para detectar clusters a diferentes densidades.

Grafo completamente conectado: Se obtiene conectando todos los puntos con similaridad positiva entre sí, y el peso correspondiente a cada arista es s_{ij}.

La elección de los parámetros para las distintas reglas no es trivial y no hay demasiados resultados teóricos sobre cómo elegirlos y el problema es que spectral clustering puede ser muy sensible a cambios en el grafo de similaridad. Para una discusión sobre este tema se puede consultar [18].

3.2.3.2. Partición de grafos

Se denomina **corte** de un grafo conexo G, al conjunto de aristas de G que al removerse de G lo desconectan. La suma de los pesos de las aristas removidas del grafo se denomina **tamaño del corte**.

Sean A y B subconjuntos de V, se define $W(A,B) = \sum_{i\in A, j\in B} w_{ij}$. La forma más simple y más directa de construir una partición de un grafo en k componentes o subgrafos de modo que las aristas entre las particiones tengan pesos muy bajos y las aristas dentro de los grupos tengan pesos altos, es resolver el problema de corte con tamaño mínimo.

$$minCut(C_1,\ldots,C_k) = min[\frac{1}{2}\sum_{i=1}^{k} W(C_i, C_i')] \tag{3.2.2}$$

donde C_i' es el complemento de C_i.

El problema es que en muchos casos minimizar el tamaño del corte produce vértices aislados. Una solución es requerir que los conjuntos C_i sean "razonablemente grandes", para lo cual se modifica la función objetivo. Las dos funciones objetivo más comunes son las denominadas RatioCut, presentada en [19] por Hagen y Kang en 1992, y NCut presentada en [20] Shi y Malik en el 2000.

Ambos criterios favorecen particiones de aproximadamente el mismo tamaño (particiones balanceadas). Las definiciones son:

$$RatioCut(C_1,\ldots,C_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(C_i, C_i')}{|C_i|} \tag{3.2.3}$$

$$NCut(C_1, \ldots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(C_i, C_i')}{vol(C_i)} \tag{3.2.4}$$

Los problemas de minimización se pueden escribir como:

$$\min_{\bigcap_i C_i = \varnothing, \bigcup_i C_i = V} RatioCut(C_1, \ldots, C_k) \tag{3.2.5}$$

$$\min_{\bigcap_i C_i = \varnothing, \bigcup_i C_i = V} NCut(C_1, \ldots, C_k) \tag{3.2.6}$$

Los problemas discretos de minimizar RatioCut o NCut son NP-hard, y spectral clustering es una forma de resolver la versión relajada[3] de los anteriores problemas de partición de grafos (ver [18] para más detalles).

Como se muestra en [18], los k autovectores correspondientes a los menores k autovalores del Laplaciano sin normalizar aproximan la solución al problema de minimización de RatioCut, mientras que los autovectores correspondientes a los menores autovalores del Laplaciano de ramdom-walk aproximan la solución al problema de minimización NCut.

3.2.3.3. Algoritmo de spectral clustering

El algoritmo de spectral clustering sin normalizar usa el Laplaciano, $L = D - W$. Primero se calculan los k autovectores correspondientes a los k menores autovalores de L, después se construye una matriz U de $n \times k$, cuyas columnas son los k autovectores y las n filas de U representan los vértice del grafo de similaridad en un espacio euclídeo $k-$dimensional. Finalmente se agrupan las filas en k clusters usando un algoritmo tipo k-means.

Las versiones normalizadas del algoritmo de spectral clustering funcionan de la misma manera, la diferencia principal reside en el tipo de Laplaciano usado. La versión del algoritmo de Shi y Malik presentado en [20], usa los autovectores del Laplaciano normalizado de random-walk (L_{rw}) y se describe en el algoritmo 1.

[3] Relajar el problema significa que se permite asumir valores reales al vector indicador de la partición.

Algoritmo 1 *Algoritmo de Spectral clustering de acuerdo a Shi y Malik[20] (en [18])*

Entrada: Matriz de similaridad $S \in R^{n \times n}$, número k de clusters

Salida: Clusters $C_1, \ldots, C_k$ con $C_i = \{j | y_j \in C_i\}$.

1: **(Computar el Laplaciano)** Construir el grafo de similaridad, sea W su matriz de adyacencia, calcular $L = D - W$.

2: **(Reducción dimensional)** Encontrar los k autovectores $u_1, \ldots, u_k$ de $Lu = \lambda Du$ correspondiente a los menores autovalores, y sea $U \in R^{n \times k}$ la matriz que contiene los vectores $u_1, \ldots, u_k$ como columnas. Para $i = 1, \ldots, n$ sea $y_i \in R^k$ el vector correspondiente a la i-ésima fila de U.

3: **(Clustering)** Clusterizar los puntos $(y_i)\, i = 1, \ldots, n \in \mathbb{R}^k$ con k-means en $C_1, \ldots, C_k$.

Spectral clustering consiste en la transformación del conjunto inicial de objetos a un conjunto de puntos, cuyas coordenadas son elementos de los autovectores. Si el grafo de similaridad es conexo, pero consiste en k subgrafos que están débilmente conectados, el espectro del Laplaciano L tendrá solo un autovalor en cero, y el resto positivos. Los $k-1$ menores autovalores positivos estarán cerca de cero, y las coordenadas de los primeros k autovectores permiten distinguir los clusters en un espacio k-dimensional.

4 Métrica PKNNG

La métrica `pknng` es un métrica basada en grafos de k-vecinos que puede ser utilizada en cualquier algoritmo de clustering basado en búsqueda sobre una matriz de distancias. Produce como resultado una matriz de disimilaridades entre pares de puntos. Aplicar la métrica a un conjunto de puntos es equivalente a transformar el espacio original y el resultado de dicha transformación es un conjunto de puntos linealmente separables. Esta propiedad hace que sea posible detectar clusters con formas arbitrarias usando `pknng` y algún algoritmo de clustering tipo k-means.

Fue presentada en [21] y [22], y sirve para evaluar distancias entre objetos sobre un manifold[1] de menor dimensión que el espacio donde se encuentran los datos.

Se basa en un proceso en el que primero se forma un grafo de k-vecinos, luego se conectan las componentes mediante un peso penalizado para lograr un grafo conexo y finalmente se calculan las distancias geodésicas entre dos puntos como la suma de las distancias del camino mínimo entre ambos.

El algoritmo funciona de la siguiente manera:

1. Calcula las distancias entre los puntos.
2. Genera un grafo k-nn.[2]
3. Elimina arcos outliers.[3]
4. Conecta las estructuras separadas agregando arcos con pesos p.

[1] $M \subseteq \mathbb{R}^D$, es un manifold n-dimensional ($n < D$) si $\forall\, p \in M$ se puede encontrar una biyección "suave" entre $\mathbb{R}^n$ y algún vecino cerca de p.

[2] Se conectan los nodos v_i y v_j si v_i esta entre los primeros k vecinos de v_j o si v_j esta entre los primeros k vecinos de v_i.

[3] El grafo k-nn determina para cada k una distribución de probabilidades sobre los arcos. Un arco es outlier si no es recíproco y su longitud es mayor que el valor del 3^{er} cuartil más 3/2 la distancia intercuartiles de dicha distribución de distancias.

5. Calcula las distancias geodésicas entre los puntos, como la suma de las distancias del camino mínimo entre dos nodos.

El algoritmo acepta como entrada un conjunto de patrones o una matriz de disimilaridad entre los mismos, en este segundo caso se omite el primer paso.

Cuando el grafo construido tiene más de una componente, se deben agregar aristas entre las mismas de modo de lograr una única componente. Los esquemas de conexión pueden ser:

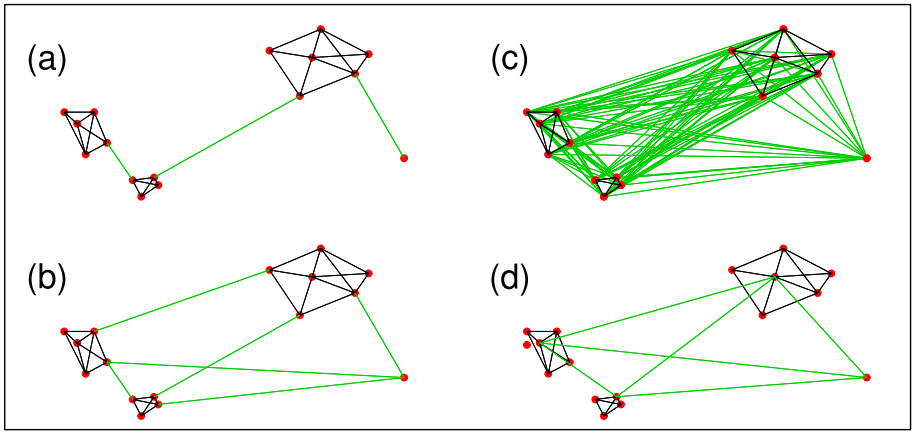


Figura 4.1

Ejemplo que ilustra los diferentes tipos de conexión usados por PKNNG, las líneas color negro corresponden a las aristas del grafo k-nn, mientras que las verdes son el resultados de las aristas agregadas por el tipo de conexión a) MinSpan. b) AllSubGraph. c) AllEdges. d) Medois. Basado en diagrama extraído de [21].

MinSpan Se adiciona el mínimo numero de arcos de longitud mínima hasta que se logra conectar el grafo.

AllSubGraph conecta cada componente a todas las restantes por el mínimo camino.

AllEdges Se conecta cada punto al resto de los puntos que no pertenecen al mismo subgrafo con caminos penalizados.

Medoids Se calcula el medoide[4] del sub-grafo y luego se conectan todos los medoides entre si.

En la figura 4.1 se ejemplifican los cuatro tipos de conexionado.

[4]Un medoide es un objeto del conjunto de datos que actúa como el mejor representante del centro del grupo.

Penalización

Para la conexión de los arcos adicionales del grafo se usa un peso que se calcula como la distancia d_{ij} entre los vértices v_i y v_j involucrados, multiplicada por una penalización que es función de la distancia d_{ij} y un factor de escala μ

$$p_{i,j} = d_{ij} f(d_{ij}, \mu) \tag{4.0.1}$$

Los tipos de penalizaciones pueden ser *a*) sin penalización, *b*) exponencial, *c*) lineal o *d*) polinómica. Para la penalización exponencial se tiene que $f(d_{ij}, \mu) = e^{\left(\frac{d_{ij}}{\mu}\right)}$.

El cálculo de distancias con `pknng` tiene parámetros que permiten manipular *a*) la cantidad de vecinos a usar para la construcción del grafo, *b*) la métrica de base (euclídea, coeficiente de correlación), *c*) el tipo de penalización, *d*) el parámetro μ para la penalización, *e*) tipo de esquema de conexión de los subgrafos.

En [22] se hace un estudio detallado sobre la variación de los parámetros que controlan cómo trabaja el algoritmo y se analizan los resultados obtenidos luego de aplicar distintos algoritmos de clustering a las distancias calculadas.

En lo que resta del capítulo usaré la métrica con algunos dataset para ilustrar el comportamiento de la misma, para esto los parámetros estarán fijos. A menos que se indique lo contrario, usaré la métrica euclídea como métrica base, el tipo de penalización exponencial y el conexionado tipo MinSpan. Para la penalización exponencial el valor de μ es invariante frente a k, esto es, μ asume el valor del tercer cuartil de las distancias al primer vecino de los datos.

Comparación de la distancia euclídea y pknng

En la Figura 4.2 se pueden observar para distintos conjuntos de datos, los histogramas de las distancias euclídeas (última columna) y los histogramas de distancias calculadas con `pknng` usando 3 y 9 vecinos, (columnas 2 y 3 respectivamente).

Los datos de las primeras tres filas tienen en común que representan un único cluster. Los histogramas de las distancias calculadas con `pknng` se ven muy diferentes entre sí para estos conjuntos de datos, incluso para un mismo conjunto de datos los histogramas varían al cambiar el parámetro k. Esto se debe a que las distancias calculadas con `pknng` son sensibles al valor de k, pero también al nivel de conectividad de los puntos.

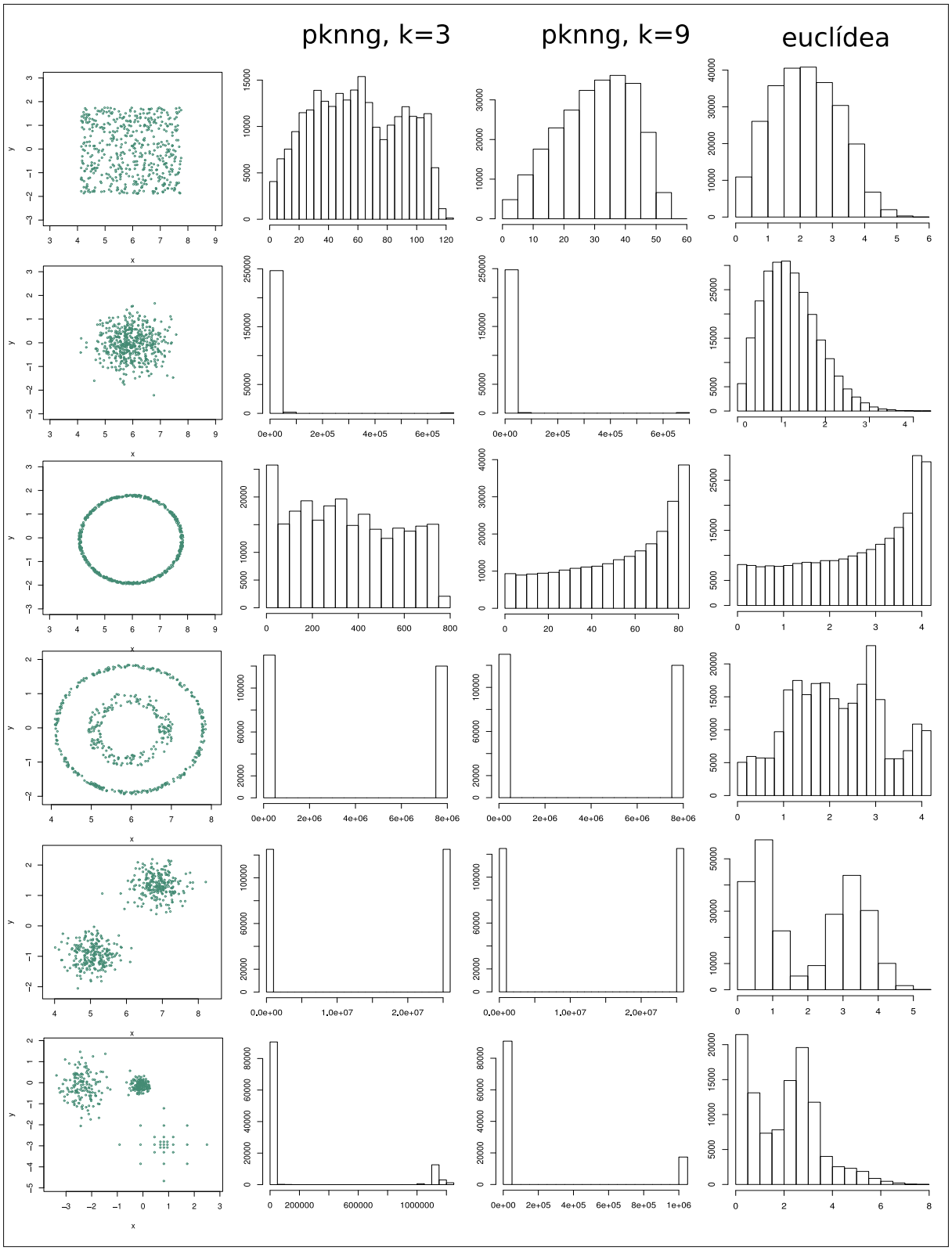


Figura 4.2

Ejemplos de distintos conjuntos de datos y los histogramas de las distancias calculadas con pknng usando 3 y 9 vecinos, (columnas 2 y 3 respectivamente) y los histogramas de las distancias euclídeas (última columna).

Como mencioné antes, la métrica `pknng` construye un grafo, cuando el grafo queda desconectado se agregan aristas pesadas para volverlo conexo. En las tres últimas filas de la figura 4.2 se puede observar el efecto de la penalización. Esto es, las distancias entre los grupos que habían quedado desconectados aumentan considerablemente haciendo más evidente la presencia de grupos.

Relación entre el parámetro k y la penalización

La penalización tiene el efecto de hacer que los objetos que están "lejos" entre sí se alejen aún más, mientras que los objetos que están "cerca" sigan estando cerca. Lo cual es un aspecto clave en la métrica.

A continuación ilustramos cómo varían las distancias gracias a la penalización y cómo disminuye el efecto de esta última al aumentar el valor de k. Para eso generamos un dataset con 300 datos en dos dimensiones cuyos atributos provienen de una distribución normal centrada en 0 y desvío standart 1, calculamos las distancias entre los pares de puntos con `pknng` variando k de dos en dos en el intervalo $[3, .., 13]$, luego repetimos el proceso pero usando `pknng` sin penalización. Tomamos las distancias máximas para cada caso y las dispusimos en las curvas de la Figura 4.3. En la gráfica se puede observar cómo se incrementan las distancias debido a la penalización. También se puede ver que a medida que k aumenta el efecto de la penalización es menor debido a que el grafo k-nn se encuentra cada vez más conectado, hasta que finalmente a partir de cierto valor de k la penalización ya no tiene ningún efecto y esto es cuando el grafo k-nn construido posee sólo una componente.

Efecto de la dimensionalidad

El término "curse of dimensionality" (maldición de la dimensionalidad) fue acuñado por Richard Bellman (1961) para describir la rapidez con que se incrementa la dificultad de los problemas a medida que el número de variables (dimensiones) se incrementa. En la actualidad se usa comúnmente en varios campos para referirse a los problemas o desafíos que surgen al trabajar en espacios de altas dimensiones.

Uno de los aspectos del llamado curse of dimensionality es la concentración de distancias, que denota la tendencia de la distancia entre pares de puntos en altas dimensiones

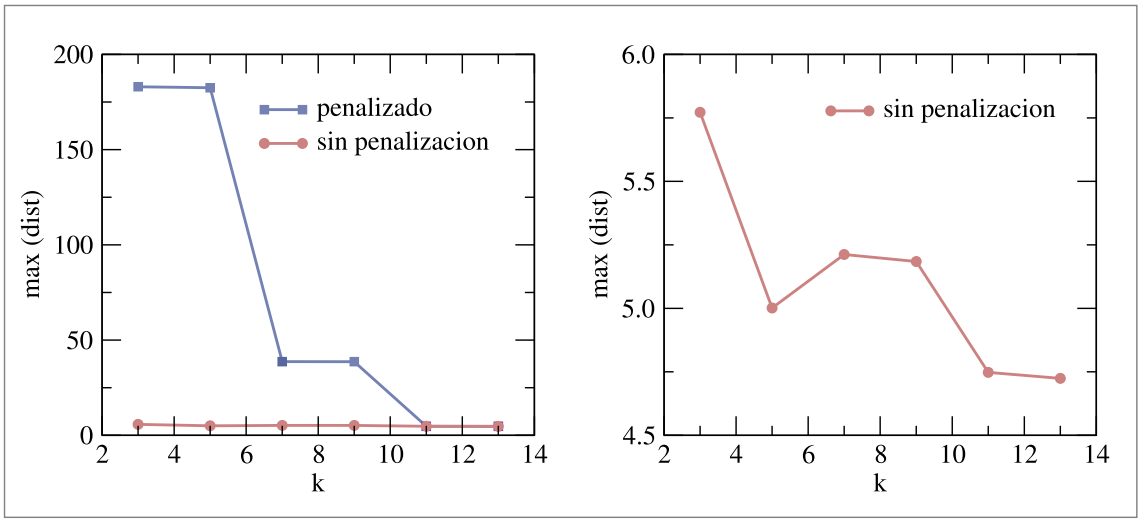


Figura 4.3

Curvas de la máxima distancia pknng obtenida para un mismo dataset en función del valor de k. La gráfica de la derecha es una ampliación de la curva de las máximas distancias pknng sin penalización que se ve en la gráfica de la izquierda.

a volverse casi iguales entre sí. En otras palabras la densidad de los datos disminuye y la distancia entre ellos aumenta y se vuelve más uniforme al aumentar la dimensión. Esto puede ocasionar que aún para valores bajos de k la penalización deje de producirse porque el grafo k-nn quedará conexo sin necesidad de agregar arcos penalizados, lo cual a su vez dificultará la tarea de distinguir clusters. Pero si los datos están en un manifold de menor dimensión entonces la métrica `pknng` tendrá más posibilidades de describir los datos correctamente.

5 Validación

El procedimiento para evaluar los resultados (partición) de un algoritmo de clustering de un modo cuantitativo y objetivo se conoce como **validación**.

Las medidas de validación suelen clasificarse en internas y externas. Las externas evalúan el resultado de un algoritmo de clustering basados en alguna estructura pre-especificada, como por ejemplo las clases correctas de los datos y en general se usan para elegir el mejor método de clustering para un dataset dado, ya que permite comparar métodos entre sí. Las internas solo se basan en la información proveniente de los datos, y se pueden usar para elegir el mejor método o para determinar el número óptimo de clusters para un conjunto de datos.

5.1. Medidas externas

Las medidas externas hacen uso de índices que comparan particiones, en el sentido de medir qué tan similares son entre sí un grupo de particiones distintas de un mismo conjunto de datos. Cuando el objetivo es la validación, en general el procedimiento consiste en la comparación de dos particiones: la evaluada (producida por el algoritmo de clustering a evaluar) y una de referencia (que puede ser la clases a la que pertenecen los datos, una partición creada a partir de otro algoritmo, etc). En [23] se presenta un repaso de medidas de similaridades entre clusters o particiones.

Una forma de comparar particiones es contar los pares de objetos que están clasificados de la misma manera en ambas particiones, ej. los objetos que están en el mismo cluster en ambas particiones. La llamada **matriz de contingencia** resume el conteo de pares de puntos en los cuales dos particiones están de acuerdo o en desacuerdo.

5.1.1. Matriz de contingencia

Sean $U = \{U_1, .., U_R\}$ y $V = \{V_1, .., V_C\}$ dos particiones sobre el mismo conjunto X de n datos. La matriz de contingencia resume el solapamiento entre los clusters de U y V (Tabla 5.1). El elemento n_{ij} de la matriz es el número de elementos que los conjuntos U_i

U/V	V_1	$\cdots$	V_C	Suma
U_1	n_{11}	$\cdots$	n_{1C}	a_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
U_R	n_{R1}	$\cdots$	n_{RC}	a_R
Suma	b_1	$\cdots$	b_C	$\Sigma_{ij} n_{ij} = n$

Tabla 5.1
Matriz de contingencia.

y V_j tienen en común, $n_{ij} = |U_i \cap V_j|$. Cualquier par de puntos x_i, x_j de un total de $\binom{n}{2}$ pares distintos de X caen en alguna de las siguientes cuatro categorías:

a) x_i, x_j están en el mismo cluster en U y en V

b) x_i, x_j están en distintos clusters en U y en V

c) x_i, x_j están en el mismo cluster en U y en distintos clusters en V

d) x_i, x_j están en el mismo cluster en V y en distintos clusters en U

La cantidad de pares que caen en cada categoría se puede calcular usando la matriz de contingencia. Sean a, b, c y d los valores correspondientes a las cantidades de pares de cada categoría. Los dos primeros valores son indicadores de acuerdo entre las particiones y los dos últimos, indicadores de desacuerdo.

5.1.2. Índice Rand

El estadístico *Rand* es un índice que fue propuesto por William M. Rand en 1971 y es usado para medir la similaridad entre dos particiones. Usando los valores de las cantidades de pares en las categorías antes enumeradas, el índice *Rand* se calcula como

$$Rand = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}} = \frac{[\binom{n}{2} + 2\Sigma_{ij}\binom{n_{ij}}{2} - [\Sigma_i \binom{a_i}{2} + \Sigma_j \binom{b_j}{2}]]}{\binom{n}{2}} \tag{5.1.1}$$

Los valores del índice están entre 0 y 1. Cuando las particiones comparadas son exactamente las mismas el valor de *Rand* es 1.

Existen algunos problemas conocidos con este índice, como el hecho de que el valor esperado del índice para dos particiones aleatorias no toma un valor constante (ej. cero). Además es sensible tanto al número de clusters como a la cantidad de elementos [24]. Con la intención de resolver las limitaciones del índice *Rand* se crearon varias medidas diferentes, una de ellas es el índice de Rand ajustado.

5.1.3. Índice Rand ajustado

El índice de Rand ajustado (ARI por sus siglas en inglés, también apodado *cRand*) fue propuesto por Hubert and Arabie en 1985 [1] y se define como:

$$ARI = \frac{\Sigma_{ij}\binom{n_{ij}}{2} - [\Sigma_i \binom{a_i}{2} \Sigma_j \binom{b_i}{2}]/\binom{n}{2}}{\frac{1}{2}[\Sigma_i \binom{a_i}{2} + \Sigma_j \binom{b_j}{2}] - [\Sigma_i \binom{a_i}{2} \Sigma_j \binom{b_i}{2}]/\binom{n}{2}} \tag{5.1.2}$$

Este índice fue propuesto para solucionar el problema de que el valor esperado de *Rand* para dos particiones aleatorias no toma un valor constante. El nuevo índice usa un ajuste que asume una distribución hipergeométrica generalizada como hipótesis nula, y se calcula como la diferencia normalizada entre el índice Rand y su valor esperado bajo la hipótesis nula $ARI = \frac{Rand - RandEsperado}{maxRand - RandEsperado}$.

El nuevo índice tiene el valor esperado cero para particiones independientes y el máximo valor 1 cuando ambas particiones son idénticas.

5.2. Medidas internas

Las medidas internas también hacen uso de índices que comparan particiones. Uno de sus objetivos es determinar la cantidad de clusters presentes en un conjunto de datos. Milligan y Cooper [25] presentan la comparación de 30 métodos para estimar la cantidad de clusters usando algoritmos jerárquicos, en [26] se presenta un estudio detallado de 11 métodos de validación interna y en [8] se comparan el comportamiento de varios de estos índices.

La mayoría de los índices internos se basan en la propiedad de que un buen algoritmo de clustering genera clusters con alta concentración interna y buena separación externa. La

suma de cuadrados intraclusters (Sum of squares within clusters, SSW) es comunmente usada para medir qué tan compacto es un cluster, mientras que la suma de cuadrados interclusters (Sum of squares Between - SSB) es una medida de separación. Dichos valores se calculan:

$$SSW(k) = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - m_i||^2 \tag{5.2.1}$$

$$SSB(k) = \sum_{i=1}^{k} |C_i| ||m_i - M||^2 \tag{5.2.2}$$

donde k es la cantidad de clusters, m_i corresponde al centro del cluster C_i, $|C_i|$ es el tamaño del cluster C_i y M es el centroide de los centros de los k clusters.

Las medidas de concentración y separación son criterios opuestos (la concentración aumenta con el número de clusters mientras que la separación disminuye) y se cumple para estas medidas que

$$SST = SSB(k) + SSW(k) \tag{5.2.3}$$

donde SST es la suma total de las distancias al cuadrado. En general los índices internos combinan ambos criterios, el índice Silhouette width presentado por Peter Rousseuw en 1987 en [27] es un ejemplo de combinaciones no lineales de compactación y separación. El Silhouette de un punto se define como:

$$S(i) = \frac{b(i) - a(i)}{max(b(i), a(i))} \tag{5.2.4}$$

donde $a(i)$ es la distancia media al objeto i de todos los objetos en el mismo grupo y $b(i)$ es la mínima distancia media del objeto i a todos los objetos de cada uno de los otros grupos. Se tiene que $S(i) \in [1-, 1]$ y el punto i está mejor clasificado cuanto más cercano a uno es el valor del $S(i)$. El índice Silhouette medio se define en base al Silhouette de un punto de la siguiente manera

$$S = \frac{1}{n} \sum_{i=1}^{k} S(i) \tag{5.2.5}$$

De forma general, si el valor devuelto por un índice de validación es inusualmente alto o bajo (según la definición del índice) para una partición dada, entonces se deduce que la partición encontrada es adecuada para los datos. El problema es definir cuándo un valor es inusualmente alto o bajo; la aproximación usada en este caso consiste en correr

el algortimo de clustering con distinta cantidad de clusters $k \in \{1, 2, .., kmax\}$ y elegir el k^* que optimiza el valor del índice. El valor óptimo para algunos índices es el mínimo mientras que para otros, como en el caso de Silhouette, es el máximo encontrado. Si se usa el máximo o el mínimo depende de la definición del índice.

Hay algunos índices como el SSW en los que hay que examinar la curva del valor del índice en función del número de clusters. El valor de SSW siempre decrece a medida que k aumenta pero si realmente hay k^* clusters en los datos entonces el decrecimiento será más lento a partir de $k^* + 1$, y ese cambio en la pendiente de la curva (fenómeno del codo, elbow) es lo que indica la cantidad de clusters.

La mayoría de los índices internos suponen tendencia al clustering (ver sección 5.3), es decir que esperan la presencia de más de un cluster en los datos por lo que no están definidos para $k = 1$. El método Gap statistic que explicaré a continuación fue presentado en [28] y está definido incluso para $k = 1$.

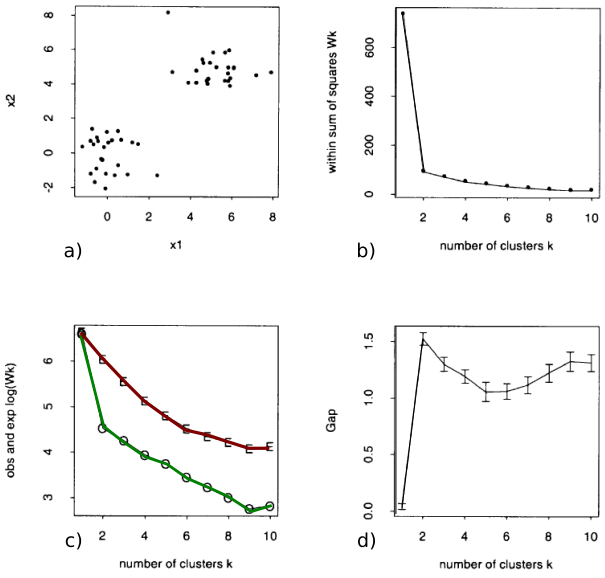


Figura 5.1

Figura extraída de [28]. Resultado de un ejemplo de 2 clusters, a) datos, b) suma de los cuadrados de las distancias intracluster W_k vs el número de clusters, c) funciones $log(W_k)$ (curva verde) y $E_n^*\{log(W_k)\}$ (curva roja), d) curva de gap: valores de Gap vs el número de clusters.

5.2.1. Gap Statistic

Gap statistic se usa para estimar el número de clusters comparando la curva del logaritmo de la dispersión intracluster (SSW) obtenida a partir de los datos, con el valor correspondiente esperado bajo una distribución uniforme generada sobre el rectángulo que contiene a los datos.

Intuitivamente utiliza el fenómeno del codo (elbow) en la curva de dispersión de la suma intracluster (ver Figura 5.1.b). Gap statistic está diseñado para ser aplicado a cualquier técnica de clustering y medida de distancia. Se define

$$Gap(k) = \frac{1}{B}\left(\sum_{b=1}^{B} log(W_{kb}^*)\right) - log(W_k) \tag{5.2.6}$$

donde $\frac{1}{B}\sum_b log(W_{kb}^*)$ indica el valor esperado de $log(W_k)$ calculado a partir de una distribución nula. Donde W_k se define

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r} \sum_{x_i, x_j \in C_r} d(x_i, x_j). \tag{5.2.7}$$

El valor óptimo k^* (número de clusters) para un conjunto dado es

$$k^* = argmin_k\{k | Gap(k) \geqq Gap(k+1) - s_{k+1}\}$$

donde

$$s_k = \sqrt{(1 + 1/B)} sd_k \tag{5.2.8}$$

y sd_k es el desvío standart de $log(W_{kb}^*)$. Notar que $SSW(k)$ (ecuación 5.2.1) y W_k son equivalentes.

Para aplicar este método es necesario elegir una apropiada distribución de referencia (distribución nula). En Tibshibani et al [28], los autores probaron que para una dimensión la distribución uniforme es la que mejor produce clusters espurios basados en el test de gap sobre todas las distribuciones unimodales. También probaron que para n dimensiones ($n > 1$) no existe generalización aplicable a la distribución de referencia. De todas formas, animados por el caso univariado, sugieren dos formas de generar la distribución de referencias para n dimensiones:

a) generar cada atributo de referencia como una distribución uniforme sobre los rangos de los datos observados.

b) generar cada atributo de referencia a partir de una distribución uniforme sobre los rangos de las componentes principales de los datos observados. Esto es, si X es la matriz $n \times p$ de datos, asumir que las columnas tienen media cero y calcular la descomposición singular $X = UDV^T$. Obtener $X_1 = XV$ y generar los atributos Z_1 sobre los rangos de X_1 como en la opción (a). Finalmente transformar $Z = Z_1V^T$ para obtener la distribución de referencia Z.

El método a) es más simple. El mérodo b) tiene en cuenta la forma de la distribución y hace el procedimiento invariante a rotaciones, siempre y cuando el algoritmo de clustering sea invariante.

La Figura 5.1 muestra un ejemplo de la aplicación del método gap statistic usando distancia euclídea y como algoritmo de clustering k-means. La cantidad de los clusters de los datos es 2 (Figura 5.1.a), en la Figura 5.1.c se observa el cambio en la pendiente para la curva de los datos observados para $k = 2$, mientras que la curva de la nula tiene un decrecimiento constante. La curva de Gap que se calcula como la diferencia de ambas curvas tiene un máximo para $k = 2$ (Figura 5.1.d) y de acuerdo al criterio de gap statistic hay dos cluster en los datos.

Extensión de Gap Statistic para clusters con formas arbitrarias

En [29] se presenta una extensión de Gap Statistic para encontrar el número de clusters presentes en datos con formas arbitrarias. Para eso define la medida de distancia denominada MED (Maximum Edge Distance) que se calcula a partir de un árbol de expansión mínima (MST) formado sobre el grafo $G(V, E)$ construido a partir de los datos, a saber, cada dato x_i representa un vértice $v_i \in V$ del grafo y cada arista e_{ik} se corresponde con la distancia euclídea entre los puntos x_i, x_k. Luego a partir de $G(V, E)$ se construye un MST y la distancia entre los puntos x_i y x_k corresponde al valor de la máxima arista en el camino entre v_i y v_k sobre el árbol construido. Esto es:

$$d^{MED}(x_i, x_k) = d^{MED}(v_i, v_k) = d_{ij}^{MED} = max(\{e \in P_{ik}\}) \tag{5.2.9}$$

donde P_{ik} es el conjunto de aristas que conforman el camino del vértice v_i al v_k.

Luego se toma como medida de distancia para el cálculo de W_k el cuadrado de las distancias MED, y los autores denominan a este valor IC-av que se define:

$$\text{IC-av}_k = \sum_{r=1}^{k} \frac{1}{2n_r} \sum_{x_i, x_j \in C_r} d^{MED}(x_i, x_k)^2 \tag{5.2.10}$$

Por lo que el gap para IC-av puede ser reescrito de la siguiente manera:

$$G_{\text{IC-av}}(k) = \frac{1}{B} \sum_{b=1}^{B} log(IC - av_{kb}^{*}) - log(IC - av_k) \tag{5.2.11}$$

5.3. Tendencia al clustering

Tendencia al clustering (cluster tendency; cluster assessment tendency) se refiere al problema de decidir cuándo los datos exhiben una predisposición a poseer grupos naturales, sin necesariamente identificar a los grupos en si [1]. Si es posible, antes de usar un algoritmo de clustering debe evaluarse la tendencia al clustering para evitar aplicar técnicas de clustering a datos sin estructuras.

Los algoritmos de clustering tienden a crear clusters ya sea que los datos posean clusters o no. Muchos métodos de análisis de cluster tendency se basan en el concepto de aleatoriedad y tratan de determinar si los datos provienen de un mismo proceso aleatorio. En algunas técnicas el problema de testear la tendencia al clustering se asimila al problema de testear la aleatoriedad espacial, es decir la aleatoriedad en las posiciones que ocupan los distintos puntos.

En [1] se recopilan varios test de aleatoriedad (la mayoría derivados de aplicaciones en astronomía y geografía donde los datos son bi-dimensionales) y se muestra cómo algunos de dichos test pueden ser extendidos a d-dimensiones, incluido el presentado en [30].

Existen otros enfoques menos formales que hacen uso de representaciones gráficas de la matriz de distancias como el sistema VAT (visual assessment tendency) [31], y posibles automatizaciones al mismo [32], que además de detectar si hay clusters presentes en los datos también pueden determinar la cantidad.

5.3.1. Test de gap como test de tendencia al clustering

En el marco del trabajo de Tibshibani et al. [28] se asume un modelo nulo de una sola componente (un sólo cluster), y se rechaza en favor de un modelo de $k-$componentes ($k > 1$) si hay evidencia suficiente para algún k, teniendo en cuenta simultáneamente todos los $k > 1$.

Es posible definir un test de cluster tendency basado en gap statistic, basta con verificar que los datos cumplen la condición

$$Gap(1) >= Gap(2) - s_2 \tag{5.3.1}$$

para asegurar que no hay clusters o existe un único cluster presente en los datos.

6 Dhclus: algoritmo de clustering jerárquico divisivo

Los métodos de spectral clustering solucionan una amplia variedad de problemas de clustering. Debido a esto se comenzaron a desarrollar un conjunto de mejoras a las formulaciones originales de Shi et al. [20] y Ng et al. [33]. En los trabajos de Filippone et al. [34] y Nascimento et al. [35] se pueden encontrar recopilaciones donde se discuten diferentes implementaciones del método original, en las cuales se introducen mejoras. Sin embargo, en estos trabajos falta tratar algunos problemas fundamentales que presenta el método: la **determinación del factor de escala** y la **cantidad de componentes** o clusters presentes en los datos, sobre todo cuando hay clusters a diferentes escalas en un mismo problema.

Los autores Zelnik-Manor y Perona [36] utilizan un método basado en densidades para resolver el problema de escala no uniforme. Nadler y Galun [37] también reconocen este problema y proponen un método basado en una regla ad-hoc para solucionarlo, a partir de la cual determinan la cantidad de componentes. Azran y Ghahramani [38] discuten el problema de la cantidad de componentes y utilizan un método de árbol para explorar una solución. Las soluciones obtenidas por estos autores son satisfactorias únicamente en un conjunto de ejemplos artificiales y reales reducidos, por lo cual pueden ser consideradas como una

Figura 6.1

Figura extraía de [1], ilustra el concepto de que el número de clusters depende de la escala. A gran escala, en forma global, se perciben cuatro clusters, pero si se inspecciona localmente cada cluster, se perciben doce.

solución parcial para el problema de la **escala no uniforme** y de la **determinación de la cantidad de componentes**.

El número de clusters en un conjunto de datos depende de la resolución con la que se ven los datos. Por ejemplo, en la figura 6.1 a una escala global se perciben cuatro clusters, pero si se inspecciona cada cluster localmente se pueden definir doce grupos. Por lo tanto, mirar los datos en múltiples escalas ayuda a analizar su estructura [3].

En este capítulo presentaremos el desarrollo de un algoritmo de clustering que hemos denominado `dhclus` (por su descripción en inglés "Divisive hierarchical clustering") el cual resuelve problemas de clusters con formas arbitrarias incluso cuando se encuentran en diferentes escalas. Además el algoritmo selecciona de forma automática sus parámetros y estima el número de clusters.

Para detectar clusters con formas arbitrarias y múltiples escalas elegimos usar un algoritmo jerárquico divisivo que aplica recursivamente spectral clustering en cada nivel, permitiendo de ese modo recalcular la similitud entre los pares de puntos de manera local en cada subproblema. Para esto último el algoritmo selecciona los parámetros que afectan la construcción de la matriz de similitud en base a una medida de calidad de los clusters producidos.

Para determinar la cantidad de clusters propusimos usar un criterio de parada basado en el test de gap presentado en la sección 5.3, que determina la presencia o no de clusters en cada subproblema. De esa manera se detiene la ejecución del algoritmo cuando el subproblema analizado no presenta más clusters.

El algoritmo propuesto realiza los siguientes pasos:

1. Dado un conjunto de datos, construye la matriz de similitud S.
2. Crea una partición con dos clusters utilizando spectral clustering.
3. Evalúa la partición generada en el paso 2. En el caso de que se corresponda con un único cluster, rechaza la partición propuesta y devuelve todas las particiones encontradas hasta el momento, sino recursivamente reparticiona los clusters encontrados.

El resultado final es una jerarquía de particiones que puede ser representada como un árbol binario. En el último nivel se encuentra la partición propuesta por el algoritmo.

En lo que resta del capítulo explicaremos las partes que componen el algoritmo `dhclus`. Discutiremos la construcción de la matriz de similitud para alimentar al algoritmo de

spectral clustering, con especial interés en la determinación del factor de escala. Luego presentaremos el criterio de parada propuesto para `dhclus`, el pseudocódigo del algoritmo desarrollado, una explicación del funcionamiento del algoritmo a través de un ejemplo y el análisis de la complejidad temporal.

6.1. Construcción de la matriz de similitud

Los algoritmos de spectral clustering toman como entrada una matriz de similitud. Si los datos a clusterizar están representados por una matriz de este tipo es posible aplicar directamente spectral clustering. En caso contrario, primero es necesario transformar el conjunto de patrones o la matriz de distancias en una matriz de similitud. La manera de construirla constituye un problema en sí mismo. A continuación presentamos tres variantes en la forma de construir la matriz de similitud basadas en el kernel gaussiano.

6.1.1. Radial basis function

Dada una medida de distancias siempre es posible convertirla en una de similitud y viceversa. El kernel gaussiano (que está contenido dentro de las funciones de base radial[1] - RBF) transforma una matriz de distancias D en una de similaridad S cuyas componentes vienen dadas:

$$S_{ij} = e^{-\frac{||d_{ij}||^2}{2\sigma^2}} \tag{6.1.1}$$

donde $\sigma \in \mathbb{R}$ y d_{ij} es el elemento ij de la matriz de distancias D. Por lo general la distancia usada es la euclídea, pero es posible usar otras.

El parámetro σ determina qué tan rápido la similitud S_{ij} decae con la distancia entre los puntos i y j. Puede ser visto como un factor de escala, ya que define qué esta cerca y qué esta lejos en el grafo, en otras palabras define el tamaño del vecindad de cada nodo, determinando de esta manera la partición de los datos. Un valor de σ demasiado grande o demasiado pequeño puede ocasionar que el algoritmo devuelva particiones incorrectas. Por lo tanto, es necesario dedicar algo de tiempo a la selección de un σ apropiado.

[1] Las funciones de base radial, son funciones a valores reales cuyos valores dependen sólo de la distancia a algún punto c llamado centro, $\phi(x, c) = \phi||(x - c)||$.

6.1.1.1. Elección automática del factor de escala para RBF

El algoritmo buscará un valor de σ de forma tal que al aplicar spectral clustering se obtenga una "buena" partición de los datos. Para distintos valores de σ se obtienen distintos pesos en las aristas del grafo de similaridad y spectral clustering puede ser muy sensible a cambios en este grafo, produciendo particiones muy diferentes de un mismo dataset. Por ejemplo, las gráficas de la figura 6.2, en las columnas 3 y 4, corresponden a diferentes particiones de un mismo dataset al que se le aplicó spectral clustering usando distintos valores de σ para calcular su matriz de similitud. Como puede observarse no todas las particiones se corresponden con los clusters del dataset.

Surge entonces la cuestión de cómo determinar σ de modo que la partición de los datos sea correcta. En Ng et al. [33] los autores demuestran que si hay k clusters en los datos, entonces para el valor adecuado de σ las componentes de los autovectores del Laplaciano tenderán a formar k-clusters "apretados". A partir de ese resultado proponen buscar sobre distintos valores de σ y elegir aquel que produzca los clusters más concentrados en el espacio de los autovectores, porque de esa manera en el espacio original la partición propuesta tiene muchas posibilidades de ser correcta (si es que hay clusters).

Siguiendo esta idea `dhclus` buscará construir una matriz de similitud de modo tal que las componentes de los dos primeros autovectores formen clusters muy concentrados. Para esto nuestra primer idea fue usar la definición de la suma intracluster (SSW) de la Ecuación 5.2.1, que es una medida de qué tan compacto es un cluster. Pero teniendo en cuenta que este índice tiende a beneficiar a clusters balanceados [39, 40] preferimos usar el promedio de las distancias al centro del cluster que también es un índice de validación interno [41] definido como:

$$BH(k) = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{n_i} \sum_{x \in C_i} ||x - m_i||^2. \tag{6.1.2}$$

donde n_i es la cantidad de elementos en el cluster C_i, k es la cantidad de clusters m_i es el centroide del cluster C_i. Por lo tanto, el algoritmo buscará σ de modo tal que $BH(2)$ computado sobre los dos primeros autovectores sea mínimo.

Hasta el momento hemos discutido acerca del criterio de búsqueda de σ, sólo nos queda definir el rango dentro del cual el algoritmo buscará a σ y garantizar que el mismo contenga valores apropiados. La idea es usar para σ algún valor de la distribución de distancias para que el parámetro guarde cierta relación con los datos.

A partir de algunos experimentos (que no se incluyen en este informe) observamos que al trabajar con distancias euclídeas, valores apropiados de σ eran valores menores a la mediana de la distribución de distancias. Entonces un rango adecuado podría ser el conjunto formado por las distancias mayores a cero y menores o iguales a la mediana. Pero esto no siempre puede ser válido por lo que decidimos no restringir el rango de búsqueda y el mismo estará formado por todos los valores de la distribución de distancias.

Debido a que valores de σ muy cercanos entre sí producen particiones de los datos muy similares o la misma; la estrategia para determinar σ consiste en usar un conjunto pequeño de fractiles[2] del histograma de distancias. En este caso el conjunto de fractiles usados es $\{9/10, 8/10, 7/10, 6/10, 2^{-1}, 2^{-2}, 2^{-3}, \ldots, 2^{-\log_2 n+1}\}$. Una vez seleccionado el fractil f_i que produce el mejor σ, es posible refinar la búsqueda aplicando nuevamente el método anterior en un nuevo rango de fractiles en torno al valor hallado (ej , $\{f_{i-1}, \ldots, f_i, \ldots, f_{i+1}\}$). Este refinamiento se puede aplicar hasta que no haya cambios en las particiones. En este trabajo se aplicó sólo un nivel de refinamiento, y en general las particiones raramente cambiaron luego de aplicar el refinamiento.

6.1.2. RBF-PKNNG

Ahora vamos a analizar el caso de usar la métrica `pknng` en lugar de la euclídea para calcular la matriz de distancias a la que se aplica el kernel gaussiano. De acuerdo a lo visto en el capítulo 4 el cálculo de las distancias con `pknng` tiene una serie de parámetros que determinan su forma de trabajo. Dichos parámetros estarán fijos: la métrica base será la euclídea, el tipo de penalización exponencial y el tipo de conexionado será el denominado MinSpan. El parámetro que controla la cantidad de vecinos para construir el grafo de k-vecinos, k, será el único parámetro de entrada el cual será ajustado automáticamente.

6.1.2.1. Elección automática del número de vecinos para pknng

Una vez aplicado `pknng` a los datos se obtiene una matriz de distancias D que se usará con el kernel gaussiano para producir la matriz de similitud S. La estrategia y el rango

[2]En una distribución de frecuencias, cierta cantidad de los datos cae en un fractil o por debajo de éste. La mediana, es el fractil que queda en medio, puesto que la mitad de los datos está por debajo de este valor.

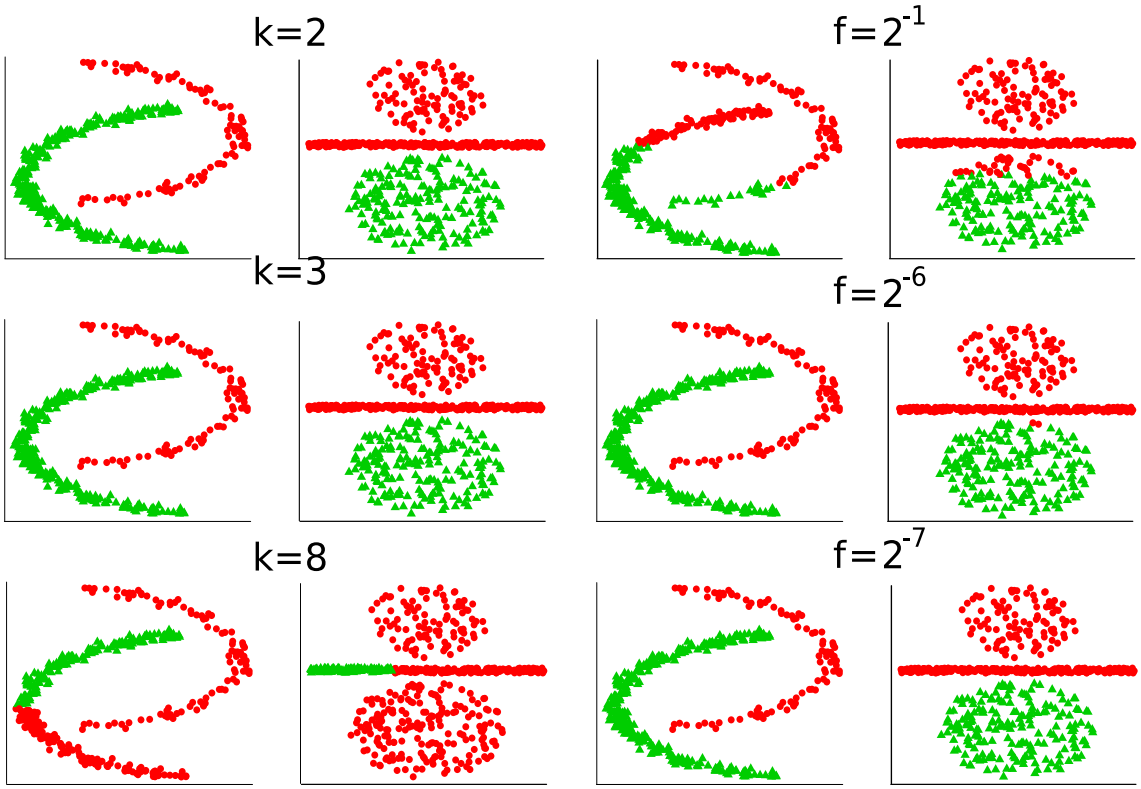


Figura 6.2

Particiones obtenidas con Local Scaling y RBF al variar los parámetros de escala del algoritmo. De izquierda a derecha, las primeras dos columnas corresponden a Local Scaling (datasets D1 y D2, respectivamente) para k en $\{2, 3, 8\}$*, las dos últimas columnas corresponden a RBF para valores de fractiles f* $\{2^{-1}, 2^{-6}, 2^{-7}\}$.

de búsqueda para σ con la métrica `pknng` serán los mismos que para la métrica euclídea, comentada en la sección 6.1.1.1.

El parámetro k para `pknng` es determinado en cada llamada recursiva de `dhclus` de acuerdo al mismo índice de calidad usado para σ, por lo que en este caso se busca la combinación (k,σ) que minimice el índice de validación propuesto (Ec. 6.1.2).

El parámetro k es responsable del grado de conectividad del grafo k-nn que se usa de base para calcular las distancias pesadas con `pknng`. A medida que k aumenta los grafos se vuelven más conexos y los valores de distancias producidas disminuyen debido a la disminución de aristas penalizadas en el grafo. Usar un valor de k demasiado grande equivale a usar `pknng` sin penalización. En el análisis de la métrica que se presenta en [22] se mostró que al clusterizar diferentes datasets variando k en un amplio rango, los mejores resultados se obtienen para k entre 2 y 7. Además en experimentos usando spectral clus-

tering y fijando manualmente el valor del parámetro k dentro del rango 2 a 13 obtuvimos los mejores resultados para $2 \leq k \leq 7$ (resultados no incluidos en este informe). Debido a esto el rango de búsqueda para k será de 2 a 9.

6.1.3. Local Scaling

En [36] los autores Zelnik-Manor y Perona proponen usar en vez de un único parámetro σ para el cálculo de la matriz de similitud, un parámetro σ_i para cada punto i, quedando el elemento S_{ij} de la matriz de similaridad definido como:

$$S_{ij} = e^{-\frac{||d_{ij}||^2}{\sigma_i \sigma_j}} \tag{6.1.3}$$

donde $\sigma_i = d_{ik}$, y d_{ik} es la distancia entre el elemento i y su k-ésimo vecino. La distancia usada es la distancia euclídea. Los autores aclaran que la elección de k es independiente de la escala y que es una función de la dimensión intrínseca de los datos. Pero no indican qué criterio usar para elegir k, aunque expresan que en sus experimentos usaron el valor $k = 7$ con el que obtuvieron buenos resultados inclusive en altas dimensiones. Por lo tanto, para asegurar que el valor de k adecuado pertenezca al rango de búsqueda, elegimos como rango los valores comprendidos entre 2 y 10. El criterio usado para elegir k será el mismo que en los casos anteriores.

6.1.4. Ejemplo

Para ilustrar el nivel de acierto del índice de validación en la determinación de σ para RBF y k para Local Scaling aplicamos el criterio propuesto una vez sobre 2 conjuntos de datos artificiales.

- **D1**: Dos clusters no linealmente separables y con distintas densidades, con (310, 100) observaciones.
- **D2**: Tres clusters con poca separación, con distintas formas y densidades con (100,300,200) observaciones. En este caso como sólo se buscarán dos cluster, una partición correcta será aquella que agrupe a dos de los clusters en uno sólo y que ninguno de los tres clusters quede dividido.

Para ambos datasets calculamos el índice de validación para las distintas particiones producidas al variar los parámetros σ en el caso de RBF y k para Local Scaling sobre los rangos propuestos. En la tabla 6.1 se encuentran los valores del índice $BH(2)$ para RBF según el correspondiente fractil para ambos dataset, y en la tabla 6.2 los correspondientes valores del índice para los distintos valores de k con Local Scaling.

	6/10	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
D1	1.04E-03	1.02E-03	8.82E-04	7.91E-04	1.14E-03	1.86E-04	**1.14E-06**	**1.30E-15**
D2	9.72E-04	9.67E-04	9.33E-04	7.00E-04	4.10E-04	2.27.67E-04	2.07.67E-04	**1.40E-05**

Tabla 6.1

Valores de $BH(2)$ para las particiones producidas con RBF según el fractil de la distribución de distancia usados para los dataset D1 y D2. Sólo se incluyeron un subconjunto de los fractiles usados por cuestiones de espacio, pero en los valores incluidos se encuentra el seleccionado por el criterio.

	2	3	4	5	6	7	8	9
D1	**3.6E-06**	**1.74E-05**	**6.40E-04**	**1.17E-03**	1.38E-03	1.33E-03	1.18E-03	1.03E-03
D2	**2.63E-10**	**5.63E-06**	**7.41E-05**	**5.96E-05**	**1.22E-04**	**2.21E-04**	4.99E-04	8.23E-04

Tabla 6.2

Valores de $BH(2)$ para las particiones producidas con Local Scaling según el valor de k para los dataset D1 y D2.

En ambas tablas están resaltados los valores del índice que corresponden a particiones correctas de los datos. Hay más de un valor de σ y k con los cuales el algoritmo produce particiones adecuadas y el criterio de selección de parámetros en cada caso tiene éxito ya que el mínimo valor se encuentra entre los valores resaltados.

En la Figura 6.2 se presentan algunas de las particiones de los datasets obtenidas durante la búsqueda de los parámetros. De izquierda a derecha, las primeras dos columnas corresponden a Local scaling (D1 y D2, respectivamente) para k en $\{2, 3, 8\}$ y de la misma manera, las dos últimas columnas corresponden a RBF en este caso los fractiles f usados $\{2^{-1}, 2^{-6}, 2^{-7}\}$.

6.2. Spectral Clustering

Una vez obtenida la matriz de similitud S se procede a crear el grafo de similaridad y computar la matriz W (adyacencia pesada). Debido a que el algoritmo usa el modelo de grafo completamente conectado (que conecta todos los vértices entre sí con peso s_{ij} - ver sección 3.2.3.1) resulta que $W = S$ y entonces sólo resta aplicar spectral clustering con las siguientes particularidades: la cantidad de autovectores es 2 (ya que en cada iteración se buscan sólo dos clusters), la definición de Laplaciano utilizado es $L = I - D^{-1}W$ del que se usan los autovectores correspondientes a los primeros dos mayores autovalores de $D^{-1}W$ (ya que L y $D^{-1}W$ tienen los mismos autovectores y los autovalores cambian de λ_i a $1 - \lambda_i$, por lo que tomar los autovectores de L correspondientes a los menores autovalores es equivalente a tomar de los autovectores de $I - L$ correspondiente a los mayores autovalores), y el algoritmo de clustering usado para separar las componentes de los autovectores es PAM.

En este paso también se eliminan outliers, en general un outlier se define como un punto que cae muy lejos de la media muestral del cluster más cercano pero en spectral clustering es un punto con muy baja conectividad. Si luego de la descomposición spectral se detectan outliers en el espacio de los autovectores entonces se marcan como outliers los datos correspondientes y se eliminan de la entrada antes de ir al paso siguiente.

6.3. Criterio de parada

Luego de clasificar los datos con spectral clustering el algoritmo decidirá en base a la partición producida si los datos debían dividirse o no. Se responden dos preguntas simultáneamente, una es si los datos presentan clusters y la otra si las particiones propuestas representan clusters válidos, de lo contrario el algoritmo rechazará la partición y dejará de dividir los datos.

Como criterio de parada el algoritmo usa el test de gap, explicado en la sección 5.3.1. Consideramos cuatro variantes, la primera consiste en usar el test de gap con la métrica `pknng` para calcular las sumas W_k y $E(W_k)$ (Ec. 5.2.7), la segunda usa la métrica MED para calcular las sumas IC-av y E(IC-av) (ver sección 5.2.1). Denominaremos al primer criterio test de gap-PKNNG y al segundo test de gap-IC. Las otras dos variantes consisten en las dos ya mencionadas salvo que se reemplazan las sumas de W_k y IC-av por sus

respectivos promedios (W_k/n_k y IC-av/n_k), la versión resultante se conoce como gap pesado (weigthed gap) presentado en [40] y en [42].

Para todos los criterios la distribución de referencia se genera sobre un box alineado con las componentes principales de los datos originales tal y como se explicó en la sección 5.2.1.*b*. Luego se clusterizan B distribuciones de referencia y se toma la media de la dispersión intracluster W^*_{kb}, $b = 1, \ldots, B$, $k = 1, 2$. Para agrupar las B repeticiones se usa spectral clustering con la matriz de similitud creada a partir de la matriz de distancias de cada distribución nula a la que se le aplica kernel gaussiano con los mismos parámetros que los usados para crear la matriz de similitud de los datos originales.

6.4. Pseudocódigo para dhclus

El algoritmo final compuesto por las partes explicadas en las secciones precedentes, es un algoritmo capaz de descubrir clusters con formas arbitrarias y en altas dimensiones. Además puede determinar automáticamente la cantidad de componentes o clusters presentes en los datos incluso cuando hay clusters a diferentes escalas. El pseudocódigo para el algoritmo `dhclus` es el que sigue.

Algoritmo 2 *dhclus*

Entrada: Datos

Salida: Clusters encontrados

1: Construir la matriz de similitud S
2: Sean cl_1 y cl_2 los clusters obtenidos al aplicar spectral clustering a S.
3: **if** $Gap(1) < Gap(2) - sd2$ **then**
4: Aplicar `dhclus` a cada partición de los datos determinadas por cl_1 y cl_2.
5: **else**
6: **return** Clases encontradas
7: **end if**

Para mostrar cómo trabaja el algoritmo, usaremos el dataset representado en la Figura 6.3.a. A partir de los datos, el algoritmo construye una matriz de similitud S con algunas de las métricas explicadas en la sección 6.1, luego aplica spectral clustring a S y propone la partición con 2 clusters (Fig. 6.3.c). Luego calcula el test de gap sobre la partición propuesta (Fig 6.3.b), el cual determina la existencia de 2 clusters. Como el test resultó

positivo el procedimiento vuelve a aplicarse a cada una de las particiones obtenidas, que son los clusters verde y rojo de la Figura 6.3.c. Como resultado del proceso se obtiene una partición final (Figura 6.3.g). Durante la ejecución de `dhclus` se aplicó spectral clustering y el test de gap para decidir la existencia de clusters a cada uno de los clusters que componen la partición final del dataset pero el proceso se detuvo en ese punto, dado que como muestra la Figura 6.3.e. el test de gap indicó la presencia de un único cluster en cada caso.

Al finalizar el algoritmo retorna las jerarquía de particiones, nivel a nivel, donde el último nivel es la partición final propuesta y, dado que en varios contextos puede ser útil, devuelve también la matriz de similitud final creada de forma conjunta entre las distintas ejecuciones del algoritmo (Figura 6.3.f). Inicialmente el algoritmo crea una matriz de similitud con la similitud de todos los pares de patrones, luego de partir los datos en cada nivel se generan las matrices de similitudes locales, los valores de la matriz de similitud original son actualizados con los valores de las nuevas submatrices que corresponden a particiones aceptadas.

6.5. Análisis de la complejidad

La complejidad de un algoritmo se refiere a la cantidad de tiempo y espacio necesarios para ejecutarlo. El tiempo necesario es una función que depende del tamaño n de la entrada.

Para una entrada de tamaño $n > 1$, en la primer ejecución `dhclus` debe

1. calcular la similitud y aplicar spectral clustering ajustando el parámetro de escala
2. calcular el test de gap

Calcular la similitud requiere X ejecuciones de spectral clustering, X creaciones de matrices de similitud más algunas comparaciones, el paso 2 requiere 1 ejecución de spectral clustering, y calcular el test de gap B ejecuciones de spectral clustering y B creaciones de matrices de similitud. La creación de las matrices de similitud tiene complejidad n^2 para RBF. Entonces la complejidad del algoritmo está dominada por la complejidad de spectral clustering, que a su vez está dominado por la complejidad de la descomposición spectral que depende del algoritmo usado, pero en el peor de los casos es $O(n^3)$.

Sea $m = B+X$ entonces el tiempo de ejecución en el primer nivel del árbol del algoritmo es n^3m+n^2m. En cada llamada recursiva m no cambia y el tamaño de la entrada se divide en 2. En el peor caso, la entrada se divide en un conjunto con $n-1$ puntos y un solo punto en el otro, y continúa de esa manera en cada iteración, de modo que el árbol final tiene una altura de $n-1$. En dicho caso la complejidad será $m(\frac{n^4+2n^3+n^2}{4} + \frac{2n^3+3n^2+n}{6})$. En el caso intermedio la entada se divide aproximadamente en partes iguales y por simplicidad supongamos que n es potencia de 2. El tiempo total en el caso de tener que llegar hasta el final del árbol (altura $log_2 n + 1$) se aproxima a

$$\begin{aligned} T &= mn^3 + mn^2 \\ &+ 2\left(m(\frac{n}{2})^3 + m(\frac{n}{2})^2\right) \\ &+ 4\left(m(\frac{n}{4})^3 + m(\frac{n}{4})^2\right) \\ &+ \cdots + n(m+m) \end{aligned} \tag{6.5.1}$$

que es equivalente a

$$\sum_{i=0}^{\log_2 n} 2^i m\left((\frac{n}{2^i})^3 + (\frac{n}{2^i})^2\right) = \sum_{i=0}^{\log_2 n} \frac{1}{2^i} m\left(\frac{n^3}{2^i} + n^2\right) \quad =^3 \quad 4(log_2 n+1)n^3m+4(log_2 n+1)^2n^2m \tag{6.5.2}$$

Por lo tanto la versión serial del algoritmo es de orden cúbico respecto al tamaño de la entrada.

Existen métodos iterativos de orden lineal para el cálculo de la descomposición spectral, como el algoritmo de Arnoldi que es un análogo al método de Lanczos para matrices Hermíticas. Estos métodos podrían utilizarse para reducir la complejidad de `dhclus` asociada al cómputo de autovectores.

Además hay que tener en cuenta que si $m > n$ entonces el orden del algoritmo en el caso promedio es peor que $O(n^4)$. Por lo que también una forma de reducir los tiempos de cómputo es reemplazar el test de gap por un criterio de parada que insuma menos tiempo de cálculo. De todas maneras, en la implementación de gap para `dhclus` usamos paralelización siempre que era posible para reducir los tiempos efectivos de cómputo.

[3] Debido a que la serie geométrica $\sum_{i=0}^{\infty} \frac{1}{2^i}$ converge a 2.

Paquete DHclus

El código del algoritmo `dhclus` está escrito en R, un lenguaje y entorno open source para cálculo y gráfico estadístico [43].

Para el desarrollo de `dhclus` utilizamos las plataformas de desarrollo sourceforge y R-forge[4]. R-forge es una plataforma ofrecida desde el 2007 por el projecto R para el desarrollo de paquetes en R de forma colaborativa. Uno de los servicios que brinda está plataforma es la posibilidad de ofrecer el paquete tanto en formato tar.gz como en binario para ser instalado a través del comando `install.packages("nombre-del-paquete", repos="http://R-Forge.R-project.org")`, siempre que el paquete haya pasado los chequeos de construcción.

El paquete `dhclus` se encuentra disponible para descarga e instalación en la plataforma R-forge, así como los paquetes de los que depende y que no están disponibles directamente desde los cran[5] de R: `icav` y `pknng`.

También puede obtenerse el código fuente desde el repositorio svn: `svn checkout svn://r-forge.r-project.org/svnroot/dhclus/`

Una cuestión que queremos destacar de la implementación de `dhclus` es que en el más alto nivel es una función genérica:

```
dhclus(data, diss=FALSE,  FUNCluster=Get.clusters, FUNTest=test, ...)
```

Los parámetros `FUNCluster` y `FUNTest` permiten que el algoritmo reciba como argumentos funciones. Por defecto recibe las funciones `Get.cluster` y `test` que son las funciones que realizan el clustering y la validación usando gap respectivamente. Pero es posible pasarle otras funciones definidas por el usuario. Para más detalle se puede consultar el manual provisto por el paquete.

[4] `https://r-forge.r-project.org/`

[5] Cran, Comprehensive R Archive Network, es una red de servidores web y ftp sincronizados alrededor del mundo que proveen el código de R, tanto el sistema base como todos los paquetes provistos por la comunidad.

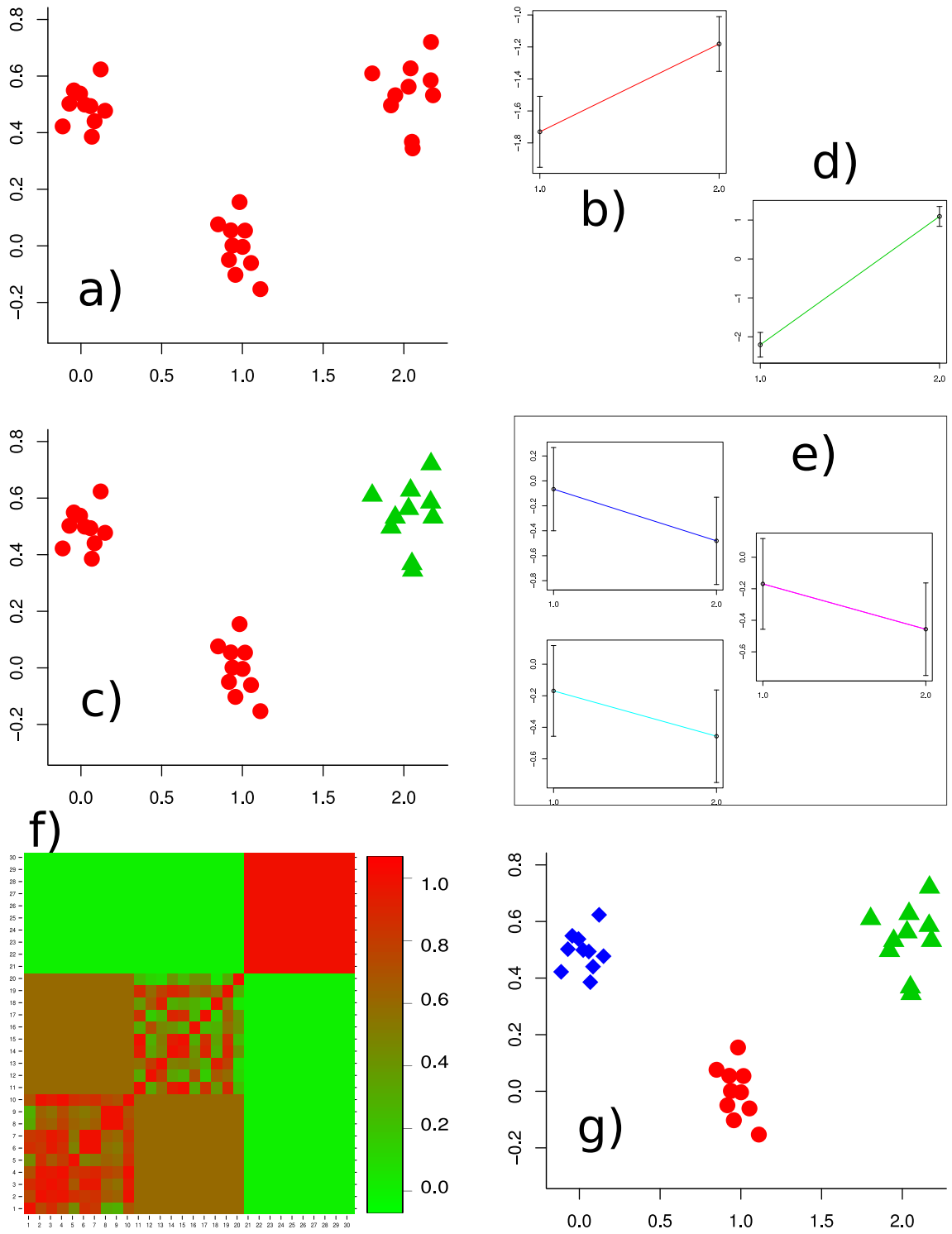


Figura 6.3

Dataset ejemplo. a) Particiones del mismo dataset en diferentes niveles del árbol luego de aplicar dhclus, c) nivel 1, g) nivel 2. Curvas de gap para las diferentes particiones propuestas, b) y d) para las particiones aceptadas e) para las particiones rechazadas. f) Matriz de similitud final de los datos creada de forma conjunta entre los distintos niveles del árbol.

7 Evaluación del algoritmo

En este capítulo presentamos los resultados de la ejecución del algoritmo sobre datasets artificiales y luego sobre datos de expresión de genes de dominio público. En cada caso describimos los datasets utilizados y presentamos: el detalle del experimento realizado, los resultados obtenidos y un análisis de dichos resultados.

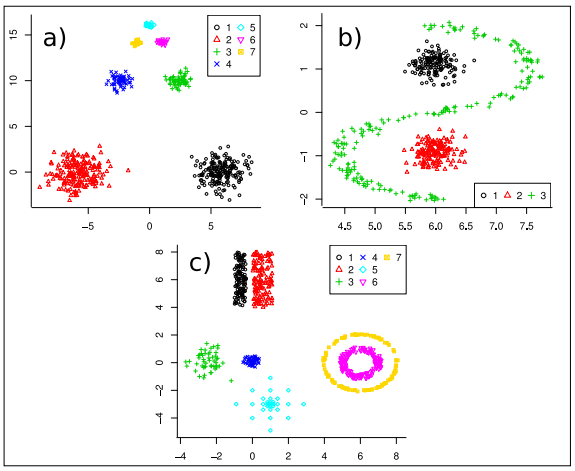


Figura 7.1

Dataset artificiales usados para evaluar el algoritmo dhclus. a) Dataset 1, b) Dataset 2, c) Dataset 3.

7.1. Evaluación sobre datasets artificiales

En esta etapa analizamos la capacidad del algoritmo de detectar clusters con formas arbitrarias, distintas densidades y escala no uniforme.

7.1.1. Descripción de los dataset artificiales

Se generaron 18 dataset con clusters de distintas formas, tamaños y densidades, de los cuales presentamos tres que generalizan los problemas analizados (Figura 7.1).

Dataset 1 : Siete Clusters (Figura 7.1.a) que están formados por variables distribuidas normalmente con 189 (nubes roja y negra), 63 (nubes verde y azul) y 21 (nubes celeste, fucsia y naranja) observaciones.

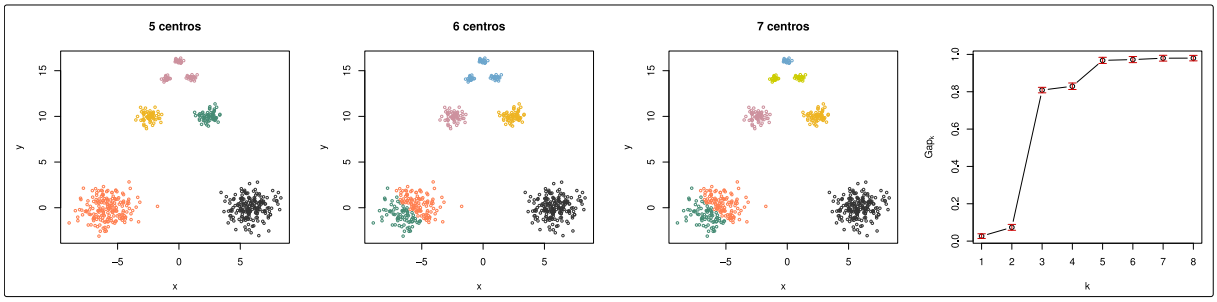


Figura 7.2

Divisiones producidas con PAM para 5, 6 y 7 centros sobre el dataset 1 y la curva gap para 1 a 8 centros. En este caso gap statistic detecta 5 clusters.

Dataset 2 : Tres clusters no linealmente separables (Figura 7.1.d), dos provienen de distribuciones normales con 150 datos (nubes negra y verde), y uno con 200 datos (cluster verde).

Dataset 3 : Siete clusters (Figura 7.1.c), dos con 120 observaciones (rectángulos negro y rojo) y distribuciones uniformes (U[0,0.65], U[0,4]) y (U[-1,-.65], U[4,8]), dos formados por variables normalmente distribuidas (nubes verde y azul) con 50 datos con centros en (-2.5,0) y en (0,0.1), un cluster con 21 datos distribuidos de manera regular (puntos celestes), y dos círculos concéntricos con 150 (círculo interior) y 100 (circunferencia exterior) observaciones con centro en (0,6) y radios ($0.8 \leq r_1 \leq 1.2$) y ($1.9 \leq r_2 \leq 2$) respectivamente.

El primer dataset (Figura 7.1.a) sirve para analizar la capacidad del algoritmo de detectar clusters a distintas escalas. Es un dataset que presenta clusters con formas simples pero la disposición y tamaño de los mismos hace que sea un problema difícil de resolver para algoritmos que minimizan la suma intracluster. Por ejemplo, el algoritmo PAM detecta incorrectamente los clusters cuando se lo utiliza con 7 centros que es la cantidad exacta de clusters presentes en los datos (Figura 7.2). El dataset 2 es un problema de separación no lineal, con clusters de distintas densidades y tamaños. El dataset 3 combina clusters de distintas densidades (nubes verde y negra, y puntos lilas), un problema de separación no lineal (dos círculos) y dos clusters elongados con poca separación entre ellos el cual también es un problema difícil para algoritmos de clustering que minimizan la suma intraclusters.

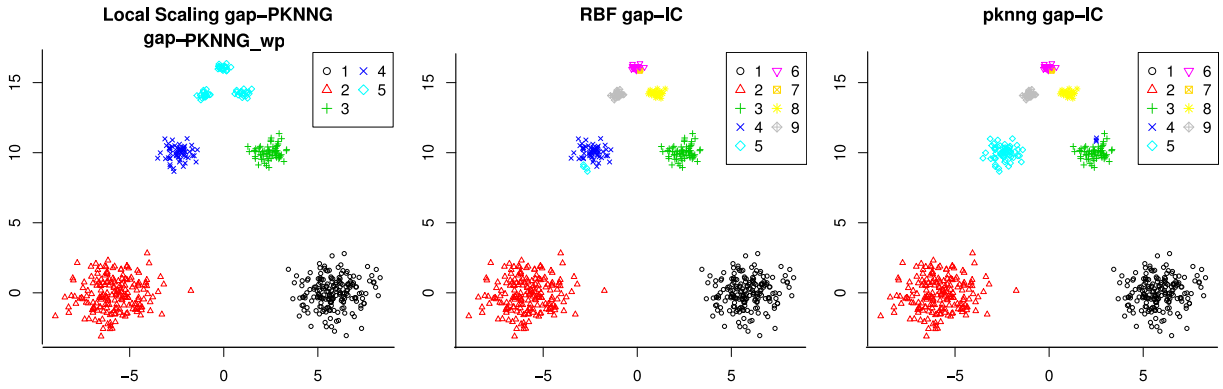


Figura 7.3

Resultados dataset1 con cRand < 1.

7.1.2. Descripción de experimentos sobre dataset artificiales

Para cada uno de los datasets se corrió el algoritmo con los cuatro criterios de parada (gap-PKNNG, gap-PKNNG_wp, gap-IC y gap-IC_wp) y con cada una de las formas de construir la matriz de similitud (pknng, RBF y Local Scaling). Para calcular los tests de gap se usaron 200 repeticiones de las correspondientes distribuciones de referencia en cada iteración.

7.1.3. Resultados

	gap-PKNNG			gap-IC			gap-PKNNG_wp			gap-IC_wp		
D1	1	1	0.97	1	1	1	1	1	0.97	1	0.99	1
D2	1	0.74	0.74	1	0.74	0.74	0.80	0.75	0.74	0.78	0.75	0.74
D3	0.74	0.77	0.69	0.85	0.89	0.89	0.76	0.99	0.69	0.95	1	0.89
	PKNNG	RBF	LS	PKNNG	RBF	LS	PKNNG	RBF	LS	PKNNG	RBF	LS

Tabla 7.1

Valores de cRand para las clases obtenidas con dhclus con las distintas combinaciones de formas de construir la similitud y los test de gap para cada uno de los dataset artificiales presentados.

Se calculó el índice *cRand* entre las particiones propuestas y las reales. Los valores del mismo se encuentran en la Tabla 7.1. Además, dada la baja dimensionalidad de los datos, se presenta los resultados de forma visual en las Figuras 7.3, 7.4 y 7.6.

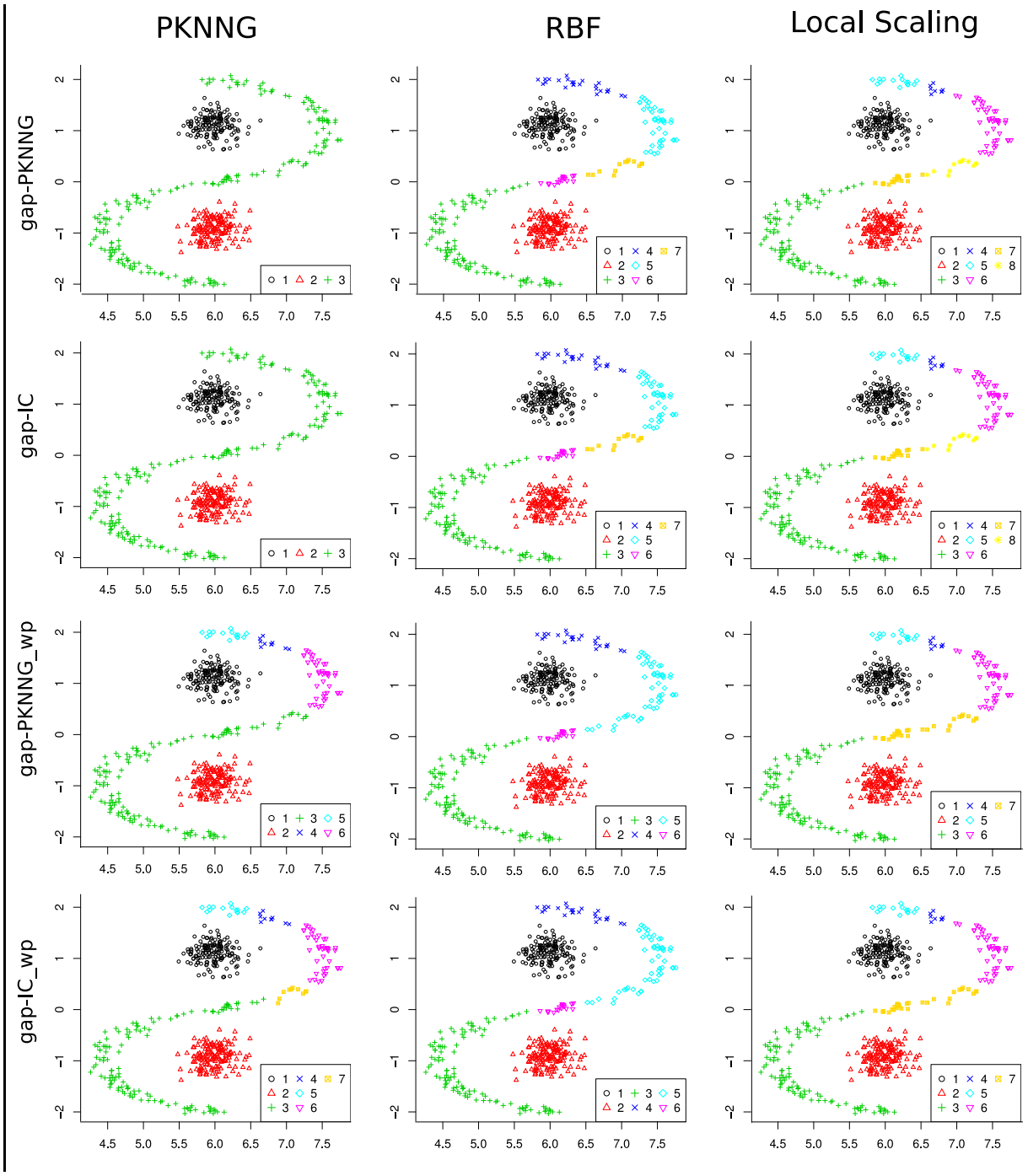


Figura 7.4

Particiones obtenidas con dhclus para el segundo dataset. Cada columna corresponde a una forma de calcular la similaridad, pknng (primer columna), RBF (segunda columna) y Local Scaling (tercer columna). Cada fila corresponde a un test de parada diferente gap-PKNNG, gap-IC, gap-PKNNG_wp y gap-IC_wp.

7.1.4. Análisis de resultados

Para el primer dataset con las distintas métricas el algoritmo encuentra la cantidad y calidad correcta de clusters, excepto en tres casos (los presentados en la Figura 7.3), un caso se corresponde con la similitud RBF con el test gap-IC_wp y dos casos correspondientes a Local Scaling con los test gap-PKNNG y gap-PKNNG_wp. En las ejecuciones en las que se usaron las versiones pesadas del test se acepto dividir alguna de las gaussianas.

La mayoría de los resultados para el primer dataset son satisfactorios (excepto en los casos mencionados). El algoritmo se comporta adecuadamente ante la presencia de clusters a distintas escalas. Esto lo atribuimos a la estrategia divisiva que posibilita recalcular la escala en cada llamada recursiva y permite la validación de las particiones propuestas a una escala local.

Para el segundo dataset en la mayoría de los casos el algoritmo encuentra una cantidad mayor de clusters. Analizando nivel a nivel las particiones producidas encontramos que cuando se usa similitud RBF-PKNNG o RBF las particiones iniciales son correctas, es decir, primero se separa una de las gaussianas del resto y luego la otra gaussiana del dataset mayor y luego sigue particionando el dataset mayor como se muestra en la Figura 7.5 columna izquierda. En esos casos es posible cortar el árbol generado en el segundo nivel y encontrar la partición adecuada. Mientras que con Local Scaling el dataset es partido en forma incorrecta desde el inicio. Esto se debe a que con Local Scaling no es posible bipartir el conjunto de datos de forma satisfactoria para ningún valor de k.

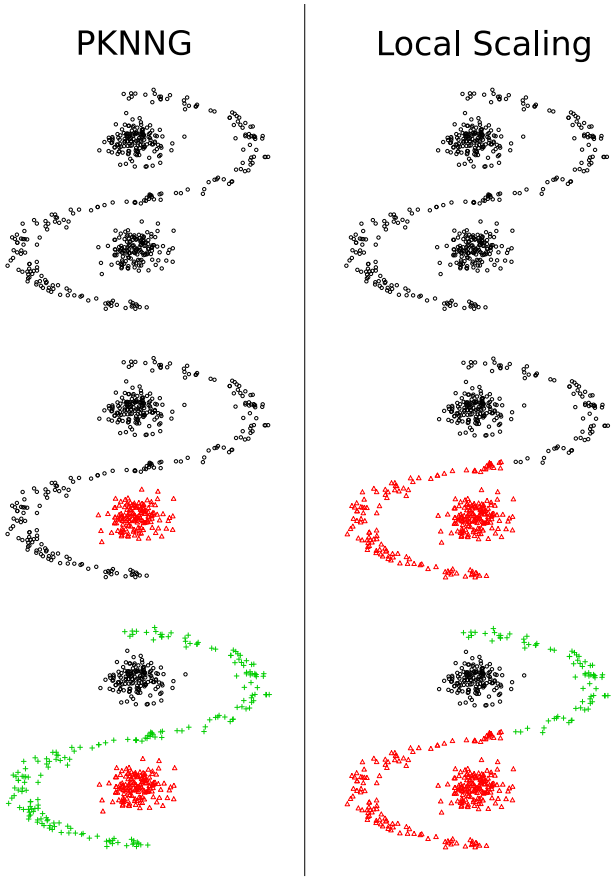


Figura 7.5

Primeras tres particiones en la jerarquía construida por dhclus para pknng y Local Scaling. Para Local Scaling la partición inicial es incorrecta.

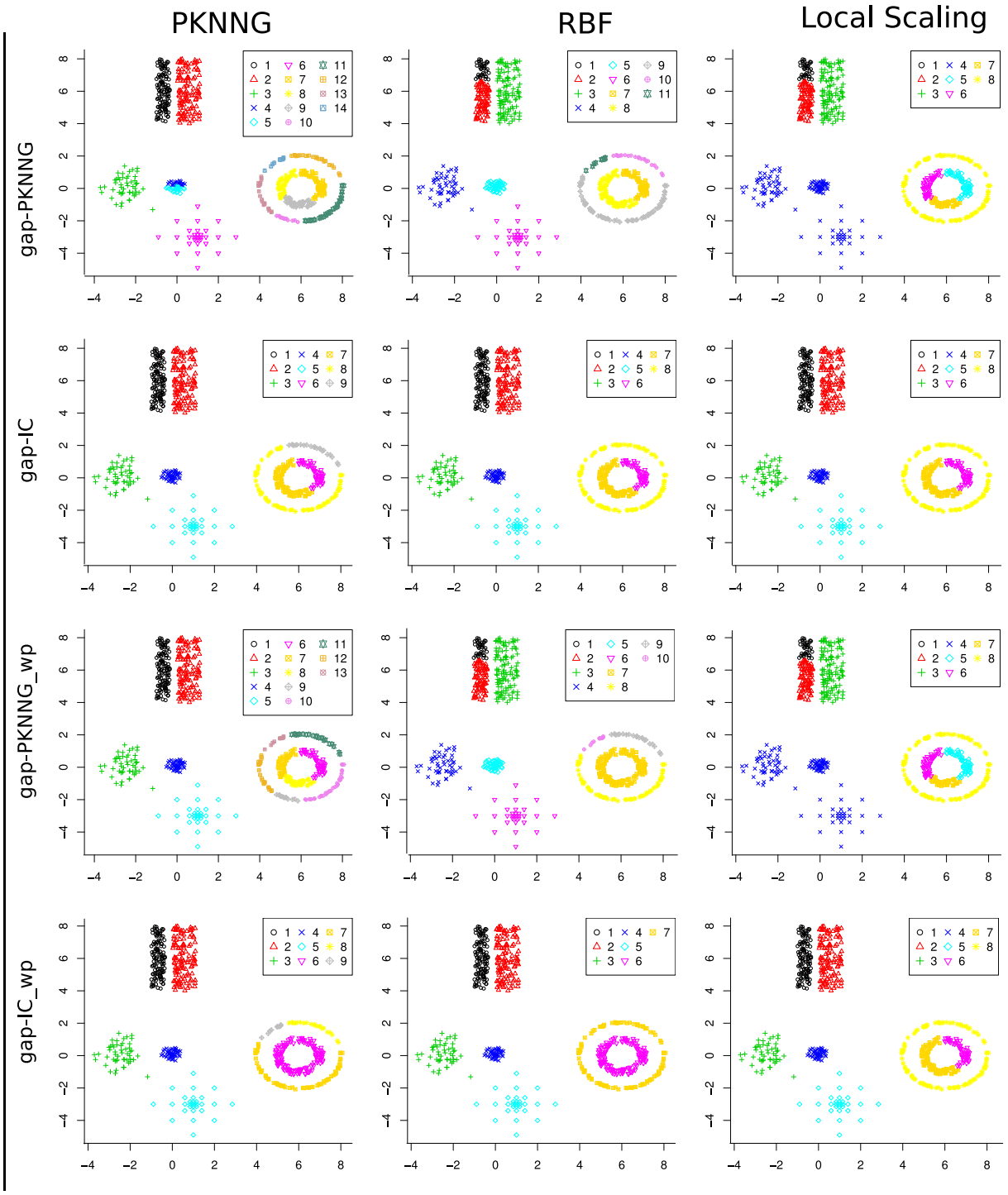


Figura 7.6

Particiones obtenidas con dhclus para el dataset 3. Cada columna corresponde a una forma de calcular la similaridad, pknng (primer columna), RBF (segunda columna) y Local Scaling (tercer columna). Cada fila corresponde a un test de parada diferente.

En los resultados para el último dataset ocurre algo similar, varios subconjuntos son partidos de más, en especial las dos circunferencias. Otra vez es posible llegar a la partición adecuada cortando el árbol a una altura apropiada. En todos los casos las particiones propuestas durante el paso de spectral clustering son correctas (siempre que se están separando clusters entre sí), después es trabajo del test de parada evaluar si debe aceptar o no la partición.

Aunque el algoritmo no encuentre las cantidad exacta de clusters, en la mayoría de los casos la solución óptima se encuentra contenida en la jerarquía de particiones devuelta por `dhclus`, por lo tanto es posible aplicar un índice de validación global basado en conectividad y encontrar la cantidad exacta de clusters en los datos. Por ejemplo para el dataset 2 usando Silhouette media con MED como medida de distancia es posible determinar que la cantidad de clusters es 3 para RBF y RBF-PKNNG, y para el dataset 3 el índice $k * SSW/SSB$ (otra vez usado con MED como medida de distancia) propuesto en [44] encuentra la cantidad correcta de clusters en todos los casos.

La sobre fragmentación de las circunferencias así como del cluster en forma de S del dataset 2 llamó nuestra atención debido a que los tests de parada están basados en conectividad. Por un lado es cierto que hay pequeños "huecos" entre los puntos que conforman el cluster y podría ser que simplemente el criterio de parada es muy sensible a pequeñas desconexiones en los datos (posiblemente debido al ajuste de escala que hace el algoritmo para tratar de encontrar la mejor separación de los datos y a que la condición de parada es evaluada de forma local). Por otro lado, si eso fuera todo, el algoritmo debería fragmentar aún más los datasets. Por lo tanto esto es una explicación parcial.

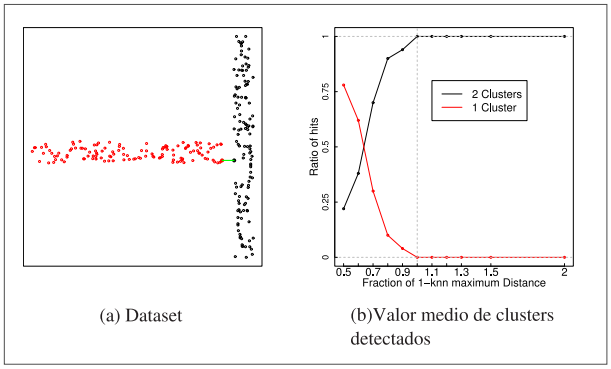
Sospechamos que están sucediendo dos situaciones en simultaneo, la primera es que la distribución que conforma el dataset no es estrictamente uniforme (porque hay huecos entre grupos de puntos), y la segunda es que la distribución de referencia nula usada para el test de gap podría no ser la más apropiada en los casos donde se produce sobre fragmentación (la importancia de la elección de una distribución de referencia nula es ampliamente conocida, fue demostrada por Gordon en [45] según lo expresado en [28]).

Para comprobar esto analizamos la mínima separación necesaria entre dos clusters para detectarlos como tal con el test de gap-PKNNG y test de gap-IC cuando la distribución de referencia es apropiada y cuando no lo es.

Primero debemos comentar un experimento presentado en Baya et al. [29] donde los autores analizan la mínima separación entre dos clusters uniformes para que gap-IC detecte

la presencia de dos clusters. El experimento consiste en generar varios sampleos provenientes de dos distribuciones uniformes dispuestos como en la Figura 7.7.a (la línea verde señala la mínima separación entre los puntos más cercanos de cada cluster) con diferente grado de separación y luego aplicar gap-IC a cada uno de los muestreos. La mínima separación entre los puntos más cercanos de cada clusters fue fijada como una fracción del valor del la máxima distancia el primer vecino del grafo 1-knn de cada cluster. Se usaron las siguientes fracciones $\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.5, 2\}$. Cada configuración se evaluó 50 veces, se acumularon los resultados del gap IC-av y se dividieron los resultados por 50 para calcular el nivel de acierto. En la Figura 7.7.b se presentan los resultados en forma de dos curvas donde el eje x muestra la separación entre los clusters y el eje y muestra la media de aciertos para 1 y 2 clusters.

A partir de los resultados mostrados en el artículo se puede concluir que bajo las condiciones del experimento para que el test de gap-IC detecte clusters, la distancia mínima entre dos puntos de los distintos clusters debe ser mayor que la máxima distancia al primer vecino del grafo k-nn de cada uno de los clusters. Hay que tener en cuenta que los dataset de los experimentos no presentan huecos en el interior de cada cluster. Pero ¿qué pasa cuando los clusters involucrados no cumplen las condiciones ideales?, ¿qué sucede cuando, por ejemplo, los clusters provienen de distribuciones uniformes independientes e igualmente distribuidas, pero que presentan pequeñas separaciones entre grupos de puntos al interior de cada cluster? ¿Qué pasa además con el test de gap-pknng en estos casos?



(a) Dataset

(b)Valor medio de clusters detectados

Figura 7.7

(a) Modelo de dataset usado para el test de mínima separación necesaria entre clusters para ser detectados con gap-IC. La línea verde conecta los puntos más cercanos entre el cluster rojo y el negro, y representa la separación entre los clusters que es una fracción del valor de la máxima separación del grafo de primer vecino, 1-knn. (b) Curvas de proporción de detección de uno y dos clusters en función de la separación.

Para responder estas cuestiones realizamos el siguiente experimento: generamos 1 dataset con dos clusters provenientes de una distribución uniforme y los dispuse como en la Figura 7.8.a y 7.8.b (es el mismo muestreo de datos, sólo que uno tiene una parte de

sus puntos rotados y trasladados) con diferentes grados de separación de modo tal que la mínima distancia entre dos puntos de los distintos clusters fuera una fracción de una distancia d de referencia. Esta distancia de referencia se midió como la máxima distancia MED de cada cluster (en vez del máximo primer vecino del grafo k-nn usado en el experimento original). La fracciones de separación usadas fueron $\{0.65, 0.9, 1.17, 1.5\}$.

frac / K	3	5	7	9	IC	3	5	7	9	IC
0.65	0	0	0	0	0	50	50	50	50	50
0.90	0	0	0	0	0	50	50	50	50	50
1.17	50	50	50	50	50	50	50	50	50	50
1.5	50	50	50	50	50	50	50	50	50	50
	Disposición 1					Disposición 2				

Tabla 7.2

Cantidad de veces que el test gap-PKNNG y gap-IC detectaron clusters para el dataset de la Figura 7.8.(a), de acuerdo a las distintos grados de separación entre las clases rosa y celeste.

Sobre el mismo dataset ejecutamos 50 veces gap-IC y gag-PKNNG variando k entre 2 y 9. Para calcular la suma intraclusters para los datos observados usamos las clases originales de los datasets y para clasificar los muestreos nulos utilizamos k-means con dos centros.

En la Tabla 7.2 presentamos la cantidad de veces que el test detectó clusters para el dataset bajo la disposición 1 (Figura 7.8.a). Para esa disposición ambos test detectan clusters a medida que la fracción de separación es mayor a la unidad. El resultado es similar al del experimento original, lo curioso sucede en el caso de la segunda disposición del dataset (Figura 7.8.b), porque en ese caso ambos test detectan clusters sin importar la separación entre los clusters .

Sin embargo, para los datos de la disposición 2 (Figura 7.8.b) las condiciones son similares a la del experimento original, incluso la disposición de los datos, excepto en el hecho de que los clusters tal vez presenten pequeños huecos, por lo que la diferencia en el resultado se puede deber a este factor.

Entrando más en detalle se tiene que el decrecimiento de la suma intraclusters para dos clusters con respecto a un sólo clusters (W_1 vs. W_2 y IC-av(1) vs. IC-av(2)) tanto para la disposición 1 y 2 es prácticamente el mismo, mientras que el decrecimientos de la esperanza muestral de la correspondiente nula ($E(W_k)$, $E(IC - av)$), en el caso de la disposición 1 es similar al de los datos observados. Pero que en el caso de la disposición

2 el decrecimiento de la esperanza de suma intracluster para la nula es menor que la de los datos observados. Esta es la razón de que se detecten clusters para la disposición 2 de los datos. La suma de la disimilaridad intracluster decae en promedio más rápido para los datos dispuestos como en la figura (Figura 7.8.b) que su distribución de referencia. Dado que eso no ocurre en el experimento presentado en [29] podemos concluir que en este caso la razón por la cual se detectan clusters aún cuando casi no hay separación entre los mismos se debe a dos cuestiones: una es que los clusters no son estrictamente uniformes y la segunda es que la distribución de referencia nula no es la más apropiada.

Esa conclusión es válida para ese dataset particular, y podría ser que con otro dataset generado bajo las mismas condiciones no sucediera lo mismo. Para verificar este punto repetimos el experimento unas 50 veces sólo para gap-IC, pero esta vez generando en cada caso un nuevo muestreo de los datos y en cada oportunidad se rotaron y trasladaron para obtener ambas disposiciones de datos. La forma de obtener las clases fue la misma que el primer experimento. La fracción para la separación usada fue $\{0.60, 0.90, 1.20, 1.50\}$, para ambas disposiciones, y los resultados de usar test de gap-IC están resumidos en la Tabla 7.3, donde se puede observar que para el caso de la disposición 2 (Figura 7.8.b) hay una tendencia mayor a detectar clusters sin que la separación entre los clusters influya demasiado.

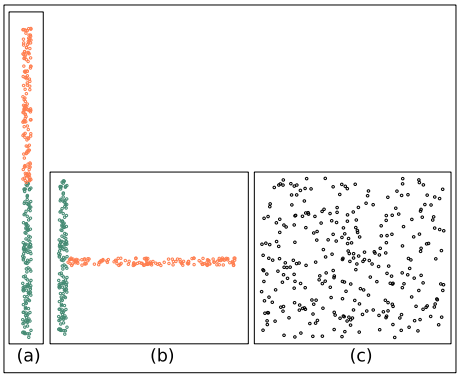


Figura 7.8
Modelos de dataset usados para el test de mínima separación. (a) Disposición 1. (b) Disposición 2. (c) Ejemplo de la correspondiente nula para el dataset con la disposición 2.

Cabe aclarar que en todos los experimentos la cantidad de puntos en uno y otro cluster es aproximadamente la misma y la densidad en ambos clusters también.

Lo que podemos concluir es que en ciertas ocasiones cuando la distribución de referencia nula no es la más apropiada para conjuntos de datos que presentan un único cluster bastará con que haya pequeñas separaciones en los datos para que entonces el test de gap-IC o test-PKNNG se incline por la presencia de clusters. También podemos concluir que cuando la distribución de referencia sea la apropiada, para detectar clusters se requiere como mínimo

fracción	0.60	0.90	1.20	1.30	1.50
disposición 1	4	7	44	50	50
disposición 2	33	46	50	50	50

Tabla 7.3

Resultados de usar test de gap-IC sobre un total de 50 sampleos de datasets provenientes del mismo tipo de distribución de acuerdo a la disposición 2 mostrada en la Figura 7.8.(b), según la fracción de separación.

una separación mayor o igual a la máxima distancia MED entre los pares de puntos de cada uno de los cluster.

7.1.4.1. Comparación con spectral clustering

Por la forma de trabajar y como queda evidenciado en los ejemplos `dhclus` es capaz de detectar clusters con formas arbitrarias, distintas densidades y escala no uniforme. Pero los algoritmos de spectral cluster particionales también son capaces de detectar clusters con formas arbitrarias. La pregunta es si `dhclus` presenta alguna ventaja sobre spectral cluster. Una de las ventajas es que trata de determinar automáticamente la cantidad de clusters. Pero una vez determinadas cabe preguntar si las particiones que proponen ambos difieren y cúal es la mejor.

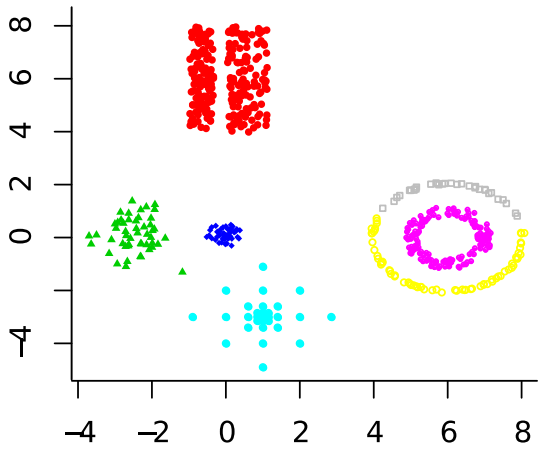


Figura 7.9

Partición encontrada por spectral clustering con 7 centros para el dataset 3.

Como comparación preliminar ejecutamos el algoritmo de spectral clustering particional pasando como argumentos la cantidad exacta de clusters. La similitud usada es RBF y la forma de elegir el σ es el mismo que el usado con `dhclus`. En los dos primeros dataset la partición hallada fue la correcta, mientras que para el dataset 3 partición propuesta fue la de la Figura 7.9 donde puede observarse que los clusters elongados son considerados un único cluster y la circunferencia mayor es partida en dos. En este caso `dhclus` obtiene mayores resultados ya que en todos los casos es posible buscar en la

jerarquía de particiones y encontrar las clases correctas, además con RBF y gap-IC_wp `dhclus` encuentra la solución exacta al problema.

7.2. Evaluación sobre datos de expresión de genes

En esta etapa analizamos la capacidad del algoritmo de resolver problemas en altas dimensiones. Para la evaluación del algoritmo sobre datos reales se incluyen cinco conjuntos de datos de *expresión de genes* de dominio público.

Se llama **expresión genética** a la transcripción de la información almacenada en un gen a una secuencia de ARN. La tecnología de microarreglos de ADN (DNA microarrays) sirve para medir la expresión de miles de genes simultáneamente bajo diferentes condiciones. Existen dos grandes clases de microarrays: los de dos canales, cDNA y los de un sólo canal. Los del tipo cDNA consisten en un sustrato sólido que tiene impreso miles de fragmentos de ADN (cDNA clones), sobre el que se hibridan simultáneamente dos muestras, una muestra en estudio y una de referencia, ambas marcadas con un fluoroscopo diferente cada una (comúnmente se utilizan cianinas[1] Cy3 -verde- y Cy5 -roja-). Luego se escanea el resultado y analiza mediante análisis de imagen los niveles relativos de expresión. En el caso de los microarrays de un sólo canal la expresión de genes de la muestra en estudio se mide en forma absoluta.

Después de normalizada la intensidad de la fluorescencia, los perfiles de expresión de genes se guardan en una matriz $E = (e_{ij})$ donde e_{ij} es el nivel de expresión del gen i de la muestra j. Con dicha matriz es posible hacer cluster análisis de dos formas diferentes, clusterizar genes o clusterizar las muestras. La hipótesis de trabajo detrás del clustering de genes es que genes que aparecen en el mismo grupo pueden tener funciones similares o comparten el mismo mecanismo de regulación de transcripción. El clustering de muestras sirve para identificar muestras en diferentes estados de enfermedad, o descubrir diferentes subtipos de cáncer [46, 47, 48].

El clustering de muestras es típicamente caracterizado por la alta dimensión $(10^3, 10^4)$ y pequeña cantidad de datos, pero algunos estudios demuestran [47] que es posible clasificar las muestras con un conjunto reducido de genes con altos niveles de expresión a los que se llama genes informativos y que el resto de los genes son irrelevantes y se pueden considerar

[1] Tinturas sintéticas fluorescentes.

ruido. La proporción de genes informativos con respecto a los irrelevantes suele ser menor a 1:10 y esto puede degradar los resultados finales. Es por ello que debe reducirse la dimensión de las muestras antes de intentar clusterizarlas [49].

A continuación describimos los datasets de expresión de genes sobre los que evaluamos el algoritmo.

7.2.1. Descripción del conjunto de datos

AML Muestras de médula ósea obtenidas de pacientes con leucemia aguda al momento del diagnóstico (antes de la quimioterapia). Este dataset fue presentado y estudiado en [47]. Se distinguen 3 clases: 11 muestras de leucemia mieloide aguda (AML), y 27 muestras de leucemia linfoblástica aguda (ALL), la que a su vez contiene dos subclases 19 de tipo T-ALL y 8 de tipo B-ALL .

ALI Cuenta con muestras de tres tipos de cáncer: 42 de linfoma difuso de células B grandes (DLBCL), 9 de linfoma folicular (FL) y 11 de la leucemia linfocítica crónica (CLL). Originalmente fue presentado en [48].

LEU Muestras de médula ósea de pacientes pediátricos con leucemia aguda que corresponden a 6 pronósticos importantes de subtipos de leucemia linfoblástica: 43 de T-ALL, 27 E2A-PBX1, 15 BCR-ABL, 79 TELAML1, 20 MLL y 64 hiperdiploides (con más de 50 cromosomas). Este dataset fue presentado y estudiado en [50].

LUNG Incluye 4 clases conocidas de carcinoma de pulmón: 139 adenocarcinomas (AD), 21 de carcinoma escamoso o epidermoide (SQ), 20 carcinoma pulmonar (COID) y 17 pulmonares normales (NL). Este dataset fue presentado y estudiado en [51].

YEAST Contiene la expresión de genes de la levadura de cerveza (Saccharomyces cerevisiae) bajo distintas condiciones de gemación[2] (diauxic shift, mitosis, sporulation, cambios abruptos de temperatura). La expresión de genes se obtuvo mediante el uso de microarrays que contienen esencialmente cada ORF[3] de este organismo secuenciado en su totalidad.

[2]La gemación es un tipo de reproducción asexual.

[3]Open Reading Frame: secuencia de nucleótidos sin el código de terminación.

El dataset original fue presentado y estudiado en [52], pero en este trabajo usamos una versión reducida en genes, la misma que se usó en [22].

Dataset	Chip	Tipo de tejido	n	Clases	Dim_1	Dim_2
AML	unicanal	medula ósea	38	3 (11-8-19)	6817	999
ALI	cDNA	Sangre	62	3 (42-9-11)	4022	1000
LEU	unicanal	medula ósea	248	6 (43-27-15-79-20-64)	12600	985
LUNG	unicanal	Pulmón	197	4 (139-17-21-20)	12600	1000
YEAST	unicanal		208	5 (14-27-121-35-11)	79	79

Tabla 7.4

Descripción de los datos reales usados en los experimentos. Las dos últimas columnas corresponden a la dimensión original del dataset (Dim_1) y la dimensión de la versión reducida (Dim_2) con la que trabajamos.

En los cuatro primeros casos el objetivo es agrupar las diferentes muestras en base a la información provista por los genes de expresión; por lo que utilizamos una versión reducida de los dataset en el número de dimensiones. Para el caso de los dataset AML y LEU usamos la versión utilizada en [21], para LUNG usamos la misma versión usada en [29] y para ALI usamos la versión reducida propuesta en [46].

Además los primeros cuatro conjuntos se normalizaron de la siguiente forma: se centraron los genes fijando su media en 0 y se escaló para igualar los desvíos estándar de cada gen a 1. Para el último dataset el objetivo es clusterizar los genes en base a las condiciones experimentales (las muestras), por lo que las variables normalizadas, siguiendo el mismo protocolo anterior, fueron las muestras.

7.2.2. Descripción de los experimentos

Se corrió doce veces el algoritmo para cada dataset. Una por cada combinación de la forma de calcular la matriz de similitud con cada test de parada.

7.2.3. Resultados

Los valores del índice *cRand* de resultados de las ejecuciones individuales de cada dataset se encuentran en la Tabla 7.5. Además para dar una idea de la complejidad de los

datos y contrastar las clases reales con los resultados de una manera visual procedimos a realizar una proyección a dos dimensiones usando las dos primeras componentes principales (Figura 7.10-7.14), cada gráfica usa un código de color diferente para cada cluster propuesto por `dhclus` y a su vez un código de símbolos para identificar a la clase a la que pertenece cada dato.

Para las Figura 7.11 - 7.14 las gráficas de la primer columna corresponden a los resultados cuyas matrices de similitud se calcularon usando la similitud RBF-PKNNG, la misma relación existe entre la segunda columna y RBF, y entre la tercera y Local Scaling. Al mismo tiempo las gráficas de la primer fila corresponden al test de gap-PKNNG, la segunda a gap-IC, la tercera a gap-PKNNG_wp y la cuarta a gap-IC_wp. Las particiones propuestas por spectral clustering en las gráficas para una misma columna son las mismas, y si los resultados finales (los graficados) difieren entre sí es debido al test de parada usado en cada caso.

	gap-PKNNG			gap-IC			gap-PKNNG_wp			gap-IC_wp		
AML	1	0.94	0.94	1	0.94	0.94	1	0.94	0.94	1	0.94	0.94
ALI	0.59	0.86	0.89	0.59	0	0	1	0.89	0.89	1	0.5	0.50
LEU	0.17	0.19	0	0.2	0.9	0.91	0.92	0.19	0	0.2	0.75	0.85
LUNG	0.39	0.39	0.37	0.64	0.39	0.37	0.64	0.65	0.60	0.63	0.64	0.6
YEAST	0.96	0.95	0.78	0.96	0.94	0.96	0.85	0.95	0.78	0.96	0.95	0.97
	PKNNG	RBF	LS	PKNNG	RBF	LS	PKNNG	RBF	LS	PKNNG	RBF	LS

Tabla 7.5

Valores de cRand para las clases obtenidas con dhclus con las distintas combinaciones de formas de construir la similitud y los tests de gap para cada uno de los dataset de expresión de genes presentados.

7.2.4. Análisis de resultados

Para el dataset AML los resultados de las ejecuciones del algoritmo bajo todos los test de gap coinciden. Sólo hay dos resultados finales, el resultado exacto de acuerdo al *cRand* se encuentra usando la similitud RBF-PKNNG lo mismo puede observarse en la Figura 7.10, la primer gráfica corresponde al resultado con RBF-PKNNG y la segunda al obtenido con RBF y Local Scaling, que en este caso coinciden.

Para el dataset ALI la partición generada por `dhclus` usando la similitud RBF-PKNNG es exacta como puede observarse la Tabla 7.5 (fila 2) para los test de gap-IC_wp y gap-

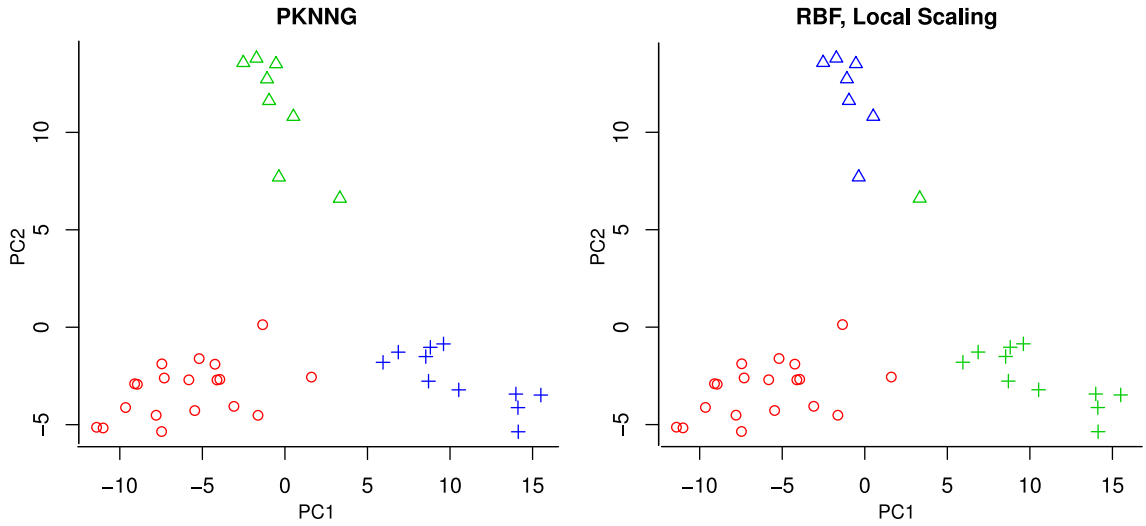


Figura 7.10

Proyección en dos dimensiones usando componentes principales para el dataset AML. La primer gráfica presenta el resultado de usar la similitud RBF-PKNNG y la segunda el de RBF y Local Scaling que coinciden.

PKNNG_wp, mientras que con RBF y Local Scaling hay 2 puntos de una clase mal clasificados como puede observarse en la Figura 7.11 columnas 2 y 3. Entre los test de parada, el test de gap-PKNNG es el que ha tenido el mejor comportamiento.

En el caso del dataset LEU el análisis es más complicado dado que ninguna de las combinaciones produce un resultado exacto, sin embargo puede observarse que la similaridad RBF-PKNNG es la que mejor clasificación propone. El resultado de la gráfica 7.12 primer columna y fila 3, es el resultado mejor puntuado, sólo clasifica mal 7 puntos y subdivide una clase en dos. Los resultados que le siguen corresponden a las gráficas de la segunda fila para RBF y Local Scaling, pero en este caso en los resultados hay mal clasificados 20 puntos en cada una (la cantidad de puntos mal clasificados se calcularon a partir de la tabla de contingencia entre las particiones propuestas y las reales.)

Los errores tempranos de clasificación se observan para Local Scaling comparando las gráficas de la Figura 7.12, filas dos y cuatro, en la cuarta fila se puede ver que en el cluster azul (conformado en su mayoría por la clase representada con triángulos apuntando hacia abajo) ha capturado puntos de otras dos clases (círculos y cruces), en la segunda fila a su vez estos puntos extras son separados del cluster (que en dicha gráfica quedó de color gris). Para RBF puede observarse algo similar comparando las mismas dos filas. Para RBF-PKNNG el caso del resultado de la gráfica de la primer columna y fila 3 es posible

subir un nivel en la jerarquía (juntar los clusters rosa y amarillo) y encontrar una partición que mejore aún más el resultado, mientras que con RBF y Local Scaling eso no es posible. Con respecto al test de parada no hay un comportamiento uniforme.

En el dataset LUNG la clase COID (símbolo x) es identificada correctamente por el algoritmo usando similaridad RBF-PKNNG y RBF; mientras que con Local Scaling se consigue un punto mal clasificado. Las clases NL (triángulos) y AD (círculos) son muy similares de acuerdo a los autores que presentaron el dataset originalmente [51]. Todas las similaridades confunden 1 a 3 puntos de clase NL con AD. Con respecto al test de parada las definiciones del gap pesado aceptan mayor o igual número de particiones que el sin pesar, sin embargo ningún test aceptó la partición entre las clases AD y SQ por lo que no es posible inferir cómo se comportaron las métricas en dicha situación a partir de los resultados.

Las tres formas de construir la similaridad estudiadas detectan sin problemas la clase 2 (triángulos) del dataset YEAST. Para el resto de las clases todas las métricas incurren en errores, el mejor resultado de acuerdo al índice *cRand* se produce con Local Scaling con muy poca diferencia con el resto de las métricas.

7.2.4.1. Comparación con spectral clustering

Para comparar con spectral clustering particional corrimos el algoritmo de spectral clustering particional pasando como argumentos la cantidad exacta de clusters. Como similitud usamos cada una de las similitudes estudiadas (RBF, RBF-PKNNG y Local Scaling) y la forma de elegir los parámetros de las similitudes es la misma que para `dhclus` Los resultados se resumen en la Tabla 7.6 y se comparan con el mejor valor de *cRand* obtenido con `dhclus` de acuerdo a la Tabla 7.5.

Para los dataset ALI, AML Y YEAST el valor del *cRand* en el mejor de los casos para dhclus y spectral clsutering particional es la misma, lo que significa que quizás todos los clusters de estos dataset están a la misma escala y que en ese caso buscar los clusters de modo particional o jerárquico no cambia los resultados. En el caso de LEU, el mejor resultado de `dhclus` es mucho mejor que el mejor del método particional, y hay que tener en cuenta que `dhclus` en ese caso devolvió un cluster de maś (Figura 7.12 fila 3 columna 1) lo que significa que volviendo un nivel en el árbol el resultado aún puede mejorarse. Eso significa que LEU posiblemente contenga clusters a diferentes escalas dado que en

Dataset	Dhclus	SP RBF	SP RBF-PKNNG	SP LS
AML	1	1	0.91	1
ALI	1	1	1	1
LEU	0.92	0.75	0.55	0.68
LUNG	0.65	0.91	0.65	0.82
YEAST	0.97	0.95	0.97	0.96

Tabla 7.6

Valores de cRand para las clases obtenidas con dhclus y Spectral clustering particional con las similitudes RBF, RBF-PKNNG y Local Scaling.

cuyo caso `dhclus` trabaja mejor que el método particional. En el caso de LUNG, `dhclus` encuentra sólo tres de los cuatros clusters presentes en los datos por lo tanto no es posible mejorar el *cRand* y dado que spectral clsutering corrió con la cantidad correcta de clusters la comparación no brinda ninguna información. Con respecto al clustering particional la similitud que obtiene mejores resultados es RBF.

7.2.5. Estabilidad

Intentamos medir cuánto varían los resultados ante variaciones en los datos. Si ante perturbaciones en los datos las particiones que genera `dhclus` no varían demasiado entonces el clustering es estable.

Para medir la estabilidad se realizaron 100 corridas del algoritmo por cada combinación similiaridad-test de parada, sobre una muestra del 95 % del conjunto de cada conjunto de datos. En base al resultado de cada ejecución se calculó el valor del índice *cRand* comparando las clases obtenidas y las verdaderas. Al hacer esto se genera una distribución de valores del índice *cRand* para cada combinación de matriz de similitud y test de parada.

Presentamos las distribuciones del índice *cRand* en gráficos tipo boxplot. Interpretando a la cantidad de clusters encontrados en cada ejecución como una variable aleatoria, también dispusimos en gráficos tipo boxplot a la cantidad de clusters.

De acuerdo a los resultados, AML es un dataset sencillo para `dhclus` en todas sus combinaciones. El algoritmo tiene buen desempeño y poca variación con todas las formas de construir la matriz de similaridad. Con respecto a comparación entre gap-PKNNG y

gap-IC, gap-IC es el mejor test. De los dos es el que mejor se comporta, siempre acepta más de 2 clusters mientras que gap-PKNNG rechaza más cuando deberían ser aceptados.

Teniendo en cuenta los resultados de las Figura 7.15 y la Figura 7.16 para el dataset ALI, la similitud RBF-PKNNG tiene el mejor comportamiento (mayor valor de *cRand* y correcta cantidad de clusters), le sigue RBF y luego Local Scaling. Con respecto al test de parada el test de gap con PKNNG se comporta mejor que con MED (gap-IC) y las versiones pesadas son mejores que la original.

Para LEU el algoritmo usando RBF-PKNNG propone mejores particiones, seguido por Local Scaling. En relación a gap-IC y gap-PKNNG depende de cómo se construyó la similiaridad: para RBF-PKNNG es mejor el test de gap-PKNNG, aunque ambos test tienen un mal desempeño y para RBF o Local Scaling gap-IC se comporta mejor. En general la versión pesada de los test se comporta mejor.

Las versiones gap pesado en los resultados del dataset LUNG tienen un valor de *cRand* un 50 % más alto que las versiones normales y buena estabilidad, con respecto a la forma de calcular la similitud no hay mucho que podamos decir sólo mirando los valores del *cRand*, pero dado que el calculo es estable y de acuerdo a las gráficas ya analizadas sobre este dataset, concluimos que las similitudes con RBF-PKNNG y RBF son las que producen las mejores particiones.

En el caso de YEAST todas las formas de construir la similaridad ofrecen particiones acertadas y gap-IC detecta mayor o igual cantidad de clases que gap-PKNNG.

En resumen las versiones de test de gap pesado aceptan mayor cantidad de particiones que las versiones sin pesar. Con respecto a la forma de construir la matriz de similitud RBF-PKNNG demostró mejor comportamiento tanto en bajas como en altas dimensiones, seguido por RBF y en tercer lugar Local Scaling. Si debemos elegir un acompañante para RBF-PKNNG como test de parada, nuestra elección estaría entre gap-PKNNG_wp y gap-IC_wp

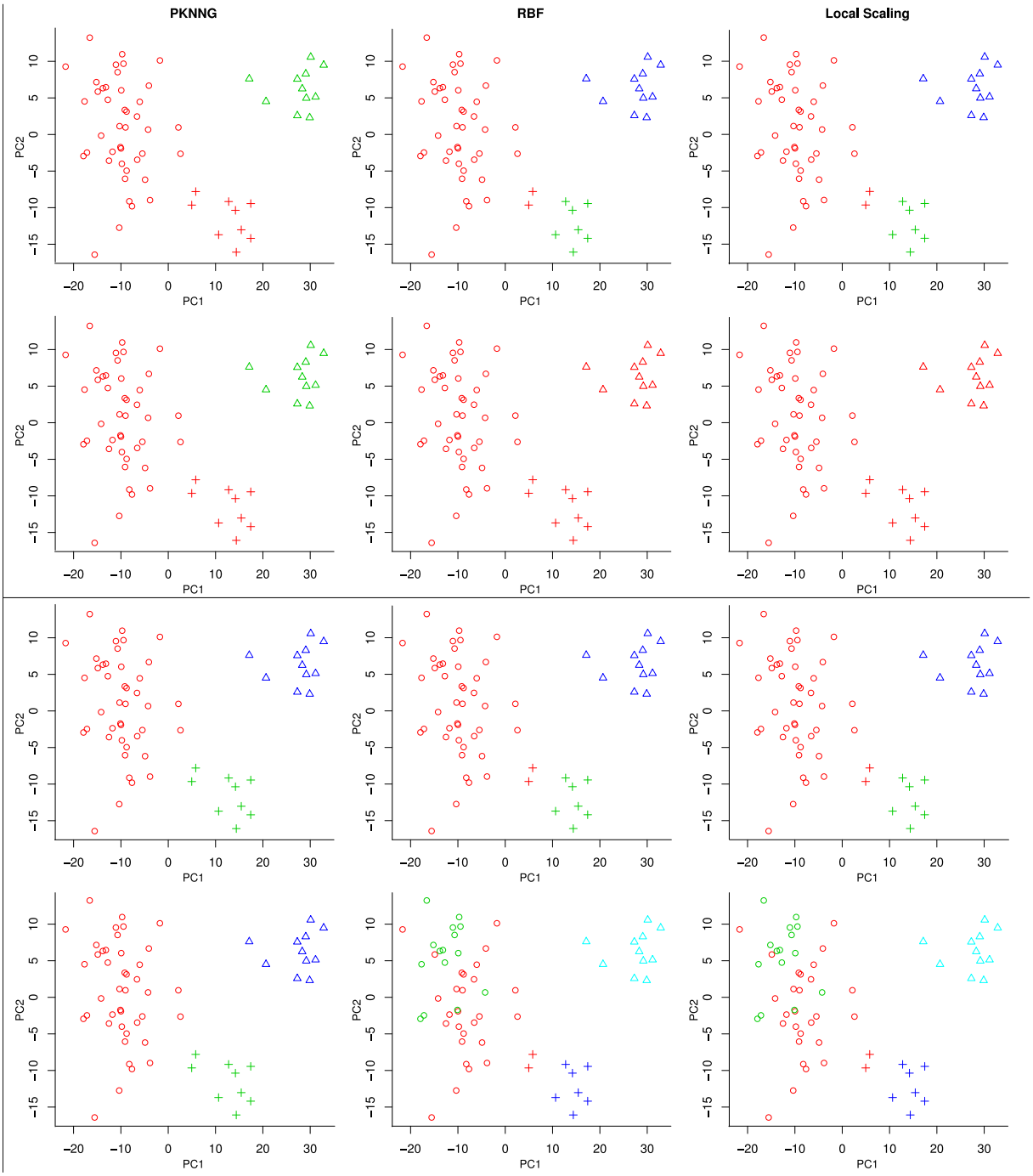


Figura 7.11

Proyección en dos dimensiones usando componentes principales para el dataset ALI. Cada columna corresponde a los resultados cuyas matrices de similitud se una forma particular. La primera corresponde a la similitud RBF-PKNNG, la segunda a RBF y la tercera a Local Scaling. Al mismo tiempo las gráficas de la primer fila corresponden al test de gap-PKNNG, la segunda a gap-IC, la tercera a gap-PKNNG-wp y la cuarta a gap-IC-wp. Las particiones propuestas en las gráficas para una misma columna son las mismas, y si los resultados finales difieren entre sí es debido al test de parada usado en cada caso.

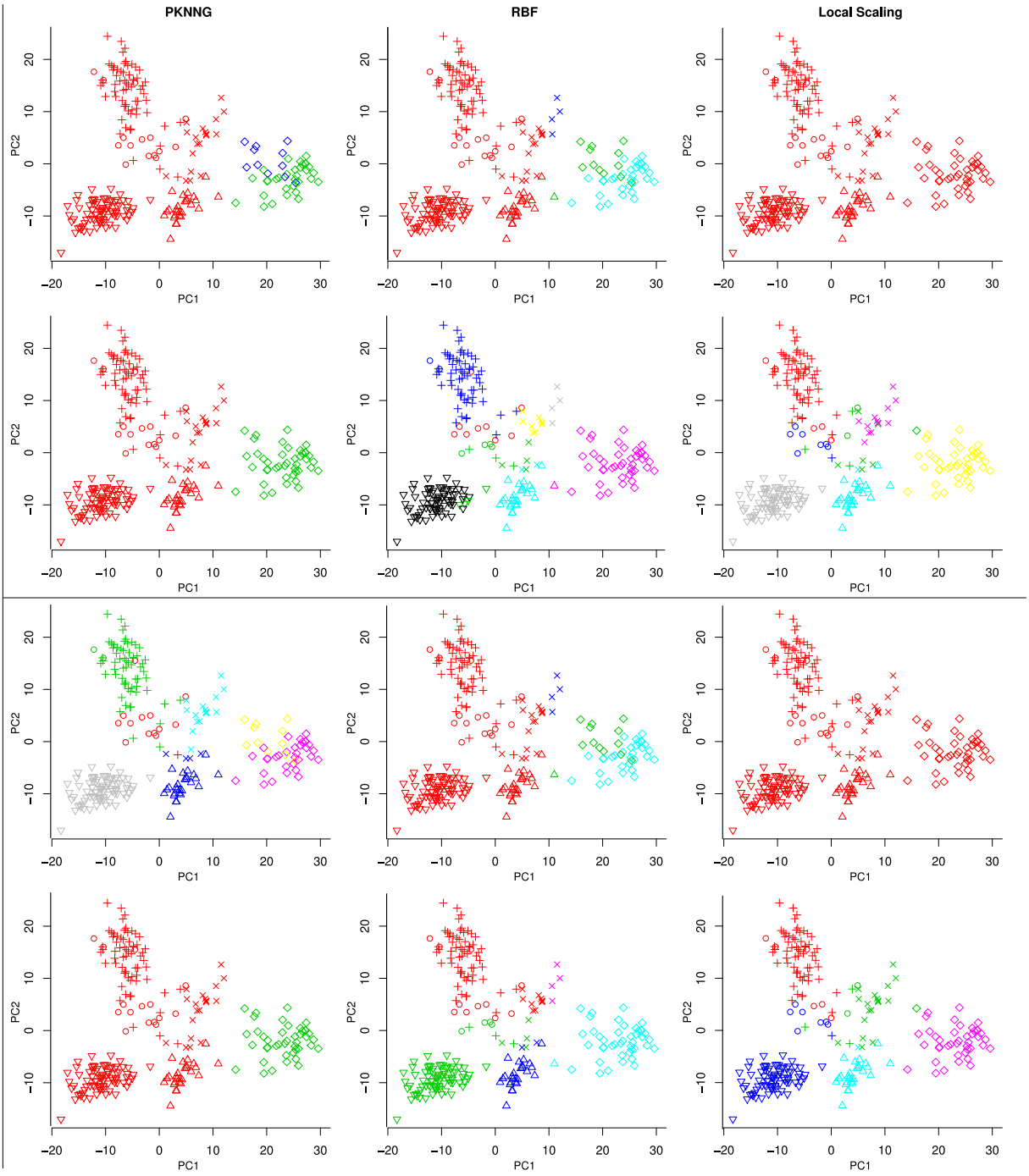


Figura 7.12

Idem Figura 7.11 para el dataset LEU.

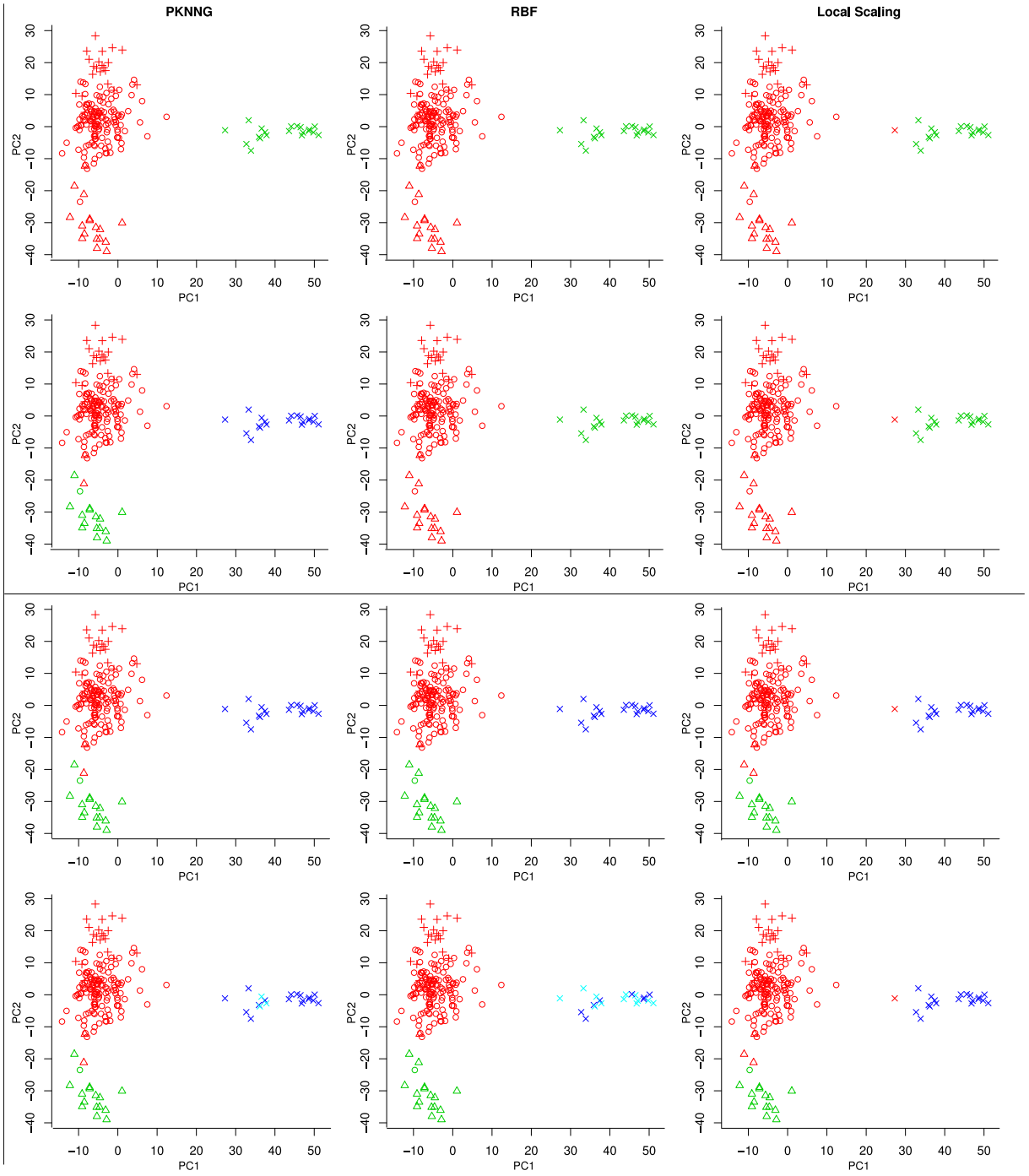


Figura 7.13

Idem Figura 7.11 para el dataset LUNG.

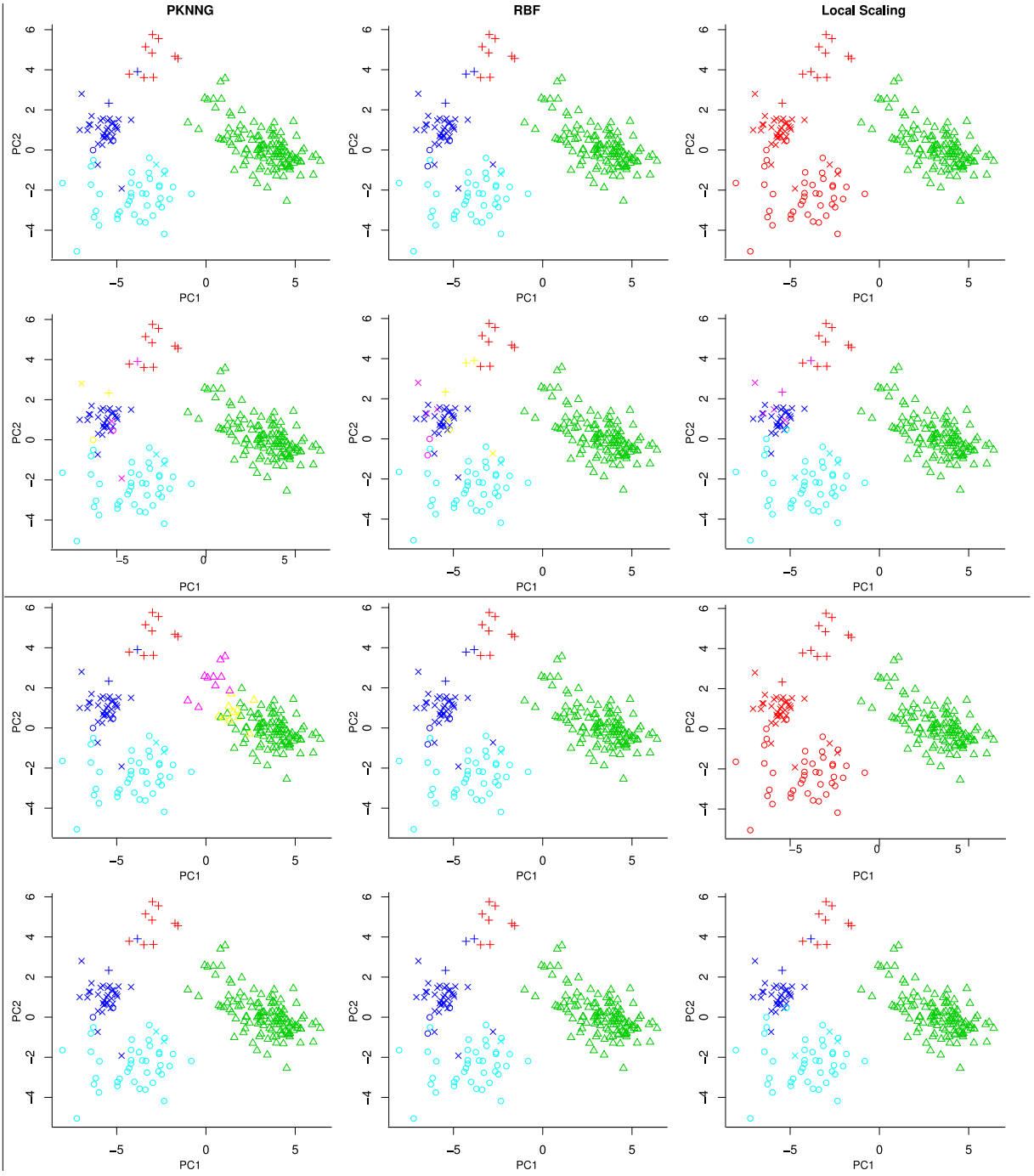


Figura 7.14

Idem Figura 7.11 para el dataset YEAST.

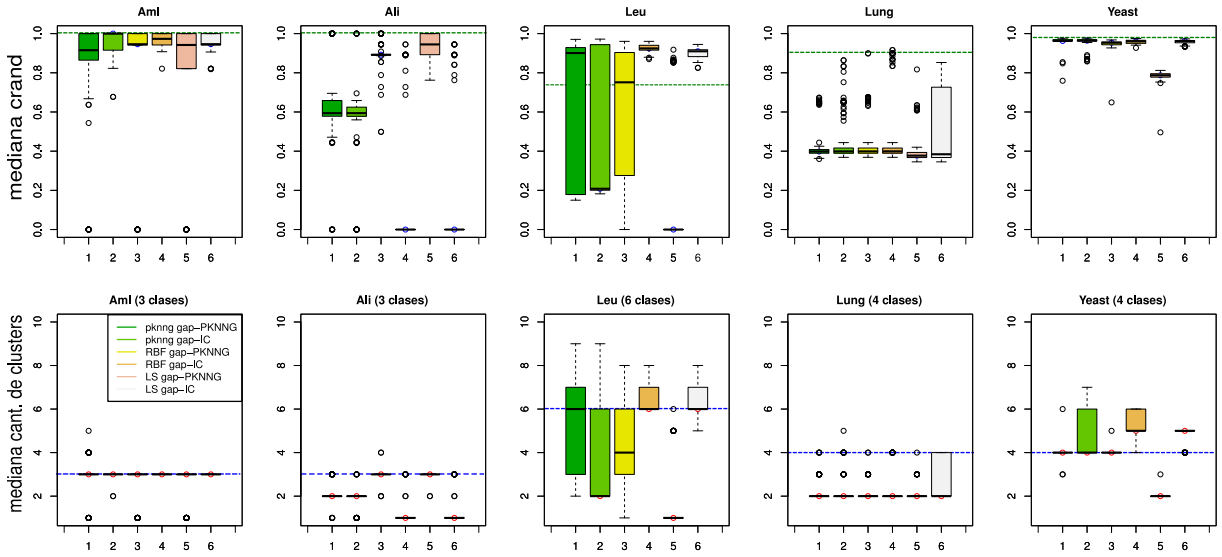


Figura 7.15

Primera fila: distribuciones del índice cRand en gráficos tipo boxplot; las líneas de puntos marcan el mejor valor de cRand alcanzado por la versión de spectral clustering particional. Segunda fila, boxplot sobre la distribución de la cantidad de clusters encontrados para cada dataset; las líneas de puntos indican la cantidad correcta de clusters.

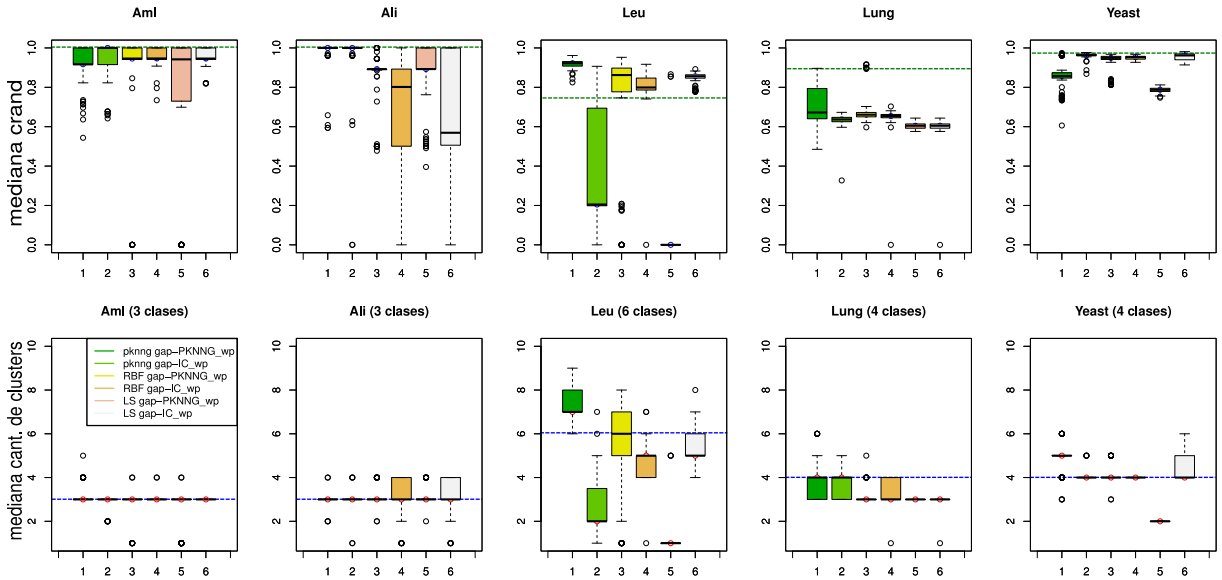


Figura 7.16

Primera fila: distribuciones del índice cRand en gráficos tipo boxplot; las líneas de puntos marcan el mejor valor de cRand alcanzado por la versión de spectral clustering particional. Segunda fila, boxplot sobre la distribución de la cantidad de clusters encontrados para cada dataset; las líneas de puntos indican la cantidad correcta de clusters.

8 Conclusiones y trabajos futuros

En este trabajo desarrollamos un algoritmo de clustering divisivo que usa spectral clustering para detectar clusters con formas arbitrarias, `dhclus`. Comparamos tres similaridades basadas en el kernel gaussiano (RBF, RBF-PKNNG y Local Scaling) para alimentar el algoritmo de clustering y discutimos una estrategia para seleccionar los parámetros que regulan la construcción de dichas similaridades. Además consideramos cuatro variantes del test de gap como criterio de parada: gap-PKNNG, gap-IC, gap-PKNNG_wp, gap-IC_wp. Evaluamos la combinación de las distintas similaridades con los criterios de parada en problemas sintéticos en bajas dimensiones y en problemas reales en altas dimensiones, analizamos la estabilidad de las soluciones ante variaciones en los datos y comparamos las soluciones encontradas con las obtenidas de aplicar un algoritmo particional de spectral clustering.

Entre las similaridades usadas para `dhclus`, los resultados obtenidos parecen indicar que la mejor similaridad es RBF-PKNNG ya que detecta correctamente las clases en mayor proporción. Cuando los datos están a una misma escala `dhclus` posee una exactitud similar a la versión particional de spectral clustering pero cuando los clusters se encuentran a diferentes escalas `dhclus` tiene más oportunidades de encontrar las clases cuando la versión particional falla. Este era uno de los principales objetivos de este trabajo y se logró gracias a la naturaleza divisiva del algoritmo que permite recalcular la similaridad en cada nivel. En este sentido también es posible k-partir los datos en cada nivel (en lugar de bi-partirlos), lo que básicamente significa reemplazar el test de tendencia al cluster como criterio de parada por un índice de validación interna u otra estrategia que determine la cantidad de clusters en cada subproblema.

Con respecto a la búsqueda del parámetros σ para la construcción de la similitud decidimos usar todo el histograma de distancias en lugar de restringirlo para evitar introducir sesgos adicionales y dejar que el criterio de búsqueda selecciones el más apropiado. Existe un efecto indirecto que el parámetro tiene sobre la decisión final de aceptar una partición

propuesta, al igual que el resto de los parámetros que intervienen en la construcción de la similitud entre los datos. El problema de la selección de dichos parámetros es que los mismos además de influir en la determinación de las particiones también influyen en la decisión del test de parada, ya que se usan para crear la matriz de similitud para las distribuciones nulas y eso afecta la aceptación de la partición. Por esta razón sería preferible desacoplar el test de parada de estos parámetros para poder hacer un análisis más directo del criterio de parada. Una manera de hacerlo sería entrenar un clasificador sencillo en base a las clases encontradas para los datos observados y usarlo para clasificar los datos nulos. De esa manera el test de parada sería independiente de los parámetros usados para crear la matriz de similitud de los datos observados. La exploración e implementación de esta solución es un posible trabajo futuro.

También es posible cambiar completamente el test de parada, sin usar test de gap. En este sentido las ideas del trabajo de Zahn [53] para detectar y describir clusters basadas en criterios sobre grafos podrían resultar interesantes para usarlas como punto de partida en conjunción al criterio desarrollado en [54] para detectar el número de cluster a través de grafos de vecinos relativos. Encontrar un test de parada independiente del algoritmo de clustering usado que sea eficiente y con alta precisión es un problema difícil pero nos parece fundamental para mejorar el algoritmo presentado.

Respecto a la estabilidad de los resultados producidos por `dhclus` debo decir que la parte del algoritmo encargada de hacer el clustering además de buena precisión presenta gran estabilidad en los resultados ante variaciones de los datos mientras que el test de parada no siempre resulta ser muy estable. Concluimos esto debido a que cuando en los resultados se observaron grandes variaciones ante cambios del dataset se debió al test de parada.

Una de las ideas del trabajo era que el test de parada sirviera para detectar la cantidad de cluster presentes en los datos, sin embargo en vista de la dificultad que conlleva el problema planteado relajamos dicho objetivo y en vez de eso se espera que el algoritmo en el mejor de los casos encuentre una cantidad mayor de clusters que la presente en los datos. Dado que el algoritmo devuelve en realidad una jerarquía de particiones, y en el último nivel se encuentra la propuesta, si la partición propuesta no es la solución óptima es siempre preferible que proponga una cantidad igual o mayor de clusters, dado que entonces esperamos que la solución óptima este contenida dentro de la jerarquía de particiones. Esto brinda la posibilidad de aplicar un índice de validación global basado

en conectividad y encontrar la cantidad exacta de clusters en los datos. En este aspecto, podría ser útil analizar la factibilidad de usar la matriz de similitud devuelta por `dhclus` en un algoritmo del estilo VAT para determinar la cantidad de clusters.

Con la versión final de la implementación del algoritmo armamos un paquete en R bajo licencia GPL3 y lo pusimos a disposición de toda la comunidad en un repositorio libre acceso.

Bibliografía

[1] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0-13-022278-X.

[2] Sergios Theodoridis and Konstantinos Koutroubas. *Pattern Recognition*. Academic Press, USA, 2003. ISBN 0-12-685875-6.

[3] Anil K. Jain, M. Narasimha Murty, and Patric J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, sep 1999. ISSN 0360-0300. doi: 10.1145/331499.331504. URL `http://doi.acm.org/10.1145/331499.331504`.

[4] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000. ISBN 0471056693.

[5] Rui Xu and D. Wunsch, II. Survey of clustering algorithms. *Trans. Neur. Netw.*, 16(3):645–678, may 2005. ISSN 1045-9227. doi: 10.1109/TNN.2005.845141. URL `http://dx.doi.org/10.1109/TNN.2005.845141`.

[6] Ana L. N. Fred and Anil K. Jain. Combining multiple clusterings using evidence accumulation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(6):835–850, jun 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.113. URL `http://dx.doi.org/10.1109/TPAMI.2005.113`.

[7] Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability)*. SIAM, Society for Industrial and Applied Mathematics, may 2007. ISBN 0898716233.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, USA, aug 2003. ISBN 0387952845.

[9] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[10] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, dec 2001. ISSN 0925-9902. doi: 10.1023/A:1012801612483. URL `http://dx.doi.org/10.1023/A:1012801612483`.

[11] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[12] Stuart Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489.

[13] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett.*, 31(8):651–666, jun 2010. ISSN 0167-8655. doi: 10.1016/j.patrec.2009.09.011. URL `http://dx.doi.org/10.1016/j.patrec.2009.09.011`.

[14] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975. URL `http://eudml.org/doc/12900`.

[15] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[16] Wilm E. Donath and Alan J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973. ISSN 0018-8646. doi: 10.1147/rd.175.0420.

[17] Ulrike Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. pages 857–864. MIT Press, 2004.

[18] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, dec 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z. URL `http://dx.doi.org/10.1007/s11222-007-9033-z`.

[19] L. Hagen and A.B. Kahng. New spectral methods for ratio cut partitioning and clustering. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 11(9):1074–1085, 1992. ISSN 0278-0070. doi: 10.1109/43.159993.

[20] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000. ISSN 0162-8828. doi: 10.1109/34.868688.

[21] Ariel E. Baya and Pablo M. Granitto. Clustering gene expression data with a penalized graph-based metric. *BMC Bioinformatics*, 12(1):2, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-2. URL `http://www.biomedcentral.com/1471-2105/12/2`.

[22] Ariel E. Baya. *Aplicación de algoritmos no supervisados a datos biológicos.* PhD thesis, Universidad Nacional de Rosario, 2011.

[23] Wagner Silke and Wagner Dorothea. Comparing clusterings - an overview. URL `http://i11www.iti.uni-karlsruhe.de/extra/publications/ww-cco-06.pdf`.

[24] Jorge M. Santos and Mark Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *Artificial Neural Networks – ICANN 2009*, volume 5769 of *Lecture Notes in Computer Science*, pages 175–184. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04276-8. doi: 10.1007/978-3-642-04277-5_18. URL `http://dx.doi.org/10.1007/978-3-642-04277-5_18`.

[25] Glenn W. Milligan and MarthaC. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985. ISSN 0033-3123. doi: 10.1007/BF02294245. URL `http://dx.doi.org/10.1007/BF02294245`.

[26] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 911–916, 2010. doi: 10.1109/ICDM.2010.35.

[27] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(0):53 – 65, 1987. ISSN 0377-0427. doi: http://dx.doi.org/10.1016/0377-0427(87)90125-7. URL `http://www.sciencedirect.com/science/article/pii/0377042787901257`.

[28] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a dataset via the gap statistic. *Journal of the Royal Statistical Society, Series B*, 63:411–423, 2001.

[29] Ariel E. Baya and Pablo M. Granitto. How many clusters: A validation index for arbitrary-shaped clusters. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(2):401–414, 2013. ISSN 1545-5963. doi: http://doi.ieeecomputersociety.org/10.1109/TCBB.2013.32.

[30] Stephen P. Smith and Anil K. Jain. Testing for uniformity in multidimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(1):73–81, 1984. ISSN 0162-8828. doi: 10.1109/TPAMI.1984.4767477.

[31] J. C. Bezdek and R. J. Hathaway. Vat: a tool for visual assessment of (cluster) tendency. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2225–2230, 2002. doi: 10.1109/IJCNN.2002.1007487.

[32] Yingkang Hu and Richard J. Hathaway. An algorithm for clustering tendency assessment. *WSEAS Trans. Math.*, 7(7):441–450, jul 2008. ISSN 1109-2769.

[33] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Avances in Neural Iinformation Processing Systems*, pages 849–856. MIT Press, 2001.

[34] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176 – 190, 2008. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2007.05.018. URL `http://www.sciencedirect.com/science/article/pii/S0031320307002580`.

[35] Maria C.V. Nascimento and André C.P.L.F. de Carvalho. Spectral methods for graph clustering – a survey. *European Journal of Operational Research*, 211(2):221 – 231, 2011. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2010.08.012. URL `http://www.sciencedirect.com/science/article/pii/S0377221710005497`.

[36] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608. MIT Press, 2004.

[37] Boaz Nadler and Meirav Galun. Fundamental limitations of spectral clustering. In *Advanced in Neural Information Processing Systems 19*. MIT Press, 2007. URL `http://books.nips.cc/papers/files/nips19/NIPS2006_0134.pdf`.

[38] Arik Azran and Zoubin Ghahramani. Spectral methods for automatic multiscale data clustering. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 190–197, 2006. doi: 10.1109/CVPR.2006.289.

[39] F. H. C. Marriott. Optimization methods of cluster analysis. *Biometrika*, 69(2):pp. 417–421, 1982. ISSN 00063444. URL `http://www.jstor.org/stable/2335416`.

[40] Mingjin Yan. *Methods of Determining the Number of Clusters in a Data Set and a New Clustering Criterion.* PhD thesis, Faculty of the Virginia Polytechnic Institute and State University, 2005.

[41] Geoffrey H. Ball and David J. Hall. Isodata: A novel method of data analysis and pattern classification. Technical report, Menlo Park: Stanford Research Institute, 1965.

[42] Mingjin Yan and Keying Ye. Determining the number of clusters using the weighted gap statistic. *Biometrics*, 63(4):1031–1037, 2007. ISSN 1541-0420. doi: 10.1111/j.1541-0420.2007.00784.x. URL `http://dx.doi.org/10.1111/j.1541-0420.2007.00784.x`.

[43] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org`.

[44] Qinpei Zhao, Mantao Xu, and Pasi Fränti. Sum-of-squares based cluster validity index and significance analysis. In *Proceedings of the 9th International Conference on Adaptive and Natural Computing Algorithms*, ICANNGA'09, pages 313–322, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 3-642-04920-6, 978-3-642-04920-0.

[45] A.D. Gordon. *Classification, 2nd Edition.* Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1999. ISBN 9781584888536. URL `http://books.google.com.ar/books?id=_w5AJtbfEz4C`.

[46] Marcilio de Souto, Ivan Costa, Daniel de Araujo, Teresa Ludermir, and Alexander Schliep. Clustering cancer gene expression data: a comparative study. *BMC Bioinformatics*, 9(1):497, 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-497. URL `http://www.biomedcentral.com/1471-2105/9/497`.

[47] T.R. Golub and D. K. Slonim. Molecular classification of cancer: Class discovery and class prediction by gene expression. *Science*, 286(5439):531–537, 1999. ISSN 00368075.

[48] Ash A. Alizadeh, Michael B. Eisen, R. Eric Davis, Chi Ma, Izidore S. Lossos, Andreas Rosenwald, Jennifer C. Boldrick, Hajeer Sabet, Truc Tran, Xin Yu, John I. Powell, Liming Yang, Gerald E. Marti, Troy Moore, James Hudson Jr., Lisheng Lu, David B. Lewis, Robert Tibshirani, Gavin Sherlock, and Wing C. Chan. Distinct types of

diffuse large b-cell lymphoma identified by gene expression profiling. (cover story). *Nature*, 403:503–511, 2000. ISSN 00280836. doi: 10.1038/35000501.

[49] D. Domingo Savio Rodríguez Baena. *Análisis de datos de Expresión Genética mediante técnicas de Biclustering.* PhD thesis, Dto. Lenguajes y Sistemas Informáticos, Universidad de Sevilla, 2006. URL `http://www.lsi.us.es/docs/doctorado/memorias/Memoria-v2.pdf`.

[50] Eng-Juh Yeoh, Mary E. Ross, Sheila A. Shurtleff, W.Kent Williams, Divyen Patel, Rami Mahfouz, Fred G. Behm, Susana C. Raimondi, Mary V. Relling, Anami Patel, Cheng Cheng, Dario Campana, Dawn Wilkins, Xiaodong Zhou, Jinyan Li, Huiqing Liu, Ching-Hon Pui, William E. Evans, Clayton Naeve, Limsoon Wong, and James R. Downing. Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. *Cancer Cell*, 1(2):133 – 143, 2002. ISSN 1535-6108. doi: http://dx.doi.org/10.1016/S1535-6108(02)00032-6. URL `http://www.sciencedirect.com/science/article/pii/S1535610802000326`.

[51] Arindam Bhattacharjee, William G. Richards, Jane Staunton, Cheng Li, Stefano Monti, Priya Vasa, Christine Ladd, Javad Beheshti, Raphael Bueno, Michael Gillette, Massimo Loda, Griffin Weber, Eugene J. Mark, Eric S. Lander, Wing Wong, Bruce E. Johnson, Todd R. Golub, David J. Sugarbaker, and Matthew Meyerson. Classification of human lung carcinomas by mrna expression profiling reveals distinct adenocarcinoma subclasses. *Proceedings of the National Academy of Sciences*, 98(24):13790–13795, 2001. doi: 10.1073/pnas.191502998.

[52] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[53] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *Computers, IEEE Transactions on*, C-20(1):68–86, 1971. ISSN 0018-9340. doi: 10.1109/T-C.1971.223083.

[54] Sanghamitra Bandyopadhyay. An automatic shape independent clustering technique. *Pattern Recognition*, 37(1):33 – 45, 2004. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/S0031-3203(03)00235-8. URL `http://www.sciencedirect.com/science/article/pii/S0031320303002358`.