Universidad de Valladolid

FACULTAD DE CIENCIAS

MÁSTER EN MATEMÁTICAS

CLUSTERING ESPECTRAL

Trabajo de Fin de Máster

Autor: Carlos Noriega González

Tutor: Pedro César Álvarez Esteban

Curso 2023-2024

Resumen

El clustering espectral es una técnica de clustering que tiene sus orígenes en la teoría de grafos. Este método asocia conjuntos de datos a los nodos de un grafo e identifica agrupaciones de nodos en este grafo basándose en las aristas que los conectan. Nuestro primer objetivo del trabajo consiste en presentar los diferentes tipos de grafos que podemos construir partiendo de nuestro conjunto de datos. Puesto que este método utiliza información de los autovalores de determinadas matrices construidas a partir del grafo, el segundo de los objetivos planteados es aprender a construir estas matrices y a interpretar su espectro.

A lo largo del segundo capítulo estudiaremos diferentes algoritmos que nos permiten utilizar los autovectores asociados a estas matrices para asignar nuestros datos a clusters. La elección de la matriz que usemos a la hora de calcular estos autovalores nos llevará a diferentes algoritmos de clustering espectral. En el tercer capítulo introduciremos una interpretación en términos de probabilidad y procesos estocásticos del funcionamiento de estos algoritmos. Finalmente, concluiremos en el cuarto capítulo con la implementación práctica del clustering espectral, tanto a datos generados de manera sintética como a datos reales. Estos últimos nos permitirán presentar una aplicación real del clustering espectral al reconocimento facial.

Abstract

Spectral clustering is a clustering technique that has its origins in graph theory. This method associates data sets to the nodes of a graph and identifies groups of nodes in this graph based on the edges that connect them. Our first objective is to present the different types of graphs we can construct from our data set. Since this method uses information from the eigenvalues of certain matrices constructed from the graph, the second objective is to learn how to construct these matrices and how to interpret their spectrum.

In the second chapter we will study different algorithms that allow us to use the eigenvectors associated with these matrices to assign our data to clusters. The choice of the matrix we use when calculating these eigenvalues will lead us to different spectral clustering algorithms. In the third chapter we will introduce an interpretation in terms of probability and stochastic processes of how these algorithms work. Finally, we will conclude in the fourth chapter with the practical implementation of spectral clustering, both to synthetically generated data and to real data. The latter will allow us to present a real application of spectral clustering to face recognition.

Lista de figuras y tablas

Figuras

- Figura 1.1: *Grafos de ε-vecindad*
- Figura 1.2: *Grafos 4-vecinos más próximos*
- Figura 1.3: *Grafo 4-vecinos más próximos y 7-vecinos más próximos mutuos*
- Figura 4.1: *Conjuntos de datos generados*
- Figura 4.2: *Aplicación de los algoritmos de clustering espectral a diferentes conjuntos*
- Figura 4.3: *Muestra de 10 imágenes de The Yale Face Database*
- Figura 4.4: *Representación gráfica de los resultados obtenidos con 4 sujetos*
- Figura 4.5: *10 autovalores de L más pequeños*
- Figura 4.6: *Representación gráfica de los resultados obtenidos con 10 sujetos*
- Figura 4.7: *25 autovalores de L más pequeños*

Tablas

- Tabla 4.1: *Matriz de contingencia general*
- Tabla 4.2: *Valores de la métrica ARI modificando k y σ*
- Tabla 4.3: *Resultados obtenidos con ambos métodos de clustering con 4 sujetos*
- Tabla 4.4: *Matriz de contingencia del clustering espectral con 4 sujetos*
- Tabla 4.5: *Métrica ARI con ambos algoritmos para 4 sujetos*
- Tabla 4.6: *Valores de la métrica ARI modificando kgraph*
- Tabla 4.7: *Resultados obtenidos con ambos métodos de clustering con 10 sujetos*
- Tabla 4.8: *Matriz de contingencia del clustering espectral con 10 sujetos*
- Tabla 4.9: *Métrica ARI con ambos algoritmos para 10 sujetos*

Índice general

Introducción **7**

1. Teoría espectral de grafos **9**
1.1. Definiciones básicas de grafos . . . 9
1.2. Funciones y grafos de similitud . . . 10
1.2.1. Grafo de ε-vecindad . . . 11
1.2.2. Grafo de k-vécinos más próximos . . . 13
1.2.3. Grafo completamente conectado . . . 15
1.3. Matrices Laplacianas . . . 15
1.4. Matrices Laplacianas normalizadas . . . 18

2. Algoritmos de clustering espectral **21**
2.1. Método de Fiedler . . . 21
2.2. Clustering espectral no normalizado - *RatioCut* . . . 25
2.3. Clustering espectral normalizado - *NCut* . . . 31
2.3.1. Clustering espectral normalizado con L_{sym} . . . 34
2.3.2. Clustering espectral normalizado con L_{rw} . . . 35
2.4. Detalles prácticos . . . 36
2.4.1. Comparación de los 3 algoritmos . . . 36
2.4.2. Uso del algoritmo *k-means* . . . 37
2.4.3. Determinación del número k de clusters . . . 38

3. Interpretación probabilística de los algoritmos **40**
3.1. *Paseos aleatorios* y cadenas de Markov . . . 40
3.2. Distancia de conmutación . . . 43

4. Implementación práctica de los algoritmos **48**
4.1. Resultados obtenidos con datos sintéticos . . . 48
4.2. Métricas de evaluación . . . 50
4.2.1. Índice de Rand . . . 50
4.2.2. Índice de Rand ajustado (ARI) . . . 51
4.2.3. Métricas utilizando la tabla de contingencia . . . 52
4.3. Aplicación del clustering espectral en *face clustering* . . . 53
4.3.1. Presentación de los datos . . . 53
4.3.2. Resultados obtenidos . . . 54

A. Código implementado en R **61**
A.1. Generación de grafos . . . 61
A.2. Clustering espectral con datos sintéticos . . . 62
A.3. *Face clustering* . . . 65

Introducción

El clustering es una de las principales tareas del aprendizaje automático no supervisado. El objetivo es asignar datos no etiquetados a diferentes grupos, con la premisa de que puntos asociados a datos similares se asignen a un mismo grupo. Las aplicaciones de las técnicas de clustering van desde la estadística, la informática y la biología hasta las ciencias sociales o la psicología. Se trata en definitiva de identificar grupos de *"comportamiento similar"* entre los datos con los que trabajamos.

Dentro de las numerosas técnicas de clustering que podemos encontrar en la actualidad, los algoritmos de clustering espectral resultan ser uno de los métodos más atractivos. Estos algoritmos fueron popularizados a principios de siglo tanto por Shi y Malik ([12]) como por Ng, Jordan y Weiss ([10]). En ambos trabajos se plantea el problema de la agrupación de datos como un problema de partición de grafos, sin hacer ninguna suposición sobre la forma de los clusters. Este hecho de no suponer una forma determinada de los datos da lugar a que sus resultados mejoren en algunos casos a los obtenidos con ciertos algoritmos tradicionales.

La mejora en ciertos casos respecto a los algoritmos de clustering tradicionales, como el algoritmo *k-means*, se debe a que estos últimos utilizan métricas esféricas o elípticas, asumiendo que los puntos asignados a un cluster son esféricos alrededor del centro de este cluster. Esta suposición sobre la forma de los datos implica una peor actuación cuando los clusters son no convexos. Para determinados problemas el clustering espectral se ha convertido en un tipo de clustering muy popular, ya que puede agrupar correctamente observaciones que en realidad pertenecen al mismo cluster, pero que están más alejadas que las observaciones de otros clusters.

Otro de los motivos que encontramos en la popularización del clustering espectral se debe a la expansión que han tenido las redes sociales en los últimos años. Los datos que provienen de estas redes son propiamente grafos en los que los usuarios se corresponden con los diferentes nodos y las aristas que los conectan se asocian con las relaciones entre ellos. Debido a esta característica intrínseca de este tipo de datos, una aplicación muy interesante del clustering espectral es la detección de comunidades en redes sociales.

Además de utilizarse para trabajar con datos provenientes de las redes sociales, las aplicaciones del clustering espectral no se quedan ahí: segmentación de imágenes y reconocimiento de objetos, agrupación de textos y clasificación de documentos, sistemas de recomendación y segmentación de clientes en diferentes comercios, análisis de tráfico y detección de anomalías en datos de red... Es decir, una larga lista de situaciones en las que los algoritmos de clustering espectral consiguen obtener grandes resultados.

El clustering espectral presenta numerosas ventajas, ya que son algoritmos fáciles de aplicar y que pueden resolverse eficazmente con programas de álgebra lineal estándar. Además, como hemos comentado previamente, se puede utilizar cuando los datos no cumplen determinados requisitos que sí imponen otros algoritmos, como la convexidad de los datos.

También debemos destacar que los algoritmos de clustering espectral son costosos desde el punto de vista computacional para grandes conjuntos de datos, ya que es necesario calcular los valores y vectores propios de grandes matrices de datos. Este aumento del tamaño de las matrices implica que la complejidad aumenta y la precisión disminuye considerablemente. Además, al igual que en otras técnicas de clustering, debemos indicar desde un primer momento el número k de clusters que debemos detectar, por lo que la calidad de los clusters obtenidos depende en gran medida de el valor que hayamos elegido para este k.

En definitiva, los algoritmos de clustering espectral son algoritmos que gozan de una gran popularidad debido a un comportamiento mejorado en determinados casos respecto a los algoritmos tradicionales, así como a la proliferación de conjuntos de datos asociados por su naturaleza a la representación en forma de grafos. Sin embargo, no debemos olvidar que son algoritmos que resultan costosos computacionalmente cuando el número de datos es muy elevado, por lo que no se suelen aplicar en determinados contextos de *big data*.

Capítulo 1

Teoría espectral de grafos

El objetivo del clustering es, dado un conjunto de puntos $x_1, ..., x_n \in \mathbb{R}^m$, agruparlos de manera que los puntos que pertenezcan a un mismo grupo presenten similitudes entre ellos y se diferencien de forma clara de aquellos que pertenecen a otro grupo. Puesto que tomamos los puntos en $\mathbb{R}^m$, estamos trabajando con un conjunto de datos con m variables. Si dotamos a estos puntos de una noción de similitud, denotada por $s_{i,j}$, podemos intentar resolver el problema haciendo uso de este valor.

Para representar estas similitudes entre los diferentes puntos podemos utilizar una herramienta matemática que resultar ser un modo natural de representar numerosos tipos de datos: los grafos.

1.1. Definiciones básicas de grafos

Un grafo es un conjunto de vértices y un conjunto de arcos que conectan pares de vértices. Estos arcos pueden ser dirigidos o no dirigidos e incluso se les puede asociar pesos determinados. En nuestro caso denotaremos al grafo con la notación $G = (V, E)$, siendo $v_i \in V$ cada uno de los vértices. Cada uno de estas vértices está asociado a una observación dentro de nuestro conjunto de datos, y cada arco perteneciente a E conecta dos vértices v_i y v_j si la similitud entre v_i y v_j, denotada por $s_{i,j}$, es positiva o mayor que un cierto umbral.

Trabajaremos con un grafo ponderado, por lo que al arco que une los vértices v_i y v_j le asociaremos el peso $w_{i,j} = s_{i,j}$. Si dos vértices no están unidos por un arco escribiremos $w_{i,j} = 0$. Utilizando el grafo definido de esta forma podemos reformular el planteamiento del clustering que hemos hecho en un principio transformándolo en la búsqueda de una partición del grafo tal que los arcos entre puntos del mismo grupo tengan pesos asociados altos; y aquellos puntos que pertenezcan a grupos distintos no estén unidos o lo hagan mediante arcos que presentan pesos bajos.

En resumen, trabajaremos con un grafo no dirigido $G = (V, E)$ donde $V = \{v_1, ..., v_n\}$ y cada arco en E que une dos vértices tiene un peso asociado $w_{i,j} > 0$. Recogeremos los pesos de cada arco en la siguiente matriz:

Definición 1.1. *El peso asociado a cada arco lo podemos representar de manera compacta con la matriz de adyacencia ponderada* $W = (w_{i,j})_{i,j=1,...,n}$.

Utilizando esta matriz podemos definir el grado de un vértice y la matriz de grados:

Definición 1.2. *El grado de un vértice $v_i \in V$ se define como la suma de los pesos de las aristas que lo conectan con otros vértices:*

$$d_i = \sum_{j=1}^{n} w_{i,j}$$

Definición 1.3. *La matriz de grados D es la matriz diagonal con elementos diagonales los grados de los vértices del grafo: $d_1, ..., d_n$.*

Si consideramos ahora un subconjunto de vértices $A \subset V$, podemos medir el *tamaño* de este subconjunto de dos maneras diferentes:

- $|A| := |\{i \mid v_i \in A\}|$: es decir, el número de vértices de A.
- $vol(A) = \sum_{i \in A} d_i$: es decir, la suma de los grados de sus vértices.

Incluiremos a continuación la noción de conectividad dentro de un grafo:

Definición 1.4. *Diremos que un subconjunto $A \subset V$ es conexo si cualquier par de vértices de A se pueden conectar por un camino tal que todos los puntos intermedios de ese camino son también vértices de A.*

Para presentar la siguiente definición denotaremos al complementario de nuestro subconjunto de vértices A como $\overline{A}$.

Definición 1.5. *El subconjunto A se denomina componente conexa si es un subgrafo conexo tal que no existen arcos entre los elementos de A y los de $\overline{A}$.*

Definición 1.6. *$A_1, ..., A_k$ forman una partición del grafo A si se verifica que $A_i \cap A_j = \emptyset \ \forall i, j$ y $A_1 \cup A_2 \cup ... \cup A_k = V$.*

Introduciremos en esta primera sección la noción de vector indicador, puesto que la utilizaremos posteriormente:

Definición 1.7. *Dado un subconjunto $A \subset V$ de vértices, denotaremos el vector $\mathbb{1}_A = (f_1, ..., f_n) \in \mathbb{R}^n$ como el vector indicador del conjunto A, tomando valores $f_i = 1$ si $v_i \in A$ y $f_i = 0$ si $v_i \notin A$*

1.2. Funciones y grafos de similitud

En esta sección presentaremos diferentes construcciones que nos permiten convertir nuestro conjunto de puntos $x_1, ..., x_n \in \mathbb{R}^m$ en un grafo con pesos asociados $s_{i,j}$ en los arcos. El objetivo de los grafos que estamos construyendo es el de representar de manera fiel los entornos de cada uno de los puntos, por lo que es importante elegir una función de similitud que sea capaz de modelar estas relaciones locales. La función de similitud escogida para asociar estos pesos $s_{i,j}$ debe cumplir que los entornos locales inducidos por la propia función mantengan la relación que presentan los datos originales. Un ejemplo de este tipo es la función de similitud Gaussiana:

$$s(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

Esta función invierte la relación entre la distancia de los dos puntos y la puntuación que se les asocia; es decir, cuanto mayor es la distancia entre los puntos que estamos considerando menor es la puntuación que se les asocia. La función de similitud Gaussiana presenta numerosas y muy variadas características que resultan ser útiles para los objetivos que estamos buscando, como por ejemplo:

- **Sensibilidad local:** esta función es localmente sensible a la distancia entre los puntos, ya que el valor que toma (la similitud asociada a los dos puntos) disminuye suavemente a medida que aumenta la distancia entre ellos: resulta sencillo comprobar que esta función decrece respecto a la distancia y que además oscila entre cero y uno. Esta característica nos permite captar estructuras y relaciones locales dentro de los datos, premisa que buscábamos a la hora de elegir nuestra función de similitud.
- **Parametrización:** la expresión de esta función presenta un único parámetro σ que controla la influencia de los puntos vecinos. Por tanto, la capacidad de poder ajustar el valor de σ nos permite adaptarlo a las características específicas de los datos, lo cual proporciona cierta capacidad de adaptación a diferentes conjuntos de datos y a diversos algoritmos de aprendizaje automático
- **Simplicidad matemática:** la formulación matemática es sencilla, y por tanto también resulta computacionalmente eficiente calcular estas puntuaciones de similitud entre puntos.

En resumen, es una función capaz de capturar relaciones complejas entre las diferentes observaciones de un conjunto de datos, lo que la hace adecuada para diversas tareas de aprendizaje automático, y en nuestro caso particular para el clustering.

Una vez que he explicado la función de similitud que utilizaré a lo largo del trabajo, presentaré a continuación los tres grafos de similitud más habituales, comentando en cada uno de ellos cuál es su construcción, en qué casos funcionan mejor o peor y cuál es la manera más adecuada de elegir los diferentes parámetros que aparecen a la hora de construirlos. Aunque no existe una gran cantidad de resultados teóricos que nos permitan afirmar cuál de estos parámetros elegir a la hora de enfrentarnos a un determinado problema, sí que he podido encontrar en [13] diferentes recomendaciones para su elección.

1.2.1. Grafo de ε-vecindad

Cada vértice de este grafo está conectado a los vértices que se encuentran dentro de una bola de radio ε, donde ε es un valor real que debe ajustarse para captar la estructura local de los datos. Puesto que la distancia entre todos los puntos que están conectados tiene una escala muy similar (como máximo ε), no consideraremos en este caso asociar pesos a cada arco, y trabajaremos por tanto con un grafo no ponderado.

La dificultad principal que nos encontramos cuando utilizamos este tipo de grafo es la elección del ε más adecuado. Esta dificultad se presenta en particular cuando trabajamos con conjuntos de datos en diferentes "escalas", es decir, en aquellos casos en los que las distancias entre puntos son distintas en las diferentes regiones del espacio en el que estemos trabajando.

A la hora de elegir el valor de ε debemos asegurarnos que el grafo resultante sea un grafo conexo. Para determinar el valor más pequeño de ε que verifica esta condición debemos elegir ε como la longitud máxima entre todas las aristas de un árbol de expansión mínimo del grafo con el que estamos trabajando.

Definición 1.8. *Se define un árbol de expansión mínimo de un grafo no dirigido y conexo G como un subconjunto del grafo tal que todos los vértices de G están conectados entre ellos con el mínimo número de aristas posible.*

Sin embargo, debemos tener en cuenta que utilizando esta forma de elegir ε podemos obtener un valor muy elevado si el conjunto de datos contiene *outliers*, puesto que elegiremos un ε lo suficientemente grande para que estos *outliers* también estén conectados. Algo similar puede ocurrir cuando trabajamos con conjuntos de datos en los que los clusters están muy diferenciados y alejados, puesto que en estos casos elegiremos de nuevo un ε que una los puntos entre los diferentes clusters, eligiendo un ε demasiado grande como para representar las relaciones locales de los datos.

Para poder visualizar la importancia que tiene el valor de ε a la hora de construir estos grafos, he decidido generar 50 observaciones en $\mathbb{R}^2$, tomando de manera aletoria 100 observaciones de una variable aleatoria con distribución uniforme en el intervalo $(0, 1)$ y distribuyéndolas en dos columnas. He construido tres grafos de este tipo a partir de estos datos, modificando el valor de ε. El código utilizado se encuentra en el apéndice A.1.

$$\varepsilon_1 = 0{,}15 \ ; \ \varepsilon_2 = 0{,}25 \ ; \ \varepsilon_3 = 0{,}5$$

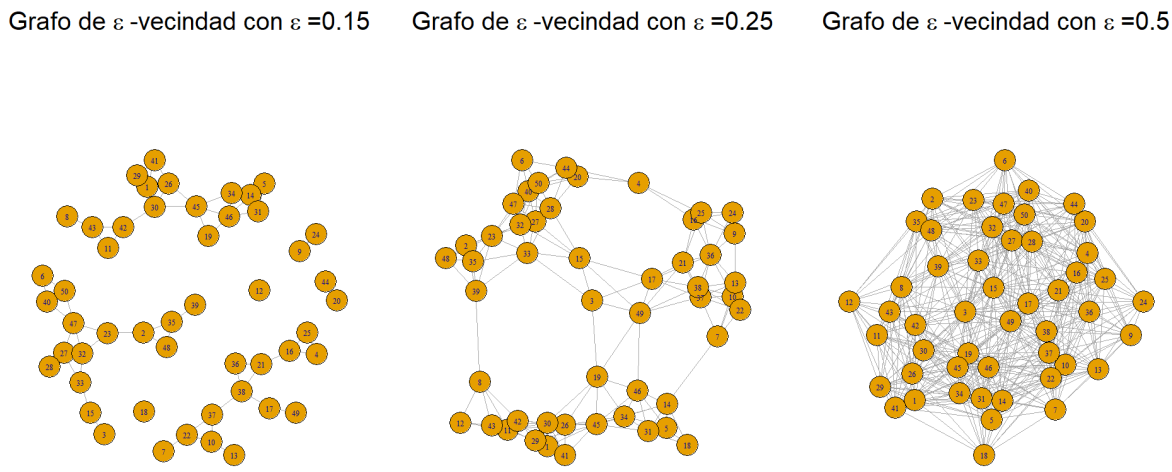

Figura 1.1: Grafos de ε-vecindad

En los tres grafos podemos comprobar la importancia de la elección de ε:

- En el primer caso, con $\varepsilon = 0{,}15$, obtenemos un grafo no conexo con 3 componentes conexas y numerosos puntos aislados, lo cuál no nos interesa puesto que al ser puntos aislados no tenemos información alguna acerca de sus puntos vecinos.
- En el segundo caso, con $\varepsilon = 0{,}25$, obtenemos un grafo conexo con más aristas que el anterior en el cual podemos intuir tres clusters bien diferenciados.
- En el último caso, con un valor de $\varepsilon = 0{,}5$, el número de aristas es muy alto, por lo que no nos aporta información acerca de las relaciones locales de los datos.

En conclusión, un ε muy pequeño puede generar un grafo no conexo y con muy pocas aristas; mientras que un ε demasiado elevado genera más aristas de la necesarias, disminuyendo la capacidad que tiene el grafo de aportarnos información.

1.2.2. Grafo de k-vécinos más próximos

En este grafo cada vértice está conectado a sus k vecinos más cercanos, donde k es un número entero el cuál debemos determinar para poder representar de la mejor manera posible las relaciones locales de los datos. Sin embargo, la definición de este grafo presenta un problema: la relación de vecindad no es simétrica, y esto nos lleva a un grafo dirigido. Para convertir este grafo dirigido a uno no dirigido tenemos dos opciones:

- La primera opción es pasar por alto que los arcos son dirigidos y tratar directamente el grafo como uno no dirigido, es decir, v_i y v_j estarán conectados si el vértice j se encuentra entre los k más próximos respecto de i o viceversa.
- La segunda opción es construir el *grafo de k-vécinos más próximos mutuos*; es decir, unir con arcos sólo aquellos vértices $\{v_i, v_j\}$ tales que uno esté entre los k vécinos más próximos del otro y viceversa.

En los dos casos anteriores, una vez que hemos construido todos los arcos asociamos a cada uno de ellos un peso dado por la similitud $s_{i,j}$ de los vértices correspondientes.

Este tipo de grafos solventan el problema que aparecía en el grafo de ε-vecindad respecto a los datos en diferentes "escalas", puesto que son capaces de conectar los puntos que se encuentran dentro de diferentes regiones del plano, aunque estas tengan distintas densidades. Sin embargo, el grafo de k-vécinos más próximos tiende también a conectar puntos pertenecientes a regiones con densidades distintas, lo cuál no resulta ser muy deseable. Una forma de solventar este problema es utilizar el grafo de k-vécinos más próximos mutuos, puesto que este grafo presenta un comportamiento a medio camino entre los dos anteriores: es capaz de trabajar en diferentes "escalas" (caso del grafo de k-vécinos más próximos), pero no conecta puntos que pertenezcan a dos regiones con densidades diferentes (caso del grafo de ε-vecindad).

Cuando trabajamos con este tipo de grafos el parámetro k debe ser elegido de tal forma que el grafo resultante sea conexo, o al menos tenga ciertas componentes conexas y la menor cantidad posible de puntos aislados. La elección de este k con conjuntos de datos que no sean excesivamente grande se puede hacer manualmente, pero con un conjunto de datos de gran tamaño deberíamos empezar probando con valores de k cercanos a $log(n)$, siendo n el número de puntos con los que estamos trabajando.

La elección de este valor k cercano a $log(n)$ proviene de unos resultados teóricos recogidos en [2], los cuáles estudian una cota para el valor de la variable $\hat{k}_n(S)$, la cual se define como el menor número entero k, tal que que el grafo de *k-vecinos más próximos mutuo* es conexo. El comportamiento de esta variable se estudia teóricamente en el artículo mencionado y se prueba, bajo ciertas hipótesis acerca de la distribución de la que provienen los datos, que existe una constante c tal que $\hat{k}_n(S) \leq c \cdot log(n)$ casi seguro para un tamaño de n suficientemente grande.

Un factor a tener en cuenta en la elección del k es que el grafo de k-vecinos mutuos presenta muchas menos aristas en comparación con el grafo usual de k-vecinos, ya que en el caso del primero de ellos la relación de pertenecer a los k-vecinos más próximos debe ser mutua. Por tanto, si elegimos el primero de los grafos frente al segundo debemos elegir un k considerablemente mayor que en el segundo, puesto que de lo contrario podemos obtener un grafo con una cantidad de aristas muy baja.

A la hora de construir este grafos he utilizado de nuevo los datos asociados a las 50 observaciones en $\mathbb{R}^2$ de la sección anterior. En primer lugar necesitamos determinar el valor de k, y en este caso la primera elección ha sido un valor de k cercano a $log(n)$, como hemos visto previamente. Puesto que tenemos un tamaño muestral $n = 50$, entonces $log(50) = 3{,}912$, por lo que he elegido $k = 4$. Los grafos obtenidos son los siguientes:

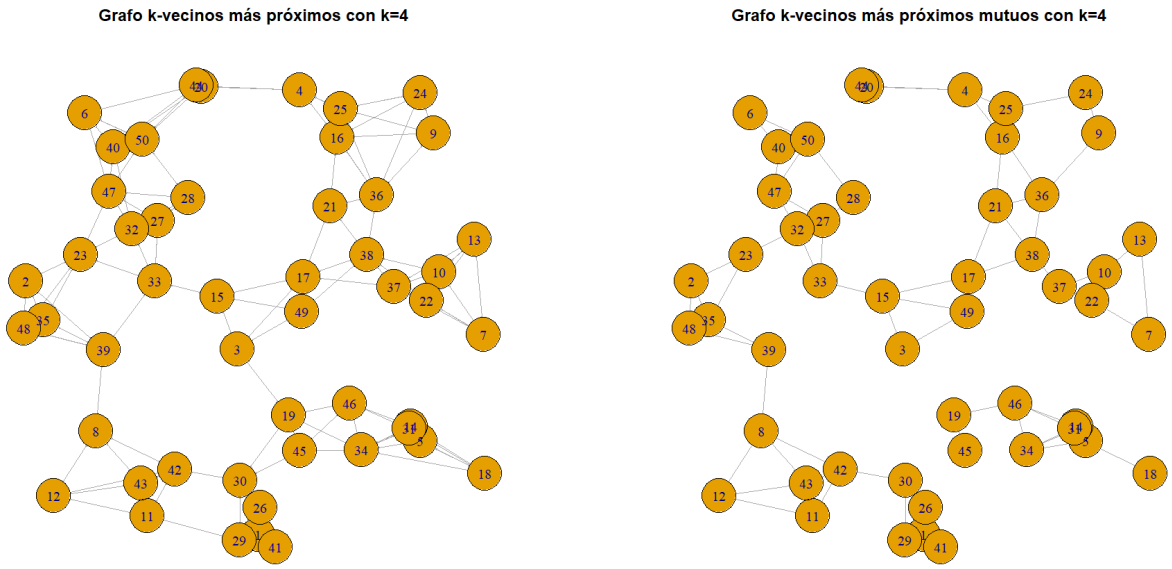

Figura 1.2: Grafos 4-vecinos más próximos

Resulta rápido comprobar a simple vista que el grafo de k-vecinos mutuos presenta un número de aristas inferior si lo comparamos con el número de aristas que presenta el grafo estándar (el de la izquierda). Por ello, he decidido volver a representar ambos grafos modificando el valor de k para el grafo de k-vecinos mutuos, eligiendo $k = 7$. Con esta modificación obtenemos los siguientes grafos:

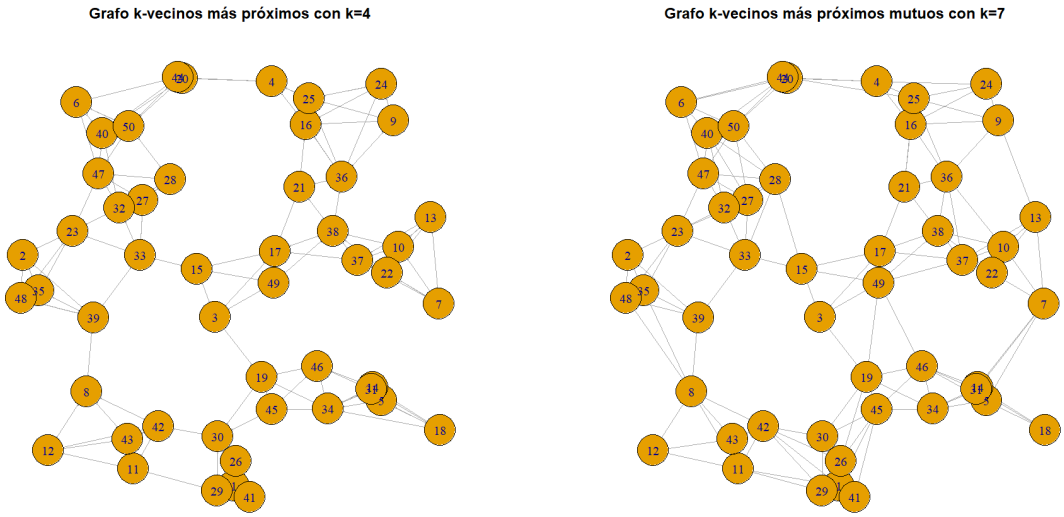

Figura 1.3: Grafo 4-vecinos más próximos y 7-vecinos más próximos mutuos

Al modificar este valor de k podemos comprobar que aparecen numerosas aristas nuevas en el grafo de k-vecinos mutuos, consiguiendo de esta forma que ambos grafos sean similares.

1.2.3. Grafo completamente conectado

En el caso de este último grafo conectamos todos los vértices que presentan una similitud positiva y asociamos a cada arco este peso. Este tipo de grafo se suele construir utilizando como función de similitud la función Gaussiana presentada anteriormente. Como hemos comentado esta función nos permitía asociar pesos altos a aquellos puntos que se encontraban muy cercanos en el espacio, mientras que se les asociaba pesos prácticamente nulos cuando la distancia entre ambos puntos aumentaba. La desventaja de utilizar este tipo de grafos radica en que la matriz resultante no es una matriz dispersa, lo cuál aumenta el coste computacional de los cálculos a realizar.

En este tipo de grafo el parámetro que debemos determinar es el σ de la función de similitud Gaussiana. La búsqueda de este parámetro se debe hacer con la misma intención de antes: modelar las relaciones locales de tal forma que la función aplicada a los puntos cercanos a un punto dado presenten valores próximos a 1. Las reglas generales que se suelen utilizar para que este conjunto de puntos con similitud cercana a 1 respecto de un punto dado no sea ni muy grande ni muy pequeño es tomar σ como la distancia media de cada punto a su k-ésimo vecino, eligiendo $k \sim log(n)$. Otra elección del valor de σ puede ser el ε obtenido como la arista de longitud máxima del árbol de expansión mínimo.

1.3. Matrices Laplacianas

A lo largo de la sección anterior hemos presentado los diferentes grafos de similitud con los que podemos trabajar y las diferentes elecciones que debemos tomar a la hora de construirlos. Suponiendo que hayamos elegido el grafo con los parámetros adecuados que se ajusten mejor a nuestros datos, es momento de empezar a trabajar con diferentes matrices que podemos obtener a partir de estos grafos.

El campo de estudio de estas matrices que vamos a construir, las cuales toman el nombre de matrices Laplacianas, se denomina Teoría Espectral de Grafos, y en las siguientes páginas iremos explicando las diferentes matrices Laplacianas que podemos generar a partir de un grafo. Los grafos con los que vamos a trabajar son grafos no dirigidos ponderados con matriz de pesos $W = (w_{i,j})$.

Presentaremos en primer lugar la matriz Laplaciana no normalizada:

Definición 1.9. *La matriz Laplaciana no normalizada se define como la diferencia de la matriz de grados D menos la matriz de pesos W.*

$$L = D - W$$

Utilizando esta definición podemos demostrar numerosas propiedades relativas a esta matriz. La demostración tanto de esta proposición como de las dos siguientes se basan en las recogidas en el artículo [13], incluyendo el desarrollo de alguna de las partes que aparecen sin completar o como indicaciones.

Proposición 1.10. *La matriz Laplaciana L satisface las siguientes propiedades:*

1. *Para todo vector $f \in \mathbb{R}^n$ se verifica:*

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

2. *L es simétrica y semi-definida positiva.*

3. *El autovalor más pequeño de L es 0, y su autovector correspondiente es $\mathbb{1}$.*

4. *L tiene n autovalores reales no negativos tales que:*

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$$

Demostración.

1. Utilizando la definición de $L = D - W$ y teniendo en cuenta la construcción de D, la cual es una matriz diagonal con el elemento diagonal i-ésimo el grado del vértice i; podemos expresar $f^T L f$ de la siguiente forma

$$f^T L f = f^T D f - f^T W f = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{i,j}$$

   Si a esta última expresión le sumamos de nuevo a ella misma obtenemos:

$$\sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{i,j} = \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{i,j} + \sum_{j=1}^{n} d_j f_j^2\right)$$

   Si utilizamos ahora la definición del grado de un vértice ($d_i = \sum_{j=1}^{n} w_{i,j}$) llegamos a la expresión pedida:

$$\begin{aligned} f^T W f &= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{i,j} + \sum_{j=1}^{n} d_j f_j^2\right) = \\ &= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{i,j} + \sum_{j=1}^{n}\sum_{i=1}^{n} w_{i,j} f_j^2\right) = \\ &= \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i^2 - f_i f_j + f_j^2) = \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2 \end{aligned} \tag{1.1}$$

2. La simetría de la matriz L la deducimos de su construcción ($L = D - W$), ya que la matriz D es una matriz diagonal (y por tanto simétrica); y la matriz W es simétrica porque estamos trabajando con un grafo G no dirigido, por lo que $w_{i,j} = w_{j,i}$.

   Utilizando la expresión del apartado anterior comprobamos que:

$$f^T L f = \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2 \geq 0 \ \forall v \in \mathbb{R}^n$$

   Esto se debe a que los pesos $w_{i,j}$ verifican ser ≥ 0, por lo que la matriz L es semidefinida positiva.

3. Puesto que la matriz L es semidefinida positiva, todos sus autovalores son ≥ 0. En este apartado vamos a probar que en particular $\lambda = 0$ es autovalor y que su autovector asociado es $\mathbb{1}$. Para ello debemos comprobar que:

$$L\mathbb{1} = 0 \iff (D - W)\mathbb{1} = 0 \iff D\mathbb{1} = W\mathbb{1}$$

$$D\mathbb{1} = (d_1, ..., d_n)^T \ ; \ W\mathbb{1} = (\sum_{j=1}^{n} w_{1,j}, ..., \sum_{j=1}^{n} w_{n,j})^T = (d_1, ..., d_n)^T$$

Puesto que $D\mathbb{1} = W\mathbb{1}$, se verifica que $L\mathbb{1} = 0$ y por tanto hemos demostrado que 0 es autovalor y $\mathbb{1}$ su autovector asociado.

4. Utilizando los tres apartados anteriores llegamos a la conclusión que la matriz L tiene autovalores no negativos y reales, ya que L es una matriz simétrica y semidefinida positiva. En particular hemos visto que el 0 es autovalor, por lo que podemos afirmar que los autovalores $\lambda_1, ..., \lambda_n$ de L verifican:

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$$

□

La siguiente proposición hace referencia a la relación existente entre el número de componentes conexas del grafo G que estamos estudiando y el espectro de L.

Proposición 1.11. *Sea G un grafo no dirigido ponderado con pesos no negativos, entonces la multiplicidad k del autovalor 0 de la matriz L coincide con el número de componentes conexas $A_1, ..., A_k$ del grafo. En este caso, el subespacio propio asociado al autovalor 0 está generado por los vectores indicadores $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$.*

Demostración. Sea l=*"nº de componentes conexas"*, comenzaremos con $l = 1$. Este caso se corresponde con un grafo en el que existe una única componente conexa, y por tanto es un grafo conexo. Queremos probar que $l = k$, y sabemos por la proposición anterior que el 0 es autovalor de la matriz L. Por ello, tomaremos f autovector asociado al autovalor 0 y comprobaremos que este vector solo puede ser el vector $\mathbb{1}$, el cual ya hemos probado que es autovalor. De esta forma comprobaremos que la multiplicidad $k = l = 1$. Sabemos que se cumple la siguiente igualdad:

$$f^T L f = \sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2 = 0$$

Puesto que los elementos $w_{i,j}$ son no negativas, esta suma vale 0 si se anulan todos los términos $w_{i,j}(f_i - f_j)^2$, lo cual nos lleva a dos posibilidades:

- v_i y v_j no están conectados, luego $w_{i,j} = 0$.
- Si v_i y v_j están conectados, entonces $w_{i,j} > 0$, por lo que $f_i = f_j$

Si repetimos este argumento para todos los vértices i, j que están conectados en el grafo concluimos que el autovector f debe ser constante para todos aquellos vértices que se pueden conectar mediante un camino en el grafo.

Teniendo en cuenta que estamos trabajando con un grafo en el que existe una única componente conexa, entonces $f = \mathbb{1}$; es decir, el autovalor 0 tiene al autovector $\mathbb{1}$ como único autovector asociado, y por tanto la multiplicidad k del autovalor 0 es 1 y coincide con el número de componentes conexas ($l = k = 1$). Además, este autovector verifica ser el vector indicador de la componente conexa.

Si consideramos ahora el caso en el que el grafo presenta l componentes conexas, podemos asumir sin pérdida de generalidad que los vértices están ordenados en función de a la componente conexa a la que pertenecen, obteniendo así una matriz de adyacencia W diagonal por bloques.

Calculando la matriz $L = D - W$ resulta sencillo comprobar que esta matriz L es también diagonal bloques, denotando a estos bloques con L_i en relación a la componente conexa que hagan referencia. Puesto que estamos trabajando con l componentes conexas, cada uno de estos bloques L_i se corresponde con la matriz Laplaciana asociada a la i-ésima componente conexa del grafo. Puesto que este L_i es la matriz Laplaciana de un grafo conexo, sabemos que cada bloque L_i tiene un autovector 0 de multiplicidad 1, y el autovector asociado es 1.

Finalmente, sabemos que el espectro de la matriz L es la unión de espectros de las matrices $L_1, ..., L_l$, por lo que la matriz L tiene el autovalor 0 con multiplicidad $k = l$. Además, los autovectores asociados a este autovalor son los correspondientes a los vectores constantes igual a 1 en los elementos pertenecientes a cada una de las componentes conexas y 0 en el resto; es decir, que el subespacio propio asociado al 0 está generado por los vectores indicadores $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$ □

1.4. Matrices Laplacianas normalizadas

Algunos de los algoritmos de clustering espectral que presentaremos posteriormente utilizan las matrices Laplacianas presentadas en la anterior sección pero normalizadas. Podemos encontrar dos definiciones diferentes de las matrices Laplacianas normalizadas.

$$L_{sym} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \quad ; \quad L_{rw} = D^{-1} L = I - D^{-1} W$$

La primera de ellas se denota L_{sym} puesto que es una matriz simétrica, mientras que la matriz L_{rw} lleva ese nombre debido a que está relacionada con la noción de *random walk*, formalización matemática de la trayectoria resultante de sucesivos paseos aleatorios. Profundizaremos en este concepto más adelante, pero antes de ello presentaremos diferentes propiedades de las matrices Laplacianas normalizadas.

Proposición 1.12. *Las matrices Laplacianas normalizadas satisfacen las siguientes propiedades:*

1. *Para todo $f \in \mathbb{R}^n$ tenemos:*

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

2. *λ es autovalor de L_{rw} con autovector asociado v si y sólo si λ es autovalor de L_{sym} con autovector $w = D^{\frac{1}{2}} v$.*

3. *λ es autovalor de L_{rw} con autovector asociado v si y sólo si λ y v resuelven el problema de valores propios generalizados $Lv = \lambda Dv$.*

4. *0 es autovalor de L_{rw} con autovector asociado $\mathbb{1}$. 0 es autovalor de L_{sym} con autovector asociado $D^{\frac{1}{2}}\mathbb{1}$.*

5. *L_{sym} y L_{rw} son matrices semidefinidas positivas y tienen n autovalores reales no negativos tales que:*
$$0 = \lambda_1 \leq ... \leq \lambda_n$$

Demostración.

1. Utilizando la definición de $L_{sym} = I - D^{-1/2}WD^{-1/2}$ y teniendo en cuenta la construcción de D y de W, podemos expresar $f^T L_{sym} f$ de la siguiente forma:
$$f^T L_{sym} f = f^T I f - f^T D^{-1/2} W D^{-1/2} f = \sum_{i=1}^{n} f_i^2 - \sum_{i,j=1}^{n} f_i f_j \frac{w_{i,j}}{\sqrt{d_i}\sqrt{d_j}}$$
En esta última expresión hemos tenido en cuenta que
$$D^{-1/2}WD^{-1/2} = \left(\frac{w_{i,j}}{\sqrt{d_i}\sqrt{d_j}}\right)_{i,j}$$
Operando de forma similar a la demostración de la Proposición 1.10 llegamos a:
$$\begin{aligned} f^T L_{sym} f &= \frac{1}{2}\left(\sum_{i=1}^{n} f_i^2 - \sum_{i,j=1}^{n} f_i f_j \frac{w_{i,j}}{\sqrt{d_i}\sqrt{d_j}} + \sum_{j=1}^{n} f_j^2\right) = \\ &= \frac{1}{2}\left(\sum_{i=1}^{n}\left(\frac{\sqrt{d_i} f_i}{\sqrt{d_i}}\right)^2 - \sum_{i,j=1}^{n} f_i f_j \frac{w_{i,j}}{\sqrt{d_i}\sqrt{d_j}} + \sum_{j=1}^{n}\left(\frac{\sqrt{d_j} f_j}{\sqrt{d_j}}\right)^2\right) = \end{aligned} \tag{1.2}$$
Si utilizamos ahora la definición del grado de un vértice ($d_i = \sum_{j=1}^{n} w_{i,j}$) llegamos a la expresión pedida:
$$\begin{aligned} f^T L_{sym} f &= \frac{1}{2}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{n} w_{i,j}\right)\left(\frac{f_i}{\sqrt{d_i}}\right)^2 - 2\sum_{i,j=1}^{n} f_i f_j \frac{w_{i,j}}{\sqrt{d_i}\sqrt{d_j}} + \sum_{j=1}^{n}\left(\sum_{i=1}^{n} w_{i,j}\right)\left(\frac{f_j}{\sqrt{d_j}}\right)^2\right) = \\ &= \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}\left(\left(\frac{f_i}{\sqrt{d_i}}\right)^2 - 2\frac{f_i f_j}{\sqrt{d_i}\sqrt{d_j}} + \left(\frac{f_j}{\sqrt{d_j}}\right)^2\right) = \\ &= \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}\left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right)^2 \end{aligned} \tag{1.3}$$

2. λ es autovalor de L_{sym} con autovector $w = D^{1/2}v \iff L_{sym}w = \lambda w \iff_{w=D^{1/2}v}$
$(I - D^{-1/2}WD^{-1/2})D^{1/2}v = \lambda D^{1/2}v \iff (D^{1/2} - D^{-1/2}W)v = \lambda D^{1/2}v \iff_{\times D^{-1/2}}$
$$\iff (I - D^{-1}W)v = \lambda v \iff L_{rw}v = \lambda v \iff$$
$\iff$ λ es autovalor de L_{rw} con autovector asociado v.

3. λ es autovalor de L_{rw} con autovector asociado $v \iff L_{rw}v = \lambda v \iff$

$$\iff (I - D^{-1}W)v = \lambda v \iff {}_{\times D}\ (D - W)v = \lambda Dv \iff Lv = \lambda Dv \iff$$

$\iff \lambda$ y v resuelven el problema de valores propios generalizados.

4. Para demostrar lo pedido comprobaremos que se verifica la expresión $L_{rw}\mathbb{1} = 0$

$$L_{rw}\mathbb{1} = 0 \iff (I - D^{-1}W)\mathbb{1} = 0 \iff \mathbb{1} = D^{-1}W\mathbb{1}$$

Si estudiamos la expresión $D^{-1}W\mathbb{1}$ resulta sencillo comprobar que $D^{-1}W = (\frac{w_{i,j}}{d_i})_{i,j}$. Si multiplicamos esta expresión por el vector $\mathbb{1}$ llegamos a

$$D^{-1}W\mathbb{1} = (\frac{\sum_{j=1}^{n} w_{i,j}}{d_i})_i = (\frac{d_i}{d_i})_i = \mathbb{1}$$

Hemos probado que el 0 es autovalor de la matriz L_{rw} con autovector asociado $\mathbb{1}$, y utilizando el apartado (2) resulta inmediato demostrar que entonces 0 es autovalor de L_{sym} con autovector $D^{1/2}\mathbb{1}$

5. L_{sym} es semidefinida positiva puesto que es simétrica y hemos probado en el apartado (1) que se verifica para todo $f \in \mathbb{R}^n$ la siguiente desigualdad:

$$f^T L_{sym} f = \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}\left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right)^2 \geq 0$$

Por tanto todos sus autovalores $\lambda_i \geq 0$ y en particular hemos probado que 0 es autovalor, por lo que se verifica la expresión: $0 = \lambda_1 \leq ... \leq \lambda_n$

L_{rw} es semidefinida positiva puesto que utilizando el apartado (2) sabemos que los autovalores de L_{rw} coinciden con los de L_{sym}, por lo que también es semidefinida positiva y se verifica $0 = \lambda_1 \leq ... \leq \lambda_n$.

□

Utilizando estas propiedades de la Proposición 1.12, podemos probar una proposición muy similar a la Proposición 1.11, la cual de nuevo relaciona la multiplicidad k del autovalor 0 para las matrices Laplacianas normalizadas con el número de componentes conexas del grafo asociado.

Proposición 1.13. *Sea G un grafo no dirigido ponderado con pesos no negativos, entonces la multiplicidad k del autovalor 0 de ambas matrices (L_{rw} y L_{sym}) coincide con el número de componentes conexas $A_1, ..., A_k$ del grafo G. Además se cumple que:*

- *Para L_{rw}, el subespacio propio asociado al autovalor 0 está generado por los vectores indicadores $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$.*
- *Para L_{sym}, el subespacio propio asociado al autovalor 0 está generado por los vectores indicadores $D^{\frac{1}{2}}\mathbb{1}_{A_1}, ..., D^{\frac{1}{2}}\mathbb{1}_{A_k}$.*

Demostración. La demostración de esta proposición es análoga a la Proposición 1.11 utilizando los distintos apartados probados en la Proposición 1.12. □

Capítulo 2

Algoritmos de clustering espectral

En este segundo capítulo presentaremos diferentes algoritmos de clustering espectral, justificando teóricamente los diferentes procedimientos llevados a cabo. Todos estos algoritmos de clustering espectral tienen dos objetivos comunes:

- Dividir el grafo en dos o más componentes de tal forma que los subgrafos obtenidos tengan un número similar de vértices
- Conseguir que el peso de las aristas que unen puntos dentro de un mismo subgrafo sea elevado y el peso de aquellas que unen vértices de subgrafos diferentes sea ínfimo.

Siguiendo la búsqueda de estos dos objetivos presentaremos en primer lugar el método de Fiedler, de cuyo planteamiento derivan algunos de los algoritmos que veremos posteriormente.

2.1. Método de Fiedler

A lo largo de las secciones 1.3 y 1.4 hemos probado que la matriz Laplaciana L, así como las matrices normalizadas L_{sym} y L_{rw} son matrices simétricas y semidefinidas positivas, con un autovalor $\lambda_1 = 0$ de multiplicidad k, el cual varía en función del número de componentes conexas del grafo asociado.

Sin embargo, el autovalor 0 no es el único autovalor de la matriz Laplaciana que nos aporta información sobre el grafo. En 1973, M. Fiedler definió en [5] el concepto de *conectividad algebraica de un grafo*:

Definición 2.1. *La conectividad algebraica de un grafo se define como el segundo autovalor más pequeño de la matriz Laplaciana, y se denota con* λ_2.

Este autovalor también se conoce como *salto espectral* o *gap*, y nos aporta información sobre la conectividad del grafo en general. A medida que el valor de λ_2 se hace más pequeño esto implica que el grafo adquiere una estructura más modular, es decir, las conexiones son sólidas entre los nodos de un mismo módulo pero escasas y más débiles entre nodos en diferentes módulos.

Esta desconexión entre las componentes de un grafo es el objetivo del clustering, y en notación matemática la búsqueda de esta *desconexión* consiste en buscar una partición del grafo, por lo que presentaremos a continuación la noción de corte para una partición $P = (A_1, A_2)$:

Definición 2.2. *Para una partición $P = (A_1, A_2)$ de un grafo G definimos el corte de P, denotado en inglés por cut(P), como la suma de los pesos del conjunto de aristas que unen un vértice $v_1 \in A_1$ con un vértice $v_2 \in A_2$.*

$$cut(P) = cut(A_1, A_2) = \sum_{i \in A_1, j \in A_2} w_{i,j}$$

Utilizando esta definición y siguiendo los objetivos mencionados previamente, esta partición $P = (A_1, A_2)$ la debemos hacer de tal forma que el número de vértices esté balanceado y buscando:

- Minimizar el peso total de las aristas entre A_1 y A_2, es decir, minimizar *cut(P)* tal y como lo hemos definido en la Definición 2.2. Para conseguir este objetivo podemos determinar un vector y tal que $y_i = +1$ si el vértice $v_i \in A_1$ y tal que $y_i = -1$ si el vértice $v_i \in A_2$. De esta forma, minimizar el peso total de *cut(P)* es equivalente a minimizar la siguiente expresión:

$$\min_y \ y^T L y = \sum_{i,j=1}^{n} w_{i,j}(y_i - y_j)^2 = 2 \sum_{i \in A_1, j \in A_2} 4w_{i,j} \tag{2.1}$$

- Maximizar el peso total de las aristas dentro de cada una de las componentes.

Puesto que el problema de minimización planteado en la ecuación 2.1 es un problema de optimización NP, debemos buscar una relajación de este problema. La búsqueda de esta relajación la haremos utilizando el principio de Rayleigh, para lo cual previamente introduciremos la siguiente definición y el siguiente lema.

Definición 2.3. *Dada una matriz $A \in \mathbb{R}^{n \times n}$ simétrica y definida positiva, definimos el cociente de Rayleigh con la siguiente fórmula:*

$$R(x) = \frac{x^T A x}{x^T x} \ ; \ x \in \mathbb{R}^n$$

Lema 2.4. *El cociente de Rayleigh verifica la siguiente igualdad:*

$$\min_{x \neq 0} R(x) = \min_{||x||=1} R(x)$$

Demostración. Supongamos que $x^* = (x_1^*, ..., x_n^*)$ es un vector cualquiera que minimiza $R(x)$ y tal que $c^2 = x^{*2}_1 + ... + x^{*2}_n = ||x^*||^2$. Si consideramos ahora el vector

$$y = (y_1, ..., y_n) = (\frac{x_1^*}{c}, \frac{x_2^*}{c}, ..., \frac{x_n^*}{c}) = \frac{x^*}{c} \ ; \ ||y||^2 = \frac{x_1^2 + ... + x_n^2}{c^2} = 1$$

Tenemos por tanto que $||y|| = 1$ y sabemos que $\min_{x \neq 0} R(x) \leq \min_{||x||=1} R(x)$. Nos falta comprobar por tanto que se verifica $\min_{x \neq 0} R(x) \geq \min_{||x||=1} R(x)$:

$$\min_{x \neq 0} R(x) = R(x^*) = \frac{x^{*T} A x^*}{x^{*T} x^*} = \frac{(x^*/c)^T A (x^*/c)}{(x^*/c)^T (x^*/c)} = \frac{y^T A y}{y^T y} = R(y) \geq \min_{||x||=1} R(x)$$

□

Estos resultados relacionados con el cociente de Rayleigh, así como el *Principio de Rayleigh* que posteriormente presentaré, los he consultado en la referencia [4]. Volviendo a nuestro problema, puesto que la matriz L con la que estamos trabajando verifica ser simétrica y definida positiva, podemos replantear $\min_{y\neq 0}\ y^TLy$ con la notación del cociente de Rayleigh:

$$\min_{y\neq 0,||y||=1}\ y^TLy = \min_{||y||=1} R(y)$$

Presentaremos la siguiente definición y el siguiente teorema que nos servirán para demostrar el principio de Rayleigh.

Definición 2.5. *Una matriz cuadrada A es diagonalizable ortogonalmente si existe una matriz ortogonal Q y una matriz diagonal D tales que:*

$$A = Q^TDQ$$

Teorema 2.6. *(Teorema espectral para matrices simétricas) Sea A una matriz simétrica real. Entonces existe una matriz diagonal D y una matriz ortogonal Q tal que $A = Q^TDQ$. Los valores propios de A son las entradas λ_i de la matriz diagonal D y las filas de Q son los vectores propios de A.*

Este resultado lo podemos encontrar en númerosos libros de álgebra; en particular, yo he utilizado la referencia [1]. Utilizando el teorema espectral sabemos que en el caso de las matrices simétricas de coeficientes reales es posible encontrar una base ortonormal de $\mathbb{R}^n$ formada por vectores propios de A.

Teorema 2.7. *(Principio de Rayleigh) Sea A una matriz simétrica y definida positiva con autovalores $\lambda_1 \leq ... \leq \lambda_n$. Entonces se verifica que:*

$$\min_{x\neq 0,||x||=1} R(x) = \lambda_1$$

Demostración. Puesto que A es simétrica, entonces por el Teorema 2.6 sabemos que existe una matriz ortogonal Q y una matriz diagonal D de forma que $A = Q^TDQ$. Por tanto, podemos reescribir el cociente de Rayleigh como:

$$R(x) = \frac{x^TAx}{x^Tx} = \frac{x^TQ^TDQx}{x^Tx} = \frac{(Qx)^TD(Qx)}{x^Tx}$$

Resulta sencillo comprobar que si tomamos $y = Q^Tx$, entonces:

$$||Q^Tx||^2 = (Q^Tx)^T(Q^Tx) = x^TQQ^Tx = x^Tx = ||x||^2$$

Si tomamos $||x||^2 = 1$, entonces $||y||^2 = ||Q^Tx||^2 = 1$, y por tanto los valores que toma $R(y)$ son:

$$R(y) = R(Q^Tx) = \frac{(QQ^Tx)^TD(QQ^Tx)}{(Q^Tx)^T(QQ^Tx)} = \frac{x^TDx}{x^Tx} = \frac{\lambda_1x_1^2 + \ + \lambda_nx_n^2}{x_1^2 + \ + x_n^2} = \lambda_1x_1^2 + \ + \lambda_nx_n^2$$

Los valores que toma esta expresión se encuentran por tanto en el intervalo $[\lambda_1, \lambda_n]$, y en particular el mínimo se alcanza con $x = (1, 0, ..., 0)$. Por tanto hemos probado que:

$$\min_{x\neq 0,||x||=1} R(x) = \lambda_1$$

□

Utilizando este principio podemos reformular la ecuación (2.1) de tal forma que llegamos a la expresión:

$$\min_{y\neq 0,||y||=1} \; y^T L y = \lambda_1 \tag{2.2}$$

Siendo λ_1 el autovalor más pequeño de la matriz L, sabemos que este $\lambda_1 = 0$, y por tanto el valor mínimo se alcanza con el autovector $y^* = \mathbb{1}$. Sin embargo, en la ecuación 2.2 no tenemos en cuenta que el número de vértices en cada subgrafo esté balanceado. Para ello podemos tomar de nuevo el vector y definido como $y_i = +1$ si el vértice $v_i \in A_1$, $y_i = -1$ si el vértice $v_i \in A_2$, y añadir una restricción nueva: $y^T \mathbb{1} = 0$.

Utilizando de nuevo el cociente de Rayleigh presentaremos el teorema de Courant-Fischer, cuya demostración he consultado en [3]:

Teorema 2.8. *(Courant-Fischer) Si A es una matriz simétrica con valores propios $\lambda_1 \leq ... \leq \lambda_n$ y los correspondientes vectores propios $f_1, ..., f_n$. Entonces:*

$$\lambda_1 = \min_{x\neq 0} R(x)$$

$$\lambda_2 = \min_{x\neq 0, x\perp f_1} R(x)$$

$$\vdots$$

$$\lambda_k = \min_{x=0, x\in S_{k-1}^{\perp}} R(x)$$

donde S_{k-1} es el subespacio generado por los autovectores $f_1, ..., f_{k-1}$.

Demostración. Tomando un valor de $k > 1$, por el teorema espectral para las matrices reales simétricas, sabemos que los vectores propios $f_1, ..., f_n$ los podemos encontrar ortonormales. Además, la condición $x \in S_{k-1}^{\perp}$ implica que $x \perp f_i$ para todo $i \in \{1, ..., k-1\}$.

Utilizando la base ortonormal $f_1, ..., f_n$ podemos expresar x como combinación lineal de los $n-k$ últimos vectores: $x = \sum_{i=k}^n c_i f_i$. Además, sabemos que podemos expresar $A = Q^T D Q$, con las filas de Q los vectores $f_1, ..., f_n$ escogidos como base ortonormal. Estudiando el cociente de Rayleigh para este vector x llegamos a:

$$\begin{aligned} R(x) &= \frac{(\sum_{i=k}^n c_i f_i)^T A (\sum_{i=k}^n c_i f_i)}{(\sum_{i=k}^n c_i f_i)(\sum_{i=k}^n c_i f_i)} = \frac{(Q(\sum_{i=k}^n c_i f_i))^T D (Q(\sum_{i=k}^n c_i f_i))}{\sum_{i=k}^n c_i^2} = \\ &= \frac{(0, ..., 0, c_k, ..., c_n)^T D (0, ..., 0, c_k, ..., c_n)}{\sum_{i=k}^n c_i^2} = \\ &= \frac{(0, ..., 0, \lambda_k c_k, ..., \lambda_n c_n)^T (0, ..., 0, c_k, ..., c_n)}{\sum_{i=k}^n c_i^2} = \frac{\sum_{i=k}^n \lambda_i c_i^2}{\sum_{i=k}^n c_i^2} \geq \frac{\lambda_k \sum_{i=k}^n c_i^2}{\sum_{i=k}^n c_i^2} = \lambda_k \end{aligned} \tag{2.3}$$

Hemos probado por tanto que para un vector $x \in S_{k-1}^{\perp}$ se verifica que $R(x) \geq \lambda_k$. Si tomamos en particular $x = f_k$ resulta sencillo comprobar que $c_k = 1$ y $c_i = 0$ para $i \neq k$, por lo que $R(x) = \lambda_k$. □

Puesto que en nuestro caso el autovector $f_1 = \mathbb{1}$, la condición de perpendicularidad $y \perp \mathbb{1}$ es equivalente a la expresión anterior $y^T \mathbb{1} = 0$, y llegamos a:

$$\min_{\|y\|_2=1, y^T\mathbb{1}=0} \; y^T L y = \lambda_2 \tag{2.4}$$

Partiendo de esta ecuación sabemos que el autovector en el que se alcanza este mínimo es el autovector asociado al autovalor λ_2, y se denomina *vector de Fiedler*. Utilizando este vector definiremos la partición de Fiedler de un grafo:

Definición 2.9. *Sea G un grafo y sea L su matriz Laplaciana asociada. Sea v_2 el vector de Fiedler de esta matriz L. Entonces la partición de Fiedler de este grafo, denotada por (A_1, A_2), se define de la siguiente forma:*

$$A_1 = \{i \in V : v_2(i) < 0\} \ ; \ A_2 = \{i \in V : v_2(i) > 0\}$$

En su forma más básica el método Fiedler afirma que podemos conseguir una partición en dos subgrafos separando los vértices según el signo de los valores del vector de Fiedler, ya que asociamos cada elemento de este vector a un vértice. En el caso de $v_2(i) = 0$ se puede asociar a uno de los dos subconjuntos arbitrariamente, aunque en el caso de que este vector de Fiedler contenga una proporción de ceros muy elevada puede indicar que este no es el método más apropiado para realizar esta partición.

El método presentado en la Definición 2.9 se utiliza para la división de un grafo en dos, consiguiendo así dos clusters. Si lo que buscamos es un mayor número de clusters sería necesario iterar este procedimiento sobre los respectivos clusters generados. Este algoritmo se conoce como algoritmo de clustering de Fiedler extendido y se utilizan los l autovectores asociados a los l primeros autovalores de L, de forma que en función del signo de estos elementos se pueden generar hasta 2^l clusters.

2.2. Clustering espectral no normalizado - *RatioCut*

El objetivo de esta segunda sección es presentar el algoritmo de clustering espectral no normalizado. Este algoritmo proviene de una relajación de un problema de minimización en el que se utiliza el concepto de *RatioCut*, por lo que presentaremos en primer lugar este concepto, para posteriormente presentar el problema que debemos minimizar y finalmente llegar al algoritmo de clustering que se obtiene con la relajación.

Si tomamos una partición $P = (A_1, ..., A_k)$ de nuestro grafo G, podemos generalizar para $k > 2$ el cálculo del valor *cut(P)* presentado en la Definición 2.2 con $k = 2$. Para ello denotaremos con $W(A, B)$ la suma de los pesos del conjunto de aristas que unen un vértice $v_1 \in A$ con un vértice $v_2 \in B$:

$$W(A, B) = \sum_{i \in A, j \in B} w_{i,j}$$

Definición 2.10. *Para una partición $P = (A_1, ..., A_k)$ de un grafo G definimos el corte de P, denotado en inglés por cut(P), como la suma del peso de las aristas entre cualquier conjunto de la partición y su complementario.*

$$cut(P) = cut(A_1, ..., A_k) = \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A_i})$$

Nótese que introducimos el término $\frac{1}{2}$ multiplicando para no contar cada arista dos veces. Utilizando este término podríamos tomar esta expresión como función objetivo a minimizar; sin embargo, no estaríamos teniendo en cuenta que los diferentes clusters deben tener cierta similitud en cuanto al número de puntos que contiene cada uno. Para conseguir esto propondremos la siguiente función objetivo:

Definición 2.11. *Para una partición $P = (A_1, ..., A_k)$ de un grafo G definimos la función RatioCut de la siguiente forma:*

$$RatioCut(A_1, ..., A_k) := \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{|A_i|} = \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{|A_i|}$$

La utilidad de esta función objetivo es que $RatioCut(A_1, ..., A_k)$ toma valores más pequeños a medida que los clusters aumentan de tamaño, puesto que en la Definición 2.11 el valor de $|A_i|$ aparece en el denominador. Sin embargo, introducir esta condición para conseguir clusters balanceados en cuanto a tamaño implica que buscar el mínimo de esta función se convierta en un problema NP, por lo que buscaremos una relajación de este problema. Al igual que hicimos con el método de Fiedler presentaremos en primer lugar el método para $k = 2$, generalizándolo después para $k > 2$.

A lo largo de las siguientes páginas presentaré numerosos lemas los cuales podemos encontrar como resultados en el artículo [13], referencia principal de este trabajo. En alguno de los casos estos resultados aparecen como igualdades sin demostrar, por lo que mi aportación en estos casos ha consistido en el desarrollo de estas igualdades.

Caso $k = 2$

En este primer caso donde $k = 2$ el objetivo es encontrar un subconjunto de vértices $A \subset V$ tal que minimice la expresión:

$$\min_{A \subset V} RatioCut(A, \overline{A}) \tag{2.5}$$

Para trabajar con este problema de minimización lo primero que vamos a hacer es definir un vector auxiliar $f = (f_1, ..., f_n) \in \mathbb{R}^n$.

Lema 2.12. *El vector $f = (f_1, ..., f_n) \in \mathbb{R}^n$ definido de la siguiente forma:*

$$f_i = \begin{cases} \sqrt{|\overline{A}|/|A|} & si\ v_i \in A \\ -\sqrt{|A|/|\overline{A}|} & si\ v_i \in \overline{A} \end{cases}$$

verifica:

1. *El vector f es ortogonal a $\mathbb{1}$ ($f \perp \mathbb{1}$)*
2. *$||f||^2 = n$*
3. *Sea L la matriz Laplaciana asociada al grafo con el que estamos trabajando, entonces:*
$$f^T L f = n \cdot RatioCut(A, \overline{A})$$

Demostración.

1. Resulta sencillo comprobar por la definición del vector f que se verifica:

$$f \cdot \mathbb{1} = \sum_{i=1}^{n} f_i = \sum_{i \in A} \sqrt{\frac{|\overline{A}|}{|A|}} - \sum_{i \in \overline{A}} \sqrt{\frac{|A|}{|\overline{A}|}} = |A|\sqrt{\frac{|\overline{A}|}{|A|}} - |\overline{A}|\sqrt{\frac{|A|}{|\overline{A}|}} =$$

$$\sqrt{|A|}\sqrt{|\overline{A}|} - \sqrt{|\overline{A}|}\sqrt{|A|} = 0$$

2.

$$||f||^2 = \sum_{i=1}^{n} f_i^2 = |A|\frac{|\overline{A}|}{|A|} + |\overline{A}|\frac{|A|}{|\overline{A}|} = |\overline{A}| + |A| = n$$

3. Utilizando este vector f y la matriz Laplaciana L llegamos a la expresión pedida:

$$\begin{aligned}
f^T L f &= \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2 = \\
&= \frac{1}{2}\sum_{i\in A, j\in\overline{A}} w_{i,j}\left(\sqrt{\frac{|\overline{A}|}{|A|}} + \sqrt{\frac{|A|}{|\overline{A}|}}\right)^2 + \frac{1}{2}\sum_{i\in\overline{A}, j\in A} w_{i,j}\left(-\sqrt{\frac{|\overline{A}|}{|A|}} - \sqrt{\frac{|A|}{|\overline{A}|}}\right)^2 = \\
&= \left(\sqrt{\frac{|\overline{A}|}{|A|}} + \sqrt{\frac{|A|}{|\overline{A}|}}\right)^2 \left(\frac{1}{2}\sum_{i\in\overline{A}, j\in A} w_{i,j} + \frac{1}{2}\sum_{i\in A, j\in\overline{A}} w_{i,j}\right) = \\
&= \left(\frac{|\overline{A}|}{|A|} + \frac{|A|}{|\overline{A}|} + 2\right) cut(A,\overline{A}) = cut(A,\overline{A})\left(\frac{|A|+|\overline{A}|}{|A|} + \frac{|A|+|\overline{A}|}{|\overline{A}|}\right) = \\
&= (|A| + |\overline{A}|)\left(\frac{cut(A,\overline{A})}{|A|} + \frac{cut(A,\overline{A})}{|\overline{A}|}\right) = n \cdot RatioCut(A,\overline{A})
\end{aligned} \tag{2.6}$$

□

En base a este lema podemos reescribir el problema de minimización de la ecuación (2.5) de la siguiente forma:

$$\min_{A\subset V} f^T L f \quad sujeto\ a\ \ f \perp \mathbb{1}\ \ ;\ \ f_i\ \ definido\ \ Lema\ 2{,}12\ \ ;\ \ ||f|| = \sqrt{n}$$

Si eliminamos la restricción sobre la definición de los f_i nos quedamos con un problema de minimización que podemos resolver utilizando el Teorema 2.8, de tal forma que el vector f que minimiza la expresión es el autovector correspondiente al segundo autovalor de menor tamaño de la matriz L, puesto que el autovalor de menor tamaño de L era el 0 con autovector asociado el $\mathbb{1}$. Este hecho implica que se verifique la condición de perpendicularidad $f \perp \mathbb{1}$, al igual que ocurría con el problema de minimización planteado para el método de Fiedler.

La modificación que plantea este algoritmo de clustering respecto al método de Fiedler consiste en tomar las coordenadas f_i del vector f como puntos en $\mathbb{R}$ y agruparlas en dos clusters $(C, \overline{C})$, utilizando un algoritmo de $k - medias$. De esta forma obtenemos la separación que buscábamos:

$$\begin{cases} v_i \in A & \text{si } f_i \in C \\ v_i \in \overline{A} & \text{si } f_i \in \overline{C} \end{cases}$$

A lo largo de esta sección hemos definido el algoritmo de clustering espectral no normalizado para $k = 2$, y al finalizar el siguiente apartado detallaremos los pasos de este algoritmo para cualquier valor de k.

Caso $k > 2$

Trabajando con un valor de $k > 2$, asociado al número de clusters que queremos encontrar, definiremos en primer lugar k vectores indicadores $h_i = (h_{i,1}, ..., h_{i,j}, ..., h_{i,n})\,;\, i \in \{1, ..., k\}$.

Lema 2.13. *Los vectores h_i, con $i \in \{1, ..., k\}$ y definidos de la siguiente forma:*

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_i|} & si\ v_j \in A_i \\ 0 & si\ v_j \notin A_i \end{cases}$$

son ortonormales entre ellos.

Demostración. Los vectores están bien definidos puesto que $A_1, ..., A_k$ son una partición del grafo y por tanto $A_i \cap A_j = \emptyset\ \forall i, j \in \{1, ..., k\}\,;\, i \neq j$. Teniendo en cuenta cómo hemos construido estos vectores resulta sencillo comprobar que son ortonormales entre ellos:

$$h_i^T \cdot h_j = \begin{cases} 0 & \text{si } i \neq j \\ \sum_{l \in A_i} \frac{1}{|A_i|} = \frac{|A_i|}{|A_i|} = 1 & \text{si } i = j \end{cases}$$

□

Una vez hemos definido estos k vectores y sabiendo que son ortonormales podemos construir la matriz $H \in \mathbb{R}^{n \times k}$, cuyas columnas son los vectores indicadores previamente construidos. Por la propiedad de ortonormalidad de las columnas sabemos que $H^T H = I$. A continuación probaremos el siguiente lema:

Lema 2.14. *Sea L la matriz Laplaciana asociada al grafo G con el que estamos trabajando. Entonces se verifica:*

$$h_i^T L h_i = \frac{cut(A_i, \overline{A}_i)}{|A_i|}$$

Demostración. Para probar este lema utilizaremos la matriz $L = D - W$ y plantearemos el siguiente producto entre la matriz L y los vectores h_i, h_i^T:

$$h_i^T L h_i = h_i^T D h_i - h_i^T W h_i$$

Obtendremos el valor de estos vectores por pasos; en primer lugar

$$(h_i^T D)_j = \begin{cases} \sum_{l \neq i} \frac{w_{j,l}}{\sqrt{|A_i|}} & \text{si } j \in A_i \\ 0 & \text{si } j \notin A_i \end{cases}$$

Por tanto el resultado de multiplicar este vector fila $h_i^T D$ por el vector columna h_i da como resultado:

$$h_i^T D h_i = \sum_{l \in A_i\,;\, j \neq i} \frac{w_{l,j}}{|A_i|}$$

Si estudiamos ahora el producto $h_i^T W$, entonces el elemento j-ésimo de ese vector es:

$$(h_i^T W)_j = \begin{cases} \sum_{l=1}^{n} \frac{w_{l,j}}{\sqrt{|A_i|}} & \text{si } j \in A_i \\ 0 & \text{si } j \notin A_i \end{cases}$$

Por tanto si hacemos de nuevo el producto de este vector con h_i lo que obtenemos es:

$$h_i^T W h_i = \sum_{l \in A_i, j \in A_i} \frac{w_{l,j}}{|A_i|}$$

Si calculamos ahora la diferencia $h_i^T D h_i - h_i^T W h_i$ llegamos a la expresión pedida:

$$h_i^T L h_i = h_i^T D h_i - h_i^T W h_i = \sum_{l \in A_i \, ; \, j \neq i} \frac{w_{l,j}}{|A_i|} - \sum_{l \in A_i, j \in A_i} \frac{w_{l,j}}{|A_i|} = \sum_{l \in A_i, j \notin A_i} \frac{w_{l,j}}{|A_i|} = \frac{cut(A_i, \overline{A}_i)}{|A_i|}$$

□

Observación 2.15. *Utilizando este lema y la construcción de la matriz H resulta inmediato ver que este producto $h_i^T L h_i$ se corresponde con el valor en la posición (i, i) de la matriz $H^T L H$, es decir:*

$$h_i^T L h_i = (H^T L H)_{i,i} = \frac{cut(A_i, \overline{A}_i)}{|A_i|}$$

Teniendo en cuenta el Lema 2.14 y esta observación reformularemos la función *RatioCut*:

$$RatioCut(A_1, ..., A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{|A_i|} = \sum_{i=1}^{k} h_i^T L h_i = \sum_{i=1}^{k} (H^T L H)_{i,i} = tr(H^T L H)$$

Debido a esta expresión de la función objetivo como la traza de una matriz podemos replantear nuestro problema de minimización:

$$\min_{A_1, ..., A_k} tr(H^T L H) \quad sujeto\ a \ \ H^T H = I \ \ ; \ \ H = (h_1, ..., h_k)$$

Partiendo de este problema de optimización discreta, plantearemos una relajación eliminando la restricción sobre la construcción de H, es decir, la condición de que las columnas deben ser los vectores $h_1, ..., h_k$. Eliminando esta restricción nos quedaremos por tanto con el siguiente problema:

$$\min_{H \in \mathbb{R}^{n \times k}} tr(H^T L H) \quad sujeto \ \ a \ \ H^T H = I \tag{2.7}$$

Hemos convertido nuestro problema original en un problema estándar de minimización de la traza de una matriz. Para resolver este problema utilizaré los teoremas que presento a continuación, los cuales vienen recogidos en el artículo [11]:

Teorema 2.16. *Sea A una matriz simétrica de tamaño $n \times n$. Existe una matriz Z de tamaño $n \times n$ para la cual se verifica:*

$$Z^T Z = I_n \quad \ y \ \ tal \ \ que \quad \ Z^T A Z = diag(\lambda_1, \lambda_2, ..., \lambda_n)$$

donde $\lambda_1, \lambda_2, ..., \lambda_n$ son los autovalores del problema $Ax = \lambda x$. Las columnas de Z son los autovectores asociados a estos autovalores de la matriz A. Además, si A es definida positiva, entonces todos los valores propios λ_i son positivos.

Utilizando este teorema se deduce el siguiente, el cual aplicaremos en nuestro problema de minimización planteado en la ecuación (2.7).

Teorema 2.17. *Sea A una matriz simétrica de tamaño $n \times n$ y sea Y^* el conjunto de todas las matrices Y de tamaño $n \times k$ para las que $Y^TY = I_k$. Entonces:*

$$\min_{Y \in Y^*} tr(Y^TAY) = \sum_{i=1}^{k} \lambda_i \quad \Longrightarrow \quad \min_{Y \in Y^*} tr(Y^TAY) = tr(X^TAX)$$

con $X \in \mathbb{R}^{n \times k}$, $X^TX = I$, $X^TAX = diag(\lambda_1, \lambda_2, ..., \lambda_k)$ y tal que X se corresponde con las primeras k columnas de la matriz Z del Teorema 2.16.

Con estos dos teoremas podemos afirmar que el subespacio generado por los autovectores asociados al conjunto de los k autovalores más pequeños de A resulta aportar una solución óptima para el problema de minimización de la traza con restricciones de ortogonalidad planteado en la ecuación 2.7.

En nuestro caso en particular, eligiendo las columnas de la matriz $H \in \mathbb{R}^{n \times k}$ como los k autovectores asociados a los autovalores $\lambda_1 \leq ... \leq \lambda_k$ de L conseguiríamos minimizar nuestra función objetivo y por tanto solo nos faltaría encontrar los clusters que estábamos buscando.

Para conseguir este objetivo tomaremos cada una de las n filas de la matriz H como puntos $y_i \in \mathbb{R}^k \;\; i \in \{1, ..., k\}$ y utilizando un algoritmo de $k - medias$ agruparemos estos n puntos en k clusters $C_1, ..., C_k$. Teniendo en cuenta estas agrupaciones asignaremos cada vértice v_i a un cluster de forma que:

$$v_i \in A_i \iff y_i \in C_i$$

Hemos llegado de esta forma al algoritmo de clustering espectral no normalizado:

Algoritmo de clustering espectral no normalizado

Input: *datos con los que vamos a trabajar (n observaciones en $\mathbb{R}^m$) y el número k de clusters*

1. *Construimos un grafo G de similitud asociado a nuestro conjunto de datos y denotamos a la matriz W como la matriz de adyacencia.*
2. *Construimos la matriz laplaciana no normalizada L asociada a nuestro grafo.*
3. *Calculamos los primeros k autovalores de L y sus autovectores asociados: $u_1, ..., u_k$.*
4. *Construimos la matriz $U \in \mathbb{R}^{n \times k}$ cuyas columnas son los autovectores $u_1, ..., u_k$*
5. *Para $i = 1, ..., n$ tomamos $y_i \in \mathbb{R}^k$ como la fila i-ésima de la matriz U.*
6. *Agrupamos los n puntos $(y_i)_{i=1}^n$ utilizando el algoritmo de clustering de $k-medias$, construyendo $C_1, ..., C_k$.*
7. *Construimos los k clusters $A_1, ..., A_k$ de forma que:*

$$A_i = \{v_j | y_j \in C_i\}$$

2.3. Clustering espectral normalizado - *NCut*

Al igual que en la sección anterior, el objetivo es llegar a un algoritmo de clustering espectral partiendo de un problema de minimización para el cual previamente debemos fijar la función objetivo a considerar. En este caso trabajaremos con una nueva función que lleva el nombre de $NCut(A_1, ..., A_k)$, y viene definida con la siguiente expresión:

Definición 2.18. *Para una partición $P = (A_1, ..., A_k)$ de un grafo G definimos la función NCut de la siguiente forma:*

$$NCut(A_1, ..., A_k) := \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{vol(A_i)} = \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{vol(A_i)}$$

La notación $vol(A)$ ya la presentamos en la sección 1.1 y la definíamos como la suma de los grados de los vértices de A. Siguiendo unos pasos muy similares respecto al apartado anterior (2.2) presentaremos el camino a seguir para llegar desde el problema de minimización de la función $NCut(A_1, ..., A_k)$ hasta el algoritmo de clustering espectral normalizado. De nuevo he utilizado la referencia [13] para este apartado y he escrito y demostrado diferentes lemas asociados a resultados que aparecen sin demostrar en el artículo referenciado previamente.

Caso $k = 2$

El primer paso a tomar es definir un vector auxiliar $f = (f_1, ..., f_n) \in \mathbb{R}^n$.

Lema 2.19. *El vector $f = (f_1, ..., f_n) \in \mathbb{R}^n$ definido de la siguiente forma:*

$$f_i = \begin{cases} \sqrt{vol(\overline{A})/vol(A)} & si\ v_i \in A \\ -\sqrt{vol(A)/vol(\overline{A})} & si\ v_i \in \overline{A} \end{cases}$$

verifica:

1. *Sea D la matriz de grados asociada al grafo con el que estamos trabajando, entonces el vector Df es ortogonal a $\mathbb{1}$, es decir, $(Df)'\mathbb{1} = 0$.*
2. *Sea D la matriz presentada en el punto anterior, entonces:*
$$f^T Df = vol(V)$$
3. *Sea L la matriz Laplaciana asociada al grafo con el que estamos trabajando, entonces:*
$$f^T Lf = vol(V) \cdot NCut(A, \overline{A})$$

Demostración.

1. Escribiremos en primer lugar el vector resultante de la multiplicación matriz por vector Df:
$$(Df)_i = \begin{cases} d_i\sqrt{vol(\overline{A})/vol(A)} & \text{si } v_i \in A \\ -d_i\sqrt{vol(A)/vol(\overline{A})} & \text{si } v_i \in \overline{A} \end{cases}$$
Multiplicando este vector por el vector constante de 1's llegamos a:
$$(Df)^T\mathbb{1} = \sum_{i\in A} d_i\sqrt{\frac{vol(\overline{A})}{vol(A)}} - \sum_{i\in\overline{A}} d_i\sqrt{\frac{vol(A)}{vol(\overline{A})}} = vol(A)\sqrt{\frac{vol(\overline{A})}{vol(A)}} - vol(\overline{A})\sqrt{\frac{vol(A)}{vol(\overline{A})}} = 0$$

2. Si trasponemos el vector columna Df definido en el apartado interior y lo multiplicamos ahora por f obtenemos:

$$(Df)^T f = f^T D^T f = f^T Df = \sum_{i\in A} d_i \frac{vol(\overline{A})}{vol(A)} + \sum_{i\in \overline{A}} d_i \frac{vol(A)}{vol(\overline{A})} = vol(\overline{A}) + vol(A) = vol(V)$$

3. Utilizando este vector f y la matriz Laplaciana L llegamos a la expresión pedida:

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

$$\frac{1}{2}\sum_{i\in A, j\in\overline{A}} w_{i,j}\left(\sqrt{\frac{vol(\overline{A})}{vol(A)}} + \sqrt{\frac{vol(A)}{vol(\overline{A})}}\right)^2 + \frac{1}{2}\sum_{i\in\overline{A}, j\in A} w_{i,j}\left(-\sqrt{\frac{vol(\overline{A})}{vol(A)}} - \sqrt{\frac{vol(A)}{vol(\overline{A})}}\right)^2 =$$

Operando de la misma forma que en el apartado 3 del Lema 2.12 llegamos a:

$$= (vol(A) + vol(\overline{A}))\left(\frac{cut(A,\overline{A})}{vol(A)} + \frac{cut(A,\overline{A})}{vol(\overline{A})}\right) = vol(V)\cdot NCut(A,\overline{A})$$

□

Este lema nos permite de nuevo reescribir el problema de minimización:

$$\min_{A\subset V} f^T Lf \;\; sujeto\ a\;\; f \perp \mathbb{1} \;\;;\;\; f_i\ definido\ Lema\ 2{,}19 \;\;;\;\; f^T Df = vol(V)$$

Si eliminamos la restricción sobre los valores que debe tomar f_i, y escribimos $g := D^{1/2}f$, entonces obtenemos el problema:

$$\min_{g\in\mathbb{R}^n} g^T D^{-1/2} L D^{1/2} g \;\; sujeto\ a\;\; g \perp D^{1/2}\mathbb{1} \;\;;\;\; ||g||^2 = vol(V) \tag{2.8}$$

Observación 2.20. *Al sustituir $f = D^{-1/2}g$ aparece la matriz $D^{-1/2}LD^{-1/2}$. Esta matriz ya la definimos en el apartado 1.4 como L_{sym}, por lo que podemos escribir la expresión 2.8 de una forma más compacta:*

$$\min_{g\in\mathbb{R}^n} g^T L_{sym} g \;\; sujeto\ a\;\; g \perp D^{1/2}\mathbb{1} \;\;;\;\; ||g||^2 = vol(V)$$

Aplicando el Teorema 2.8 la solución a este problema de minimización viene dada por el autovector g asociado al segundo autovalor de menor tamaño de la matriz L_{sym}, por lo que si sustituimos ahora $f = D^{-1/2}g$ lo que obtenemos es el autovector asociado al segundo autovalor de menor tamaño de L_{rw} (Proposición 1.12.2)

Caso $k > 2$

Si trabajamos con un número de clusters $k > 2$ vamos a definir k vectores indicadores $h_i = (h_{i,1}, ..., h_{i,j}, ..., h_{i,n})\,;\, i \in \{1, ..., k\}$.

Lema 2.21. *Los vectores h_i, con $i \in \{1, ..., k\}$ y definidos de la siguiente forma:*

$$h_{i,j} = \begin{cases} 1/\sqrt{vol(A_i)} & si\ v_j \in A_i \\ 0 & si\ v_j \notin A_i \end{cases}$$

verifican las siguientes afirmaciones:

1. *Sea D la matriz de grados asociada al grafo G con el que estamos trabajando, entonces se verifica:*
$$h_i^T D h_i = 1$$

2. *Sea $H \in \mathbb{R}^{n\times k}$ la matriz construida con los vectores indicadores h_i como columnas, y sea D la matriz de grados. Entonces se verifica que:*
$$H^T D H = I$$

3. *Sea L la matriz Laplaciana asociada al grafo G con el que estamos trabajando, entonces se verifica:*
$$h_l^T L h_l = \frac{cut(A_l, \overline{A}_l)}{vol(A_l)}$$

Demostración.

1. Comprobaremos en primer lugar que $h_i^T D h_i = 1$:
$$(h_i^T D)_j = \begin{cases} d_i/\sqrt{vol(A_i)} & \text{si } v_j \in A_i \\ 0 & \text{si } v_j \notin A_i \end{cases} \implies h_i^T D h_i = \sum_{j\in A_i} \frac{d_j}{vol(A_i)} = \frac{vol(A_i)}{vol(A_i)} = 1$$

2. En el punto anterior hemos probado que los elementos de la diagonal de la matriz $H^T D H$ son iguales a 1. Nos falta comprobar que $h_i^T D h_j = 0$ si $i \neq j$:
$$h_i^T D h_j = \sum_{l\in A_i, l\in A_j} \frac{d_l}{vol(A_i)} \frac{d_l}{vol(A_j)} = 0$$
Este sumatorio es igual a 0 puesto que $A_1, ..., A_k$ es una partición, y por tanto la intersección $A_i \cap A_j = \emptyset \ \forall i \neq j$

3. Utilizando la Proposición 1.10 sabemos que:
$$h_l^T L h_l = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}(h_{l,i} - h_{l,j})^2$$
Por tanto utilizando la definición de los vectores h_l llegamos a la expresión pedida:
$$h_l^T L h_l = \frac{1}{2} \sum_{i\in A_l \,;\, j\notin A_l} w_{i,j} \frac{1}{vol(A_l)} + \frac{1}{2} \sum_{i\notin A_l \,;\, j\in A_l} w_{i,j} \frac{1}{vol(A_l)} = \frac{\sum_{i\in A_l \,;\, j\notin A_l} w_{i,j}}{vol(A_l)} = \frac{cut(A_l, \overline{A_l})}{vol(A_l}$$

□

Una vez hemos definido estos k vectores construimos esta matriz H y podemos reescribir la expresión $NCut(A_1, ..., A_k)$:
$$NCut(A_1, ..., A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{vol(A_i)} = \sum_{i=1}^{k} h_i^T L h_i = Tr(H^T L H)$$

Utilizando esta expresión replantearemos el problema de minimización:
$$\min_{H\in\mathbb{R}^{n\times k}} tr(H^T L H) \quad sujeto \ \ a \ \ H^T D H = I \tag{2.9}$$

Si escribimos ahora $T = D^{1/2}H$ entonces $T^TT = (D^{1/2}H)^T(D^{1/2}H) = H^TDH$. Utilizando esta matriz T reescribimos de nuevo el problema (2.9):

$$\min_{T\in\mathbb{R}^{n\times k}} tr(T^TD^{-1/2}LD^{-1/2}T) \quad sujeto \ \ a \ \ T^TT = I \tag{2.10}$$

Al igual que en la Observación 2.20, podemos escribir la expresión (2.10) de una forma más compacta:

$$\min_{T\in\mathbb{R}^{n\times k}} tr(T^TL_{sym}T) \quad sujeto \ \ a \ \ T^TT = I$$

La solución a este problema la obtenemos aplicando el Teorema 2.17, el cual afirma que la solución es la matriz T cuyas columnas son los autovectores asociados a L_{sym}. Una vez que hemos obtenido estos autovectores encontramos dos algoritmos de clustering espectral normalizado diferentes, en función de los autovectores que usan.

2.3.1. Clustering espectral normalizado con L_{sym}

Si utilizamos directamente los autovalores de la matriz L_{sym} entonces el algoritmo de clustering resultante es el planteado en la referencia [10] por los autores Andrew Y.Ng, Michael I. Jordan y Yair Weiss. El algoritmo es el siguiente:

Algoritmo de clustering espectral normalizado

Input: *datos con los que vamos a trabajar (n observaciones en $\mathbb{R}^m$) y el número k de clusters*

1. *Construimos un grafo G de similitud asociado a nuestro conjunto de datos y denotamos a la matriz W como la matriz de adyacencia.*
2. *Construimos la matriz laplaciana normalizada L_{sym} asociada a nuestro grafo.*
3. *Calculamos los primeros k autovalores de L_{sym} y sus autovectores asociados: $u_1, ..., u_k$.*
4. *Construimos la matriz $U \in \mathbb{R}^{n\times k}$ cuyas columnas son los autovectores $u_1, ..., u_k$*
5. *Construimos la matriz $T \in \mathbb{R}^{n\times k}$ normalizando las filas de la matriz U; es decir, tomamos*
$$t_{i,j} = \frac{u_{i,j}}{(\sum_k u_{i,k}^2)^{1/2}}$$
6. *Para $i = 1, ..., n$ tomamos $y_i \in \mathbb{R}^k$ como la fila i-ésima de la matriz T.*
7. *Agrupamos los n puntos $(y_i)_{i=1}^n$ utilizando el algoritmo de clustering de $k-$medias, construyendo $C_1, ..., C_k$.*
8. *Construimos los k clusters $A_1, ..., A_k$ de forma que:*
$$A_i = \{v_j | y_j \in C_i\}$$

2.3.2. Clustering espectral normalizado con L_{rw}

Otra posibilidad reside en utilizar los autovectores de la matriz L_{sym} para obtener los autovectores de la matriz L_{rw}, utilizando las relaciones entre ambos estudiadas en la Proposición 1.12:

- En primer lugar, sabemos que $H = D^{-1/2}T$, por lo que la matriz H tiene como columnas los autovectores de la matriz L_{sym} multiplicados por la matriz $D^{-1/2}$. Utilizando el segundo apartado de la Proposición 1.12 sabemos que v es autovalor de L_{rw} si y sólo si w es autovector de L_{sym} y se verifica que $v = D^{-1/2}w$. Por tanto, la matriz H construida de esta forma contiene los autovectores asociados a los k primeros autovalores de la matriz L_{rw}
- Utilizando ahora el tercer apartado de esa Proposición 1.12 sabemos que los autovectores de L_{rw} resuelven el problema de valores propios generalizados $Lv = \lambda Dv$, por tanto el algoritmo plantea este problema de autovalores generalizados para obtener los elementos de la matriz H, aunque hemos visto que también se pueden obtener a partir de la matriz L_{sym}

Este es el algoritmo planteado por Jianbo Shi y Jitendra Malik en el artículo [12]:

Algoritmo de clustering espectral normalizado

Input: *datos con los que vamos a trabajar (n observaciones en $\mathbb{R}^m$) y el número k de clusters*

1. *Construimos un grafo G de similitud asociado a nuestro conjunto de datos y denotamos a la matriz W como la matriz de adyacencia.*
2. *Construimos la matriz laplaciana no normalizada L asociada a nuestro grafo.*
3. *Calculamos los primeros k autovalores generalizados del problema $Lu = \lambda Du$ y sus autovectores generalizados asociados: $u_1, ..., u_k$.*
4. *Construimos la matriz $U \in \mathbb{R}^{n \times k}$ cuyas columnas son los autovectores generalizados $u_1, ..., u_k$*
5. *Para $i = 1, ..., n$ tomamos $y_i \in \mathbb{R}^k$ como la fila i-ésima de la matriz U.*
6. *Agrupamos los n puntos $(y_i)_{i=1}^n$ utilizando el algoritmo de clustering de $k-$medias, construyendo $C_1, ..., C_k$.*
7. *Construimos los k clusters $A_1, ..., A_k$ de forma que:*

$$A_i = \{v_j | y_j \in C_i\}$$

2.4. Detalles prácticos

A lo largo de los dos últimos apartados hemos presentado 3 algoritmos diferentes de clustering espectral, por lo que pueden surgir numerosas dudas acerca de su funcionamiento o de cuándo es más adecuado elegir cada uno de ellos. Estas son preguntas que aparecen de manera natural al enfrentarnos a un problema real, por lo que en este apartado voy a intentar responder a varias de ellas.

2.4.1. Comparación de los 3 algoritmos

Una cuestión fundamental relacionada con el clustering espectral es la elección de la matriz Laplaciana asociada al grafo con el que estemos trabajando, por lo que debemos tener en cuenta en primer lugar si trabajamos con la matriz Laplaciana sin normalizar o la normalizada; y en segundo lugar, si elegimos el clustering espectral normalizado, debemos decidir si utilizar L_{rw} o L_{sym} para calcular los vectores propios. Por ello recordemos que en cada uno de los tres algoritmos de clustering espectral que hemos visto se utiliza una de estas matrices:

- *Clustering espectral no normalizado*: partimos de un problema de minimización de $RadioCut(A_1, ..., A_k)$ y lo resolvemos utilizando los autovalores de la matriz L.
- *Clustering espectral normalizado con L_{sym}:* buscamos minimizar $NCut(A_1, ..., A_k)$ utilizando los autovalores de la matriz L_{sym}.
- *Clustering espectral normalizado con L_{rw}*: para resolver el problema de minimización de la $NCut(A_1, ..., A_k)$ utilizamos los autovalores de la matriz L_{rw}.

Si recordamos los dos objetivos principales de un algoritmo de clustering son agrupar los puntos de tal manera que los puntos que se encuentran en diferentes clusters sean distintos, lo que matemáticamente podemos plantear como $\min_{A_1,...,A_k} cut(A_1, ..., A_k)$; así como conseguir que los puntos dentro de un mismo cluster presenten similitudes.

Ambas funciones ($RadioCut(A_1, ..., A_k)$ y $NCut(A_1, ..., A_k)$) incluyen en su formulación el primer objetivo, puesto que aparece en ambas $cut(A_i, \overline{A_i})$; por lo que debemos estudiar cuál de los tres algoritmos hace un mayor hincapié en la búsqueda del segundo objetivo. Para ello vamos a tener en cuenta las similitudes entre los puntos dentro de un mismo cluster, las cuáles podemos denotar con $W(A, A) = \sum_{i \in A, j \in A} w_{i,j}$

Lema 2.22. *Sea V el conjunto de vértices del grafo G con el que estamos trabajando, y sea $A \subset V$, entonces se verifica:*

$$W(A, A) = vol(A) - cut(A, \overline{A})$$

Demostración.

Desarrollando $W(A, A)$ llegamos a:

$$W(A, A) = \sum_{i \in A, j \in A} w_{i,j} = \sum_{i \in A, j \in V} w_{i,j} - \sum_{i \in A, j \notin A} w_{i,j} = W(A, V) - W(A, \overline{A})$$

Sabemos por definición que $W(A, \overline{A}) = cut(A, \overline{A})$, y resulta sencillo comprobar que

$$W(A, V) = \sum_{i \in A, j \in V} w_{i,j} = \sum_{i \in A} \sum_{j \in V} w_{i,j} = \sum_{i \in A} d_i = vol(A)$$

Por lo que se verifica que $W(A, A) = vol(A) - cut(A, \overline{A})$ □

Teniendo en cuenta este lema, maximizar la similitud entre puntos de un mismo cluster A se consigue maximizando $vol(A)$ y minimizando $cut(A, \overline{A})$. Escribiendo de nuevo la definición de la función $NCut$:

$$NCut(A_1, ..., A_k) := \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{vol(A_i)}$$

Podemos comprobar que al minimizar esta expresión lo que buscamos precisamente es minimizar los términos $cut(A_i, \overline{A}_i)$ y maximizar $vol(A_i)$, por lo tanto los dos algoritmos de clustering no normalizado si contemplan este segundo objetivo, ya que ambos buscan minimizar la función $NCut$.

Si estudiamos ahora el caso del algoritmo de clustering no normalizado podemos comprobar fácilmente que este segundo objetivo no está presente, puesto que la expresión que se busca minimizar es:

$$RatioCut(A_1, ..., A_k) := \sum_{i=1}^{k} \frac{cut(A_i, \overline{A}_i)}{|A_i|}$$

En este caso en el denominador aparece $|A_i|$, es decir, buscamos que el tamaño de los clusters esté balanceado pero sin tener en cuenta que los puntos pertenecientes a un mismo cluster deben ser puntos similares. Por tanto, a la hora de tomar la primera decisión acerca de que tipo de algoritmo usar, es recomendable utilizar un algoritmo de clustering espectral normalizado.

La última recomendación sobre qué algoritmo utilizar se basa en la Proposición 1.13, ya que está proposición afirma que los autovectores de la matriz L_{rw} son los vectores indicadores de las componentes conexas (si es que existen) del grafo con el que estamos trabajando. Sin embargo, los autovectores de la matriz L_{sym} se corresponden con estos mismos vectores multiplicados por la matriz $D^{1/2}$, lo que puede dar lugar a una peor interpretabilidad de los mismos.

En definitiva, teniendo en cuenta tanto la búsqueda de los dos objetivos principales de todo algoritmo de clustering como la interpratibilidad y eficiencia en la práctica, el algoritmo que reúne estos requisitos es el algoritmo de clustering espectral normalizado propuesto por Jianbo Shi y Jitendra Malik en el artículo [12].

2.4.2. Uso del algoritmo *k-means*

Los tres algoritmos que hemos desarrollado presentan un esquema bastante similar en el cuál buscamos los autovectores de las matrices Laplacianas para posteriormente confeccionar una matriz con ellas y tomar cada una de estas filas como los puntos originales con los que estábamos trabajando. La motivación a la hora de dar estos pasos viene dada por la relación vista tanto en la Proposición 1.11 como en la Proposición 1.13 entre autovectores y vectores indicadores.

Suponiendo el caso ideal en el que el grafo con el que estamos trabajando presente k componentes conexas desconectadas, entonces los autovectores de las matrices Laplacianas L y L_{rw} coinciden con los vectores indicadores de esas componentes conexas. Por tanto, si suponemos estar en esta situación, tomar las filas de la matriz que hemos construido nos lleva a encontrarnos vectores de la forma $(0, ..., 1, ..., 0)$ con el 1 en la posición i-ésima, indicando por tanto que este punto corresponde al cluster A_i.

Esta breve explicación nos permite entender la transformación que hemos realizado desde nuestros puntos originales a las filas de una matriz que contiene vectores indicadores (en la situación ideal) de los clusters que queremos generar. En esta situación ideal el uso del algoritmo de $k - means$ resultaría trivial puesto que agruparía todos los vértices que tuvieran el 1 en la misma posición; sin embargo, en los problemas reales no se da esta situación y por tanto el algoritmo $k-means$ agrupa las filas de la matriz según la distancia euclídea entre ellas, construyendo posteriormente los clusters al asociar cada una de estas filas con el punto x_i.

2.4.3. Determinación del número k de clusters

Seleccionar el número óptimo de clusters es un paso muy importante en los algoritmos de clustering, puesto que determinar la cantidad adecuada de grupos a generar nos permite obtener resultados más significativos. Existen diferentes métodos para seleccionar el número óptimo de clusters k generales a todos los algoritmos de clustering, por lo que en primer lugar presentaremos brevemente alguno de estos métodos y posteriormente me centraré en un método propio de los algoritmos de clustering espectral. Alguno de los métodos generales a todos los algoritmos son:

- *Coeficiente de Silhouette*: este valor es una medida de la similitud que presenta un punto x respecto a su propio cluster y en comparación con otros clusters; es decir, mide tanto la cohesión en un cluster como la separación entre él mismo y el resto de clusters. Se define utilizando los siguientes dos valores:

  - $a(x)$: distancia promedio de x a todos los demás puntos en el mismo cluster.
  - $b(x)$: distancia promedio de x a todos los demás puntos en el cluster más cercano.

  Utilizando estos dos valores se determina este coeficiente con la siguiente fórmula:

  $$s(x) = \frac{b(x) - a(x)}{max(a(x), b(x))}$$

  Este coeficiente oscila entre -1 y $+1$. Un valor alto de este coeficiente indica que el punto x está bien cohesionado con su propio cluster y mal cohesionado con los clusters vecinos. Para determinar el número de clusters k idóneo debemos comprobar cuál es el k con el que se verifica que la mayoría de los puntos de nuestro conjunto de datos presentan un valor alto de este coeficiente, y por tanto podemos fijar este k como el número de clusters que queremos conseguir.

- *Método del codo:* otro de los métodos más usados para determinar el valor idóneo de k consiste en ejecutar el algoritmo de clustering con el que estemos trabajando para diferentes valores de k y estudiar el valor de $WCSS$:

$$WCSS = \sum_{i=1}^{k} \sum_{x \in A_i} ||x - \mu_i||^2$$

  Este valor calcula la suma de las distancias al cuadrado de cada objeto a su centroide más cercano μ_i. Si representamos en el eje de las X el número de clusters k y en el eje de las Y este valor, el método del codo plantea que el número óptimo de clusters se encuentra en el punto de inflexión de esta curva.

Estos dos métodos son ampliamente conocidos y aplicables a los diferentes algoritmos de clustering; sin embargo, existe un método heurístico que permite determinar el k óptimo utilizando el concepto de *eigengap*. Con este concepto hacemos referencia a la diferencia en valor absoluto entre dos autovalores consecutivos del Laplaciano asociado a los datos con los que estamos trabajando.

Este método plantea que el número óptimo de clusters viene dado por el valor de k que maximiza esta diferencia entre valores propios consecutivos. El objetivo es por tanto, suponiendo que ordenamos los autovalores $\lambda_1 \leq ... \leq \lambda_n$, encontrar el k en el que se verifique que los valores $\lambda_1, ..., \lambda_k$ son muy pequeños, y en el que a partir del λ_{k+1} este valor aumenta considerablemente.

La justificación teórica del uso de este método la podemos encontrar en la Proposición 1.11: suponiendo que estamos en el caso ideal en el que existen k componentes conexas completamente desconectadas entre ellas en nuestro grafo, entonces la multiplicidad del autovalor 0 es igual a k ($\lambda_1 = ... = \lambda_k = 0$), mientras que $\lambda_{k+1} > 0$, por tanto existe este *eigengap*.

Capítulo 3

Interpretación probabilística de los algoritmos

Una nueva visión que podemos aportar al clustering espectral es introducir el concepto de *paseos aleatorios* en el grafo de similitud construido, de tal forma que podemos reconvertir nuestro problema de clustering en un problema de particionar un grafo consiguiendo que estos *paseos aleatorios* se mantengan el mayor tiempo posible en el mismo cluster, saltando lo menos posible de cluster en cluster. Este nuevo punto de vista nos va a permitir mostrar que estos algoritmos de clustering tienen un importante fundamento probabilístico.

3.1. *Paseos aleatorios* y cadenas de Markov

Antes de plantear la equivalencia teórica entre estos dos problemas vamos a aportar un marco teórico a este nuevo concepto, definiendo en primer lugar esta noción de *paseo aleatorio*:

Definición 3.1. *Un paseo aleatorio en un grafo (random walk en inglés) es un proceso aleatorio que comienza en un vértice y en cada paso se desplaza a otro vértice.*

- *Cuando el grafo es no ponderado, el vértice al que se desplaza se elige uniformemente al azar entre los vecinos del vértice actual.*
- *Cuando el grafo es ponderado, se desplaza a un vértice vecino con una probabilidad proporcional al peso de la arista que los une.*

Estos procesos de paseo aleatorio son un caso particular de procesos más generales llamados cadenas de Markov. Una cadena es un proceso en tiempo discreto en el que una variable aleatoria X_n va cambiando con el paso del tiempo. Las cadenas de Markov tienen la propiedad de que la probabilidad de que $X_n = j$ sólo depende del estado inmediatamente anterior del sistema X_{n-1}.

Un paseo aleatorio se puede representar mediante su matriz de transición P:

Definición 3.2. *La matriz de transición P es una matriz cuadrada $n \times n$ que denota la probabilidad de ir de cualquier vértice del grafo a cualquier otro vértice.*

$$P_{i,j} = P[\textit{partiendo de } v_i \textit{ , ir a } v_j]$$

En el caso de un grafo G ponderado, como vimos en la definición 3.1, la probabilidad de ir de un vértice v_i a un vértice v_j es proporcional al peso de la arista (i, j), y viene dado por la siguiente expresión:

$$P = (p_{i,j})_{i,j=1,...,n} \quad tal \quad que \quad p_{i,j} = \frac{w_{i,j}}{d_i}$$

Es decir, la matriz P se obtiene a partir del producto de matrices $P = D^{-1}W$. Si recordamos la definición de $L_{rw} = I - D^{-1}W$, podemos comprobar rápidamente la relación existente entre P y L_{rw}, ya que $L_{rw} = I - P$. Este es el motivo por el que justificábamos en el apartado 1.4 el nombre de la matriz L_{rw} por su estrecha relación con el concepto de *random walk*.

Esta matriz de transición nos permite estudiar la distribución de probabilidad que sigue nuestra posición en el grafo a medida que vamos dando cada uno de los pasos que componen el paseo aleatorio.

Definición 3.3. *Se dice que el vector $\pi = (\pi_1, ..., \pi_n)$ es una distribución de probabilidad si se verifica que $\pi_i \geq 0$ para todo $i \in \{1, ..., n\}$ y que $\sum_{i=1}^{n} \pi_i = 1$.*

Si empezamos con una distribución π en los n vértices del grafo (donde π es un vector de tamaño $n \times 1$) y damos un paso, entonces nuestra distribución sobre los nodos viene dada por el producto $P^T\pi$. Una de las propiedades de los paseos aleatorios que nos interesa es la distribución de nuestra posición en el grafo si realizamos un paseo aleatorio durante un número infinito de pasos. En este caso, comenzando en π y ejecutando el paseo aleatorio para un número infinito de pasos obtenemos una distribución sobre los nodos la cual se corresponde con el siguiente límite:

$$lim_{t\to\infty}(P^T)^t\pi$$

En general, esta distribución límite puede depender de la posición inicial en el grafo e incluso puede no existir; sin embargo, en determinadas condiciones la distribución límite siempre existe y es independiente de la posición de partida. Cuando se cumplen estas condiciones, esta distribución límite verifica la siguiente ecuación:

$$\pi = P^T\pi$$

Esta distribución lleva el nombre de distribución estacionaria de una cadena de Markov:

Definición 3.4. *Se dice que una distribución de probabilidad $\pi = (\pi_1, ..., \pi_n)$ es estacionaria para una Cadena de Markov con matriz de probabilidades de transición $P = (p_{i,j})$ si verifica $\pi = P^T\pi$*

El significado de esta distribución estacionaria π es que si la variable aleatoria inicial de nuestra cadena X_0 tiene una distribución π, entonces la distribución de X_n también es π $\forall n \in \mathbb{N}$; es decir, es una distribución que no cambia con el paso del tiempo.

Lema 3.5. *El vector*

$$\pi^* = (\frac{d_1}{vol(V)}, ..., \frac{d_n}{vol(V)})$$

es una distribución estacionaria para un paseo aleatorio sobre G.

Demostración. Para probar este lema debemos comprobar que se verifica la ecuación $\pi^* = P^T\pi^*$

$$\pi_j^* = \sum_{i=1}^{n} \frac{w_{i,j}}{d_i}\frac{d_i}{vol(V)} = \frac{\sum_{i=1}^{n} w_{i,j}}{vol(V)} = \frac{d_j}{vol(V)} = \pi_j^*$$

Por tanto se verifica que $\pi^* = P^T\pi^*$. □

Este lema nos permite conocer una distribución estacionaria de un paseo aleatorio por nuestro grafo G.

Relación entre $NCut$ y los *paseos aleatorios*

El marco teórico presentado a lo largo del apartado anterior nos permite plantear y demostrar la siguiente proposición, la cual muestra la relación existente entre los algoritmos de clustering espectral presentados y el concepto de paseo aleatorio. Esta proposición la podemos encontrar en el artículo de Marina Meilă y Jianbo Shi ([9]):

Definición 3.6. *Un grafo bipartito es un grafo cuyos vértices se pueden separar en dos conjuntos disjuntos, de manera que las aristas no pueden unir vértices de un mismo conjunto.*

Proposición 3.7. *Sea G un grafo conexo y no-bipartito. Asumimos que el paseo aleatorio definido con la sucesión de variables aleatorias $(X_t)_{t\in\mathbb{N}}$ comienza con X_0 siguiendo la distribución estacionaria π^*. Si para subconjuntos disjuntos $A, B \subset V$ denotamos por $P(B|A) := P(X_1 \in B|X_0 \in A)$, entonces se da la igualdad:*

$$NCut(A, \overline{A}) = P(A|\overline{A}) + P(\overline{A}|A)$$

Demostración. Para demostrar la igualdad queremos estudiar $P(A|\overline{A})$ (estudiaremos este primer sumando puesto que juegan un papel simétrico). Para calcular esta probabilidad lo que haremos es estudiar la siguiente probabilidad condicionada:

$$P(\overline{A}|A) = P(X_1 \in \overline{A}|X_0 \in A) = \frac{P(X_1 \in \overline{A}, X_0 \in A)}{P(X_0 \in A)}$$

Estudiaremos los dos términos del cociente por separado:

$$P(X_0 \in A, X_1 \in \overline{A}) = \sum_{i\in A, j\in\overline{A}} P(X_0 = i, X_1 = j) = \sum_{i\in A, j\in\overline{A}} \pi_i^* p_{i,j} =$$

$$= \sum_{i\in A, j\in\overline{A}} \frac{d_i}{vol(V)}\frac{w_{i,j}}{d_i} = \frac{cut(A, \overline{A})}{vol(V)}$$

$$P(X_0 \in A) = \sum_{i\in A} \pi_i^* = \sum_{i\in A} \frac{d_i}{vol(V)} = \frac{vol(A)}{vol(V)}$$

Calculando el cociente de ambos términos llegamos a la expresión pedida:

$$P(\overline{A}|A) = P(X_1 \in \overline{A}|X_0 \in A) = \frac{P(X_1 \in \overline{A}, X_0 \in A)}{P(X_0 \in A)} = \frac{\frac{cut(A,\overline{A})}{vol(V)}}{\frac{vol(A)}{vol(V)}} = \frac{cut(A, \overline{A})}{vol(A)}$$

□

Esta proposición nos muestra que al minimizar $NCut$ para una determinada partición $(A, \overline{A})$ también estamos minimizando la probabilidad de que un paseo aletorio que comienza en A salga de A, y viceversa. Es decir, hemos generado una partición de forma que el paseo aleatorio que comienza en una de las partes tiende a permanecer en él, y rara vez pasa de un conjunto de la partición a otro.

3.2. Distancia de conmutación

En esta sección comenzaremos presentando dos magnitudes básicas que se pueden calcular a partir de la definición de una cadena de Markov y su matriz de probabilidades de transición P: el tiempo medio de primer paso y el tiempo medio de conmutación.

Definición 3.8. *El tiempo medio de primer paso $m(k|i)$ es el número medio de pasos que necesita un caminante aleatorio para alcanzar el estado k por primera vez, cuando parte del estado i.*

Este tiempo $m(k|i)$ se puede definir de la siguiente forma:

$$m(k|i) = \begin{cases} m(k|k) = 0 & sii = k \\ m(k|i) = 1 + \sum_{j=1}^{n} p_{i,j} m(k|j) & sii \neq k \end{cases}$$

Estas cantidades pueden calcularse utilizando algunos algoritmos específicos relacionados con las cadenas de Markov o utilizando la pseudoinversa de la matriz Laplaciana, la cual presentaremos posteriormente. Esta medida no es simétrica, sin embargo podemos estudiar el siguiente valor:

$$n(i,j) = m(j|i) + m(i|j)$$

Podemos comprobar fácilmente que esta expresión cumple las siguientes condiciones:

$$n(i,j) \geq 0 \ ; \ n(i,j) = 0 \iff i = j \ ; \ n(i,j) = n(j,i) \ ; \ n(i,j) \leq n(i,k) + n(k,j)$$

Es decir, esta expresión proporciona una medida de distancia entre cualquier par de vértices del grafo, conocida como *distancia de conmutación*, y a la cual denotaremos por $c_{i,j} = n(i,j)$.

Definición 3.9. *Se define la distancia de conmutación $c_{i,j}$ entre dos vértices v_i y v_j de un grafo como el número de pasos esperados que tarda un paseo aleatorio en viajar desde el vértice v_i hasta el vértice v_j y volver.*

La distancia de conmutación tiene una propiedad interesante puesto que el valor asociado a dos nodos disminuye cuando el número de caminos que los conectan aumentan y cuando la longitud de estos caminos disminuye. Es decir, debido a su definición esta medida decrece a medida que existen más caminos y más cortos entre ambos vértices. Otras distancias consideradas entre los vértices de grafos, como la distancia del *camino más corto* o *geodésica*, no captan el hecho de que los nodos fuertemente conectados están a menor distancia que los débilmente conectados; por lo que para los algoritmos de clustering, en los cuales nos interesa la relación local de los nodos con su entorno, resulta útil estudiar la distancia de conmutación entre vértices.

Debido a estos beneficios que presenta la distancia de conmutación frente a otras distancias conocidas entre vértices de un grafo, resulta interesante estudiar cómo podemos calcular esta distancia de conmutación entre dos vértices v_i y v_j. Para ello necesitaremos construir la pseudoinversa de la matriz laplaciana L. El concepto de pseudoinversa generaliza el concepto de inversa de una matriz a matrices que no son de rango completo o no son cuadradas.

Observación 3.10. *Recordamos que L no es invertible puesto que sabemos que $\lambda = 0$ siempre es un autovalor de esta matriz Laplaciana.*

En primer lugar, utilizando la descomposición ortogonal de la matriz L vista en el Teorema 2.6, sabemos que L admite una descomposición de la forma $L = Q^T \Lambda Q$, con Q matriz ortogonal cuyas filas son los autovectores de L y Λ matriz diagonal cuyos elementos diagonales son los autovalores $\lambda_1, ..., \lambda_n$. Utilizando esta descomposición definiremos la inversa generalizada L^- de la siguiente forma:

Definición 3.11. *Se define la inversa generalizada de L con la expresión*

$$L^- := Q^T \Lambda^- Q$$

donde Λ^+ es una matriz diagonal con elementos $1/\lambda_i$ si $\lambda_i \neq 0$ y 0 si $\lambda_i = 0$.

Podemos estudiar diversas propiedades de esta matriz inversa generalizada que utilizaremos posteriormente.

Teorema 3.12. *La inversa generalizada L^- de la matriz Laplaciana verifica las siguientes propiedades:*

1. *L^- es simétrica.*
2. *Si $(\lambda_i, \mu_i)_{i=1}^n$ son valores y vectores propios, respectivamente, de L; entonces:*
   - *Si $\lambda_i \neq 0$, $(\frac{1}{\lambda_i}, \mu_i)$ son valores y vectores propios, respectivamente, de L^-.*
   - *Si $\lambda_i = 0$, $(0, \mu_i)$ son valores y vectores propios, respectivamente, de L^-.*
3. *L^- es semidefinida positiva.*

Demostración.

1. $(L^-)^T = (Q^T \Lambda^- Q)^T = Q^T (\Lambda^-)^T (Q^T)^T = Q^T \Lambda^- Q = L^-$, por lo que L^- es simétrica.
2. Si multiplicamos L^- por un autovector μ_i de L con autovector $\lambda_i \neq 0$, recordando que las filas de Q son $\mu_i, ..., \mu_n$ y que $Q^T Q = I$, llegamos a:

$$L^- \mu_i = Q^T \Lambda^- Q \mu_i = Q^T \Lambda^- \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = Q^T \begin{bmatrix} 0 \\ \vdots \\ \frac{1}{\lambda_i} \\ \vdots \\ 0 \end{bmatrix} = \frac{1}{\lambda_i} \mu_i$$

   Hemos probado por tanto que $(\frac{1}{\lambda_i}, \mu_i)$ son valores y vectores propios, respectivamente, de L^-. El razonamiento es el mismo si $\lambda_i = 0$.

3. Según la Proposición 1.11 sabemos que L es definida positiva, por lo que todos sus autovalores $\lambda_i \geq 0$. Utilizando el punto anterior sabemos que L y L^- tienen los mismos autovalores, por lo que L^- también es semidefinida positiva.

□

Utilizando esta matriz $L^- = (l^-_{ij})_{i,j=1,...,n}$ podemos calcular los valores del tiempo medio de primer paso $m(k|i)$ y de la distancia de conmutación $c_{i,j}$. Los resultados que presentaré a continuación se pueden encontrar en el artículo [6].

Lema 3.13. *El tiempo medio de primer paso $m(k|i)$ se calcula con la siguiente expresión:*

$$m(k|i) = \sum_{j=1}^{n} (l^+_{i,j} - l^+_{i,k} - l^+_{k,j} + l^+_{k,k}) d_j$$

Teorema 3.14. *Sea $G = (V, E)$ un grafo conexo no dirigido. Denotemos por $c_{i,j}$ la distancia de conmutación entre el vértice v_i y el vértice v_j , por $L^+ = (l^+_{ij})_{i,j=1,...,n}$ la inversa generalizada de L, y por e_i al elemento i-ésimo de la base canónica. Entonces tenemos:*

$$c_{i,j} = vol(V)(l^-_{i,i} - 2l^-_{ij} + l^-_{jj}) = vol(V)(e_i - e_j)^T L^- (e_i - e_j)$$

Demostración. La distancia $c_{i,j}$ la podemos expresar como la siguiente suma:

$$c_{i,j} = n(i,j) = m(i|j) + m(j|i)$$

Utilizando el Lema 3.13 conocemos la expresión en función de la matriz L^- de los valores $m(i|j)$, por lo que podemos utilizar este desarrollo para llegar a la igualdad pedida:

$$\begin{aligned} c_{i,j} = m(i|j) + m(j|i) &= \sum_{k=1}^{n} (l^-_{j,k} - l^-_{j,i} - l^-_{i,k} + l^-_{i,i}) d_k + \sum_{k=1}^{n} (l^-_{i,k} - l^-_{i,j} - l^-_{j,k} + l^-_{j,j}) d_k = \\ &= \sum_{k=1}^{n} (l^-_{j,k} - l^-_{j,i} - l^-_{i,k} + l^-_{i,i} + l^-_{i,k} - l^-_{i,j} - l^-_{j,k} + l^-_{j,j}) d_k = (l^-_{i,i} + l^-_{j,j} - 2l^-_{i,j}) \sum_{k=1}^{n} d_k = \\ &= vol(V)(l^-_{i,i} + l^-_{j,j} - 2l^-_{i,j}) \end{aligned} \tag{3.1}$$

Si expresamos el último término de la igualdad 3.2 en forma matricial llegamos a la igualdad pedida:

$$c_{i,j} = vol(V)(l^-_{i,i} - 2l^-_{ij} + l^-_{jj}) = vol(V)(e_i - e_j)^T L^- (e_i - e_j)$$

□

Si consideramos el espacio de nodos generado por $\{e_i\}_{i=1}^n$, asociando el vector $e_i = (0, ..., 1, ..., 0)$ con el vértice del grafo v_i; entonces esta expresión nos permite estudiar la distancia de conmutación entre los vértices del grafo. De hecho, podemos ir un paso más allá, ya que podemos considerar esta distancia como una distancia euclídea sobre los vértices del grafo. Es decir, podemos construir una inmersión en $\mathbb{R}^n$ que lleve los vértices v_i del grafo a puntos $x'_i \in \mathbb{R}^n$ tales que las distancias euclídeas entre los puntos x'_i coinciden con la distancia de conmutación entre los vértices del grafo. Para probar esto presentaremos el siguiente lema:

Lema 3.15. *Considerando la descomposición ortogonal de $L^- = Q^T\Lambda^- Q$ y utilizando la transformación:*

$$\begin{cases} e_i = Q^T x_i \\ x'_i = (\Lambda^-)^{\frac{1}{2}} x_i \end{cases} \qquad \forall i \in \{1, ..., n\}$$

llegamos a la expresión

$$c_{i,j} = n(i,j) = vol(V)(x'_i - x'_j)^T(x'_i - x'_j) = vol(V)||x'_i - x'_j||^2$$

Demostración. Si consideramos la descomposición $L^- = Q^T\Lambda^- Q$ estudiada en la Definición 3.11, en la que Q es una matriz ortogonal que tiene como filas los autovectores de L^-; podemos desarrollar la expresión vista en el Teorema 3.14 y llegar a la igualdad pedida:

$$\begin{aligned} c_{i,j} &= vol(V)(e_i - e_j)^T L^-(e_i - e_j) = vol(V)(Q^T x_i - Q^T x_j)^T L^-(Q^T x_i - Q^T x_j) = \\ &= vol(V)(Q^T(x_i - x_j))^T L^-(Q^T(x_i - x_j)) = vol(V)(x_i - x_j)^T(Q^T)^T L^- Q^T(x_i - x_j) = \\ &= vol(V)(x_i - x_j)^T Q Q^T \Lambda^- Q Q^T(x_i - x_j) = vol(V)(x_i - x_j)^T \Lambda^-(x_i - x_j) \end{aligned} \tag{3.2}$$

Haciendo ahora la transformación $x_i = (\Lambda^-)^{-\frac{1}{2}} x'_i$, llegamos a:

$$c_{i,j} = vol(V)((\Lambda^-)^{-\frac{1}{2}}(x'_i - x'_j))^T \Lambda^-((\Lambda^-)^{-\frac{1}{2}}(x'_i - x'_j)) = vol(V)(x'_i - x'_j)^T(x'_i - x'_j)$$

□

Con este lema hemos probado que en este subespacio de $\mathbb{R}^n$ la distancia entre los puntos x'_i se corresponde exactamente con la distancia de conmutación entre los vértices del grafo v_i. Por ello esta distancia en $\mathbb{R}^n$ tomará el nombre de *Distancia Euclídea en Tiempo de Conmutación*(ECTD). Esta matriz L^- se puede relacionar con los elementos $\{x'_i\}_{i=1}^n$ de la siguiente forma:

Lema 3.16. *L^- es la matriz de Gram del conjunto de vectores $\{x'_i\}_{i=1}^n$; por lo que podemos escribir $L^- = X'(X')^T$, considerando X' la matriz cuyas filas son los vectores $\{x'_i\}_{i=1}^n$.*

Demostración. Puesto que una matriz de Gram es aquella que define el producto escalar de un conjunto de vectores, debemos probar que $(x'_i)^T x'_j = l^+_{i,j}$:

$$(x'_i)^T x'_j = ((\Lambda^-)^{\frac{1}{2}} Q e_i)^T((\Lambda^-)^{\frac{1}{2}} Q e_j) = e_i^T Q^T(\Lambda^-)^{\frac{1}{2}}(\Lambda^-)^{\frac{1}{2}} Q e_j = e_i^T L^- e_j = l^-_{i,j}$$

□

Relación entre el clustering espectral y la distancia de conmutación

Una vez que hemos estudiado la distancia de conmutación y la inmersión que se puede hacer de los nodos en $\mathbb{R}^n$, podemos plantear si existe alguna relación entre el clustering espectral y esta distancia de conmutación. En el clustering espectral tomamos las primeras k columnas de la matriz Q^T, construyendo de esta forma una matriz a la que denotamos por $U \in \mathbb{R}^{n\times k}$. Posteriormente, aplicando un algoritmo de clustering como puede ser $k - means$, asociamos los vértices del grafo a las filas de esta matriz U. La inmersión estudiada utilizando la distancia de conmutación asocia a cada vértice v_i el vector $x'_i = (\Lambda^-)^{\frac{1}{2}} Q e_i$, el cual se corresponde con la fila i-ésima de la matriz $Q^T(\Lambda^-)^{\frac{1}{2}}$.

Teniendo en cuenta estos dos planteamientos podemos encontrar que las dos diferencias principales residen en que cada uno de los elementos de x'_i quedan multiplicados por el valor inverso de los autovalores $\lambda_1, ..., \lambda_n$, algo que no ocurre en el caso del clustering espectral; así como que en el clustering espectral solo se toman las k primeras columnas, mientras que para la inmersión utilizando la $ECTD$ se toman las n columnas de $Q^T(\Lambda^+)^{\frac{1}{2}}$. A pesar de estas diferencias, bajo ciertas condiciones en las que no profundizaremos se puede justificar que el último paso de los algoritmos de clustering espectral, en el cual se construyen clusters con $k - means$ basándose en las distancias euclídeas entre las filas de la matriz U; se puede interpretar como la construcción de clusters de los vértices del grafo basándose en la distancia de conmutación.

Capítulo 4

Implementación práctica de los algoritmos

En este cuarto capítulo voy a poner en práctica los algoritmos que he presentado en el capítulo anterior de dos formas diferentes: en primer lugar voy a generar observaciones que permitan mostrar diferentes situaciones en las que debido a la distribución de estos datos *sintéticos* los algoritmos de clustering espectral obtienen unos resultados mucho mejores respecto a otros algoritmos tradicionales, como el algoritmo $k-means$; mientras que posteriormente voy a trabajar con unos datos reales para ilustrar el funcionamiento de estos algoritmos en el mundo real.

4.1. Resultados obtenidos con datos sintéticos

Los primeros conjuntos de datos con los que he trabajado son tres conjuntos que he generado con la intención de mostrar aquellos tipos de datos en los que se puede ver de una manera clara las ventajas de aplicar los algoritmos de clustering espectral frente a otros algoritmos, como el algoritmo k-means. El primer conjunto de datos está compuesto por tres grupos diferenciados de observaciones, cada uno de ellos con forma circular y centrados en el origen. El segundo conjunto presenta dos grupos perfectamente separados, cada uno de ellos con forma semicircular; mientras que el último se corresponde con dos grupos de observaciones asociados a espirales entrelazadas.

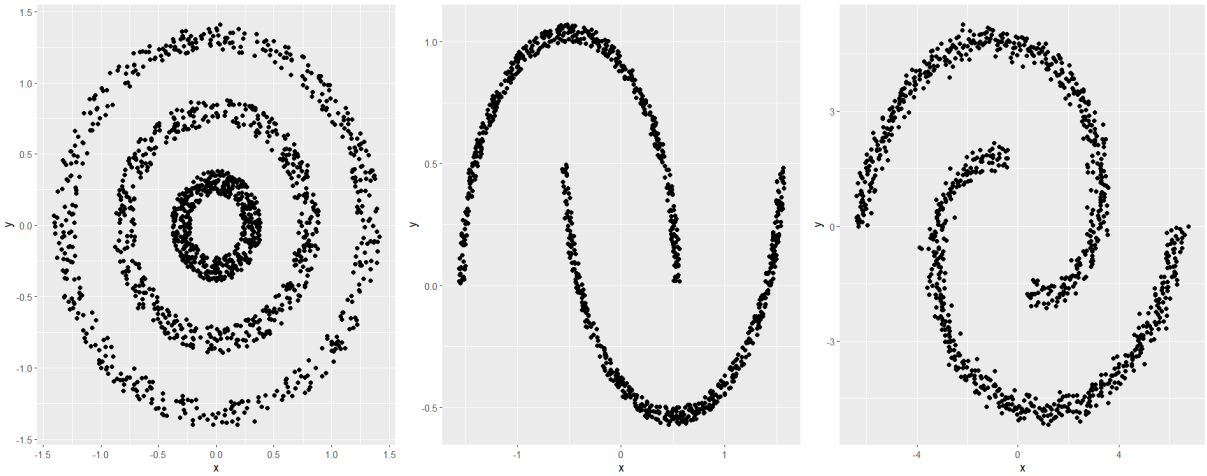

Figura 4.1: Conjuntos de datos generados

En la primera fila de la imagen podemos observar el resultado de aplicar el algoritmo k-means a estos conjuntos de datos, mientras que en la fila de abajo observamos el resultado de aplicar clustering espectral.

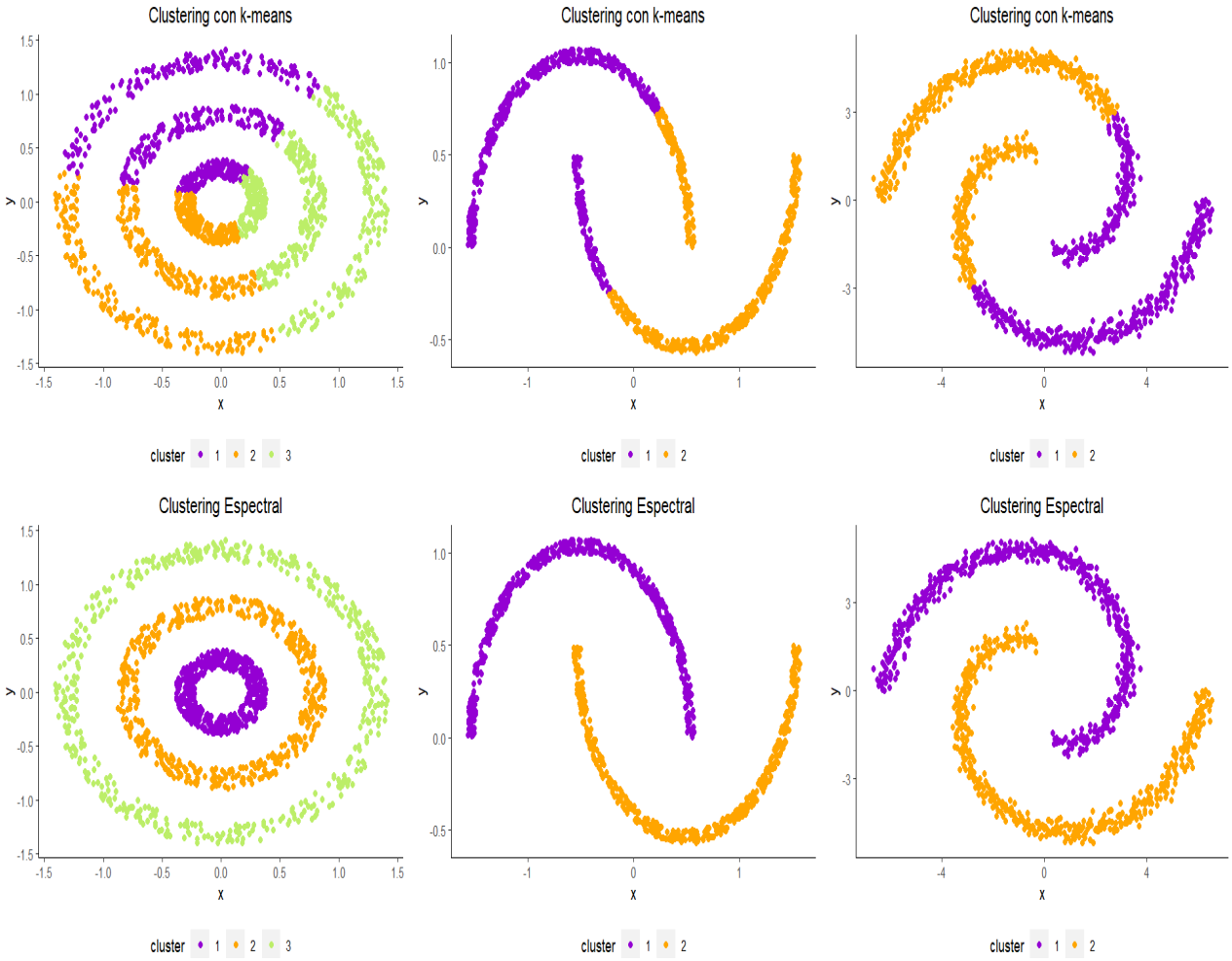

Figura 4.2: Aplicación de los algoritmos de clustering espectral a diferentes conjuntos

Las diferencias son notables en cuanto a la calidad del clustering resultante, ya que mientras que k-means no es capaz de agrupar a los datos correctamente; los algoritmos de clustering espectral separan de manera perfecta los diferentes clusters. La razón a esto se encuentra en la forma no convexa de nuestros tres conjuntos de datos.

Recordemos en primer lugar que el principio básico del algoritmo k-means es minimizar la suma de las distancias al cuadrado de cada punto a determinados centros. Este principio implica la formación de clusters con forma esférica, por lo que en el caso de conjuntos de datos que contienen agrupaciones no convexas el algoritmo k-means no funciona bien. En nuestro caso podemos comprobar que el clustering con k-means asocia puntos a un mismo cluster que en realidad pertenecen a diferentes grupos. Los algoritmos de clustering espectral solucionan este problema de una manera satisfactoria, ya que se basan en la conectividad entre los nodos del grafo al que se ha asociado el conjunto de datos correspondiente. Los puntos que están conectados mediante numerosas aristas se colocan en el mismo cluster, aunque la distancia entre ellos sea mayor que la distancia de uno de ellos a otro posible punto que pertenece a otro cluster.

Tomando como ejemplo los dos primeros gráficos de cada fila de la figura 4.2 observamos que los puntos que pertenecen al conjunto más exterior, coloreados de verde, presentan distancias entre ellos que son elevadas. Sin embargo, puesto que estos puntos están conectados en el grafo con los puntos más cercanos, y así sucesivamente; el clustering espectral es capaz de detectar que los nodos asociados a estos puntos se corresponden con una componente conexa del grafo y por tanto genera este cluster de manera satisfactoria.

En resumen, esta primera aproximación a la puesta en práctica de los conocimientos adquiridos durante los capítulos anteriores nos permite comprobar que en efecto los algoritmos de clustering espectral son muy útiles en determinadas situaciones donde los datos presentan estas formas no convexas. El código para construir estos conjuntos de datos y aplicar el algoritmo de clustering espectral no normalizado lo he escrito en lenguaje R y se encuentra en el Apéndice A.2. En ese apéndice podemos encontrar en primer lugar la definición de una función llamada *clusteringespectral()*, a la cual pasamos los datos con los que va a trabajar y el número de clusters que queremos encontrar; y nos devuelve cuál es el clustering resultante de aplicar el algoritmo.

4.2. Métricas de evaluación

Aunque en el apartado anterior resultaba sencillo comprobar la calidad del clustering esto no es lo habitual en los problemas asociados a situaciones real. Para ello se introducen las métricas de evaluación, las cuáles son medidas cuantitativas utilizadas para evaluar el rendimiento y la eficacia de un modelo estadístico o de aprendizaje automático. Estas métricas proporcionan información sobre el rendimiento del modelo y ayudan a comparar diferentes modelos o algoritmos.

En esta sección introduciré dos métricas de evaluación que posteriormente utilizaremos para evaluar el comportamiento de nuestro algoritmo de clustering sobre los datos con los que vamos a trabajar. Estas dos medidas son el *índice de Rand* y el *índice de Rand ajustado (ARI)*. La referencia utilizada para desarrollar estos conceptos es [8].

4.2.1. Índice de Rand

El índice de Rand es una métrica de evaluación que se utiliza para evaluar la calidad de una técnica de clustering. Esta métrica compara cómo se agrupan los pares de observaciones en el cluster predicho frente al verdadero cluster al que pertenecen, evaluando de esta forma el grado de concordancia entre los clusters producidos por dos métodos o algoritmos diferentes. La fórmula utilizada para calcularlo es la siguiente:

$$R = \frac{a+b}{\binom{n}{2}} \tag{4.1}$$

- a representa el número de pares de elementos que pertenecen al mismo cluster en ambas agrupaciones.
- b denota el número de pares de elementos que se asignan a diferentes clusters en ambas agrupaciones.
- n representa el número total de observaciones con las que estamos trabajando.

Debemos tener en cuenta que $a+b$ representa el número de acuerdos entre ambas agrupaciones y el denominador representa el número total de pares que podemos construir en un conjunto de n elementos, por lo que el índice de Rand indica la probabilidad de que ambas agrupaciones coincidan en un par elegido al azar. Puesto que este índice se corresponde con una probabilidad, varía entre 0 y 1: un valor de 1 significa una concordancia completa entre las dos agrupaciones con las que estamos trabajando, lo que significa que todos los pares de puntos han sido asignados al mismo cluster o a diferentes clusters en ambos métodos; sin embargo, un valor de 0 indica que no existe ninguna concordancia más allá de la que puede atribuirse al azar. El uso de este índice tiene diversas aplicaciones:

- La primera de ellas es que nos permite evaluar el rendimiento de los algoritmos de clustering comparando sus resultados con la agrupación de referencia (los verdaderos grupos). Esto nos permite determinar hasta qué punto el algoritmo ha agrupado puntos de datos similares.

- Nos permite ajustar los parámetros de un determinado algoritmo, ya que podemos experimentar con diferentes parámetros y utilizar el valor de este índice como una medida objetiva para comparar los diferentes resultados obtenidos y seleccionar la configuración óptima. También podemos probar diferentes algoritmos y quedarnos con aquel que obtenga unos mejores resultados. En ambos casos elegiremos aquellos algoritmos que obtengan unas puntuaciones más altas, ya que producen agrupaciones que se ajustan mejor a la realidad.

- Por último, nos aporta una herramienta para la tarea de selección de características. Cuando trabajamos con numerosas variables resulta interesante identificar un subconjunto de características relevantes para reducir la dimensión de nuestros datos. El índice de Rand puede utilizarse como criterio para evaluar la eficacia de diferentes subconjuntos de características, y quedarse con aquellos subconjuntos de variables que obtengan puntuaciones más altas, ya que se consideran que aportan más información.

Sin embargo, este índice también presenta ciertas limitaciones. En primer lugar requiere una agrupación de referencia para efectuar la comparación, algo que en numerosos problemas reales resulta muy complicado. Además presenta limitaciones cuando los clusters tienen tamaños significativamente diferentes o cuando existen patrones de agrupación complejos, ya que este índice trata por igual todas los diferencias entre las agrupaciones, independientemente de la naturaleza de estas diferencias. Por último, no proporciona información sobre otros aspectos específicos de la calidad de la agrupación, como la compacidad de los clusters o la separación entre ellos. Para intentar mitigar alguna de estas limitaciones y trabajar con una métrica más robusta se suele hacer uso de una versión ajustada del índice de Rand.

4.2.2. Índice de Rand ajustado (ARI)

El índice de Rand presentado en la sección anterior no tiene en cuenta el papel que juega el azar a la hora de agrupar los datos. El Índice de Rand ajustado (ARI) es una variante que tiene en cuenta el azar a la hora de evaluar la similitud entre dos agrupaciones. Este nuevo índice proporciona una medida que pueda dar un valor negativo cuando el acuerdo es peor de lo esperado respecto al que conseguiríamos utilizando simplemente el azar.

La fórmula del ARI es la siguiente:

$$ARI = \frac{R + E}{1 - E} \tag{4.2}$$

- R es el valor del índice de Rand definido anteriormente.
- E es el valor esperado del índice de Rand para agrupaciones aleatorias.

Por tanto, esta fórmula toma el índice de Rand (R) y lo ajusta teniendo en cuenta el valor esperado de este índice debido al azar (E). Este valor oscila entre -1, para aquellas agrupaciones completamente opuestas, y 1 para agrupaciones idénticas. El valor 0 indica una concordancia entre ambas agrupaciones que no mejora la que existiría con una hecha de forma aleatoria. El Índice de Rand ajustado se utiliza en el análisis cluster porque proporciona una medida más precisa de la similitud entre las agrupaciones al tener en cuenta también la componente aleatoria.

4.2.3. Métricas utilizando la tabla de contingencia

Las fórmulas vistas para calcular los índices en los dos apartados anteriores las podemos reescribir de tal manera que podamos implementarlas haciendo uso de los elementos de la tabla de contingencia. Para ello presentaremos en primer lugar este concepto y la notación que utilizaremos para nombrar sus elementos:

Definición 4.1. *Dado un conjunto de n elementos $\{x_1, ..., x_n\}$, y dos agrupaciones de estos elementos que denotaremos por $X = \{X_1, \ldots, X_r\}$ e $Y = \{Y_1, \ldots, Y_s\}$, definiremos los elementos de una tabla de contingencia $[n_{ij}]$ de forma que cada entrada n_{ij} denota el número de objetos en común entre X_i e Y_j: $n_{ij} = |X_i \cap Y_j|$.*

X $\diagdown$ Y	Y_1	Y_2	$\ldots$	Y_s	$a_i = \sum_{j=1}^{s} n_{ij}$
X_1	n_{11}	n_{12}	$\ldots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\ldots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\ldots$	n_{rs}	a_r
$b_j = \sum_{i=1}^{r} n_{ij}$	b_1	b_2	$\ldots$	b_s	

Tabla 4.1: Tabla de contingencia

Utilizando esta notación podemos reescribir las fórmulas presentadas en las expresiones (4.1) y (4.2):

$$R = \frac{\binom{n}{2} + 2\sum_{i,j}\binom{n_{i,j}}{2} - \left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right]}{\binom{n}{2}} \tag{4.3}$$

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}} \tag{4.4}$$

Aunque en la práctica obtendré los valores ARI utilizando un comando de R, resulta interesante conocer la posibilidad del cálculo de estas métricas utilizando la tabla de contingencia. Por ello, cuando presente los resultados de los diferentes experimentos incluiré también esta matriz de contingencia.

4.3. Aplicación del clustering espectral en *face clustering*

Una de las aplicaciones más relevantes de los algoritmos de clustering consiste en el *face clustering*. Este proceso consiste en agrupar los elementos en un conjunto de fotos o vídeos de forma que cada uno de los clusters contenga imágenes de una misma persona. Idealmente, este proceso da como resultado que cada cluster contenga únicamente imágenes de un mismo sujeto. El *face clustering* nos permite responder a preguntas como cuántas personas diferentes estaban presentes en una determinada foto o vídeo, así como quiénes eran estas personas.

Este proceso está muy relacionado con la tecnología del reconocimiento facial, la cual se utiliza para identificar o confirmar la identidad de una persona utilizando imágenes o vídeos de su rostro. Esta tecnología se emplea usualmente para permitir el acceso a aplicaciones o sistemas, y en los últimos años se ha convertido en un campo de investigación muy activo, ya que permite de una forma rápida y sencilla la identificación de una persona.

Existen diferentes enfoques para el proceso de *face clustering* que utilizan diferentes modelos y técnicas de clustering, en función de la representación del rostro con la que se disponga. En nuestro caso trabajaremos con imágenes hechas a diferentes sujetos, sobre las cuales trataremos de aplicar los algoritmos que hemos estudiado de clustering espectral para conseguir agruparlas de la mejor forma posible. Para ello presentaremos en primer lugar los datos con los que vamos a trabajar, y posteriormente pondremos en marcha los algoritmos que he implementado en R y analizaremos los resultados obtenidos.

4.3.1. Presentación de los datos

Los datos que hemos utilizado en este experimento práctico consisten en un dataset de R llamado *yale*, el cual se corresponde con una muestra de 2000 imágenes asociadas a 10 sujetos extraídas de la base de datos pública de la Universidad de Yale, denominada *The Yale Face Database*. Este conjunto de datos fue desarrollado y utilizado en el artículo [7], el cual trata sobre el desarrollo de un método generativo para el reconocimiento de rostros humanos bajo variaciones en la iluminación y el punto de vista. Los detalles técnicos que presentaré a continuación se pueden encontrar de manera más extendida en el enlace: `http://cvc.cs.yale.edu/cvc/projects/yalefacesB/yalefacesB.html`

The Yale Face Database contiene las imágenes de 10 sujetos en 9 poses diferentes: triste, somnoliento, sorprendido, con un ojo guiñado,etc.; y cada una de estas imágenes asociadas a las 9 poses está repetida con 64 condiciones de iluminación diferentes. A mayores, para cada sujeto en una determinada pose se capturó también una imagen con iluminación ambiental. Por tanto, el número total de imágenes es de 5850 imágenes.

Las imágenes de esta base de datos se tomaron con un equipo de iluminación especialmente diseñado para este propósito, puesto que está equipado con 64 luces estroboscópicas (emiten destellos breves en una rápida sucesión) controladas por ordenador. Las 64 imágenes de un sujeto en una pose determinada se tomaron con una velocidad de fotogramas de la cámara muy alto (30 fotogramas/segundo), permitiendo que se pudieran hacer estas 64 imágenes en unos 2 segundos. De esta forma se consigue que sólo haya pequeños cambios en la posición de la cabeza y la expresión facial entre las 64 imágenes.

Figura 4.3: Muestra de 10 imágenes de *The Yale Face Database*

Cada una de las imágenes del dataset original están en blanco y negro y tienen un tamaño de 32×32 píxeles. Sin embargo, el dataset propio de R con el que trabajamos ha comprimido estas imágenes a un tamaño de 30×20 píxeles, y ha representado cada uno de estos 600 píxeles con un valor númerico asociado al correspondiente valor en la escala de grises del píxel. Los elementos del dataset consisten en una matriz x de tamaño 2000×600 en la cual cada fila corresponde a una imagen vectorizada de los rostros de 10 sujetos con distintas poses y condiciones de iluminación; así como un vector y que indica el sujeto al que pertenece cada una de las fotografías.

El objetivo planteado a la hora de utilizar este dataset es analizar el comportamiento del clustering espectral sobre esta base de datos y comprobar si somos capaces de agrupar las imágenes de un mismo individuo en un mismo cluster a pesar de las diferentes configuraciones que presentan estas imágenes.

4.3.2. Resultados obtenidos

Una vez que he presentado los datos con los que vamos a trabajar es momento de poner en práctica el algoritmo de clustering espectral no normalizado que he implementado en R y he desarrollado en el Apéndice A.3. Este algoritmo se ejecuta mediante una función llamada *clusteringespectral*:

```
clusteringespectral <- function(datos,niveles,kgraph,kcluster,sigma)
```

Los parámetros de entrada son los siguientes:

- *datos*: los datos con los que vamos a trabajar
- *niveles*: un vector con la categoría a la que pertenece cada observación. He incluido este parámetro para representar los clusters reales, aunque en un programa de clustering habitualmente no disponemos de esta etiqueta.
- *kgraph:* hace referencia al k que debemos elegir para generar el grafo de k-vecinos más próximos.
- *kcluster*: se corresponde con el número k de clusters que queremos encontrar.
- *sigma*: es el valor de σ que utilizaremos para la función de similitud gaussiana.

Para representar los datos gráficamente he aplicado el Análisis de Componentes Principales (PCA) a los datos con los que estamos trabajando. El PCA es un método estadístico que permite simplificar la complejidad de conjuntos de datos con muchas dimensiones a la vez que conserva la mayor parte posible de información. Aunque he realizado el clustering con los datos completos, es decir, con las 600 variables correspondientes a cada observación, he aplicado este método para poder hacer la representación de los datos en dos dimensiones. Para conseguir esta representación me he quedado con las dos primeras componentes que obtenemos al aplicar este método.

El planteamiento que he decidido hacer para el estudio de este experimento consiste tanto en comparar el comportamiento del algoritmo $k-means$ respecto al clustering espectral para un mismo número k de clusters, así como la evolución que presentan los algoritmos de clustering espectral a medida que incrementamos el número de muestras con el que estamos trabajando y por tanto el número de clusters (de sujetos diferentes) que queremos detectar.

Primer experimento con 4 sujetos

En este primer experimento he decidido tomar 800 muestras de las 2000 que dispone el dataset. Estas 800 muestras pertenecen a 4 sujetos diferentes, 200 imágenes de cada uno. El procedimiento que llevaré a cabo en primer lugar consiste en determinar con qué combinación de parámetros obtenemos un valor más elevado de la métrica ARI. Para ello generamos la siguiente malla de hiperparámetros:

$$kgraph = (4, 5, 6) \ ; \ \ sigma = (0{,}5; 2; 3{,}5)$$

Aplicando la función *clusteringespectral* con estos parámetros obtenemos los siguientes valores de la métrica ARI:

$k-graph \diagdown \sigma$	0.5	2	3.5
4	0.3253214	0.3253214	0.3253214
5	0.9244779	0.9244779	0.9244779
6	0.6281403	0.6281403	0.6281403

Tabla 4.2: Valores de ARI modificando $kgraph$ y σ

Podemos obtener dos conclusiones de esta tabla: en primer lugar observamos que el valor de σ no modifica el valor de la métrica ARI, por lo que utilizaremos $\sigma = 2$; también observamos que se alcanza el valor más alto de esta métrica con $kgraph = 5$. Una vez que hemos fijado ambos parámetros podemos analizar el comportamiento del clustering espectral frente al del algoritmo k-means de una manera visual, utilizando la representación obtenida en dos dimensiones con el PCA y coloreando cada punto tanto en función del cluster al que ha sido asignado (en dos gráficos) como al sujeto al que pertenece en la realidad.

Recordemos que aunque este dataset está construido de tal manera que todas las imágenes tengan asociado el sujeto al que pertenecen, en la realidad el problema al que nos enfrentaríamos sería con unas imágenes no etiquetadas en las que no sabemos a qué sujeto pertenece cada una.

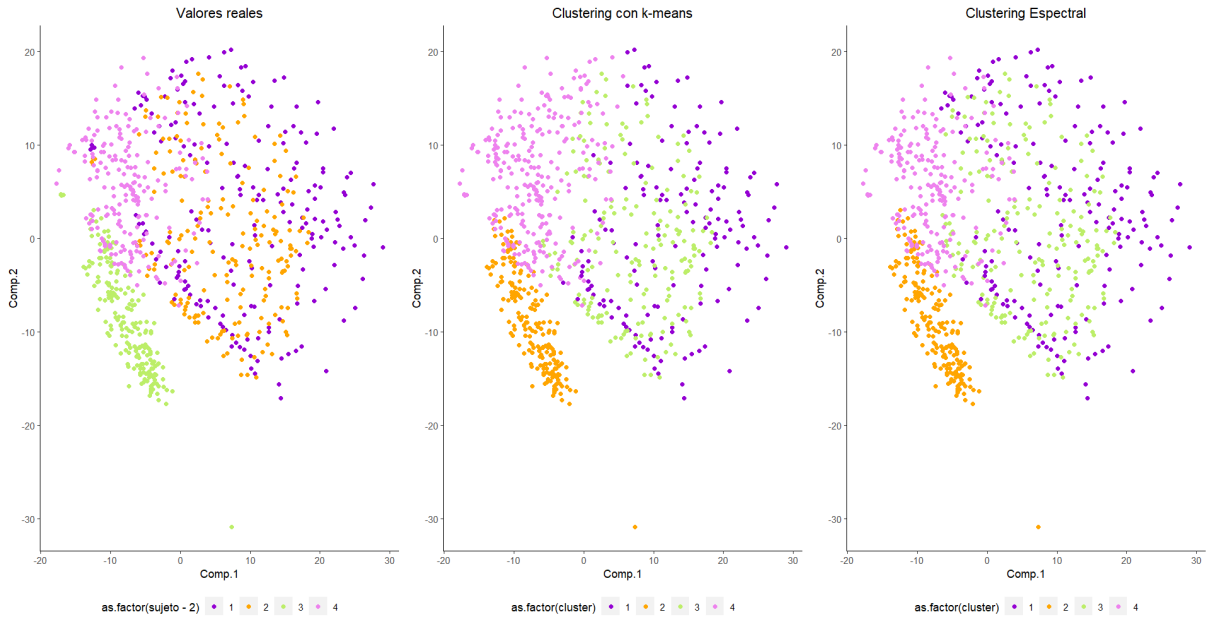

Figura 4.4: Representación gráfica de los resultados obtenidos con 4 sujetos

Visualmente podemos comprobar que el gráfico que aparece a la izquierda, el cual se corresponde con los agrupaciones reales en base a los diferentes sujetos, no dista mucho de los otros dos gráficos obtenidos tanto con $k-means$ como con el clustering espectral. Sin embargo, resulta díficil comprobar de una forma clara si en efecto las agrupaciones que proporcionan los algoritmos se corresponden con la realidad. Para poder afirmar de una manera más fiel cuál de los algoritmos presenta un comportamiento más satisfactorio vamos a representar en una tabla el número de imágenes que han sido asociadas a cada cluster con ambos algoritmos.

Cluster	1	2	3	4
K-Means	153	194	187	266
Clustering Espectral	186	195	205	214

Tabla 4.3: Resultados obtenidos con ambos métodos de clustering

Lo primero que podemos observar al estudiar esta tabla es que el clustering espectral presenta unos clusters más equilibrados en cuanto a número de elementos, en detrimento del método de $k-means$, el cual presenta un número más desigual de elementos en cada uno de los clusters. A la hora de estudiar los algoritmos de clustering espectral comentamos que se perseguía la búsqueda de clusters con un número no muy desequilibrado de elementos en cada grupo, ya que si esto ocurría se podía deber a una influencia no deseada de outliers. Por tanto este objetivo se cumple de una manera más satisfactoria utilizando el clustering espectral.

Esta tabla nos puede dar una idea de la distribución en cuanto al número de elementos asignados a cada cluster, pero no nos permite comprobar si los elementos asociados a cada cluster se corresponden con las imágenes de un mismo sujeto, es decir, si los clusters creados coinciden elemento a elemento con los grupos reales originales. Para evaluar esta *calidad* de nuestro clustering presentaremos la matriz de contingencia para el algoritmo de clustering espectral.

Pred. ╲ Real	1	2	3	4	
1	186	0	0	0	186
2	0	195	0	0	195
3	9	0	196	0	205
4	5	5	4	200	214
	200	200	200	200	

Tabla 4.4: Matriz de contingencia del clustering espectral

Como hemos visto en el apartado 4.2.3, podemos obtener el valor de la métrica ARI haciendo uso de los elementos de esta matriz. Dado que resulta pesado realizar estas cuentas, he calculado el valor de esta métrica para el caso del clustering espectal y para el caso del algoritmo $k-means$ haciendo uso del comando ARI de R, obteniendo los siguientes resultados:

Algoritmo	ARI
K-Means	0.7941662
Clustering Espectral	0.9244779

Tabla 4.5: Métrica ARI con ambos algoritmos

Comprobamos de esta forma que el algoritmo $k-means$ presenta un peor comportamiento que el clustering espectral, puesto que la métrica ARI es sensiblemente inferior.

Por último, he decidido mostrar un gráfico en el que podemos observar los 10 autovalores λ_i más pequeños asociados a la matriz Laplaciana L del grafo generado con el conjunto de datos con el que estamos trabajando.

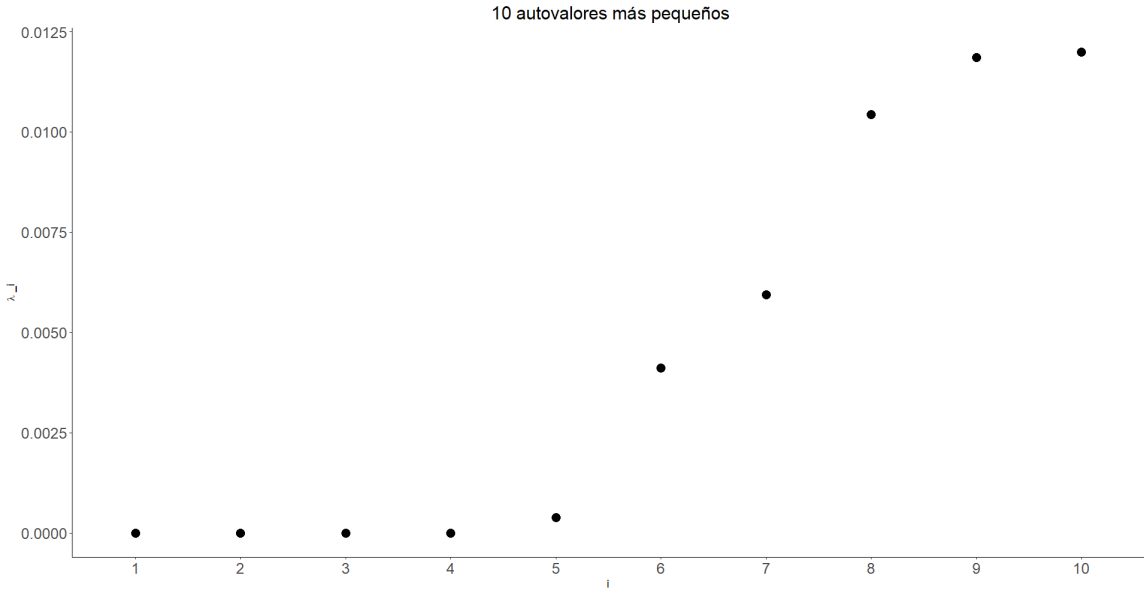

Figura 4.5: 10 autovalores de L más pequeños

Si recordamos la sección 2.4.3, podíamos determinar el número óptimo de clusters buscando aquel k en el que se produzca un mayor salto entre autovalores consecutivos λ_k y λ_{k+1}. Observando la figura 4.5 podemos comprobar que los 4 primeros autovalores son nulos, mientras que el autovalor λ_5 ya toma un valor positivo distinto de 0. Esto nos indica que $k = 4$ es el número de clusters que debemos intentar encontrar en este conjunto de datos, suponiendo que no conociéramos que las imágenes estaban asociadas a 4 sujetos diferentes.

Segundo experimento con 10 sujetos

Repetiremos el mismo esquema respecto a la sección anterior pero en este caso trabajaremos con las 2000 imágenes del dataset, y por tanto con 10 sujetos. Puesto que en el caso anterior el valor de σ no modificaba en absoluto el valor de la métrica ARI y teniendo en cuenta que el tiempo de computación aumentará ya que estamos trabajando con más datos, variaremos solamente $kgraph = (12, 13, 14)$. Aplicando la función *clusteringespectral* con la variación de este parámetro obtenemos los siguientes valores de la métrica ARI:

$kgraph$	ARI
12	0.8177594
13	0.8213903
14	0.8170502

Tabla 4.6: Valores de la métrica ARI modificando $kgraph$

Puesto que el valor más alto de la métrica ARI se alcanza con $kgraph = 13$, fijaremos este parámetro y analizaremos de nuevo el comportamiento del clustering espectral frente al del algoritmo $k - means$. Utilizando la representación gráfica que aparece en la siguiente imagen resulta muy complicado sacar alguna conclusión, por lo que recurriremos a las mismas técnicas que utilizamos antes.

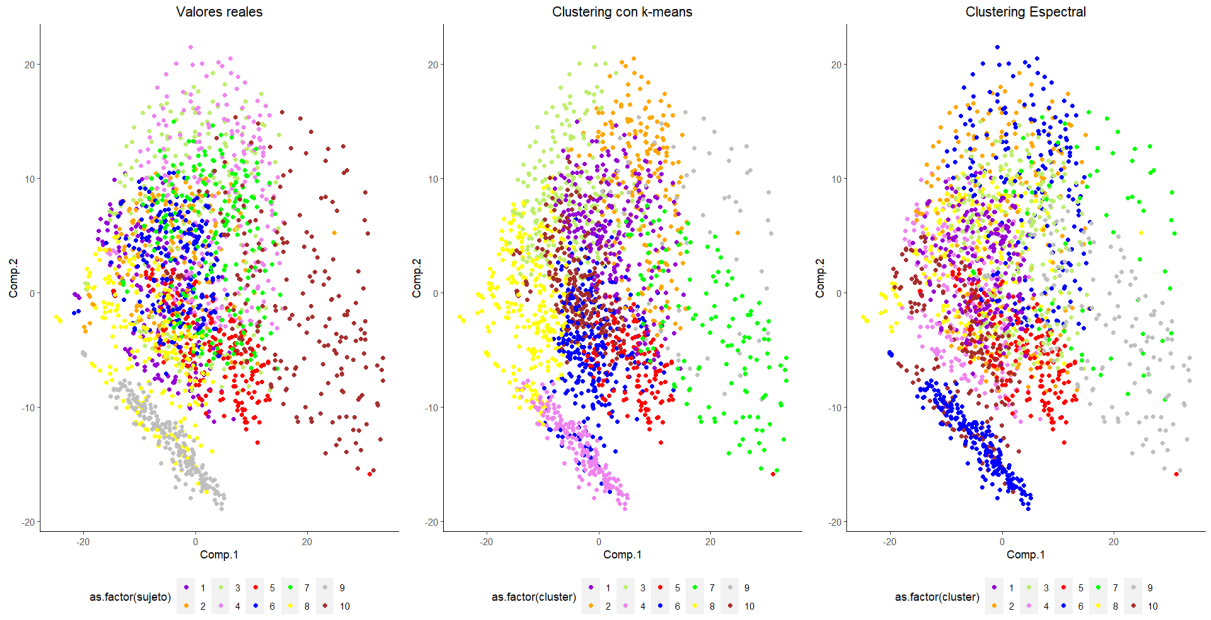

Figura 4.6: Representación gráfica de los resultados obtenidos con 10 sujetos

En primer lugar, mostramos el cardinal de cada uno de los clusters generados por ambos algoritmos. En ambos casos podemos encontrar que existe un cluster muy numeroso, el 6, en el que se agrupan más de 300 imágenes; y uno mucho más pequeño que el resto, el 9 y el 7 respectivamente. Este hecho se puede deber a dos sujetos que presenten rasgos faciales similares y por tanto nuestro algoritmo no es capaz de distinguirlos, asociando numerosas imágenes de los diferentes sujetos a un mismo cluster y generando un cluster restante de tamaño muy inferior.

Cluster	1	2	3	4	5	6	7	8	9	10
K-Means	230	205	228	177	143	313	127	285	63	229
Clustering Espectral	205	190	195	216	200	396	65	224	135	174

Tabla 4.7: Resultados obtenidos con ambos métodos de clustering

En la siguiente tabla mostraremos el valor de la métrica ARI para ambos algoritmos, comprobando de nuevo que el clustering espectral presenta un comportamiento mucho mejor que el algoritmo $k - means$.

Algoritmo	ARI
K-Means	0.3998836
Clustering Espectral	0.8213903

Tabla 4.8: Métrica ARI con ambos algoritmos

Este buen papel desempeñado por el clustering espectral se muestra de una manera muy clara estudiando la tabla de contingencia resultante de aplicar este algoritmo. Esta tabla presenta una forma prácticamente diagonal, señal de que la mayoría de imágenes han sido asociadas al cluster correcto.

Pred. ╲ Real	1	2	3	4	5	6	7	8	9	10	
1	198	0	5	0	0	0	2	0	0	0	205
2	0	190	0	0	0	0	0	0	0	0	190
3	0	0	195	0	0	0	0	0	0	0	195
4	0	0	0	194	0	0	0	0	0	22	216
5	0	0	0	0	200	0	0	0	0	0	200
6	0	0	0	0	0	200	196	0	0	0	396
7	0	0	0	0	0	0	0	0	65	0	65
8	2	10	0	6	0	0	2	200	0	4	224
9	0	0	0	0	0	0	0	0	135	0	135
10	0	0	0	0	0	0	0	0	0	174	174
	200	200	200	200	200	200	200	200	200	200	

Tabla 4.9: Tabla de contingencia con 10 sujetos

Por último mostraremos en la siguiente gráfica los 25 autovalores de menor tamaño de la matriz Laplaciana L para estudiar el número de clusters k que deberíamos intentar encontrar:

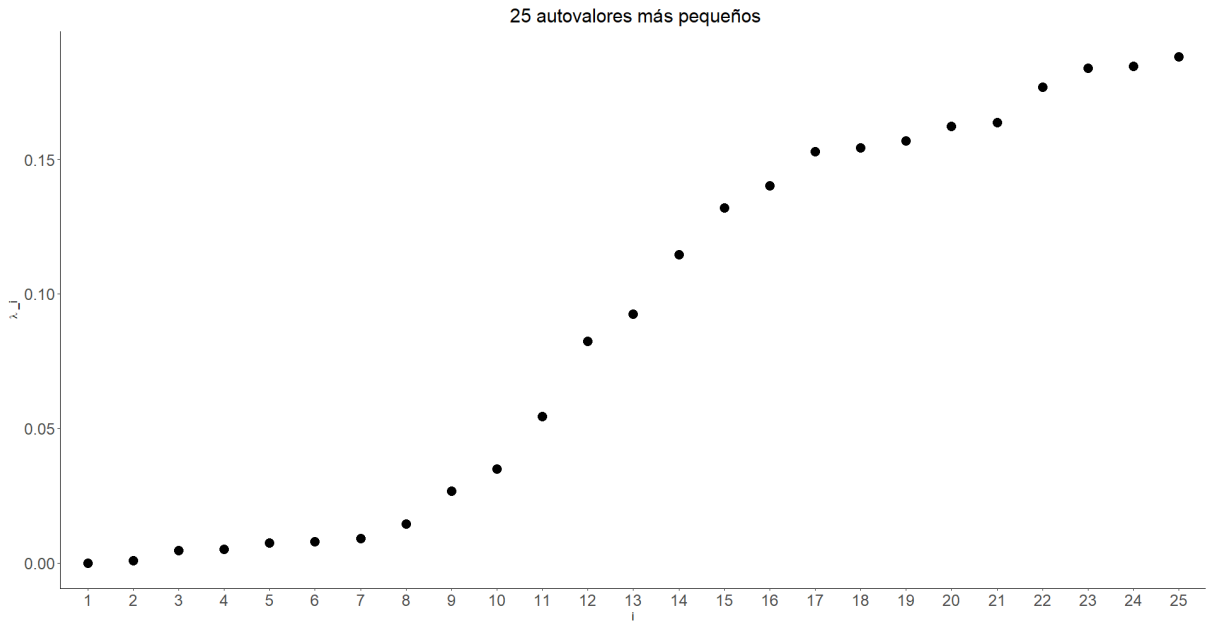

Figura 4.7: 25 autovalores de L más pequeños

En este gráfica podemos ver que los primeros 8 autovalores crecen de una manera muy lenta, crecimiento que se acelera a medida que consideramos la diferencia entre λ_8 y λ_9, así como la diferencia entre λ_{10} y λ_{11}. Por ello, determinaremos que el número óptimo de clusters se encontraría entre $k = 9$ y $k = 11$. Esta conclusión coincide con los datos con los que estamos trabajando, puesto que sabemos que existen fotos asociadas a 10 sujetos.

Apéndice A

Código implementado en R

A.1. Generación de grafos

El siguiente código se ha utilizado para dibujar los grafos de las figuras 1.1,1.2 y 1.3:

```
#LIBERIAS
install.packages("Rdimtools")
library(Rdimtools)
install.packages("igraph")
library(igraph)
install.packages("cccd")
library(cccd)
#Generamos las observaciones
x <- matrix(runif(100),ncol=2)
##Grafo epsilon-vecindad
eps1 = aux.graphnbd(x,type=c("enn",0.15))
eps2 = aux.graphnbd(x,type=c("enn",0.25))
eps3 = aux.graphnbd(x,type=c("enn",0.5))
G1 <- as.undirected(graph.adjacency(eps1$mask, weighted = F))
G2 <- as.undirected(graph.adjacency(eps2$mask, weighted = F))
G3 <- as.undirected(graph.adjacency(eps3$mask, weighted = F))
par(mfrow=c(1,3))
plot(G1)
title(main=expression("Grafo de " * epsilon ~ "-vecindad con " * epsilon
    ~ "=0.15"), cex.main=3)
plot(G2)
title(main=expression("Grafo de " * epsilon ~ "-vecindad con " * epsilon
    ~ "=0.25"), cex.main=3)
plot(G3)
title(main=expression("Grafo de " * epsilon ~ "-vecindad con " * epsilon
    ~ "=0.5"), cex.main=3)
#Grafo 4-vecinos mas proximos (mutuos y no mutuos)
G4 <- nng(x,k=4)
G4m <- nng(x,k=4,mutual=TRUE)
par(mfrow=c(1,2))
plot(G4,edge.arrow.size = 0.01, main="Grafo k-vecinos m s pr ximos con
    k=4")
plot(G4m, main="Grafo k-vecinos m s pr ximos mutuos con k=4")
#Grafo 4-vecinos mas proximos no mutuos y 7-vecinos mas proximos mutuos
G4 <- nng(x,k=4)
G7m <- nng(x,k=7,mutual=TRUE)
par(mfrow=c(1,2))
```

```r
plot(G4,edge.arrow.size = 0.01, main="Grafo k-vecinos mas proximos con k=4")
plot(G7m, main="Grafo k-vecinos mas proximos mutuos con k=7")
```

A.2. Clustering espectral con datos sintéticos

El algoritmo que está escrito a continuación utilizando el lenguaje de R es el algoritmo de clustering espectral no normalizado, presentado en la Sección 2.2:

```r
#LIBRERIAS
library(tidyverse)
library(gridExtra)
library(ggplot2)
#Funcion definida para el clustering espectral
clusteringespectral <- function(datos,kgraph,kcluster,sigma) {
  set.seed(26)
  km <- kmeans(datos, kcluster)
  ## Guardamos los resultados de este clustering para posteriormente graficarlo
  kmeans_clusters <- km$cluster
  kmeans_resultados <- cbind(datos,cluster = as.factor(kmeans_clusters))
  #Hacemos el grafico
  kmeans_plot=ggplot(data=kmeans_resultados,aes(x=x, y=y, color = cluster)) + geom_point() + scale_color_manual(values = c('1' = "darkviolet",'2'="orange",'3' ="darkolivegreen2")) + ggtitle("Clustering con k-means") + theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(), panel.background = element_blank(), axis.line = element_line(colour = "black"), plot.title = element_text(hjust = 0.5), legend.position="bottom")

  #Ahora vamos a implementar el algoritmo de clustering espectral:

  #1. Construimos un grafo G de similitud asociado a nuestro datos: grafo de k-vecinos mutuos y construimos la matriz de adyacencia W del grafo:
  S <- as.matrix(dist(datos))
  D <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
  for (i in 1:nrow(datos)) {
  #Nos quedamos con los kgraph-vecinos mas proximos de cada nodo
    vecinos <- order(S[i,])[2:kgraph]
    # Asignamos el valor 1 a los vecinos
    D[i,][vecinos] <- 1
  }
  # Buscamos los vecinos mutuos
  D=D+t(D)
  D[ D == 2 ] = 1
  #Una vez tenemos las aristas del grafo podemos construir la matriz de adyacencia W. Utilizaremos como pesos la funcion de similitud gaussiana
  W <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
  for (i in 1:nrow(datos)) {
    for (j in 1:nrow(datos)) {
      if (D[i,j]==1){
        W[i,j]=exp(-S[i,j]/(2*sigma^2))
      }
    }
  }
```

```
38    # Calculamos los grados de cada nodo sumando W por columnas
39    grados = colSums(W)
40
41    #2. Construimos la matriz laplaciana no normalizada L asociada a
       nuestro grafo. Tambien calcularemos L_sym y L_rw
42
43    laplaciana = diag(grados) -W
44    laplacianasym = ( diag(nrow(datos)) - diag(grados^(-1/2)) %*% W %*%
       diag(grados^(-1/2)) )
45    laplacianarw= ( diag(nrow(datos)) - diag(grados^(-1)) %*% W )
46
47    #3. Calculamos los primeros k autovalores de L y sus autovectores
       asociados: u1, ..., uk
48    valpropios = eigen(laplaciana, symmetric = TRUE)
49    n = nrow(laplaciana)
50    autovalores= valpropios$values
51    autovectores= valpropios$vectors
52
53    #4. Construimos la matriz U     Rn k cuyas columnas son los
       autovectores u1, ..., uk
54
55    U = valpropios$vectors[,(n - (kcluster-1)):(n)]
56
57    #5.Para i= 1,...,n tomamos yi como la fila i-esima de la matriz U
58    #6. Agrupamos los n puntos (yi) i=1,...,n utilizando el algoritmo de
       clustering de  k medias ,construyendo C1, ..., Ck
59    set.seed(26)
60    sc = kmeans(U, kcluster)
61    sc_results = cbind(datos, cluster = as.factor(sc$cluster))
62    # Dibujamos la separacion obtenida mediante el clustering espectral
63    sc_plot = ggplot(data = sc_results, aes(x=x, y=y, color = cluster)) +
64      geom_point() +
65      scale_color_manual(values = c('1' = "darkviolet",
66                                    '2' ="orange",
67                                    '3' ="darkolivegreen2")) +
68      ggtitle("Clustering Espectral") +
69      theme(panel.grid.major = element_blank(), panel.grid.minor = element
       _blank(),
70            panel.background = element_blank(), axis.line = element_line(
       colour = "black"),
71            plot.title = element_text(hjust = 0.5), legend.position="
       bottom")
72    grid.arrange(kmeans_plot, sc_plot, nrow=1)
73  }
74
75  #Generamos los datos con los que vamos a trabajar
76
77  #CLUSTERS CIRCULARES
78
79  ## Establecemos los limites superiores e inferiores de la distancia del
       punto al centro
80
81  min_vals <- c(0.05, 0.5, 1.5)
82  max_vals <- c(0.15, 0.8, 2)
83  x <- c()
84  y <- c()
85
86  ## Generamos las observaciones
87  set.seed(26)
```

```r
for(i in 1:3){
  radio <- sqrt(runif(500, min = min_vals[i], max = max_vals[i]))
  angulo <- runif(500)
  theta <- angulo * 2 * pi
  xaux <- radio * cos(theta)
  yaux <- radio * sin(theta)
  x <- c(x, xaux)
  y <- c(y, yaux)
}
datos <- as.data.frame(cbind(x, y))
## Graficamos estos datos
ggplot(data=datos, aes(x=x, y=y)) + geom_point()
#Aplicamos el clustering espectral
clusteringespectral(datos,8,3,0.25)

#MEDIAS LUNAS
x <- c()
y <- c()
angulos <-c(0,0.5)
angulos2<-c(0.5,1)
centro <- c(-0.5,0.5)
centro2 <- c(0, 0.5)
## Generamos las observaciones
for(i in 1:2){
  radio <- sqrt(runif(500, min = 1, max = 1.15))
  angulo <- runif(500, min=angulos[i], max=angulos2[i])
  theta <- angulo * 2 * pi
  xaux <- centro[i]+radio * cos(theta)
  yaux <- centro2[i]+radio * sin(theta)
  x <- c(x, xaux)
  y <- c(y, yaux)
}
datos <- as.data.frame(cbind(x, y))
## Graficamos estos datos
ggplot(data=datos, aes(x=x, y=y)) + geom_point()
#Aplicamos el clustering espectral
clusteringespectral(datos,8,2,0.25)

#ESPIRALES
install.packages("KODAMA")
library(KODAMA)
datos=spirals(n=c(500,500),sd=c(0.2,0.2))
datos=as.data.frame(datos)
## Graficamos estos datos
ggplot(data=datos, aes(x=x, y=y)) + geom_point()
#Aplicamos la funci n de clustering espectral
clusteringespectral(datos,8,2,0.25)
```

A.3. *Face clustering*

En este apéndice incluiremos todo el código utilizado para obtener los resultados mostrados en la sección 4.3.

```
#LIBRERIAS
#Dataset a utilizar
install.packages("PPCI")
library(PPCI)
library(tidyverse)
library(gridExtra)
library(ggplot2)
library(factoextra)
#Matriz de confusion
install.packages('caret')
library(caret)
#Calidad del clustering
install.packages("MixGHD")
library(MixGHD)

#Funcion definida para el clustering espectral
clusteringespectral <- function(datos,niveles,kgraph,kcluster,sigma) {
  datospca <- princomp(datos)
  datos2 <- cbind(datospca$scores[,1:2],sujeto=niveles)
  plotreal = ggplot(data = as.data.frame(datos2), aes(x=Comp.1, y=Comp
    .2, color = as.factor(sujeto))) +
    geom_point() +
    scale_color_manual(values = c('1' = "darkviolet",
                                  '2' ="orange",
                                  '3' ="darkolivegreen2",
                                  '4' = "violet",
                                  '5' = "red",
                                  '6' = "blue",
                                  '7' = "green",
                                  '8' = "yellow",
                                  '9' = "grey",
                                  '10' = "brown")) +
    ggtitle("Valores reales") +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element
    _blank(),
          panel.background = element_blank(), axis.line = element_line(
    colour = "black"),
          plot.title = element_text(hjust = 0.5), legend.position="
    bottom")

  set.seed(26)
  km <- kmeans(datos, kcluster)
  ## Guardamos los resultados de este clustering para posteriormente
    graficarlo
  kmeans_clusters <- km$cluster
  kmeans_resultados <- cbind(datos2, cluster = as.factor(kmeans_clusters
    ))
  #Hacemos el gr fico
  kmeans_plot = ggplot(data = as.data.frame(kmeans_resultados), aes(x =
    Comp.1, y = Comp.2, color = as.factor(cluster))) +
    geom_point() +
    scale_color_manual(values = c('1' = "darkviolet",
                                  '2' ="orange",
                                  '3' ="darkolivegreen2",
```

```
                                    '4' = "violet",
                                    '5' = "red",
                                    '6' = "blue",
                                    '7' = "green",
                                    '8' = "yellow",
                                    '9' = "grey",
                                    '10' = "brown")) +
    ggtitle("Clustering con k-means") +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
          panel.background = element_blank(), axis.line = element_line(colour = "black"),
          plot.title = element_text(hjust = 0.5), legend.position="bottom")

#Ahora vamos a implementar el algoritmo de clustering espectral:
#1. Construimos un grafo G de similitud asociado a nuestro datos: grafo de k-vecinos mutuos
#y construimos la matriz de adyacencia W del grafo:
S <- as.matrix(dist(datos))
D <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
for (i in 1:nrow(datos)) {
  #Nos quedamos con los kgraph-vecinos m s pr ximos de cada nodo
  vecinos <- order(S[i,])[2:kgraph]
  # Asignamos el valor 1 a los vecinos
  D[i,][vecinos] <- 1
}
# Buscamos los vecinos mutuos
D=D+t(D)
D[ D == 2 ] = 1

#Una vez tenemos las aristas del grafo podemos construir la matriz de adyacencia W.
#Utilizaremos como pesos la funci n de similitud gaussiana

W <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
for (i in 1:nrow(datos)) {
  for (j in 1:nrow(datos)) {
    if (D[i,j]==1){
      W[i,j]=exp(-S[i,j]/(2*2^2))
    }
  }
}

# Calculamos los grados de cada nodo sumando W por columnas
grados = colSums(W)

#2. Construimos la matriz laplaciana no normalizada L asociada a nuestro grafo
##Tambi n calcularemos L_sym y L_rw

laplaciana = diag(grados) -W
laplacianasym = ( diag(nrow(datos)) - diag(grados^(-1/2)) %*% W %*% diag(grados^(-1/2)) )
laplacianarw= ( diag(nrow(datos)) - diag(grados^(-1)) %*% W )

#3. Calculamos los primeros k autovalores de L y sus autovectores asociados: u1, ..., uk
valpropios = eigen(laplaciana, symmetric = TRUE)
```

```
  n = nrow(laplaciana)
  autovalores= valpropios$values
  autovectores= valpropios$vectors

  #4. Construimos la matriz U de R^nxk cuyas columnas son los
    autovectores u1, ..., uk

  U = valpropios$vectors[,(n - (kcluster-1)):(n)]

  #5. Para i= 1,...,n tomamos yi como la fila i-esima de la matriz U
  #6. Agrupamos los n puntos (yi) i=1,...,n utilizando el algoritmo de
    clustering de  k medias ,construyendo C1, ..., Ck
  set.seed(26)
  sc = kmeans(U, kcluster)
  sc_results = cbind(datos2, cluster = as.factor(sc$cluster))
  # Dibujamos el grafico de clustering espectral
  sc_plot = ggplot(data = as.data.frame(sc_results), aes(x=Comp.1, y=
    Comp.2, color = as.factor(cluster))) +
     geom_point() +
     scale_color_manual(values = c('1' = "darkviolet",
                                   '2' ="orange",
                                   '3' ="darkolivegreen2",
                                   '4' = "violet",
                                   '5' = "red",
                                   '6' = "blue",
                                   '7' = "green",
                                   '8' = "yellow",
                                   '9' = "grey",
                                   '10' = "brown")) +
     ggtitle("Clustering Espectral") +
     theme(panel.grid.major = element_blank(), panel.grid.minor = element
    _blank(),
           panel.background = element_blank(), axis.line = element_line(
    colour = "black"),
           plot.title = element_text(hjust = 0.5), legend.position="
    bottom")
  grid.arrange(plotreal, kmeans_plot, sc_plot, nrow=1)
clusterscreados <- cbind(km$cluster,sc$cluster)
return(clusterscreados)
}

#Primer experimento: cargamos los datos y nos quedamos con 800 imagenes
    asociadas a 4 sujetos
dataset=yale
imagenes= dataset$x[401:1200,]
sujetos = dataset$c[401:1200]
#Planteamos una malla y buscamos la combinacion con la que obtengamos
    una mejor metrica ARI
kgraphvector <- c(4,5,6)
sigmavector <- c(0.5,2,3.5)
ARImatriz <- matrix(0, 3, 3)
for (i in 1:3){
  for (j in 1:3){
     clusterscreados<-clusteringespectral(imagenes,sujetos,kgraphvector[i
    ],4,sigmavector[j])
     resultados <- cbind(sujetos,clusterscreados)
     ARImatriz[i,j] <- ARI(sujetos,resultados[,3])
  }
}
```

```
ARImatriz
#Mejor clustering
clusterscreados<-clusteringespectral(imagenes,sujetos,kgraphvector[2],4,
    sigmavector[2])
resultados <- cbind(sujetos,clusterscreados)
#M trica ARI
ARI(sujetos,resultados[,3]) #ARI con clustering espectral
ARI(sujetos,resultados[,2]) #ARI con kmeans
#Matriz de confusion
matriz <- confusionMatrix(data=as.factor(resultados[,3]) , reference =
    as.factor(resultados[,1]-2))
matriz
# Grafica de autovalores
datos<- imagenes
niveles<- sujetos
kgraph <- 5
kcluster<-4
sigma<- 2
S <- as.matrix(dist(datos))
D <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
for (i in 1:nrow(datos)) {
  vecinos <- order(S[i,])[2:kgraph]
  D[i,][vecinos] <- 1
}
D=D+t(D)
D[ D == 2 ] = 1
W <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
for (i in 1:nrow(datos)) {
  for (j in 1:nrow(datos)) {
    if (D[i,j]==1){
      W[i,j]=exp(-S[i,j]/(2*2^2))
    }
  }
}
grados = colSums(W)
laplaciana = diag(grados) -W
valpropios = eigen(laplaciana, symmetric = TRUE)
autovalores= valpropios$values
autovalores10 <- cbind(c(1:10),autovalores[800:791])
ggplot(data = as.data.frame(autovalores10), aes(x = as.factor(V1), y =
    V2)) + geom_point() + ggtitle(" 10 autovalores m s peque os") +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_
    blank(),
        panel.background = element_blank(), axis.line = element_line(
    colour = "black"),
        plot.title = element_text(hjust = 0.5), legend.position="bottom"
    ) + xlab("i")+ ylab( lambda ~ "_i")


#Segundo experimento: cargamos los datos y nos quedamos con 2000
    imagenes asociadas a 10 sujetos

dataset=yale
imagenes= dataset$x
sujetos = dataset$c

#Aplicamos esta funcion
kgraphvector <- c(12,13,14)
ARImatriz <- matrix(0, 3)
```

```
for (i in 1:3){
    clusterscreados<-clusteringespectral(imagenes,sujetos,kgraphvector[i],10,2)
    resultados <- cbind(sujetos,clusterscreados)
    ARImatriz[i] <- ARI(sujetos,resultados[,3])
  }
ARImatriz

#Aplicamos la funcion con el valor correspondiente a la mejor m trica obtenida
clusterscreados<-clusteringespectral(imagenes,sujetos,13,10,2)
resultados <- cbind(sujetos,clusterscreados)
#M trica ARI
ARI(sujetos,resultados[,3])
ARI(sujetos,resultados[,2])
#Matriz de confusion
matriz <- confusionMatrix(data=as.factor(resultados[,3]) , reference = as.factor(resultados[,1]))
matriz
#Autovalores
datos<- imagenes
niveles<- sujetos
kgraph <- 13
kcluster<-10
sigma<- 2
S <- as.matrix(dist(datos))
D <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
for (i in 1:nrow(datos)) {
  vecinos <- order(S[i,])[2:kgraph]
  D[i,][vecinos] <- 1
}
D=D+t(D)
D[ D == 2 ] = 1
W <- matrix(0, nrow=nrow(datos), ncol = nrow(datos))
for (i in 1:nrow(datos)) {
  for (j in 1:nrow(datos)) {
    if (D[i,j]==1){
      W[i,j]=exp(-S[i,j]/(2*2^2))
    }
  }
}
grados = colSums(W)
laplaciana = diag(grados) -W
valpropios = eigen(laplaciana, symmetric = TRUE)
autovalores= valpropios$values
autovalores25 <- cbind(c(1:25),autovalores[2000:1976])
ggplot(data = as.data.frame(autovalores25), aes(x = as.factor(V1), y = V2)) + geom_point() + ggtitle(" 25 autovalores m s peque os") +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "black"),
        plot.title = element_text(hjust = 0.5), legend.position="bottom") + xlab("i")+ ylab( lambda ~ "_i")
```

Bibliografía

[1] Titu Andreescu, *Essential linear algebra with applications*, (pp. 337-482)., Springer, 2016.

[2] Maria R Brito, Edgar L Chávez, Adolfo J Quiroz, and Joseph E Yukich, *Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection*, Statistics & Probability Letters **35** (1997), no. 1, 33–42.

[3] Mark Bun, *Lecture notes 14: Conductance, cheeger's inequality*, Spring (2022).

[4] Ernie Croot, *The rayleigh principle for finding eigenvalues*, Georgia Institute of Technology, School of Mathematics, Tech. Rep (2005).

[5] Miroslav Fiedler, *Algebraic connectivity of graphs*, Czechoslovak mathematical journal **23** (1973), no. 2, 298–305.

[6] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens, *Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation*, IEEE Transactions on knowledge and data engineering **19** (2007), no. 3, 355–369.

[7] Athinodoros S. Georghiades, Peter N. Belhumeur, and David J. Kriegman, *From few to many: Illumination cone models for face recognition under variable lighting and pose*, IEEE transactions on pattern analysis and machine intelligence **23** (2001), no. 6, 643–660.

[8] Lawrence Hubert and Phipps Arabie, *Comparing partitions*, Journal of classification **2** (1985), 193–218.

[9] Marina Meilă and Jianbo Shi, *A random walks view of spectral segmentation*, International Workshop on Artificial Intelligence and Statistics, PMLR, 2001, pp. 203–208.

[10] Andrew Ng, Michael Jordan, and Yair Weiss, *On spectral clustering: Analysis and an algorithm*, Advances in neural information processing systems **14** (2001).

[11] Ahmed H Sameh and John A Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM Journal on Numerical Analysis **19** (1982), no. 6, 1243–1259.

[12] Jianbo Shi and Jitendra Malik, *Normalized cuts and image segmentation*, IEEE Transactions on pattern analysis and machine intelligence **22** (2000), no. 8, 888–905.

[13] Ulrike Von Luxburg, *A tutorial on spectral clustering*, Statistics and computing **17** (2007), 395–416.