22 de marzo 2024

DISEÑO DE ALGORITMOS DE CLUSTERING Y MÉTODOS DE EVALUACIÓN

DAVID GARCÍA, LLUÍS ITARTE, MARCOS ROCA, DIEGO RODRÍGUEZ

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona
FIB

Índice

Introducción

El algoritmo *k-means* consiste en clasificar n elementos en k clústeres. Es decir, dado un conjunto de tamaño *n* y un entero *k*, al final de la ejecución del algoritmo, los elementos de este conjunto quedan subdivididos en k agrupaciones de tal manera que los elementos de cada clúster serán parecidos entre ellos.

La entrada consiste en n elementos. Cada elemento representará un punto de nuestro dataset. Se especificará también la dimensionalidad que tendrá cada punto, donde cada componente será un valor del conjunto de los números reales. Además, podremos tener un entero opcional *k*, que en caso de formar parte de la entrada, el resultado será dividido en *k* clústeres y en caso de no formar parte, se tendrá que encontrar la *k* óptima para los n elementos. También, dependiendo del dataset, podremos saber el valor de su groundtruth, comparando así la asignación de nuestros algoritmos en relación a la que se nos da y saber cuán buena es.

Para poder hacer un estudio formal de los algoritmos de clustering, primero tuvimos que entender el algoritmo de *k-means* y después hacer variaciones en torno a él. Así mismo, hemos decidido implementar como variantes el algoritmo de *k-means++,* que calcula inicializa los centroides de manera diferente, y *k-medoids*, que difiere con el *k-means* en el cálculo del centroide de los clústeres.

No solo conformes con hacer la implementación de los algoritmos de clustering, queríamos comprobar cuán buenos eran. Por ese motivo y para comprobar la calidad de la solución, implementamos un algoritmo de control de calidad intra clúster, que nos dé información sobre si los elementos de un clúster están bien integrados (average Silhouette), un algoritmo de control de calidad externa, que nos informe sobre la calidad de cada clúster en comparación con los demás (rand Index) y por último un algoritmo para calcular la k-óptima y, por lo tanto, en caso de haber una k especificada en la entrada nos diga si es la óptima o existe alguna mejor.

Finalmente, la experimentación. Consiste en hacer pruebas de nuestra implementación con la finalidad de medir, de ambos algoritmos implementados en la primera parte, su eficiencia y la solución obtenida. Para esto usaremos los datasets ofrecidos y los algoritmos implementados en la segunda parte para observar la calidad de los outputs de los algoritmos *k-means*, *k-means++* i *k-medoids*.

Además, para hacer nuestros propios juegos de prueba, hemos implementado un generador de datos sintéticos siguiendo el modelo gaussiano, de tal forma que podemos comprobar con mayor exactitud si nuestros algoritmos funcionan como esperábamos.

1 Primera parte: Algoritmo de k-means

Como se ha mencionado anteriormente, implementaremos los algoritmos de *k-means* y su variante *k-means++,* y también el *k-medoids*. Ambos algoritmos són muy parecidos.

Las tres variantes de clustering empiezan con la inicialización de los centros de los clústeres. Para *k-means* i *k-medoids* se escoge una instancia aleatoria de la entrada por cada clúster y se usa como centroide, en cambio para cada uno de los centros de los clusters del *k-means++* se usa un punto del dataset aleatorio.

Una vez ya se han inicializado todos los centroides, se entrará en un bucle cuya condición de salida será cuando entre dos iteraciones consecutivas los centroides no cambien entre ellos. Dentro del bucle mencionado, se seguirán los siguientes pasos:

Sea $\forall \{e_i , c_j\}$, i ∈ elementos, j ∈ clústeres, el par formado por el elemento e_i en el cluster c_j.

1. Reorganización de cada elemento al clúster con mejor centroide para ese elemento. La calidad del centroide según cada elemento se determinará con el cálculo de la distancia euclidiana entre los elementos y los centroides. Para cada par se calculará la distancia euclidiana, y se asignará e_i al clúster con menor distancia.

2. Cálculo de los centroides. Para este paso, cada uno lo hará de una manera diferente:
   Definimos ahora $\{c_j , p_j\}$, j ∈ clústeres, el par formado por el cluster c_j y su pivote p_j.

   A) k-means i k-means++

   Para cada elemento dentro de un mismo clúster, se sumarán los valores de sus componentes y se dividirá entre el total de elementos que forman el clúster. El resultado será el nuevo valor medio del pivote p_j en el clúster c_j.

   B) k-medoids

   En este caso, en vez de crear un nuevo pivote, se seleccionará un punto ya existente. Para ello, por cada cluster se recorrerán sus datos y se escogerá aquel cuyo sumatorio de distancias al resto de elementos sea menor. Es decir, nuestro p_j será la mediana de entre todos los puntos del clúster c_j.

En caso de estar ejecutando la solución con *k-means*, el bucle acabará en el caso en que de una iteración a la siguiente, no cambie de posición ningún centroide.
En caso de estar usando *k-medoids*, será algo parecido, ya que acabará la ejecución en caso de que de una iteración a otra, ningún elemento cambie de clúster.

Además hemos decidido añadir un máximo de iteraciones en caso de que se tarde en encontrar la solución óptima en cualquiera de los algoritmos. Aún así, como ya hemos comentado, si la solución no varía entre iteraciones cortamos el bucle, pues al ser algoritmos costosos no queremos seguir iterando.

2 Segunda parte: Control de calidad

En este apartado se explicarán los algoritmos usados para comprobar la buena funcionalidad de los algoritmos implementados en la primera parte.

2.1 Evaluación interna

Para tratar de evaluar el clustering formado de manera interna, hemos utilizado el famoso algoritmo llamado *Average Silhouette.* Como bien indica su nombre, consiste en retornar la silueta del clúster para así poder reconocer más claramente sus límites.

Primero, debemos analizar cada clúster individualmente por dentro como conjunto de elementos. Es decir, medir que cada punto del clúster está bien asignado a su cúmulo. Como se ha mencionado anteriormente, hemos implementado el algoritmo Silhouette que consiste en lo siguiente:

Supongamos que los datos ya han sido agrupados mediante *k-means*, *k-means++* o *k-medoids*, en k clusters. Para un punto de datos $i \in C_i$ (punto de datos i en el clúster C_i), tenemos:

$$a(i) = \frac{1}{|C_I|-1} \sum_{j \in C_I, i \neq j} d(i,j)$$

siendo $|C_I|$ el número de datos asignados a ese cluster, $d(i,j)$ sea la distancia euclídea entre los puntos de datos i y j en el cluster C_I, (se puede calcular con cualquier métrica, como la distancia Euclidiana o la Manhattan, en nuestro caso hemos usado la euclídea), y sea $\sum_{j \in C_I, i \neq j} d(i,j)$ el sumatorio de todas las distancias entre el punto i y todos los otros puntos dentro del mismo cluster.

Entonces, los que nos da esta expresión es la distancia media entre el punto i y todos los otros puntos de datos del mismo cluster, (dividimos por $|C_I| - 1$ porque no incluimos la distancia **d(i, i)** en la suma). Podemos interpretar **a(i)** como una medida de lo bien que esta que i está asignada a su cluster (cuanto más pequeño sea el valor, mejor es la asignación).

Luego se define la diferencia media del punto i a algún cluster C_K como la media de la distancia desde i a todos los puntos en C_K (donde $C_K \neq C_I$).

Para cada punto $i \in C_i$, definimos ahora:

$$b(i) = min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

Cogeremos la distancia media mínima de i a todos los puntos de cualquier otro cluster, del cual i no es miembro. El cluster con esta diferencia media más pequeña se dice que es el "cluster vecino" de i porque es el siguiente cluster que mejor se ajusta al punto de datos.

Ahora definimos un valor silueta de un punto de datos i.

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \; si \left|C_I\right| > 1$$
$$s(i) = 0 \; si \left|C_I\right| > 1$$

también puede ser escrito como:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

De la anterior definición podemos sacar que $-1 \leq s(i) \leq 1$.

Para que s(i) este cerca de 1 es necesario que a(i) << b(i). Como que un valor pequeño de a(i) significa que esta i está bien emparejada, además un valor grande de b(i) implica que i está mal emparejado con su cluster vecino. Por lo tanto si un s(i) es cercano a 1 significa que los puntos de datos están apropiadamente agrupados. Si s(i) está cerca de -1 por la misma lógica decimos que i sería más apropiado si se agrupara en el cluster vecino. Un s(i) cercano a 0 significa que el dato está en el borde de 2 clusters.

El coste de este algoritmo sería $O(n^2)$ ya que se tienen que calcular todas las distancias entre cada pareja de puntos del dataset.

Entonces encontramos una versión simplificada para el Silhouette clustering cuyo coste en cambio de ser $O(n^2)$ es $O(nk)$ donde K es el número de cluster que tenemos. Cuyo algoritmo es:

Suponiendo que para el clustering con centros μC_I (coordenadas de los centroides) para cada cluster C_I tenemos:

$$a'(i) = d(i, \mu C_I)$$
$$b'(i) = min_{Cj \neq Ci} d(i, \mu C_I)$$

Donde $a'(i)$ es la distancia desde el punto i hasta el centroide del mismo cluster y $b'(i)$ es la distancia mínima del punto i hasta un centroide de un cluster en el que el punto i no pertenece. Estas distancias se pueden calcular igual que en la implementación anterior. Entonces $s(i)$ quedara:

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

Para acabar si los centroides de cada cluster han sido calculada con k-medoids, se le puede decir *'medoid silhouette'*, donde a'(i) y b'(i) se calculan igual pero si cada punto de datos está asignado a su medoide más cercano sabemos que $a'(i) \leq b'(i)$ y por lo tanto:

$$s'(i) = \frac{b'(i)-a'(i)}{b'(i)} = 1 - \frac{a'(i)}{b'(i)}$$

2.2 Evaluación externa

Para realizar el análisis externo aplicaremos el algoritmo de *Rand Index*, de tal manera que podremos comparar la asignación de los clusters de manera externa y ver cuánto se parecen entre ellas.

Inicialmente, partiremos de dos asignaciones de puntos asignados a k-clústers para poder evaluar y obtener el índice. Si por ejemplo, comparamos la asignación que nos da el *k-means* respecto a la asignación dada por el *k-medoids*, con un buen valor de índice, significa que las asignaciones elaboradas con los diferentes algoritmos son consistentes entre sí.

Analizaremos el método evaluativo basándonos en el paper publicado por Hubert & Arabie, en 1985, titulado "Comparing Partitions". En dicha publicación, se habla al detalle de más variantes de comparaciones del Rand Index y evaluaciones entre clusters de forma externa.

En un principio, el algoritmo que habíamos pensado era demasiado costoso, suponiendo una cota cuadrática. Ahora, hemos podido reducirla a un coste lineal, gracias a utilizar una estructura con hashing. En concreto, usaremos unordered maps, donde la clave será el identificador del punto (de 1 a n), y el valor será el cluster al que están asignados.

Formaremos una tabla de contingencia dados los unordered maps, donde cada elemento de la tabla $n_{ij} = |U_i \cap V_j|$, que expresa el número de objetos que comparten clusters en U_i y V_j .

Partition V

Partition U	Class	v_1	v_2	...	v_C	Sums
	u_1	n_{11}	n_{12}	...	n_{1C}	$n_{1\cdot}$
	u_2	n_{21}	n_{22}		n_{2C}	$n_{2\cdot}$
	.	.	.		.	.
	.	.	.		.	.
	.	.	.		.	.
	u_R	n_{R1}	n_{R2}	...	n_{RC}	$n_{R\cdot}$
	Sums	$n_{\cdot 1}$	$n_{\cdot 2}$	...	$n_{\cdot C}$	$n_{\cdot\cdot} = n$

Representación de la tabla de contingencia

A partir de aquí, tendremos que encontrar los elementos de la tabla que concuerdan entre asignaciones, llamados también como *agreements*. Se pueden medir a partir de los pares de objetos que están en la misma clase tanto de U como de V, o bien en diferentes clases en ambas, o bien que están en la misma clase en U pero en diferente clúster en V, o finalmente en distinto clúster en U pero el mismo clúster en V.

Nosotros nos centraremos en encontrar los *agreements* para los pares de objetos que estén en el mismo clúster tanto en X, como al obtener la ejecución en Y, y los pares de objetos diferentes tanto en X como en la clase Y.
De tal manera, y apoyándonos del paper, la fórmula resultante es la siguiente:

$$A = \binom{n}{2} + \sum_{i=1}^{R}\sum_{j=1}^{C} n_{ij}^2 - \frac{1}{2}\left[\sum_{i=1}^{R} n_{i\cdot}^2 + \sum_{j=1}^{C} n_{\cdot j}^2\right]$$

Para hacer el respectivo cálculo, hemos dividido nuestro programa en tres fases. Cabe destacar que todos los valores serán obtenidos de la tabla de contingencia. En la primera de las fases, obtendremos el valor de (n sobre 2). En la segunda, haremos el cálculo para obtener el sumatorio de las celdas al cuadrado. Por último, la fase para obtener el sumatorio de cada $n_i^{\,2}$, $n_j^{\,2}$ recorriendo todas las filas y columnas, respectivamente.

Una vez calculado dicho valor A, basta con dividir A / (n sobre 2) para obtener el índice que usa Rand. Hubiéramos podido calcular otras variaciones como la de Hubert, que sugiere (A-D) / (n sobre 2), donde D es el número total de desacuerdos o *disagreements*, o por ejemplo la de Arabie y Boorman, que sugiere D / (n sobre 2) . Pero como ya hemos dicho, nos quedaremos con la que Rand, en 1971, llevó a cabo:

$$A/\binom{n}{2}$$

En la experimentación veremos que si obtenemos un valor lo más cercano a 1 posible, es que la clusterización con el algoritmo que ha sido ejecutada, es buena asignación. De la misma manera, se penalizará mucho más a un conjunto inicial pequeño que contenga un fallo, a diferencia que tengamos un fallo en un dataset considerablemente más grande.

2.3 Optimización de k

Por último tenemos que implementar un método para encontrar la k-óptima (siendo k el número de clusters) para conseguir una k que no sea muy baja, debido a su mínima partición ni una k muy alta, donde tendríamos muy pocos elementos en cada cluster. Sabiendo esto, necesitamos encontrar una k que nos proporcione un equilibrio entre los dos casos extremos mencionados.

Para poder realizar este método implementaremos el algoritmo propuesto Elbow Method:
Describimos el Elbow Method como la suma cuadrada de los errores

$$(\text{SSE}) = \sum_{k=1}^{K} \sum_{x_i \in S_k} \left| X_i - S_k \right|^2 ,$$

donde la k es el número de cluster, la X_i un punto del cluster y la S_k el centro de un k-cluster.

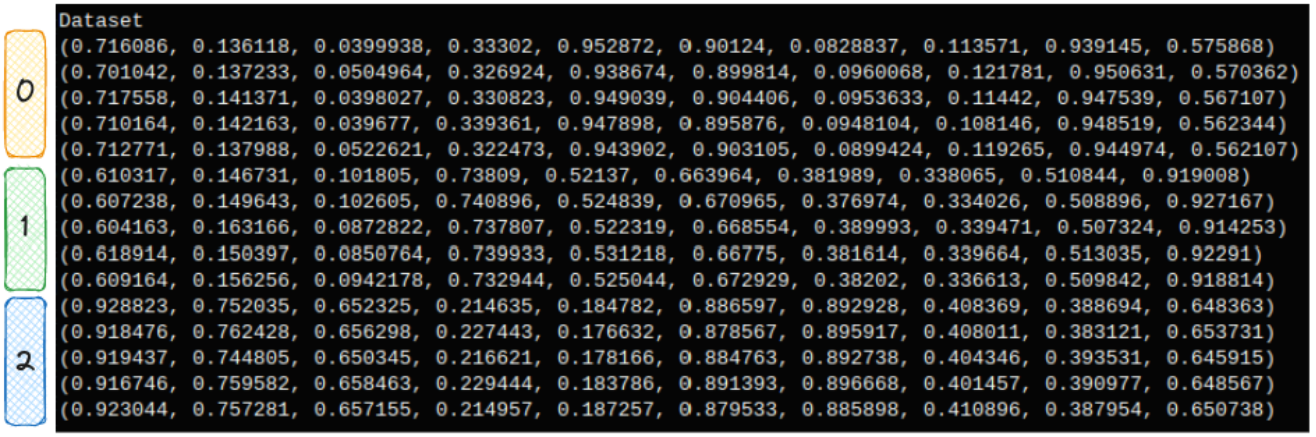
Deberemos aplicar los algoritmos de resolución incrementando la k de forma que vayamos calculando el error que toma en cada k diferente. Una vez hemos hecho los cálculos hasta una k = 20, nos quedaremos con la k tal que $max(SSE_{k-1} - SSE_k)$. Aun siendo cierto que existen SSE inferiores con otros valores de k, buscamos el punto donde la diferencia ha sido la más significativa.

2.4 Generador de datos sintéticos

No conformes con los datasets que se nos proporcionaban a la hora de realizar la práctica, decidimos crear nuestro propio generador de datos de entrada. Se basa en un modelo de creación de clusters de manera gaussiana, pues a partir de k puntos representativos, donde k es el número total de clústers, se generarán puntos alrededor, dados un radio y un ángulo sigma para poder ir rotando los nuevos puntos que vayamos añadiendo.

Al iniciar el programa, pediremos al usuario que nos proporcione el número de clusters que desea, la dimensionalidad de los puntos de todo el dataset, y cuántos puntos querrá por cluster. El total de puntos se obtendrá a partir de la multiplicación de los puntos que haya en cada clúster por el total de clústeres. Interactivamente, se vería algo así:

```
Vols generar un dataset o vols utilitzar un que ja existeix?
1) Generar dataset nou
2) Escollir dataset existent
1
Escull el numero de clusters: k = 3
Escull ara les dimensions dels punts: d = 10
Ara, quants punts voldràs per cluster? 5
Perfecte! S'acaba de crear un dataset a mida.
```



3 Tercera parte: Experimentación

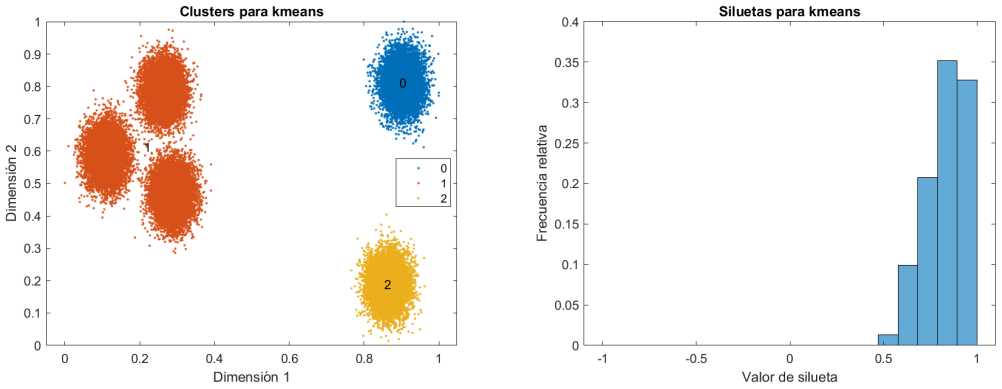
3.1 Algoritmos de la primera parte

Para experimentar y comprobar la calidad de nuestros algoritmos haremos 10 pruebas en cada algoritmo diferente para los datasets 1, 2, 3 i 6 que són los que tienen una k definida. De esta manera podremos ver como se desenvuelve cada algoritmo en los diferentes datasets. Para crear todos los experimentos hemos creado un script (adjunto en el zip con el nombre de *script.sh*) y un código de Matlab (incluido en el zip como *creadorGrafiques.m*). Todas las de los clústeres y centroides creados y los histogramas de las siluetas están incluidos en el apéndice.

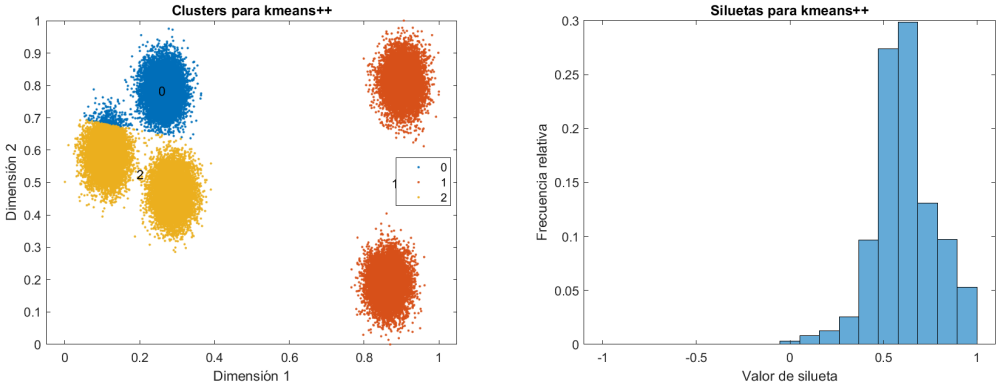
En este apartado comentaremos un único experimento en los tres algoritmos diferentes hecho con el dataset 6 ya que es el único con d = 2 y por lo tanto el único que puede estar completamente representado con 2 dimensiones.

Experimento 1. Dataset 6

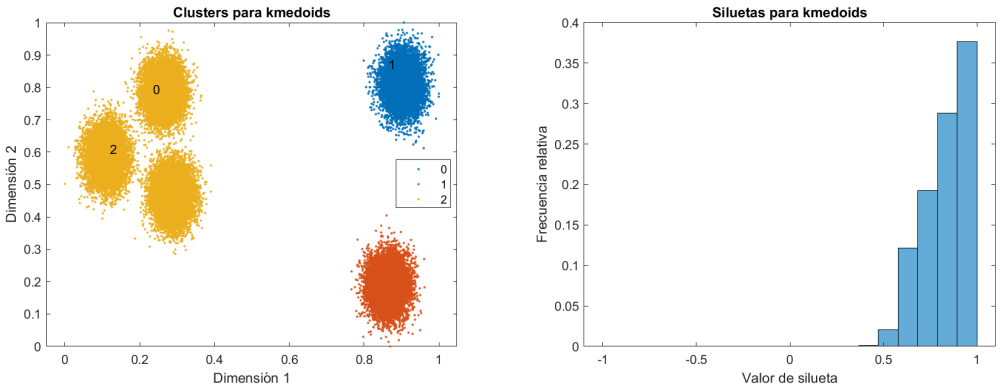
- **Output usando el algoritmo K-means**



- **Output usando el algoritmo K-means++**



- **Output usando el algoritmo K-medoids**



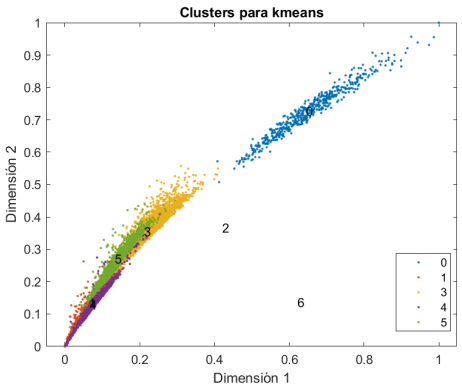
Para empezar, se puede observar fácilmente que la repartición de puntos en los 3 clusters ha sido diferente en el *k-means++* respecto a los otros dos. En el dataset 6 podemos ver como hay 5 agrupaciones de instancias y en los 10 experimentos hechos en este dataset se podía observar una de las siguientes salidas; uno en el que los tres grupos de instancias de la izquierda forman parte de un cluster y cada una de las dos agrupaciones de la derecha forman parte de otros dos clusters diferentes (como se puede ver en este ejemplo para los algoritmos de *k-means* y *k-medoids*) o bien las dos agrupaciones de la derecha forman parte de un mismo cluster y las tres agrupaciones de la izquierda están divididas por la mitad y cada una de las mitades forma parte de un cluster diferente (como se puede ver en el output del algoritmo *k-means++*). Esto es debido a que el cálculo del centroide por primera vez es aleatorio, por lo tanto dependiendo de cómo se instancian esta primera vez, la agrupación de cada cluster después de estabilizarse acabará de una de estas dos maneras.

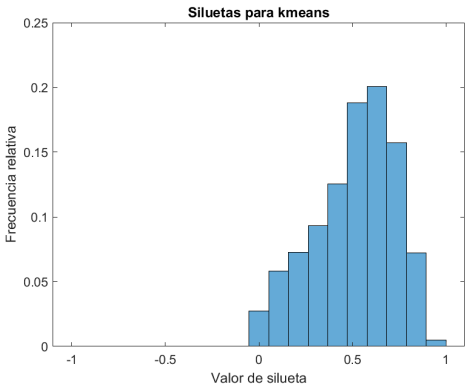
En cuanto a los histogramas de las siluetas, podemos ver que si el experimento termina como el output proporcionado por el *k-means* o el *k-medoids*, este gráfico nos dará mejores resultados que si acaba como en el *k-means++*. Esto es debido a que la de las distancias entre las instancias y el centroide de su respectivo cluster es menor.

Como se ha comentado en las explicaciones de los algoritmos, los centroides del *k-medoids* usan instancias de los datasets como centros de los clusters mientras que el *k-means* y el *k-means++* crean sus propios puntos aleatoriamente. Esto produce algo curioso que queremos comentar y para ello usaremos el experimento 5 del dataset 1 aunque no podamos representar todas sus dimensiones.

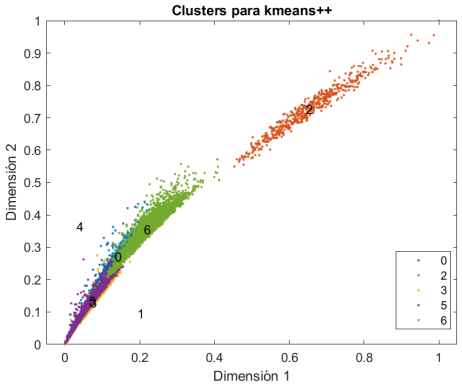
Experimento 4. Dataset 1

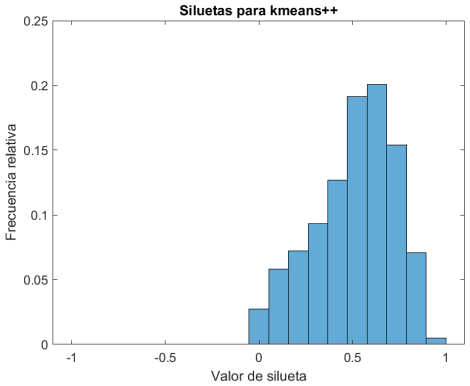
- **Output usando el algoritmo K-means**



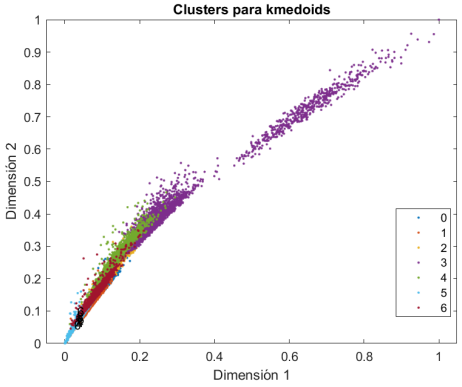


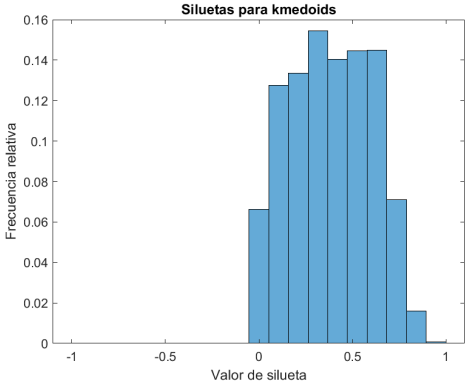
- **Output usando el algoritmo K-means++**





- **Output usando el algoritmo K-medoids**





Como se puede observar en este experimento y en muchos de los que hemos hecho con el primer dataset, tanto en experimento del *k-means* como en el del *k-means++*, hay centroides que no tienen ningún cluster asignado, mientras que en los experimentos hechos con el *k-medoids* siempre habrá 6 clusters diferentes. Esto es debido a que como acabamos de comentar, los centroides del *k-medoids* usan una instancia ya existente y por lo tanto en todo momento habrá una instancia que esté lo suficientemente cerca a cada centroide y desde un principio habrán 6 clusters diferentes definidos. Mientras que en los dos otros algoritmos el

centroide es creado en un punto aleatorio y esto causa que si por un centroide *c* no existe ninguna instancia que esté más cerca de este centroide que de otro, entonces *c* no formará parte de ningún cluster y no se actualizará en ninguna de las iteraciones.

3.2 Tiempos de ejecución

A continuación comentamos los tiempos de ejecución para cada uno de los experimentos hechos y su correspondiente tiempo medio y tanto para el *k-means* como para el *k-means++* como para el *k-medoids*.

Queremos remarcar que los tiempos de ejecución han sido más lentos e inconsistentes de lo habitual por el ordenador en el que han sido ejecutados para hacer los experimentos. Experimentando en otro ordenador diferente, con el dataset 1 se ha ejecutado el código con los siguientes tiempos:

- **K-means**: 2.7548s
- **K-means++**: 0.9921s
- **K-medoids**: 8.5151s

Mientras que en los experimentos nos salen la media de tiempos siguiente:

- **K-means**: 23.85s
- **K-means++**: 35.78s
- **K-medoids**: 50.31s

- ***K-means***

13.62	128.2	23.28	18.68	19.38	19.13	19.05	19.74	19.01	16.99

Tiempo medio: 29.78 s.

- ***K-means++***

12.97	118.14	16.83	17.56	13.18	14.59	11.16	17.98	9.80	20.75

Tiempo medio: 25.29 s.

- ***K-medoids***

402.88	605.56	483.29	446.89	733.97	397.83	447.32	433.30	416.48	514.33

Tiempo medio: 488.18 s

Entre los tiempos del *k-means* i *k-means++* podemos observar que en la mayoría de casos el *k-means++* es más rápido que el *k-means* esto se debe a que el *k-means++* en canvio de generar las coordenadas de las centroides de forma aleatoria entre 0 y 1, coje como coordenada un punto al azar de los ya existentes. Entonces podemos decir que el *k-means++* tiene una mejor inicialización que el *K-means*, eso provocará que se tenga que hacer un menor nombre de iteraciones hasta que los centroides de una iteración a otra no varíen.

En el caso que el *K-means* es más rápido al *k-means++* puede deberse a que en el *k-means*, aun siendo una inicialización aleatoria, los centroides generados tengan una mejor posición que los del *k-means++*. Esto causa que el tiempo de ejecución del k-means sea más rápida que el *k-means++* en ese experimento en concreto.

En cuanto al tiempo del tercer algoritmo, su lentitud es causada por el cálculo de los centroides. Mientras que en los otros dos, los centroides se calculan con un coste $\theta(n * d)$ sea n el número de instancias del dataset y d el número de dimensiones. En cambio en el algoritmo k-medoids, por cada instancia, se tiene que calcular la distancia media a todas las demás instancias de su clúster para seleccionar el mejor centroide. Esto hace que el coste de recalcular los centroides en cada iteración será de $O(n^2 * d)$.

En el segundo experimento, en todos los algoritmos, podemos ver una anomalía en el tiempo. Esto puede haber ocurrido por una singularidad interna de la máquina, y como los experimentos han sido generados por un script y por lo tanto se ejecutan secuencialmente uno detrás del otro.

También podemos observar otra irregularidad en el experimento número 5 hecho con el algoritmo de k-medoids, causado probablemente por una razón similar.

3.3 Comparación con Rand Index

Para cada dataset, haremos comparaciones entre los algoritmos, y si coincide que los datasets están etiquetados, entonces también los compararemos con el groundtruth.

A) Para un dataset generado por nosotros mismos (k = 3, d = 8, n = 3000):

Número de Exp.	k-means	k-means++	k-medoids
Experimento 1	0,77	1	0,72
Experimento 2	0,77	0,77	0,73
Experimento 3	0,77	1	1
Experimento 4	1	1	1
Experimento 5	0,33	0,77	1
Experimento 6	1	1	1
Experimento 7	0,33	1	1
Experimento 8	0,33	1	1
Experimento 9	0,77	1	1
Experimento 10	0,77	1	0,72
Promedio	**0,69**	**0,96**	**0,92**

Por un lado tenemos la asignación de la generación de datos de forma gaussiana, que nos dice exactamente cada punto a qué cluster pertenece unívocamente, ya que hemos tenido mucho cuidado a la hora de crearlos. Si obtenemos las ejecuciones de cada algoritmo ejecutado, vemos que los valores obtenidos con una dimensión d = 8, los índices son prácticamente 1, la cuál cosa nos hace pensar que nuestros algoritmos clasifican sumamente bien.

B) A continuación, de entre los datasets existentes, sólo compararemos el 1, 2, 3 con el groundtruth, pues estos son los que contienen etiqueta. Véase la tabla a continuación. Dichos valores están al completo en el apéndice.

Número de Exp.	k-means	k-means++	k-medoids
Dataset 0	0,69	0,96	0,92
Dataset 1	0,84	0,89	0,85
Dataset 2	0,51	0,50	0,51
Dataset 3	0,84	0,84	0,84

Aquí también vemos como notablemente el resultado de comparar clústeres de manera externa es bastante elevado, menos el caso del Dataset 2, pues ha coincidido que para los tres algoritmos, el índice comparativo no es lo suficientemente bueno.

C) Los datasets 4, 5, 6 como no tienen etiqueta, se comparan entre ellos. He aquí la media de 10 experimentos, usando la k que te dan o la que se calcula implícitamente.

Dataset 4 (k = ??, d = 10, n = 980)

Número de Exp.	k-means	k-means++	k-medoids
k-means	1	0,89	0,82
k-means++	0,89	1	0,84
k-medoids	0,82	0,84	1

Dataset 5 (k = ??, d = 24, n = 5456)

Número de Exp.	k-means	k-means++	k-medoids
k-means	1	0,94	0,88
k-means++	0,94	1	0,88
k-medoids	0,88	0,88	1

Dataset 6 (k = 3, d = 2, n = 50000)

Número de Exp.	k-means	k-means++	k-medoids
k-means	1	0,83	0,95
k-means++	0,83	1	0,98
k-medoids	0,95	0,98	1

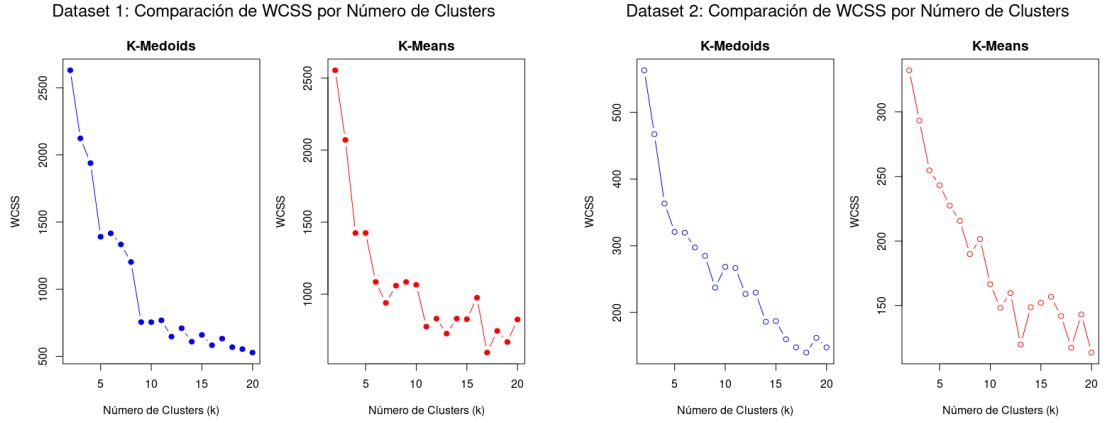
Observamos que en los tres casos son matrices simétricas, pues el resultado del Rand Index calcula la diferencia entre ambos, sin tener en cuenta cuál de las asignaciones pasas antes como parámetro para calcular el índice.

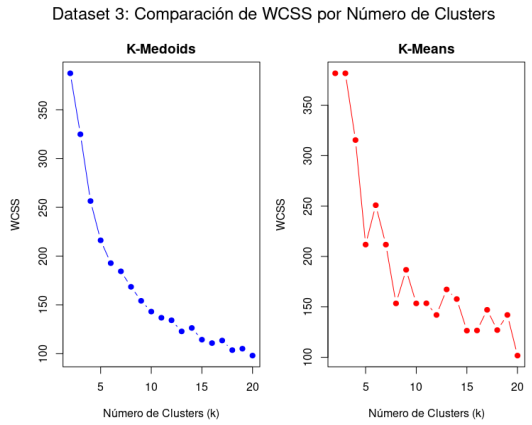
Al ver unos resultados con índices tan elevados, podemos concluir que haciendo la ejecución con un algoritmo u otro, al final acaba saliendo un resultado muy parecido. Esto sería una gran ventaja, ya que nos está diciendo que ambos algoritmos son consistentes; es decir, que su solución se asemeja bastante. Si no necesitáramos tanta precisión a la hora de clasificar clústeres, podríamos escoger el algoritmo con menor coste, por ejemplo, ya que así nos ahorraríamos mucho tiempo, obteniendo una solución muy parecida al ejecutar otro algoritmo.

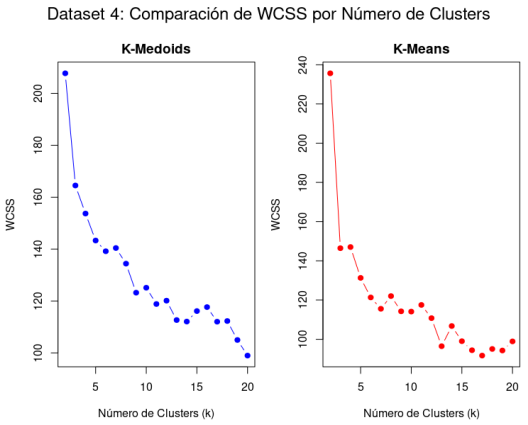
Por tanto, aquí juega un papel muy importante la siguiente pregunta: ¿Qué queremos, una buena solución aunque tengamos que pagar un elevado coste? ¿O nos conformamos con una solución decente, siendo la ejecución de ésta mucho más rápida? Eso, en efecto, va a criterio de cada quién.

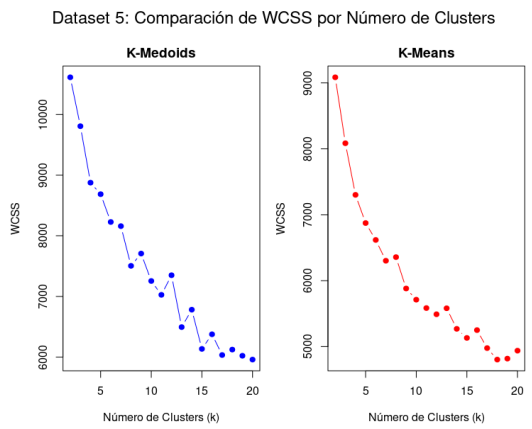
3.4 Computación k-óptima

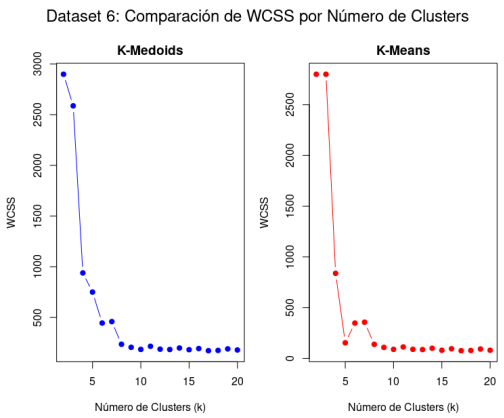
A continuación, se muestran las gráficas del Elbow Method para encontrar la k-óptima. Al ser un problema clasificado como NP-Hard, es difícil determinar con total precisión cuál es la mejor k, o la más conveniente.











También hay polémica y es criticado debido a que es bastante subjetivo a la hora de coger el k-Óptimo, pero en este proyecto nos hemos guiado por hacer una evaluación de una búsqueda entre el rango de k = 2 a k = 20 y cogiendo el mejor k a partir del valor del WCSS calculado de la siguiente forma: $max(WCSS) - (max(WCSS) - min(WCSS)) * 0.9$ y la k que tenga el WCSS más cercano a este será el óptimo.

Para el primer dataset, podemos observar que hay una bajada drástica en el WCSS ejecutando k-medoids, por eso a simple vista podríamos decir que la k se encuentra en la posición 9. Además si utilizamos la fórmula explicada previamente, podemos ver que: $2600 - (2600 - 500) * 0,9 = 710$, lo que equivaldría a la k = 9, que es el primer resultado de WCSS inferior a 1000. Es por eso que podemos deducir que la k-óptima será k = 9.

Para los datasets 4 y 5 donde tenían interrogante, las k-óptimas serían: 12 para el dataset 4 y 14 para el dataset 5, siguiendo el cálculo realizado como en el caso del dataset 1 previo.

Bibliografía

Francisco Sanz, Clustering y cómo funciona, the Machine Learners © 2024

https://www.themachinelearners.com/k-means/

Departamento de Matemática Aplicada, Aprendizaje no supervisado, Clustering, Universidad Politécnica de Madrid

https://dcain.etsin.upm.es/~carlos/bookAA/03.1_Clustering-K-Means.html

Admin, Agrupamiento con Minería de datos, Estrategias de Trading

https://estrategiastrading.com/k-means/

Lorena Ramírez, Algoritmo de k-means, IEBS

https://www.iebschool.com/blog/algoritmo-k-means-que-es-y-como-funciona-big-data/

Remolinator contributors, Agrupamiento de datos eficiente, Remolinator

https://remolinator.com/k-means-agrupamiento-de-datos-eficiente/

Wikipedia contributors, K-means clustering, Wikipedia

https://en.wikipedia.org/wiki/K-means_clustering

Wikipedia contributors, K-means++ clustering, wikipedia

https://es.wikipedia.org/wiki/K-means%2B%2B

Geeks For Geeks contributors K-means++ Algorithm

https://www.geeksforgeeks.org/ml-k-means-algorithm/

David Arthur, Sergei Vassilvitskii, The Advantages of careful seeding

https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf

Wikipedia contributors, K-medoids, Wikipedia

https://es.wikipedia.org/wiki/K-medoids

Eesha Farrukh, K-medoids algorithm, Educative

https://www.educative.io/answers/what-is-the-k-medoids-algorithm

Zubair Ahmed, K-medoids algorithm, Medium

https://medium.com/@princeans.ansari2881/k-medoids-clustering-an-approach-to-unsupervised-learning-algorithm-2468c4448051

Geeks for Geeks contributors, Gaussian Mixture Model, Geeks for Geeks

https://www.geeksforgeeks.org/gaussian-mixture-model/

Wikipedia contributors, Silhouette (Clustering), Wikipedia

https://en.wikipedia.org/wiki/Silhouette_(clustering)

Wikipedia contributors, Rand index, Wikipedia

https://en.wikipedia.org/wiki/Rand_index

Lawrence Hubert & Phipps Arabie, Comparing Partitions, Paper

https://link.springer.com/article/10.1007/BF01908075

Wikipedia contributors, Elbow method (clustering), Wikipedia

https://en.wikipedia.org/wiki/Elbow_method_(clustering)

M A Syakur, Integration K-Means Clustering Method and Elbow Method For Identification of The Best Customer Profile Cluster

https://iopscience.iop.org/article/10.1088/1757-899X/336/1/012017/pdf

Geeks For Geeks contributors Elbow Method for optimal value of k in KMeans

https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

Basil Saji, Elbow Method for Finding the Optimal Number of Clusters in K-means

https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/