In [1]:
```
from mpl_toolkits.mplot3d import Axes3D
from scipy.io import loadmat
#from sklearn import datasets
import pandas as pd
from sklearn.datasets import make_blobs, make_moons, make_circles
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.cluster import AgglomerativeClustering
%matplotlib inline
%load_ext autoreload
%autoreload 2
from skeleton import *
```

Clustering

El proceso de clustering o agrupamiento, es un procedimiento de aprendizaje no supervisado utilizado para descrubir/explorar la estructura de una colección de objetos. El objetivo es dividir el conjunto de objetos en grupos/clusters/clases de objetos **similares** mientras son separados de objetos **no similares**. Las clases generadas representan una generalización sobre los objetos y sus características.

Formalemete, el problema puede definirse como sigue: utilizando como entrada una colección de objetos X y un valor entero K se debe obtener un conjunto $G = \{G_1, G_2, \ldots, G_K\}$. Donde cada objeto $x_i \in X$ es un vector definido en $\mathbb{R}^N$, se cumple que $1 < K \leq |X|$. Al finalizar el proceso cada elemento x_i debe pertenecer a uno de los grupos en G

Definiciones adicionales

- m_k es el número de elementos en el grupo G_k, es decir $m_k = |G_k|$
- c_k será utilizado para referirse al centroide de todos los elementos en G_k, el cuál puede ser:
  - la media geométrica

$$c_k = \frac{1}{m_k} \sum_{i=1}^{m_k} x_i$$

  - la mediana
  - uno de los elementos en el grupo(centro)
- Se utilizará M para hacer referencia al conjunto de centroides $\{c_1, c_2, \ldots, c_k\}$.
- En el problema de clustering a considerar cada objeto x_i solo pertenece a un grupo, por lo tanto para cualquier par de grupos G_A, G_B donde $A \neq B$, se cumple que:

$$G_A \cap G_B = \emptyset$$

.

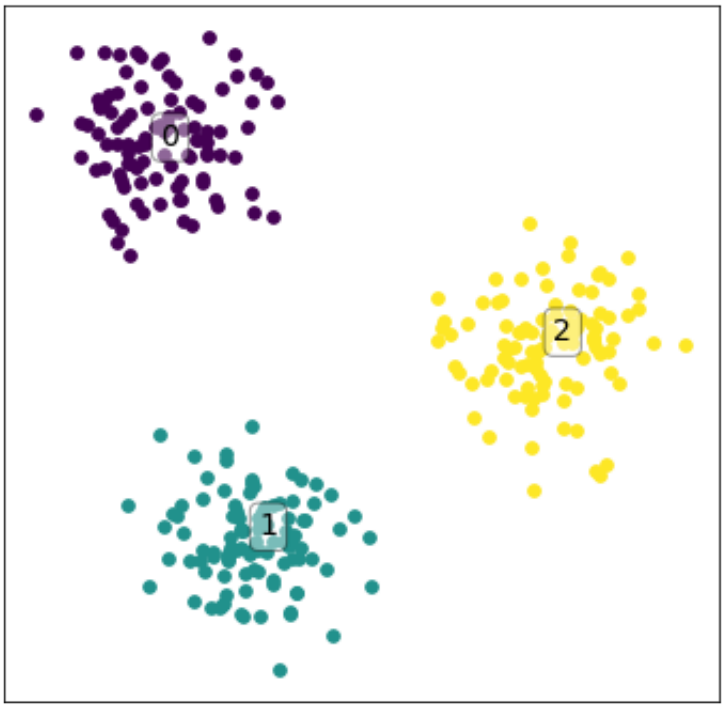
In [2]:
```
# Un ejemplo de 3 clusters bien definidos con 100 puntos cada uno
# se utiliza un distribución gaussiana
from sklearn.datasets.samples_generator import make_blobs
blobs_centers = [[-4, 4], [-2, -4], [4, 0]]
blobs_data, blobs_labels = make_blobs(n_samples=300,centers=blobs_centers,ran
blobs_classes =dict(zip(set(blobs_labels),blobs_centers))
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:143: F
utureWarning: The sklearn.datasets.samples_generator module is  deprecated in
version 0.22 and will be removed in version 0.24. The corresponding classes /
functions should instead be imported from sklearn.datasets. Anything that cann
ot be imported from sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

In [3]:
```
blobs_classes
```

Out[3]:
```
{0: [-4, 4], 1: [-2, -4], 2: [4, 0]}
```

In [4]:
```
# La función plotCusters está definida en skeleton.py
plotClusters(blobs_data,blobs_labels,centroids=blobs_classes)
```



Como mencionamos el proceso de clustering es una tarea de exploración de datos, lo cual implica que normalmente no se cuenta con información relativa a los resultados esperados del análisis. Algunos de los resultados esperados podrían ser:

- El número de grupos K
- El tamaño de los grupos (número de elementos en cada cluster)
- La densidad de los grupos
- La posición de los clusters

Existen muchas tareas donde se aplica el proceso de clustering, las cuales van desde aplicaciones de recuperación de información, biología, medicina, negocios, entre otros.

Podemos sumarizar el proceso de clustering como sigue:

1. Definición del dominio, lo cual implica:
    - identificar los objetos a analizar
    - determinar el propósito de generar grupos
    - el conjuto de características que describen los objetos
2. Definir una función de comparación, es decir como se determina que tan parecidos son los objetos
3. Definir como medir la calidad de los gurpos encontrados.
4. Determinar el algoritmo de clustering a utilizar

Medidas de distancia/similitud

Una ves definido el dominio del problema es necesario contar un mecanismo para comparar los objetos en la colección. Lo anterior con la finalidad de decir que tan similares o distante es un objeto x_i con respecto de otro x_j. Como ya hemos visto en la sección anterior, para comparar objetos definidos en un espacio vectirial, se puede utilizar una función de distancia d o bien una función de similitud sim. Distancia y similitud están relacionados, desde que dos objetos que están a una distancia pequeña se puede decir que son similares. En la siguente tabla se resumen algunas de las funciones de distancia reportadas en la literatura:

Nombre	Función
Minkowski	$L_q(x_i, x_j) = \sqrt[q]{\sum_{k=1}^{N}(x_{ik} - x_{jk})^q}$
Coseno (simulitud)	$\cos(x_i, x_j) = \frac{\sum_{k=1}^{N} x_{ik} * x_{jk}}{\sqrt{\sum_{k=1}^{N} x_{ik}^2}\sqrt{\sum_{k=1}^{N} x_{jk}^2}}$
Coseno (distancia)	$dcos(x_i, x_j) = 1 - \cos(x_i, x_j)$
Divergencia de Kullback-Liebler	$kld(x_i, x_j) = \sum_{k=1}^{N} x_{ik} * \log \frac{x_{ik}}{x_{jk}}$

Medidas de calidad

Evaluar si un agrupamiento G es **"bueno"** o no, no es una tarea simple, desde que no existe un criterio definitivo para determinarlo. Sin embargo se han propuesto muchos criterios de evaluación, los cuales podemos dividir en externos e internos.

Medidas internas

Este tipo de medidas evalúan que tan compactos son los clusters mediante el uso de una medida de similitud o distancia. Esta clase de métricas generalmente intentan medir la cohesión en cada cluster (intra-cluster), la separación de los diferentes clusters (inter-cluster) o bien una combinación de ambas. Algunos ejemplos son:

La suma de los errores al cuadrado (SSE o inercia)

SSE es la medida más simple y consiste en calcular la suma de la distancia de todos los objetos con respecto de los centroides de sus respectivos grupos. Se cálcula mediate la siguente formula:

$$SSE = \sum_{k=1}^{K} \sum_{\forall x_i \in G_k} (x_i - c_k)^2$$

donde G_k es el k-ésimo cluster y c_k es el centroide del grupo. SSE es una forma simple de medir que tan diferentes son los elementos en un cluster con respecto de su centro geométrico. Esta medida permite maximizar la similitud entre objetos en el mismo grupo mediante la minimización del SSE.

Coeficiente de Silueta

El coeficiente de silueta trata de minimizar la distancia intracluster mientras maximiza la distancia inter-cluster. El coeficiente $sil(G)$ para un conjunto de clusters G se determina como sigue:

1. Para un elemento $x_i \in G_A$ su distacia promedio a_i con respecto de todo los elementos en su mismo cluster se cálcula como sigue:

$$a_i = \frac{1}{m_A - 1} \sum_{\forall_{x_j} \in G_A} d(x_i, x_j)$$

1. Se cálcula su distancia promedio con mínima b_i con respecto a los elemento que no están en el mismo cluster al que pertenece x_i

$$b_i = \min_{G_B \neq G_A} \frac{1}{m_B} \sum_{\forall_{x_j} \in G_B} d(x_i, x_j)$$

1. Con el fin de tratar de maximizar la separación entre x_i y los elementos que no están en su mismo grupo, mientras se minimiza la distancia promedio con los que se encuentran en su mismo grupo; se realiza el siguiente cociente:

$$sil(x_i) = \frac{b_i - a_i}{\max(a_i, b_i)}$$

1. Se suma el coeficiente para todos los elementos en un mismo clusters

$$sil(G_k) = \frac{1}{m_k} \sum_{x_i} sil(x_i)$$

1. Finalmente se suman los coeficientes de cada grupo para obtener el coeficiente global

$$sil(G) = \frac{1}{K} \sum_{k=1}^{K} sil(G_k)$$

Está métrica tiene un costo mucho mayor que el SSE ya que en el peor caso se requiere conocer la distancia entre todos los elementos de la colección lo cual no es factible para coleccion muy grandes.

Medidas externas

Las medidas externas requeiren que se conozcan a priori el número de grupos, así como la pertenencia de los elemetos a cada uno de ellos. Algunas de la medidas que pueden utilizarse son $precision$, $recall$ y F_1. Si recodamos:

$$recall = \frac{tp}{tp + fn}$$

$$precision = \frac{tp}{tp + fp}$$

$$F_1 = 2\frac{precision \cdot recall}{precision + recall}$$

donde tp son los verdaderos positivos y fn los falsos negativos, en el contexto de clustering dado un grupo G_k los tp serían los elementos que fueron asignados a G_k y sí pertenecen a G_k. Por otro lado los fn serían los elementos que fueron asignados G_k sin pertenecer a él. Los falsos positivo fp son los elementos que fueron asignados a un cluster diferente a G_k.

Índice Rand Ajustado (ARI)

De forma breve, ARI mide las similitudes de las particiones de dos asignaciones diferentes, ignorando las posibles permutaciones y ajustando la puntuación cerca de los valores 0 cuando las particiones se generan al azar. La versión no ajustada del índice Rand (RI) se describe mediante la siguiente fórmula:

$$RI = \frac{a + b}{C_2^n},$$

donde C_2^n es el número total de pares que se pueden formar a partir de una base de datos de tamaño n, es decir, $\binom{n}{2}$. Sea T el conjunto de particiones "reales" (ground truth) y P el conjunto de particiones creado por el algoritmo de clustering, a el número de pares que perteneces al mismo grupo en T y P y b el número de pares que están en diferentes particiones tanto P como en T. La versión ajustada (ARI) se define mediante la siguiente ecuación:

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]},$$

donde E es el valor esperado.

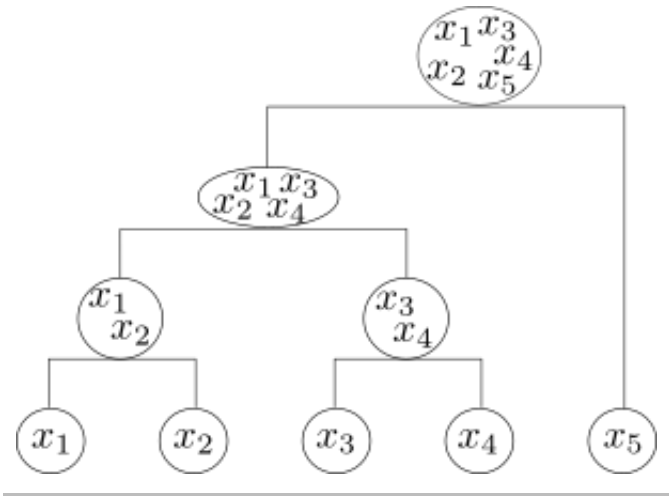
Estas y otras métricas de calidad de clustering ya están implementadas en `sklearn.metrics` https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

Algoritmos de Clustering

Los diferentes algoritmos de clustrering pueden ser divididos con respecto a como crean los grupos, esto es en si lo hacen de forma jerárquica o particional. Un algoritmo particional crea el conjunto de clusters dividiendo los objetos en grupos disjuntos, lo cual se puede realizar hasta obtener grupos con un único elmento. Mientras que los métodos jerárquicos permiten la existencia de sub-grupos, es decir grupos anidados.

Algoritmos particionales

El proceso jerarquíco puede verse como un árbol binario (dendrograma) en el que las hojas son los clusters que contienen un solo elemento, los nodos internos representan la unión de sus dos hijo y la raíz sería un único cluster que contiene todos los objetos en la colección.



Dendrograma (Clustering Jerárquico)

Los métodos jerárquicos puede construirse de forma aglomerativa o divisiva. En ambos casos, el proceso de agrupamiento puede describirse mediante el árbol dendorgrama, donde las hojas son los grupos que contienen un solo elemento. Los nodos internos representan la unión de sus dos hijos (aglomeración) o la división de un nodo padre (división). La raíz es un clúster único que contiene todos los objetos de la colección. El dendrograma representa una estructura jerárquica para los datos. Este es más informativo que el agrupamiento por algoritmos particionales. La agrupación jerárquica tiene la ventaja de que no necesariamente se requiere el número de grupos K como parámetro de entrada. Además, los algoritmos jerárquicos suelen ser deterministas (es decir, se logra la misma jerarquía para el mismo conjunto de entradas-salidas). Sin embargo, son más costosos que los algoritmos particionales

En la siguiente sección describiremos brevemente en que consites una estrategía aglomerativa

Criterios de unión (linkage)

El criterio de unión de clusters define que medida se utiliza para unir los grupos en cada nivel de dendograma. Suponga que dos elementos x_i y x_j tal que $x_i \in G_p$, $x_j \in G_q$ y c_p, c_q son la media geométrica(centroide) de los grupos G_p y G_q respectivamente. Se puede elegir que clusters mezclar mediante alguno de los siguientes criterios:

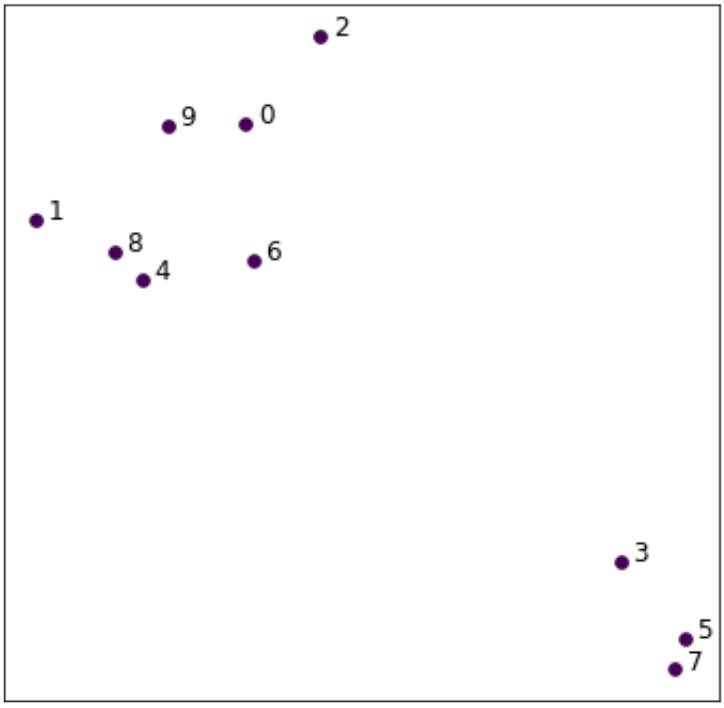
- **Single**: Une los dos clusters G_p y G_q que contienen el el par (x_i, x_j) tal que su distancia es mínima.
- **Complete**: Se eligen G_p y G_q contienen el par de elementos (x_i, x_j) tal que su distancia máxima es la menor.
- **Centroid**: Se unen G_p y G_q tal que la distancia de sus centroides es mínima.
- **Ward**: Se unen los dos clusters que minimizan en incremento de la varianza de clustering.

Librerías de clustering jerárquico

scipy

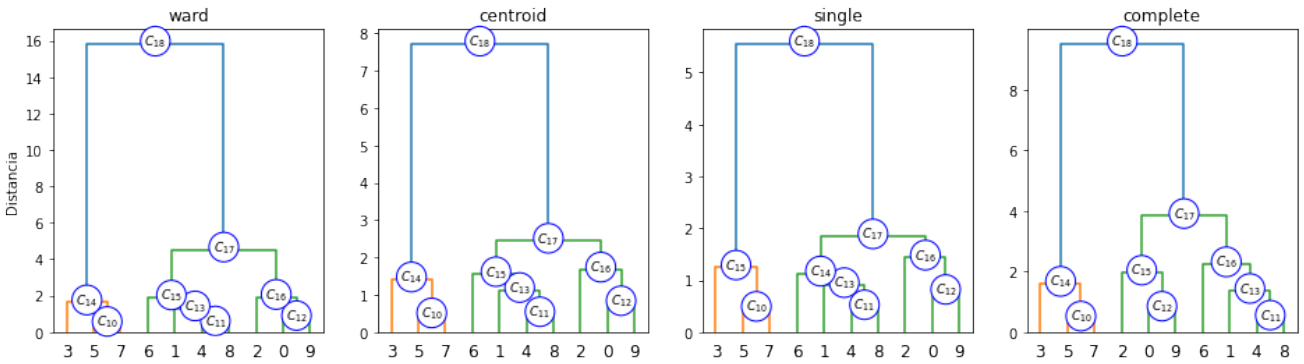
La biblioteca scipy proporciona una implementación eficiente de clustering jerárquico que permite utilizar diferentes criterios de `linkage` y generar un dendograma de los datos. El modulo se encuentra en `scipy.cluster.hierarchy`. A continuación presentamos un ejemplo del como utiliza la librería.

In [5]:
```
# Importamos la bibliotea de clustering jerarquíco
from scipy.cluster.hierarchy import dendrogram, linkage
# Generamos un ejemplo datos con tres clusters
X,y=make_blobs(n_samples=10, random_state=5)
plotClusters(X,[1 for x in range(len(X))])
for i,z in enumerate(X):
    x,y=z
    plt.annotate(f"${i}$",(x+0.2,y),ha='center', size=12)
# graficamos los clusters
```



In [6]:
```
def anota_dendograma(D, ax):
      bbox_props = dict(fc="white", ec="b",boxstyle='circle')
      ll=list(set(np.array(D['dcoord']).flatten()))
      ll.sort()
      for j, d in zip(D['icoord'], D['dcoord']):
        x = 0.5 * sum(j[1:3])
        y = d[1]
        #plt.plot(x, y, 'sr', markersize=10)
        ax.annotate("$C_{%d}$" %(ll.index(y)+len(X)-1) , (x, y), xytext=(0, 8
                    ,ha='center', bbox=bbox_props)
```

In [7]:
```
# Obtenemos distintons clustering e gráficamos el dendrogram
fig,axes=plt.subplots(1,4, figsize=(16,4))
axes[0].set_ylabel('Distancia')
for i,lk in enumerate(('ward','centroid','single','complete')):
      Z=linkage(X,lk)
      Dz=dendrogram(Z,ax=axes[i])
      axes[i].set_title(lk)
      anota_dendograma(Dz,axes[i])
```



En el dendrogrma la altura de cada nodo representa el costo (distancia) de unir los clusters, correspondientes. Para el ejemplo, en el caso de complete unir 4 y 8 tiene un coste similar a unir 5 y 7, pero unir C_{16} y C_{13} es más costoso.

La matriz Z devuelta por linkage de dimensión $n-1 \times 4$. En la i-ésima iteración, los grupos con índices $Z_{i,0}$ y $Z_{i,1}$ son conbinaddo para formar un nuevo grupo. $Z_{i,2}$ es la distancia entre los grupos $Z_{i,0}$ y $Z_{i,1}$. El cuarto valor $Z_{i,3}$ representa el número de observaciones originales en el grupo recién formado.

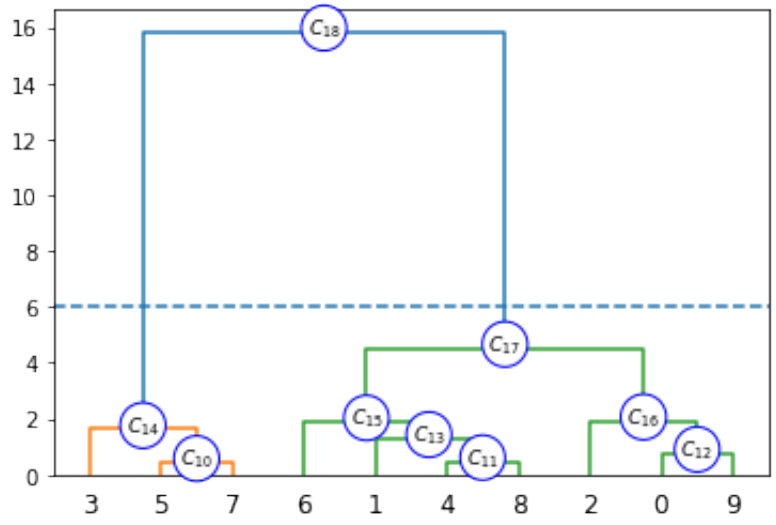
In [8]:
```
print("El la ultima iteración se unen los clusters:", (Z[8,0], Z[8,1]))
print("El costo es de:", Z[8,2])
print("El número de elementos en C18 es:", Z[8,3])
```

```
El la ultima iteración se unen los clusters: (14.0, 17.0)
El costo es de: 9.519054141442918
El número de elementos en C18 es: 10.0
```

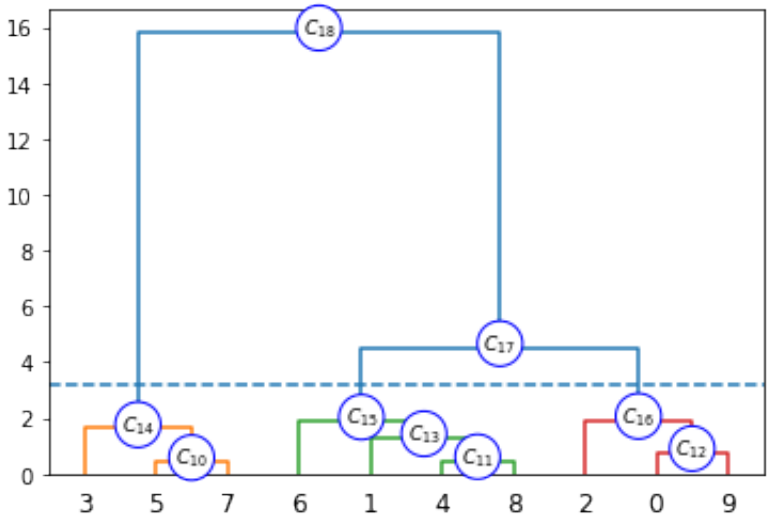
Cortar el dendograma

El dendrograma muestra la similitud entre la jerarquía de clusters, pero también puede utilizarse para determinar el número de clusters, así como los elementos en cada uno. Esto de puede conseguir mediante un "corte" horizontal a una determinada altura del dendrograma, el número de ramas que sobrepasan por encima del será el número de clusters.

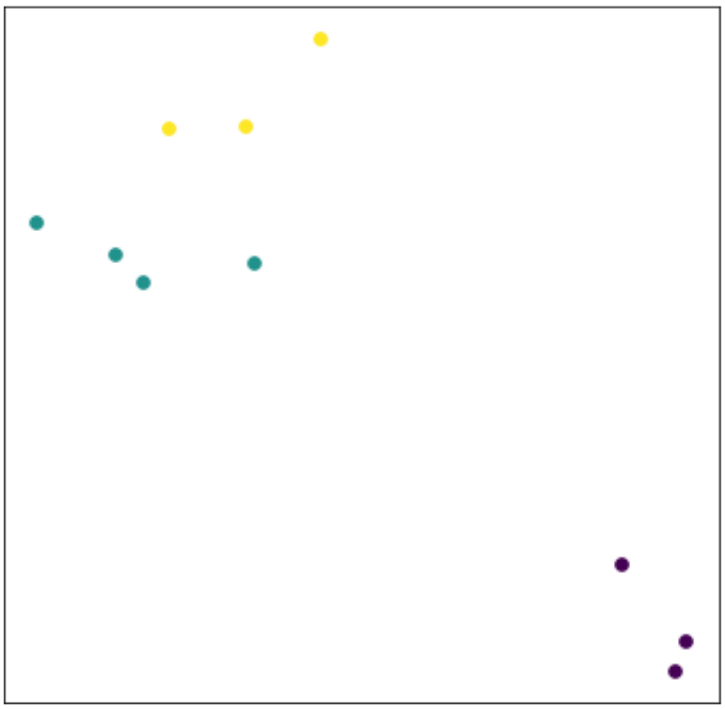
In [9]:
```
# cortando a una altura de 6 tendráimo dos clusters
Z=linkage(X,'ward')
Dz=dendrogram(Z)
plt.axhline(6,0,10,linestyle='dashed')
anota_dendograma(Dz,plt)
```



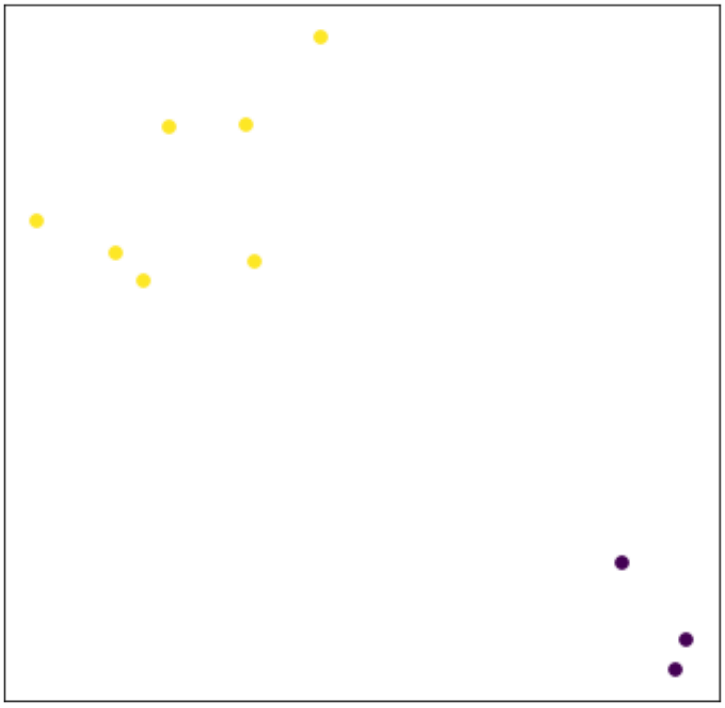
In [10]:
```
# cortando a una altura de 3.2 tendríamos 3 clusters
Z=linkage(X,'ward')
Dz=dendrogram(Z,color_threshold=3.2) # con la variable color_threshold especi
plt.axhline(3.2,0,10,linestyle='dashed')
anota_dendograma(Dz,plt)
```



In [11]:
```
# Podemos obtener las etiquetas de los grupos mediante la funcion fcluster
K=3 # este sería el número de clusters
yp=fcluster(Z, K, criterion='maxclust') # obtener las etiqutas para cada clus
plotClusters(X,yp)
```



In [12]:
```
# Para dos clusters
yp=fcluster(Z, 2, criterion='maxclust')
plotClusters(X,yp)
```



Algoritmos particionales

En este tipo de algoritmos sin duda el más conicido es K-Means

K-Means

Es un algoritmo basado en particionado. El algoritmo requiere como entrada la colección X y el número de clusters K. El proceso comienza seleccionando K elementos de X, los cuales son utilizados como los centroides iniciales. Posteriormente cada punto es asignado a su centroide más cercano, todos los elementos asignados a un mismo centroide forman un cluster. Una vez asignados todos lo elementos se cálcula el centroide c_k para cada uno de los grupos. Se repiten el proceso de asignación y actualización de los centroides hasta que no hay cambios en los centroides. Podemos resumir el algoritmo en el siguiente listado:

Algoritmo básico para K-Means

---

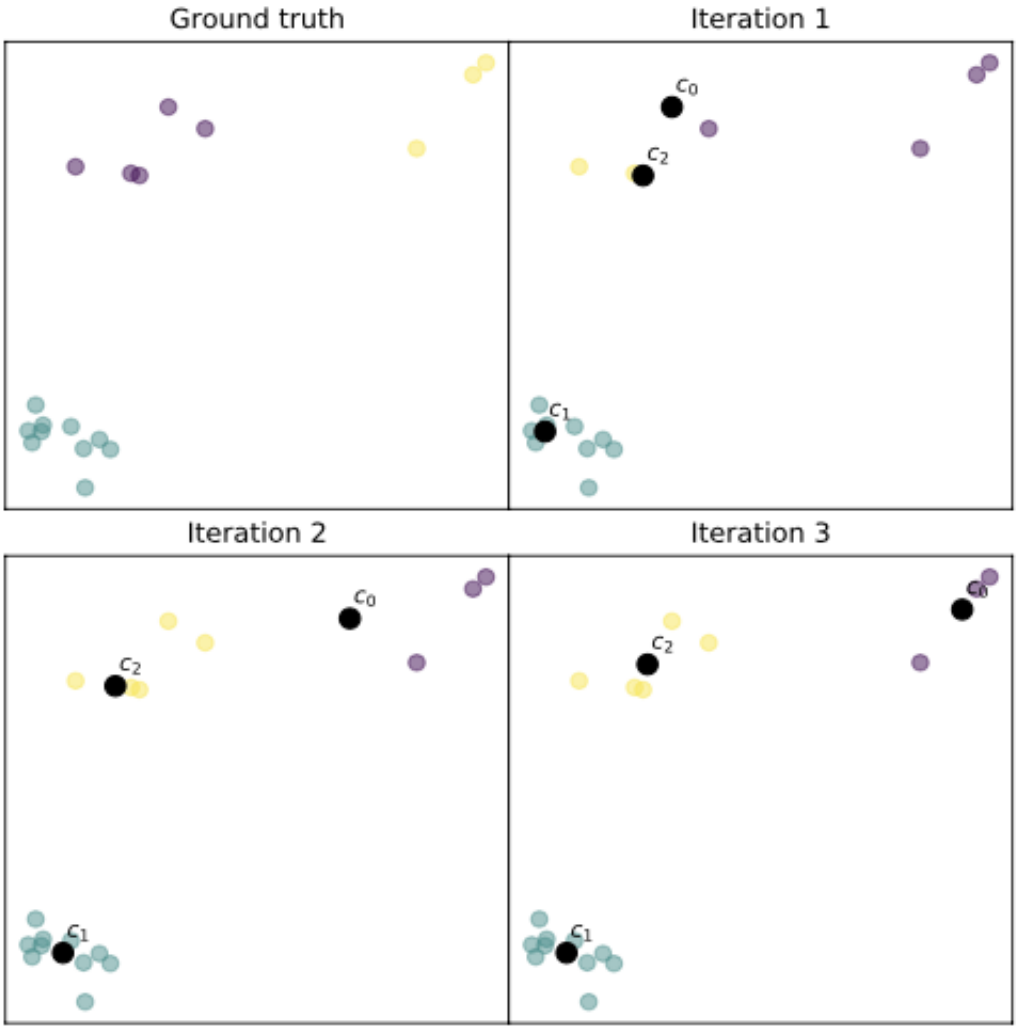
1. Seleccionar K objetos de X como centroides iniciales
2. **repetir**
3. Para cada $x_i \in X$ asignar x_i al grupo representado por su centroide más cercano
4. Recálcular los centroides en base a los grupos obtenidos
5. **hasta** que los centroides no cambien

Para el paso 3 se utiliza alguna función comparación, ya sea de distancia o similitud. En esta punto algunas de las consideraciones que hay que tomar para asegurar convergencia del

método K-Means se resumen en la siguiente tabla:

Función de comparación	valor para el centroide	Función objetivo
Distancia Euclidiana	media	$\min \sum_{k=1}^{K} \sum_{\forall x_i \in G_k} (x_i - c_k)^2$
Distancia Manhattan	mediana	$\min \sum_{k=1}^{K} \sum_{\forall x_i \in G_k} \lvert x_i - c_k \rvert$
Similitud Coseno	media	$\max \sum_{k=1}^{K} \sum_{\forall x_i \in G_k} \cos(x_i - c_k)$

La siguiente figura muestra el proceso de convergencia del K-Means para un conjunto de 90 elementos y $K = 3$ clusters:

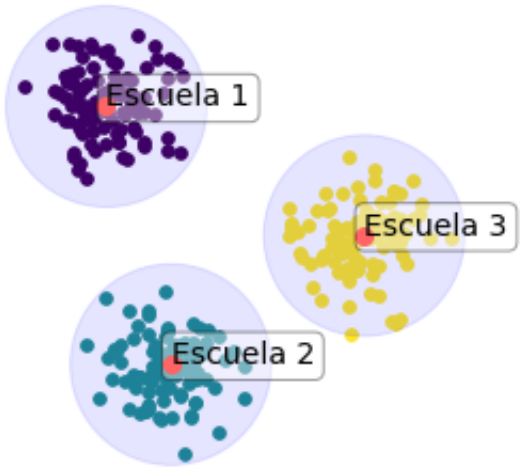


Iteraciones K-Means

K-Centros y Farthest First Traversal

El problema de los K-Centros consiste en dado un número limitado de facilidades (K) encontrar la forma de ubicarlas de tal forma los usuarios más alejados tengan un coste mínimo para acceder a dichas facilidades. Como ejemplo suponga que hay n personas en edad escolar, de las cuales conoce la ubicación de su vivienda. Se pretende construir K escuelas, pero eligiendo las posiciones de tal forma que la distancia r que los niños más alejados tengan que recorrer para acudir a clase sea mínima. En este las distintas ubicaciones pueden verse como un conjuto de puntos en un espacio bidimensional. Para solucionar el problema deben elegirse las posiciones en las que debe construirse cada escuela minimizando la distancia máxima r.

Si consideramos que los puntos que generamos previamente en los blobs corresponden a las posiciones de los habitantes y se quieren construir 3 escuelas; entonces una solución simple es ubicarlas en la posición correspondiente al centroide de cada nube de puntos:

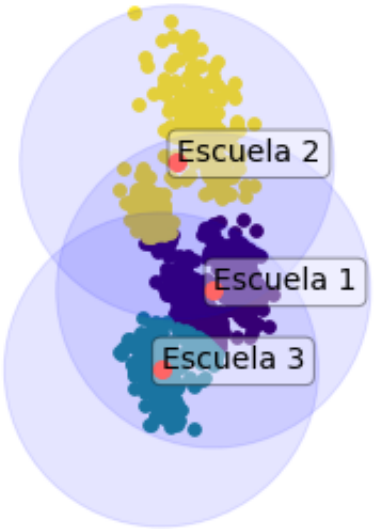


Solución ideal para los K-centros cuando los datos siguen una distribución gaussiana

desafortunadamente no todos los datos provienen de distribuciones normales con varianza unitaria. Por ejemplo no es tan obvio utilizar la misma idea para los siguientes datos:

Dato con varianza diferente de 1

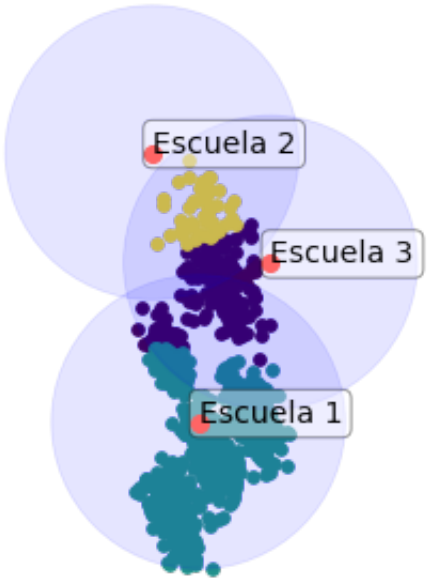
Aún así podríamos utilizar K-means para encontrar tres clusters y construir la esculeas en los centroides.



Utilizando KMeans para 100 ejecuciones se obtiene en promedio una distancia máxima de $r = 37$ y un $SSE = 128141.13$

Como puede observar en la figura K-Means tiende a favorecer la zona más densa (ahí quedaron 2 escuelas). Lo cual podría ser bueno, ya quiere decir que la mayoría de los estudiantes tienen que desplazarse poco, pero es muy probable que para este caso la distancia máxima no sea mínima.

Para el caso en que se desea minimizar la distancia máxima resulta más convemiente resolver el problema mediante el uso de Farthest First Traversal FFT.



Utilizando FFT para 100 ejecuciones se obtiene en promedio una distancia máxima de $r = 33.38$ y un $SSE = 24198.58$

Farthest First Traversal (FFT)

FFT es un algoritmo que permite seleccionar K centros de una colección de objetos. Un

centro a diferencia de un centroide es un elemento que forma parte de la colección de datos. En FFT cada centro es seleccionado de tal forma que cualquier objeto $x_i \in X$ está a lo más una distancia r de uno de los centros. Al igual que KMeans, FFT es un algoritmo de pariticionado, solo que en este caso cada $G_k \in G$ será defino en base a la partición generada por los centros, utilizaremos R_C para referiremos al conjunto de centros generados por el algoritmo.

Este algoritmo fue inicialmente propueste por Teofilo Gónzalez como un aproximación a la solución del problema de K-Centros, la cual es a lo más dos veces la solución óptima. Utilizando FFT podemos calcular un conjunto R_C, conde cada $c_j \in R_C$ son lejanos entre si. Esto se consigue simulando un recorrido, donde en cada iteración k, c_k es seleccionado como el elemento más alejado de los elementos ya seleccionados. Sea r_k la distancia utilizada para seleccionar el centro c_k, entonces en la iteración k se cumple que:

- Cualquier par de objetos $x_i, x_j \in X$ se encuentran al menos a distancia r_k el uno del otro.
- Cualquier elemento $x_i \in X$ se ecuentra a lo más a distancia r_k de uno de los centros $c_j \in R_C$ (cada elemento se asigna a la partición de su centro más cercano)

Para facilitar la definición del algoritmo definimos $d_{\min}(x)$ como la distacia entre un elemento x y su centro más cercano en R_C, entonces:

$$d_{\min}(x) = \min\{d(x, c) \mid c \in R_C\}$$

Algoritmo Farthest First Traversal

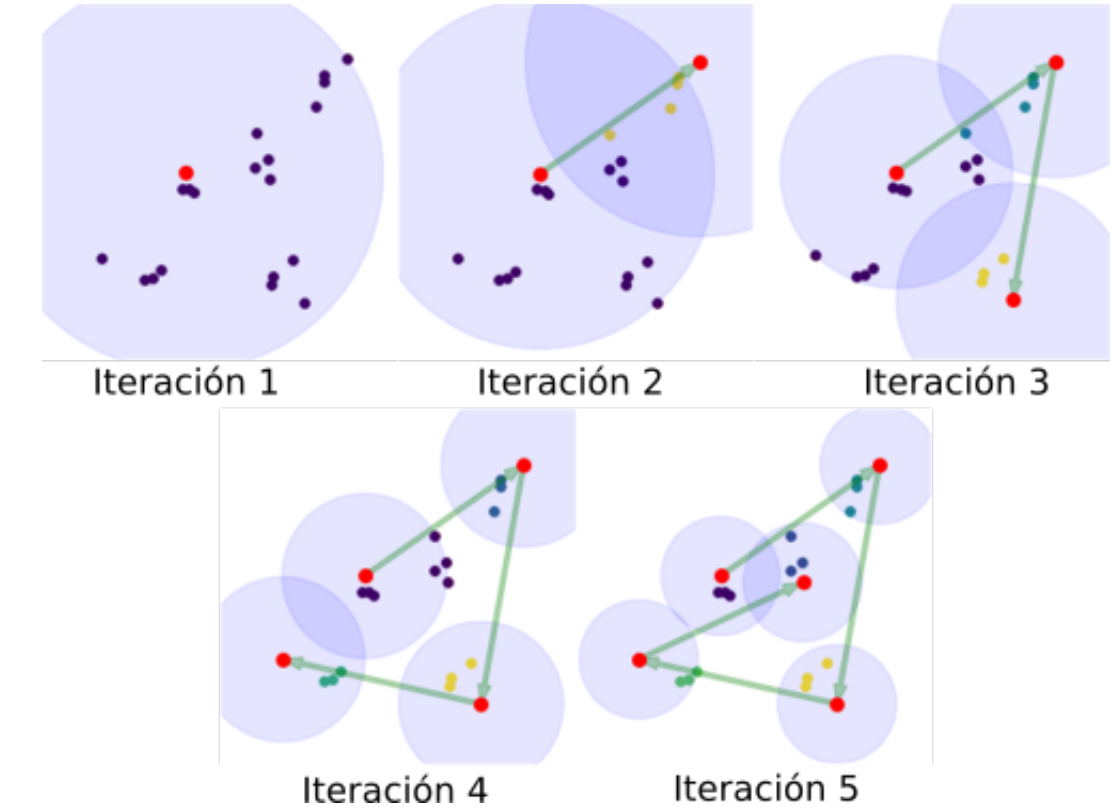
---

1. $c_1 \leftarrow x_i$, donde x_i es seleccionado aleatoriamente de X
2. $R_C \leftarrow \{c_1\}$
3. **Mientras** $|R_C| < K$
4. $\quad c_n \leftarrow \arg\max\{d_{\min}(x) \mid x \in X \setminus R_C\}$
5. $\quad r \leftarrow \max\{d_{\min}(x) \mid x \in X \setminus R_C\}$
6. $\quad R_C \leftarrow R_C \cup \{c_n\}$
7. $G \leftarrow \{G_1, G_2, \dots G_k\}$

---

donde cada G_k es el conjuto de los elemento x_i que tiene como su centro más cercano al centro c_k.

La siguiente figura muestra la evolución de algoritmo FFT, puede observarse que la distancia máxima disminuye a cada iteración; asi como el orden en que se eligen los puntos.



FFT para $K = 5$ sobre 5 nubes de puntos generadas asumiendo una distribución normal

FFT tiene la ventaja adicional que su complejidad está acotada por $O(DKn)$ donde D es la dimensión de los datos, n el número de elementos en la base de datos y K el número de centros. La solución mencionada es la mejor solución conocida para el problema. Por otro lado la solución obtenida es a lo más 2 veces la solución ópitma. Si utilizamos KMeans la solución es $O(KDnI)$ donde I es el número de iteraciones. Finalmente hay que notar que los dos algoritmos optimizan variables difierentes, mientras que FFT minimiza la distancia máxima de los datos a uno de los centros; K means optimiza el error cuadrático SSE.

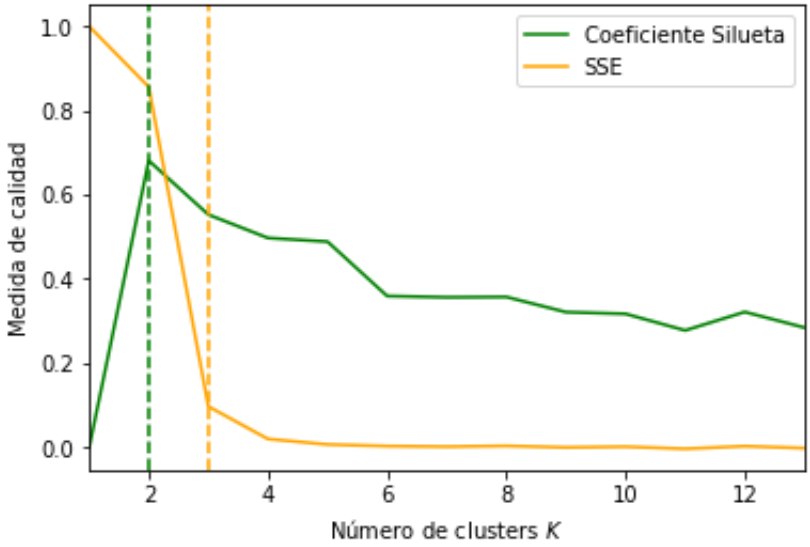
Determinando el número de clusters

Este problema ha sido objeto de muchos estudios, pero solo mencionaremos brevemente dos métodos de los muchos que existen en la literatura.

Método de Elbow: Consiste en calcular el valor del SSE para valores continuos de K y elegir el valor en que se da el cambio máximo en el valor de SSE .

Método de la silueta: En este caso de cálcula el valor del coeficiente de silueta y se elige el valor de K que presente el valor máximo del score.

En la siguente gráfica se muestra el valor de ambos criterios para el data set de iris utilizando. Como puede verse para este caso utilizando el criterio de Elbow el valor óptimo sería $K = 3$; mientras que si utilizamos Silueta esto sería $K = 2$.



Ejemplos de criterios de selección del valor de K para la DB Iris

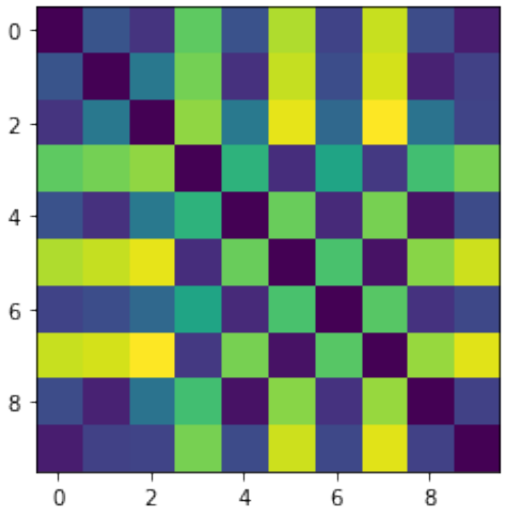
Evaluación visual de la tendencia (de clúster)

El enfoque para la identificación visuaal de clusters (Visual Assessment Tendency, VAT) utiliza la distancia entre todos los pares de elememento. Con el uso del las distancias es pposible obtener como una imagen de $n \times n$ píxeles, básicamente un mapa de calor. Utilizando los datos de las nubes de puntos dadas por X y las funciones `pdist` y `squareform` podemos obener la martiz de distancias D como sigue:

In [35]:
```
from scipy.spatial.distance import pdist, squareform
d=pdist(X, 'euclidean')
m,n=np.max(d),len(X)
D=squareform(d)/m #Normalizamos entre el valor máximo
```

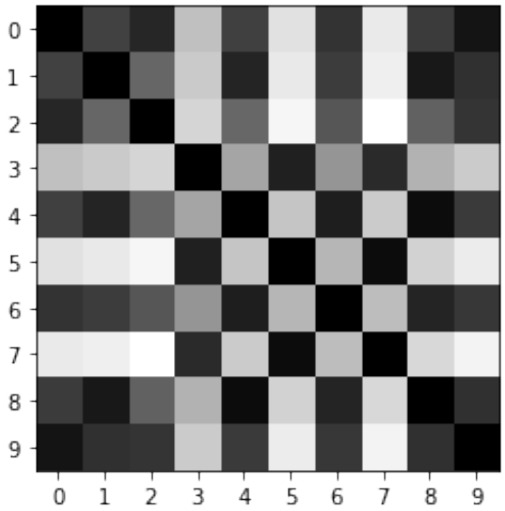
con D podemos crea un mapa de calor como sigue:

In [36]:
```
fig, ax = plt.subplots()
im = ax.imshow(D)
```



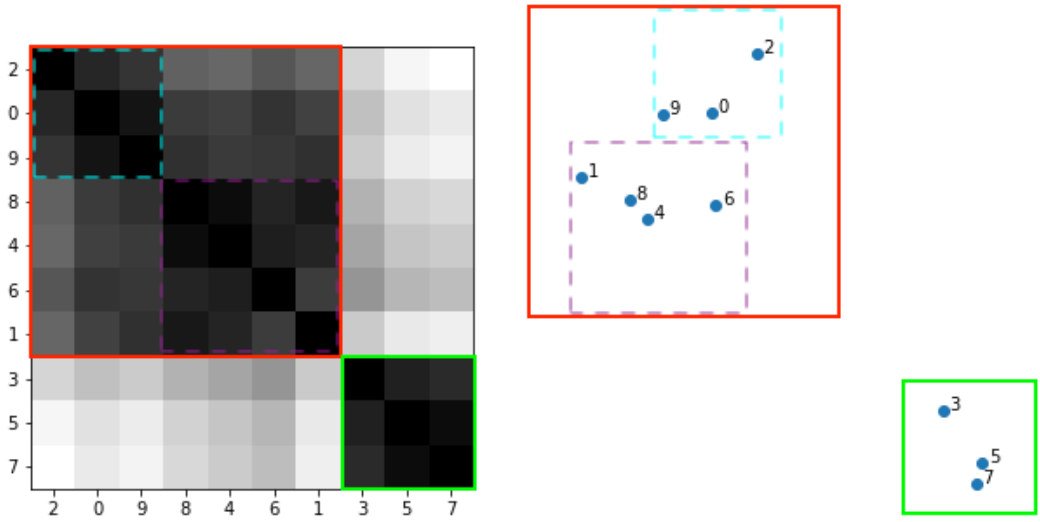
En en la imagen podemos observar que elementos más cercanos presentan colores más oscuros (por ejemplo en la diagonal). Sin embargo, de esa imagen no es posible identificar un patrón (clusters). VAT utiliza la mattiz de distancias D y construyes una imagen de **intensidad** I, la cual es conocida como imagen de disimilitud. Donde valor de cada del píxel (i, j) depende del valor de D_{ij}. Note que $D_{ij} = 0$ corresponde al valor más obscuro (negro si se realiza en scala de grises) y el valor más claro corresponde a 1 en la matriz normalizada (el valor máximo en la matriz original). Los valores intermedios de D_{ij} producen píxeles con niveles intermedios de gris, continuación se muestra la imagen previa en escala de grises.

In [61]:
```
fig, ax = plt.subplots()
ax.set_xticks(list(range(n)))
ax.set_xticklabels([f"{i}" for i in range(n)])
ax.set_yticks(list(range(n)))
ax.set_yticklabels([f"{i}" for i in range(n)])
im=ax.imshow(D, cmap='gray')
```



In [ ]:
```
ax.set_xticklabels
```

La utitilidad de una imagen de intensidad para evaluar visualmente la tendencia en los clusters depende del orden de las filas y columnas en D. Al reordenar los elementos $x_i, x_2, \ldots, x_n$ como $x_{ki}, x_{k2}, \ldots, x_{kn}$ de modo que, si x_{ki} es cerca de x_{kj}, implique que x_{ki} es similar a x_{kj}. En este caso, el imagen de distancias ordenadas correspondiente (D_o) mostrará la tendencia de los grupos en los datos mediante bloques oscuros de píxeles a lo largo de la diagonal principal.



Matriz de distancia ordenada - Clusters

La imagen anterior se obtiene al procesar los elementos en la matriz de distancias D (en lugar de usar los objetos o los datos del objeto directamente). Como puede observarse tenemos dos más oscuras la submatriz $(2, 1)$ y la submatriz $(3, 7)$. Otra alternativa podría ser dividir la región $(2, 1)$ en las submatrices $(2, 9)$, $(8, 1)$ (tres clusters). Si contrastamos la matriz de distancias con los datos originales etiquetados (lado derecho de la figura) es posible ver claramente los dos clusters y relacionar las etiquetas en los elementos con el orden de las filas/columnas del mapa de intensidad.

El algoritmo para ordenar los renglones de la matriz es similar al algoritmo para encontrar el árbol de expansión minima de **Prim** Las principales diferencia entre VAT y Prim son:

1. No se genera el árbol de expansión mínima, solo de identifica el orden que los elementos de agregan al árbol.
2. El vertice incial, se escoge como el indice del renlón que continene la distancia máxima.

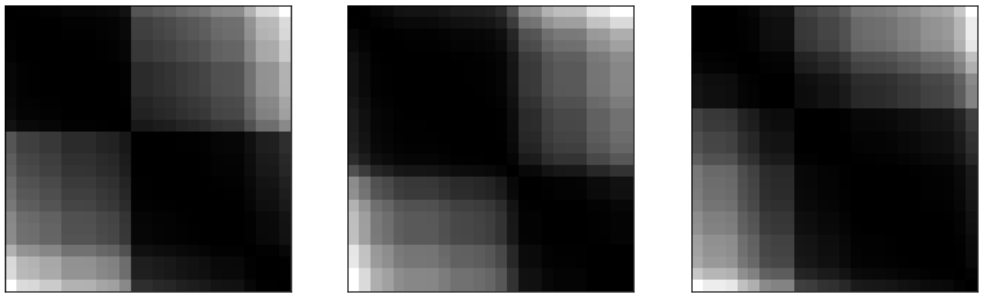
A continuación se lista el seudocódigo para VAT (En [5] puede encontrar una versión del seudocódigo estilo MATLAB).

Algoritmo VAT para ordernar la matriz de distancias

Entrada: Matriz de distancias D

1. $i_{max}, j_{max} \leftarrow argmax(D)$
2. Agregar i_{max} a la lista de visitados V
3. $S \leftarrow \{1, 2, \dots n\}$
4. **Mientras** $|V| < n$:
5. Agrerar al final de V el indice j tal $i \in V \wedge j \notin V \wedge min(D_{ij})$
6. **Para cada** $i \in S$:
7. $p \leftarrow V[i]$
8. **Para cada** $j \in S$:
9. $q \leftarrow V[j]$
10. $D_{o_{ij}} = D_{pq}$
11. Regresar D_o

Debido a que VAT es una estrategía visual, puede ser complicado analizar el mapa de intesidad para conjuntos de datos grandes. Una estrategia común es aplicar VAT sobre una muestra de los datos. La muestra puede ser selccionada de forma aleatoria o bien mediante una estrategía muestreo más sofisticada (En [6] utilizan FFT para este propósito, aunque ahí es denominado como *Maximin sampling*). La siguente figura muestra tres corridas de VAT sobre una muestra aleatoria de 25 elementos del data set de Iris.



VAT sobre una muestra aleatoria de 25 elementos (Iris dataset)

In [ ]:

In [ ]:

Actividad

Evaluación de algoritmos de clustering

Para la actividad correspondiente a clustering deberá proporcionar implementaciones propias de los algoritmos de clustering FFT y VAT. Si le resulta conveniente puede utilizar la plantilla porporcionada en el archivo **skeleton.py**. Una vez que cuente con sus implementaciones realizar las siguientes actividades:

1. Elija dos datasets pueden ser de https://scikit-learn.org/stable/datasets/toy_dataset.html o de de cualquiera de los sitios previamente utilizados (kaggle, UCI, etc).
2. Con los métodos implementados y al menos una estrategía jerárquica agrupe los datos en cada uno de los problemas elegidos, utilizar al menos dos funciones de distancia diferentes.
3. Mediante el uso de los métodos de Elbow, Silueta y VAT determine el número "óptimo" de clusters para los problemas elegidos
4. Realice una comparativa de la calidad de los clusters obtenidos por cada una de las estrategías. Utilice medidas internas, y también externas cuando se cuenta con los datos ya etiquetados.
5. Utilizando las ténicas de visualización que ya conoce, aplique un análisis exploratorio de los grupos obtenidos para cada problema.
6. Realice un reporte de las estrategias que haya aplicado. El documento deberá incluir las siguientes secciones:
    A. Introducción
    B. Sección(es) que describa el análisis realizados y los resultados obtenidos para cada dataset.
    C. Conclusiones
    D. Referencias

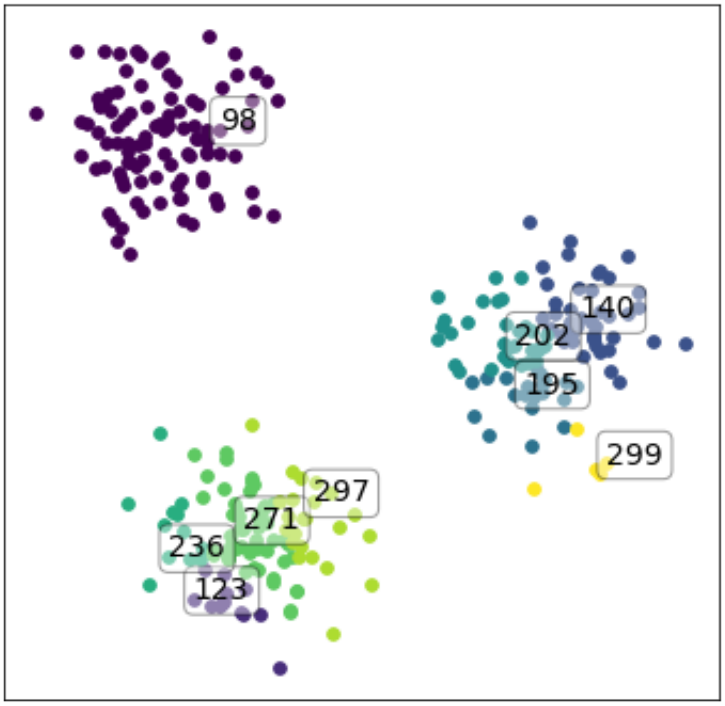
Nota: Subir un documento en formato pdf. Si utiliza estrategias adicionales por favor incluir las referencias. Si tienes alguna pregunta respecto al material o la actividad puedes ingresar al foro de dudas.

Referencias

1. Manning, C., Raghavan, V. and Schütze H. (2009). Introduction to Information Retrieval (Chapter 16). Cambridge: Cambridge University Press, Online Edition (https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf).
2. Rokach, L., & Maimon, O. (2005). Clustering methods. In Data mining and knowledge discovery handbook (pp. 321-352). Springer, Boston, MA.
3. Tan, P. N. (2018). Introduction to data mining (Chapter 8). Pearson Education India.
4. Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38, 293-306.
5. Bezdek, J. C., & Hathaway, R. J. (2002, May). VAT: A tool for visual assessment of (cluster) tendency. In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290) (Vol. 3, pp. 2225-2230). IEEE.
6. Ibrahim, O. A., Keller, J., Bezdek, J. C., & Popescu, M. (2020, July). Experiments with Maximin Sampling. In 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (pp. 1-7). IEEE.

Ejemplo de uso de randomClustering

In [13]:
```
data,target=make_blobs(n_samples=50, random_state=3, centers=3)
cl=Clustering(n_clusters=9)
cl.fit(blobs_data)
yp=cl.predict(blobs_data)
plotClusters(blobs_data,yp,centroids=cl.centroids_)# graficar los clusters co
```



In [14]:
```
from sklearn.metrics import  silhouette_score as sil
```

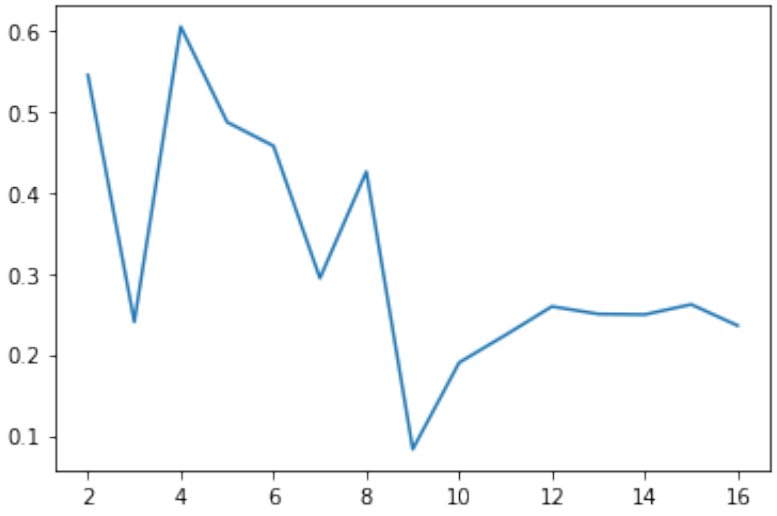
In [15]:
```
S=[]
K=list(range(2, int(np.sqrt(len(blobs_data)))))
for k in K:
    cl=Clustering(n_clusters=k)
    cl.fit(blobs_data)
    S.append(sil(blobs_data,cl.labels_))
```

In [16]:
```
K
S
```

Out[16]:
```
[0.5445107809949886,
 0.24026939906883182,
 0.6041650408782978,
 0.486519289207828,
 0.4571665232148976,
 0.2941687592519827,
 0.4255202532816396,
 0.08334488135273267,
 0.19008640497471965,
 0.2240932116641066,
 0.25926148246637354,
 0.24996395152461284,
 0.2492987956174003,
 0.26176136631256186,
 0.23571100059557673]
```

In [17]:
```
plt.plot(K,S)
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x7fcd855d1040>]
```



In [18]:
```
np.argmin(np.diff(S))
```

Out[18]:
```
6
```

In [ ]: