# 1. Análisis de agrupamiento (*clustering*)

En ocasiones es posible dividir una colección de observaciones en distintos subgrupos, basados solamente en los atributos de las observaciones; cuando se puede hacer esta separación, se facilita el entendimiento de la población o el proceso que generó las observaciones. La intención es realizar división de datos en *grupos* (*clusters*) de observaciones que son más similares dentro de un grupo que entre varios grupos. Los grupos son formados ya sea agregando observaciones o dividiendo un gran grupo de observaciones en una colección de grupos más pequeños.

El proceso de generación de grupos incluye dos variedades de algoritmos:

1. Combinar obervaciones entre un número fijo de grupos buscando maximizar la similitud dentro de cada grupo

2. Comenzar con grupos de un solo elemento y, recursivamente combinarlos; alternativamente, se puede comenzar con un sólo grupo y recursivamente dividir nuevos grupos

En esta sección se revisarán dos algoritmos populares y representativos de cada una de estas propuestas: agrupación jerárquica y $k$-medias. Sea $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ el conjunto de datos, con $\mathbf{x}_i$ un vector de atributos medidos en una unidad observacional.

## 1.1. Agrupamiento por $k-$medias

A diferencia del agrupamiento jerárquico por aglomeración, en este algoritmo, el analista establece el número de grupos; el algoritmo comienza asignando arbitrariamente los vectores del conjunto de datos $D = \{x_1, \ldots, x_n\}$ a $k$ grupos; estos grupos iniciales se denotan como $\{A_1, \ldots, A_k\}$. Después, se calculan los *centroides* de cada grupo; estos centroides son medias multivariadas de las observaciones de cada grupo y cada centroide representa al grupo para el cálculo de las distancias. También es posible asignar arbitrariamente $k$ medias iniciales y con ellas generar los grupos.

Es muy poco probable que la configuración inicial sea una buena solución, por tanto el algoritmo itera entre dos pasos:

♢ asignar cada observación al grupo más cercano

♢ recalcular los centroides de cada grupo

Si al menos una observación es reasignada en otro grupo, los centroides cambiarán y debe realizarse otra iteración; el algoritmo continuará pasando entre estos dos pasos hasta que ya no hay reasignaciones. En ese momento, cada observación pertenece al grupo que le es más cercano. A partir de la configuración aleatoria inicial, la suma de cuadrados dentro de cada grupo ha sido minimizado; esto se debe a que dicha suma de cuadrados es equivalente a la suma de distancias euclideanas entre las observaciones y los centroides; además, mover cualquiera de las obervaciones incrementará la suma de distancias (y la suma de cuadrados dentro de los grupos). Por tanto, el algoritmo ha alcanzado *una mejor* asignación posible para las observaciones en los grupos y *un mejor* cálculo de los centroides.

Este algoritmo tiene un atractivo considerable porque minimza una función objetivo popular: la suma de cuadrados. Su inconveniente es que debe generar una configuración inicial aleatoria: una configuración diferente por lo general regresa un resultado distinto.

El centroide del grupo $A_i$ es la media multivariada:

$$\overline{\mathbf{x}}_i = n_i^{-1} \sum_{\mathbf{x}_j \in A_i} \mathbf{x}_j,$$

donde, $n_i$ es el número de observaciones que pertenecen a $A_i$ y $\mathbf{x}_j = [x_{j,1}, \ldots, x_{j,h}]^T$; el número de atributos es $h$ y el $l-$ésimo elemento de $\overline{\mathbf{x}}_i$ es:

$$\overline{x}_{i,l} = n_i^{-1} \sum_{\mathbf{x}_j \in A_i} x_{j,l},$$

para $l = 1, \ldots, h$; la distancia entre cada observación $\mathbf{x}_j$ y un grupo $A_i$, se define como la distancia entre la observación y el centroide del grupo. Como $\overline{\mathbf{x}}_i = [\overline{x}_{i,1}, \ldots, \overline{x}_{i,h}]^T$, la distancia euclideana cuadrada es:

$$d_E^2 (\mathbf{x}_j, \overline{\mathbf{x}}_i) = \sum_{l=1}^{h} (x_{j,l} - \overline{x}_{i,l})^2.$$

Se puede utilizar la distancia cuadrada en lugar de la distancia, dado que el ordenamiento de más cercano a más distante será el mismo.

**Ejemplo**

Clasificar las muestras siguientes utilizando $k = 2$:
$[8, 10], [3, 10.5], [7, 13.5], [5, 18], [5, 13], [6, 9], [9, 11], [3, 18], [8.5, 12], [8, 16]$

**Ejemplos con Python**

## 1.2. Agrupamiento jerárquico por aglomeración

Este enfoque comienza con cada observación definiendo un grupo de un sólo elemento: $\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}$, donde $n$ es el número de observaciones. El algoritmo reduce iterativamente el conjunto de grupos combinando grupos similares. La combinación de los conjuntos $A$ y $B$ en alguna iteración se representa como:

$$(A, B) \longrightarrow A \cup B$$

La elección de conjuntos a combinar se puede determinar calculando la distancia entre los grupos y combinando la pareja con la menos distancia intergrupal (más adelante se presentan otras opciones). Por tanto, es requiere una métrica para obtener la distancia entre grupos; por ejemplo, la distancia entre los grupos $A$ y $B$ puede definirse como la distancia más corta entre cualquier vector de $A$ y cualquier vector de $B$. Matemáticamente:

$$d(A, B) = min \{d_v (\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\},$$

donde $d_v$ es una distancia entre vectores. La distancia euclideana es muy popular para este cálculo. También se puede utilizar la distancia Manhattan (*city-block*), la distancia *city-block* entre $\mathbf{x}_0$ y cada $\mathbf{x}_i \in X$ se define como:

$$d_C (\mathbf{x}_i, \mathbf{x}_0) = \sum_{j=1}^{p} |x_{i,j} - x_{0,j}|,$$

donde $p$ es el número de elementos de los vectores. Se pueden obtener grupos más compactos utilizando la métrica basada en centroides, éste se obtiene para cada grupo como:

$$\overline{\mathbf{x}}_A = n_A^{-1} \sum_{\mathbf{x}_k \in A} x_k,$$

donde $n_A$ es el número de observaciones en el grupo $A$; entonces la distancia entre $A$ y $B$ es:

$$d_{cent}(A, B) = d(\overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$$

Como se describió anteriormente, el algoritmo combinará grupos hasta obtener uno sólo; pero es necesario revisar conjuntos de grupos intermedios para identificar el que resulte más útil según sea el caso. Para poder identificar el agrupamiento óptimo, se puede utilizar un *dendrograma*, el cual muestra el historial de los agrupamientos, se realiza lo siguiente:

1. Determinar la mayor distancia vertical para la que no hay intersección con otros grupos
2. Trazar una línea horizontal entre ambos extremos
3. El agrupamiento óptimo es igual al número de líneas verticales cruzadas por el trazo anterior

Para el ejemplo de la figura 1 la mejor elección sería 4.

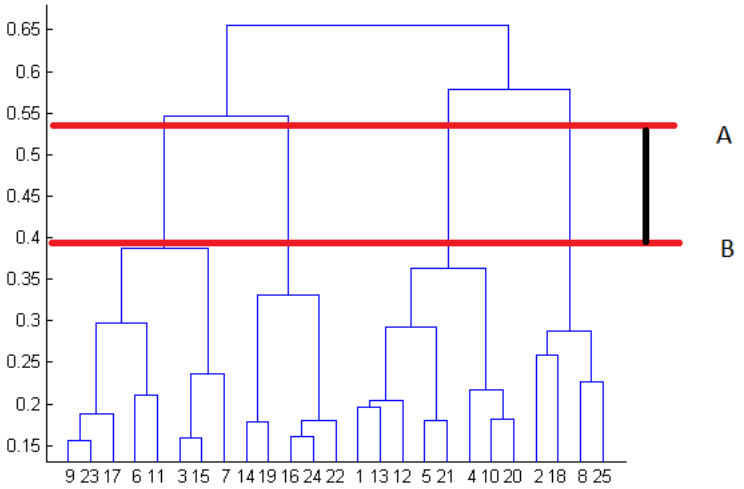



Figura 1: Ejemplo de uso de dendrogramas

**Criterio de enlazamiento de grupos**

Existen otros tipos de enlazamiento entre los grupos, cada uno entrega soluciones distintas.

♢ ***Single linkage***: utiliza la distancia más corta entre dos vectores de cada grupo
$L(A, B) = min\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\}$

♢ ***Complete linkage***: toma en cuenta la distancia más grande entre dos vectores de cada grupo
$L(A, B) = max\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\}$

♢ ***Average linkage***: usa la distancia promedio entre todos los vectores de cada grupo

♢ $L(A, B) = \dfrac{1}{n_A n_B} \sum_{k=1}^{n_A} \sum_{l=1}^{n_B} d_v(\mathbf{x}_k, \mathbf{x}_l), \mathbf{x}_k \in A, \mathbf{x}_l \in B$

♢ ***Ward linkage***: la distancia es la suma de las diferencias al cuadrado en los grupos

♢ $d_v(A,B) = (\overline{\mathbf{x}}_A - \mathbf{x}_k)^2 + (\overline{\mathbf{x}}_B - \mathbf{x}_l)^2, \mathbf{x}_k \in A, \mathbf{x}_l \in B$

**Ejemplo**

Para el conjunto de datos unidimensionales $\{7, 10, 20, 28, 35\}$, obtener el agrupamiento jerárquico y dibujar el dendrograma correspondiente.

**Ejemplos con Python**

### 1.2.1. Modelos de mezclas gausianas (*Gaussian Mixed Models*)

Similar a k-medias, en estos modelos se establece una cantidad $k$ de grupos para iniciar y se inicializan tanto la medias como las varianzas.

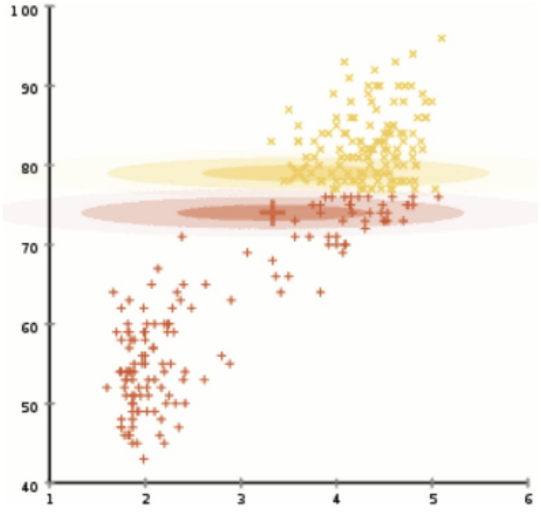



Figura 2: Paso inicial de un modelo de mezclas gaussianas

Iterativamente actualiza la media y la varianza de cada grupo, así como la probabilidad de pertenencia de cada punto a cada grupo y el peso del mismo (cantidad de muestras de cada grupo).

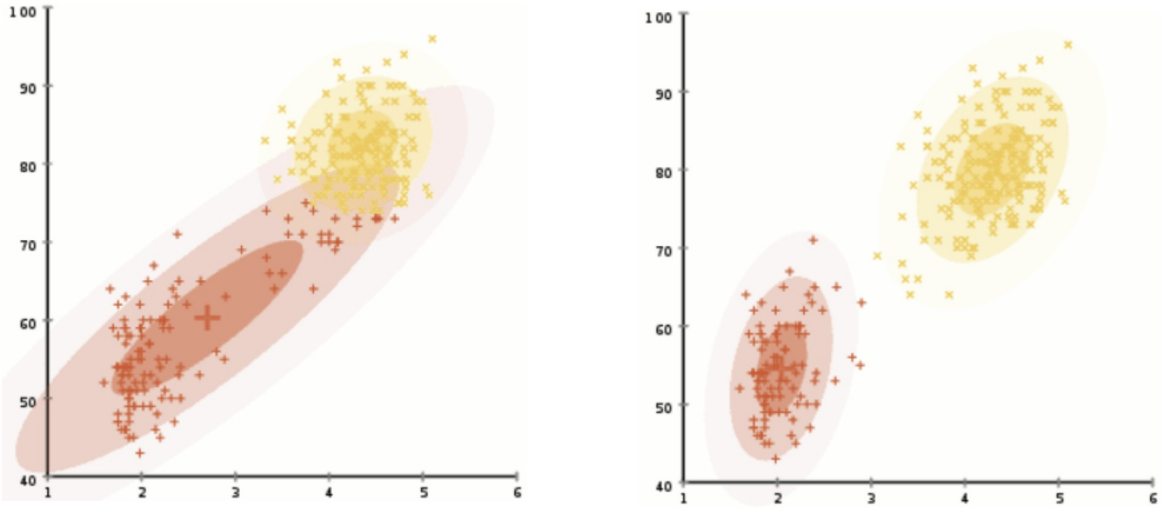



Figura 3: Evolución de un modelo de mezclas gaussianas

*Esperanza-Maximización* (*Expectation-Maximization*), es la técnica más popular para determinar los parámetros de un modelo de mezclas gaussianas. Se compone de dos pasos:

♢ Paso $E$: asignar de forma probabilística cada muestra a cada clase, basándose en la hipótesis actual de los parámetros.

♢ Paso $M$: actualizar las hipótesis de los parámetros, en función de las asignaciones actuales.

En el paso $E$ se calcula el valor esperado de las asignaciones de grupos. En el paso $M$ se obtiene la nueva máxima verosimilitud de las hipótesis.

**Modelos de mezclas gaussianas vs k-medias**

Si bien el algoritmo inicia con un número predefinido de grupos al igual que k-means, existen diferencias fundamentales:

♢ k-medias establece un círculo (hiperesfera) al centro del grupo, con radio definido por el punto más distante

♢ GMM funciona mejor cuando los datos no se ajustan a circunferencias (hiperesferas)

♢ k-medias realiza clasificación dura: devuelve la clase a la que pertenece una muestra, mientras GMM hace clasificación suave: devuelve la probabilidad de pertenencia a cada clase.

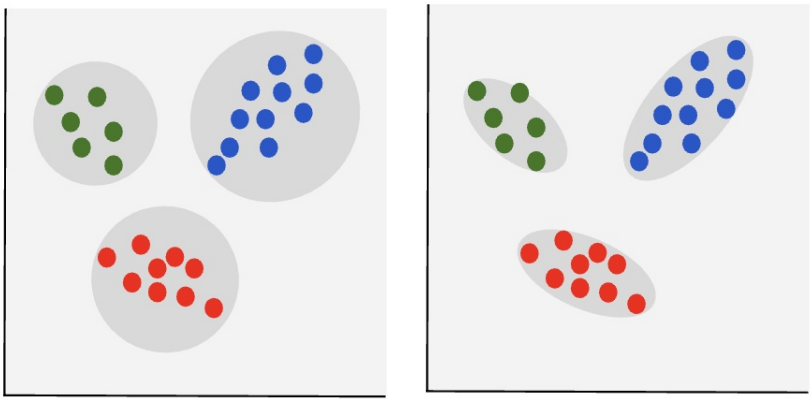


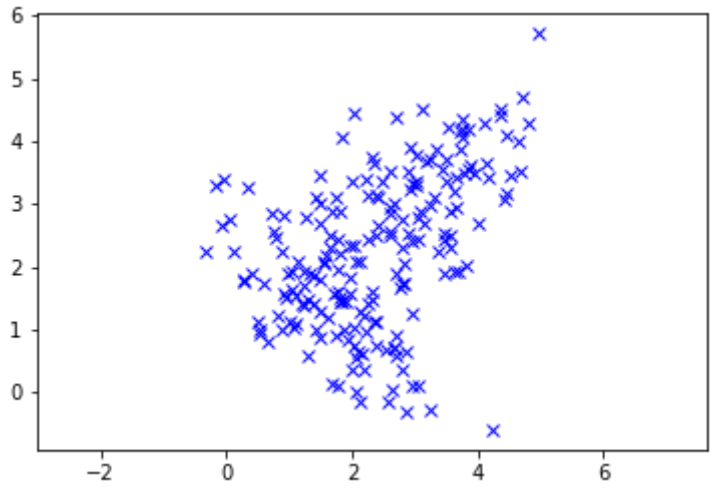
Figura 4: k-medias vs GMM

**Ejemplos con Python**

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
```

```
# https://bit.ly/3Bic3iu
X_train = np.load('data.npy')
```

```
plt.plot(X_train[:,0], X_train[:,1], 'bx')
plt.axis('equal')
plt.show()
```

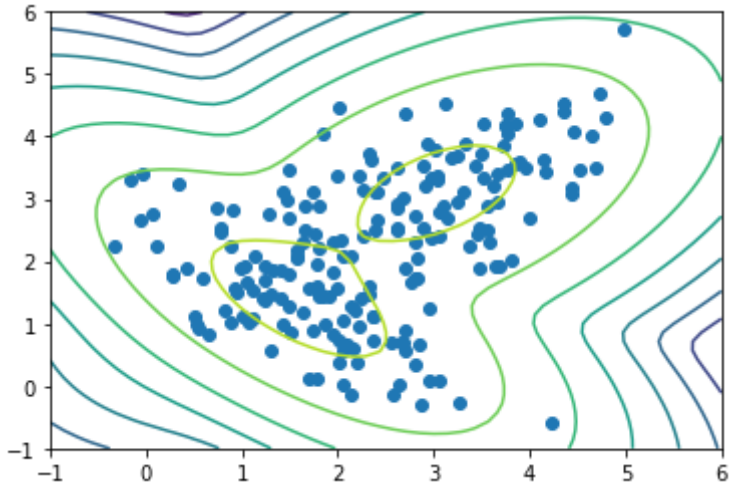



```
gmm = GaussianMixture(n_components=2)
gmm.fit(X_train)

print("Medias: \n", gmm.means_)
print("Covarianzas: \n",gmm.covariances_)

```

```
Medias:
 [[3.04641134 3.10654272]
 [1.60718016 1.35251723]]


Covarianzas:
 [[[ 0.83656079  0.37865596]
  [ 0.37865596  0.72727426]]

 [[ 0.74995307 -0.5010097 ]
  [-0.5010097   0.74377694]]]
```

```
X, Y = np.meshgrid(np.linspace(-1, 6), np.linspace(-1,6))
XX = np.array([X.ravel(), Y.ravel()]).T
Z = gmm.score_samples(XX)
Z = Z.reshape((50,50))

```

```
5
6   plt.contour(X, Y, Z)
7   plt.scatter(X_train[:, 0], X_train[:, 1])
8   plt.show()
9
```




```
1
```