

---

---

# Clase 02 - Clustering

— Aprendizaje Automático  
Aplicado —

---

---

# Clustering

Conjunto de técnicas para agrupar un conjunto de datos en grupos (clusters) de manera que tengan sentido o sean útiles para alguna tarea

¿Quién le da el sentido?

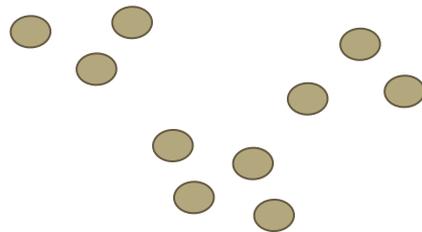
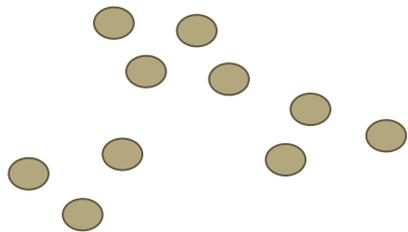
Los grupos se arman según la información misma de los datos:

Buscamos subconjuntos de datos relacionados entre sí, y a su vez separados de los otros

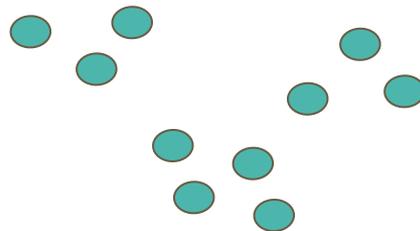
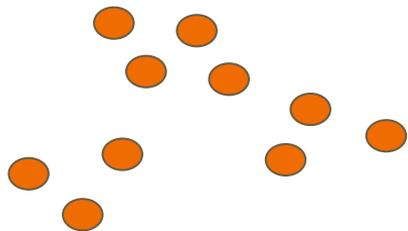
Técnica exploratoria

¿Es igual a clasificación?

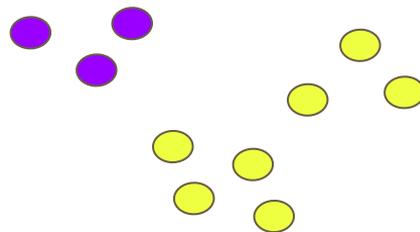
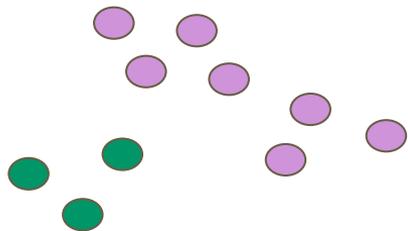
# Ejemplo



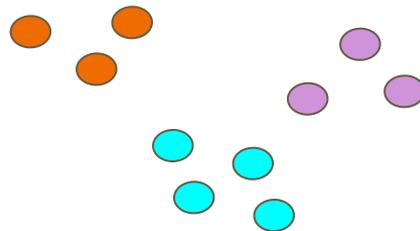
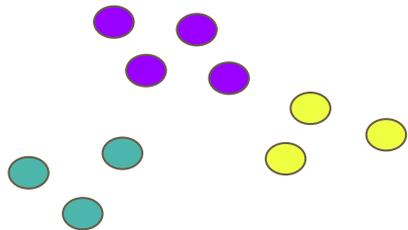
## Ejemplo - 2 clusters



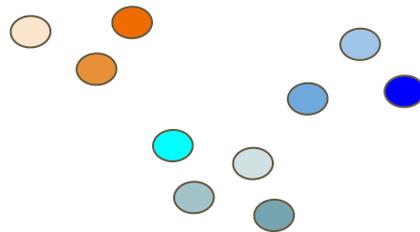
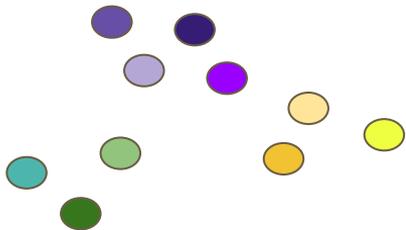
## Ejemplo - 4 clusters



## Ejemplo - 6 clusters



## Ejemplo - 10 clusters?



# Clustering

Intentamos identificar clusters “naturales”

¿Siempre existen?

Los métodos de clustering intentan encontrar este tipo de clusters de manera automática

Queda a nuestro criterio darles una interpretación semántica

# Clustering

Biología

Segmentación de mercado

Análisis de redes sociales

Recuperación de información

# K-means

# K-means

Cada cluster se representa mediante un punto en el espacio (denominado centroide)

Tengo  $K$  de estos puntos

Los puntos que queden más cerca del centroide  $c_i$  que de cualquier otro centroide corresponden al cluster  $C_i$

*En python:* `sklearn.cluster.KMeans`

# K-means

Comienzo sorteando K puntos aleatoriamente (centroides)

Para cada punto del conjunto de entrenamiento, calculo a qué distancia de los centroides está y le asigno el que quede más cerca

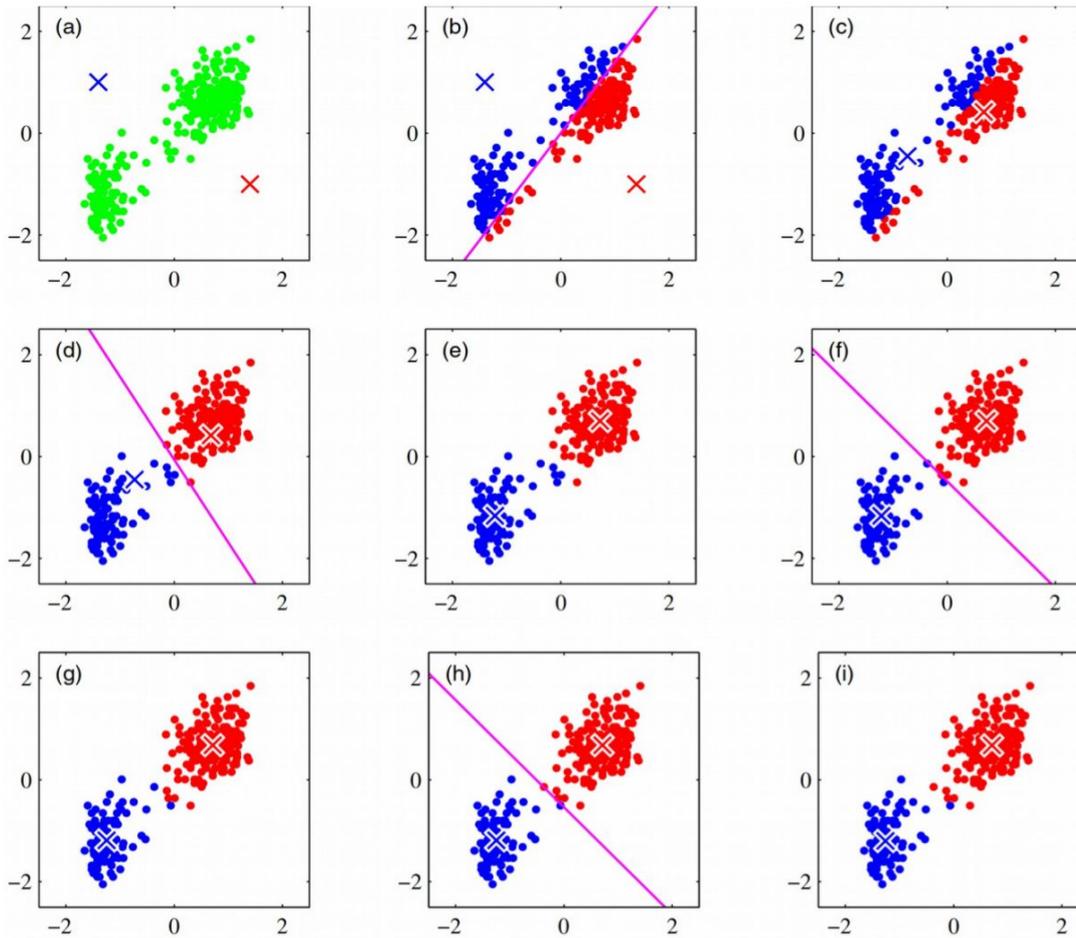
Tomo los nuevos K conjuntos y calculo sus centroides geométricos para generar los nuevos K puntos

Repito hasta que converja:

Los puntos ya no cambian de conjunto entre una iteración y la siguiente

(Si son muchos puntos y queda oscilando, que menos de cierto % cambie)

# Ejemplo K-means



# K-means

Noción de distancia:

La más habitual es la distancia euclídea

En otras aplicaciones puede servir más la distancia coseno

Se puede ver como un problema de optimización:

Costo = Suma del cuadrado de las distancias (SSE, sum of squared error)

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist(x, c_i)$$

*En python: atributo `inertia_` de `sklearn.cluster.KMeans`*

# K-means

En general es muy rápido y converge en pocas iteraciones

Muy sensible a los puntos de inicialización: cae en óptimos locales

Los algoritmos generalmente realizan varias pasadas y se quedan con la mejor

No da buenos resultados si los datos no se adaptan al tipo de cluster que espera K-means

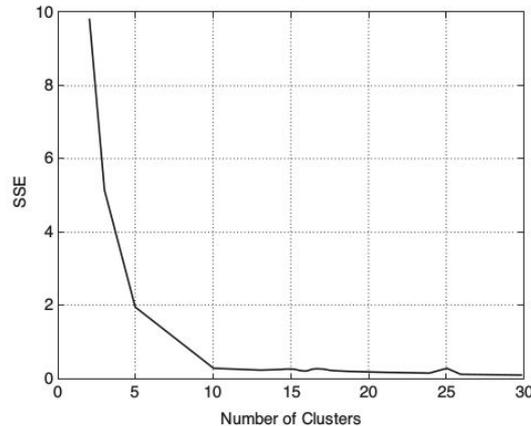
# K-means

¿Cómo elijo el mejor K?

Una opción es analizar la función de costo

La función tiende a 0 cuando aumenta K

Puede verse que en cierto momento se “desacelera” el descenso del valor



# K-means

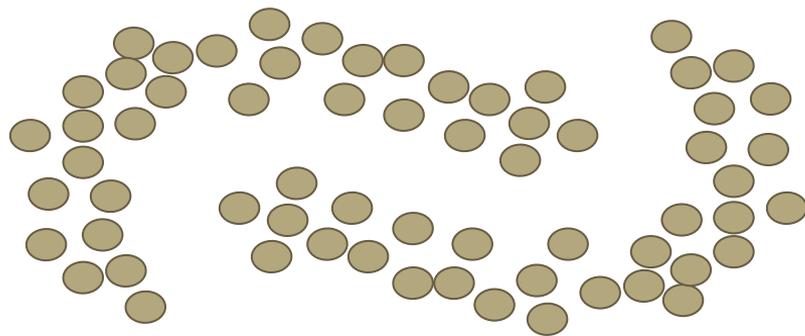
Se puede usar para clasificar!

¿Cómo?

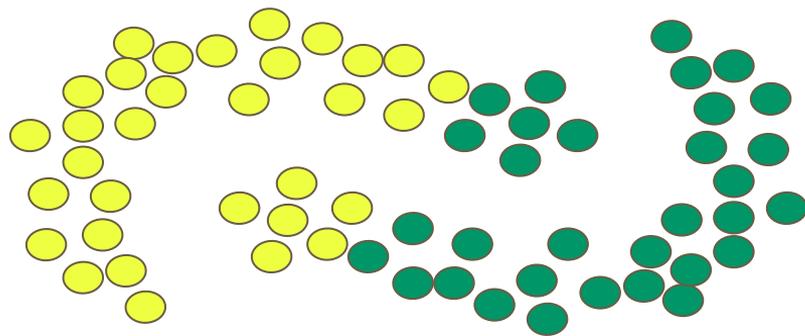
*En python: método `predict()` de `sklearn.cluster.KMeans`  
No todos los algoritmos de clustering lo tienen!*

# DBSCAN

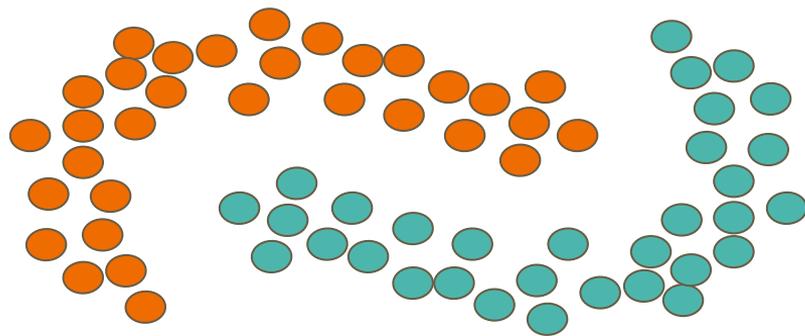
# Ejemplo



# Ejemplo



# Ejemplo



# DBSCAN

Método basado en densidad de los datos

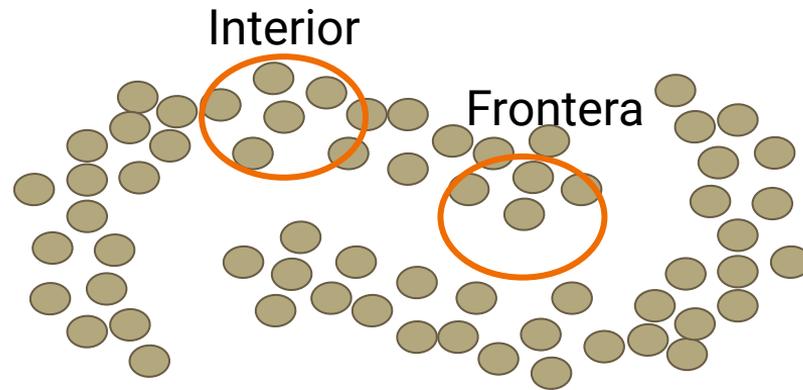
Trazamos un radio *epsilon* desde cada punto y vemos cuántos vecinos tiene

Los categorizamos en puntos “interiores”, “de frontera” o “ruido”

Armamos clusters con todos los puntos interiores posibles limitados por puntos de frontera

*En python:* `sklearn.cluster.DBSCAN`

# DBSCAN



# DBSCAN

El algoritmo determina la cantidad de clusters a formar

Pueden quedar puntos fuera de todo cluster (ruido)

Tiene problemas cuando hay clusters con diferentes densidades

Es más lento de calcular

# Evaluación

# Evaluación

Métricas no supervisadas:

Mediciones de la coherencia interna de los clusters encontrados según los datos

Métricas supervisadas:

Si el conjunto está etiquetado, podemos utilizar las etiquetas como información externa y ver qué tanto se parecen los clusters a la distribución de etiquetas

# Métricas no supervisadas

Cohesión = Suma de distancias de los puntos dentro de un mismo cluster

Separación = Suma de distancias de los puntos de clusters distintos

Una buena clusterización busca que la cohesión sea baja y la separación alta

Cohesión + Separación = Una constante que es la distancia total entre todos los pares de puntos

# Coeficiente de Silueta

Dados un conjunto de datos y los clusters encontrados, para cada punto  $i$  defino:

$a_i$  = promedio de distancias del punto  $i$  a los demás puntos de su cluster

$b_i$  = mínimo de los promedios de distancias del punto  $i$  a cualquier cluster que no sea el suyo

Índice de silueta de  $i$ :  $s_i = (b_i - a_i) / \max(a_i, b_i)$

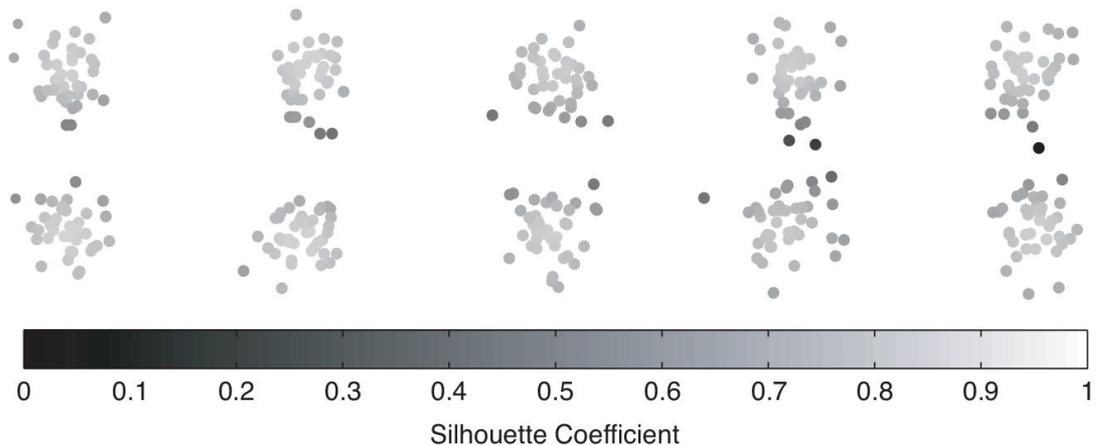
$a_i$  mide cohesión,  $b_i$  contempla separación

# Coeficiente de Silueta

Cada  $s_i$  varía entre -1 y 1:

valores positivos son mejores

si tiene valor negativo, el punto es más cercano a otro cluster que al que tiene asignado

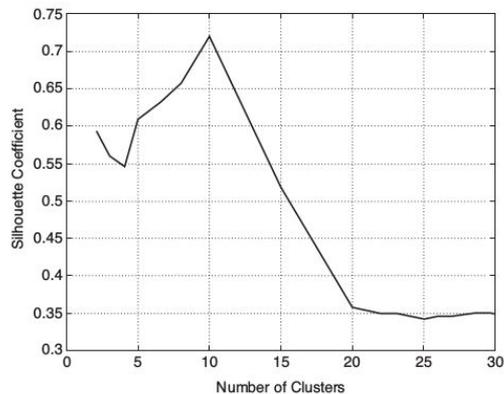


# Coeficiente de Silueta

El coeficiente de silueta es el promedio de los  $s_i$  para todos los puntos

Cuanto mayor sea su valor, mejores serán los clusters

Puede servir para encontrar la cantidad de clusters ideal



*En python:* `sklearn.metrics.silhouette_score`

# Métricas supervisadas

En estos casos conozco, además del conjunto de clusters, la etiqueta asociada a cada dato

Homogeneidad: Si cada cluster contiene datos correspondientes a una sola etiqueta

Complejidad: Si cada cluster contiene todos los datos correspondientes a una sola etiqueta

v-score: Media armónica entre H y C

*En python:* `sklearn.metrics.homogeneity_score`

`sklearn.metrics.completeness_score`

`sklearn.metrics.v_measure_score`

# Índice Rand

Puntos: p1, p2, p3, p4, p5

Clusters: C1={p1, p2, p3} y C2={p4, p5}

Etiquetas: L1={p1, p2} y L2={p3, p4, p5}

Armo una matriz de pertenencia para los clusters y una para las etiquetas

C	p1	p2	p3	p4	p5
p1	1	1	1	0	0
p2	1	1	1	0	0
p3	1	1	1	0	0
p4	0	0	0	1	1
p5	0	0	0	1	1

L	p1	p2	p3	p4	p5
p1	1	1	0	0	0
p2	1	1	0	0	0
p3	0	0	1	1	1
p4	0	0	1	1	1
p5	0	0	1	1	1

# Índice Rand

Armo la matriz conjunta

CL	p1	p2	p3	p4	p5
p1	11	11	10	00	00
p2	11	11	10	00	00
p3	10	10	10	01	01
p4	00	00	01	11	11
p5	00	00	01	11	11

Cuento las cantidades  $f_{11}, f_{00}, f_{10}, f_{01}$  (la matriz es simétrica, no repetir)

$$\text{Índice Rand: } RI = (f_{11} + f_{00}) / (f_{01} + f_{11} + f_{00} + f_{10})$$

# Índice Rand

RI varía entre 0 y 1

En general se utiliza el Índice Rand Ajustado (ARI) que resta la esperanza de que se hayan elegido los clusters por azar

ARI varía entre -1 y 1, siendo 0 el azar y 1 el match perfecto

*En python:* `sklearn.metrics.rand_score`

`sklearn.metrics.adjusted_rand_score`