

# **Tema 06. Aprendizaje no supervisado**

**Autor: Ismael Sagredo Olivenza**

# 6.1 Introducción

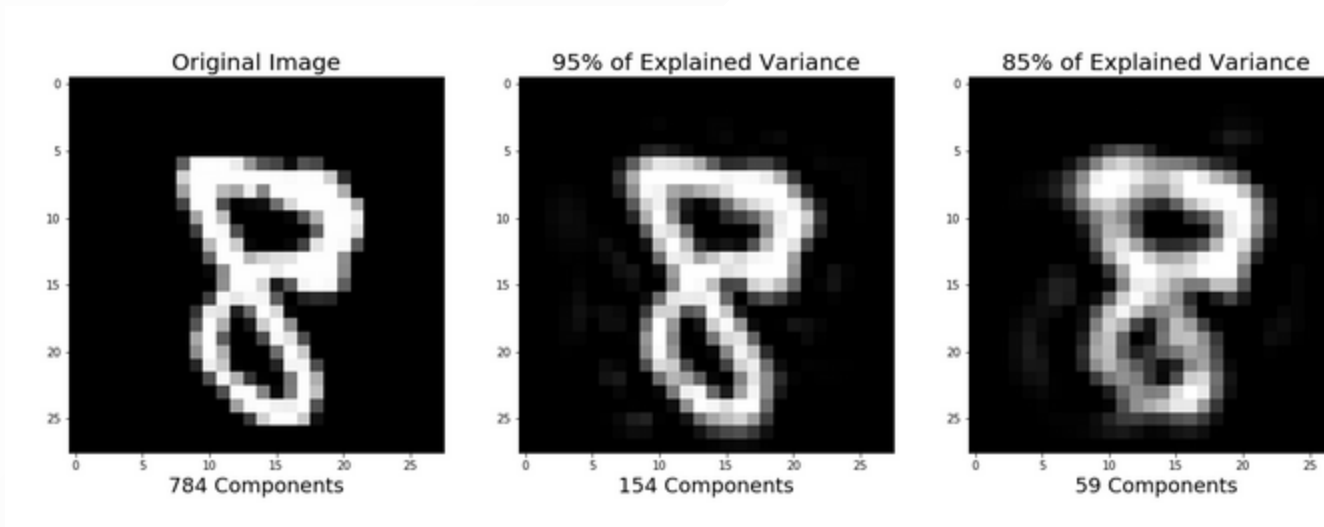
No siempre se conocen las clases o valores esperados de los datos. Si no se conocen, no podemos hacer aprendizaje supervisado y tenemos que recurrir al aprendizaje no supervisado.

También se aplica aprendizaje no supervisado cuando queremos extraer las características más relevantes de un conjunto de datos.

Comenzaremos con algunas técnicas básicas de reducción de dimensionalidad y continuaremos con los algoritmos de agrupamiento clásicos.

## 6.2 Reducción de dimensionalidad

- Datos con alta dimensionalidad (gran número de atributos) son difíciles de interpretar y procesar.
- Datos con una baja dimensionalidad son más fáciles de procesar.



Fuente: <https://www.codementor.io/@mgalarny/pca-using-python-scikit-learn-pandas-ensfh3huz>

## 6.2.1 Principal Component Analysis (PCA)

PCA nos permite reducir la dimensionalidad de los datos de entrada sin perder toda la información.

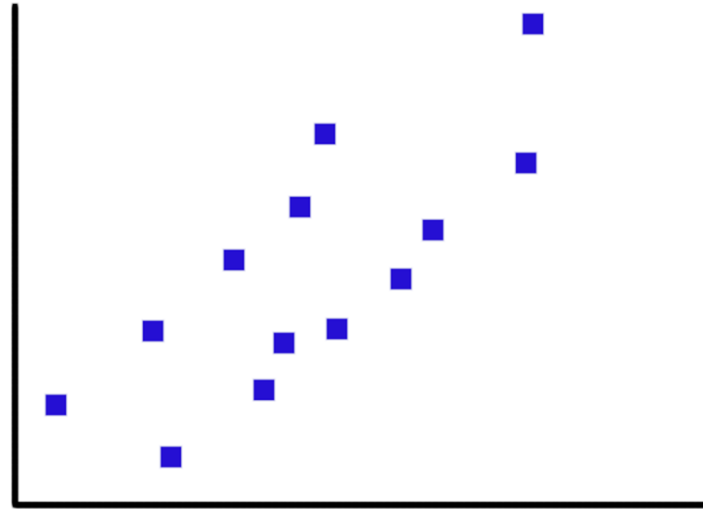
PCA transforma linealmente **las variables correlacionadas** en un número menor de **variables no correlacionadas**. Esto se hace proyectando (producto punto) los datos originales en el espacio reducido utilizando la descomposición en valores y vectores propios de la matriz de covarianza/correlación

- PCA proyecta la información a un espacio de dimensión menor.
- PCA es **efectivo cuando la correlación es alta**. Podemos mirar primero la correlación y después analizar si podría ser útil o no aplicar PCA.

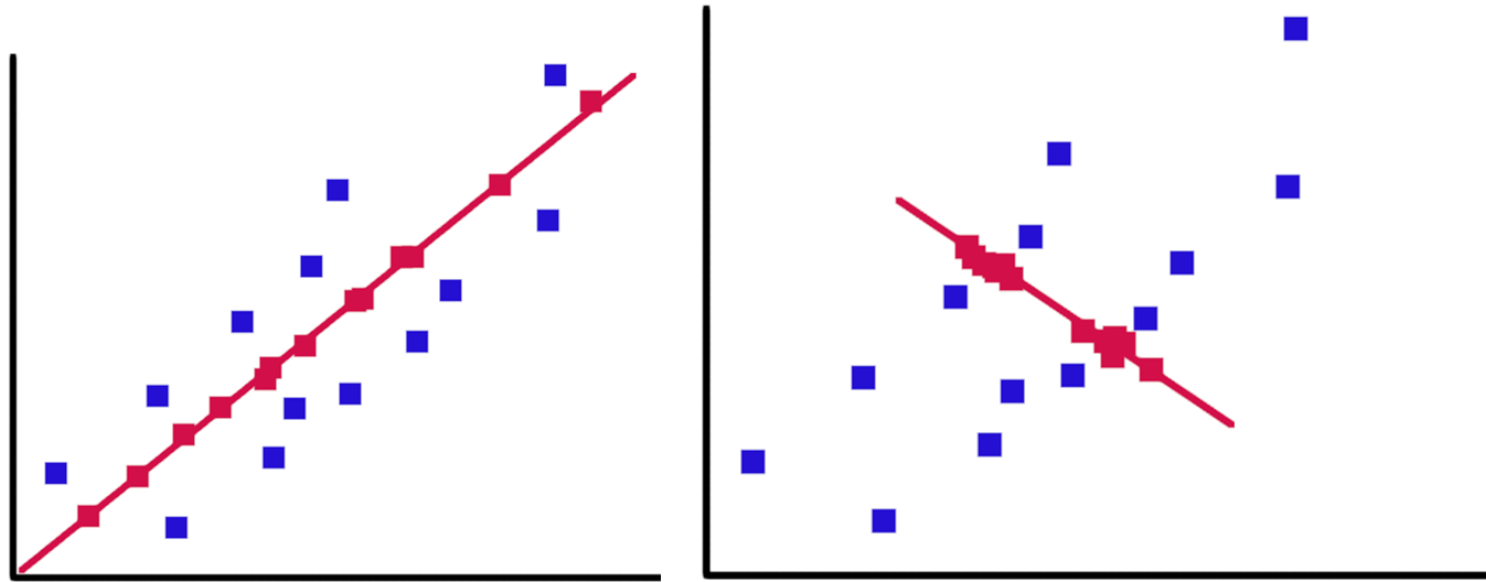
IDEA: Encontrar una dirección (un vector) sobre la que proyectar los datos para minimizar el error de proyección.

Para reducir de  $n$  dimensiones a  $k$  dimensiones debemos encontrar  $K$  vectores, sobre los que proyectar los datos para minimizar el error de proyección.

Veamos un ejemplo: Vamos a imaginar un problema de 2 dimensiones por simplicidad que queremos convertir en 1 dimensión



Ahora queremos proyectar estos puntos a 1 dimensión. ¿Cómo lo hacemos? Podemos hacerlo de dos formas:



La primera gráfica tiene mucha más información que la segunda (mayor varianza)

PCA debe obtener la mayor varianza de los datos proyectados, manteniendo las variables implicadas como independientes entre sí.

Existen dos formas de calcularlo:

- Método basado en la matriz de correlación: datos dimensionalmente no homogéneos
- Método basado en la matriz de covarianzas: cuando los datos son dimensionalmente homogéneos



## 6.2.2 Método basado en correlaciones

El método parte de la matriz de correlaciones. Para cada uno de los  $n$  individuo del conjunto de entrenamiento con  $m$  variables de entrada construimos la matriz de correlación como sigue:

$$R = |r_{ij}| = \frac{cov(F_i, F_j)}{\sqrt{var(F_i) \cdot var(F_j)}}$$

La matriz es simétrica y por tanto diagonalizable. Los valores propios de la diagonal deben sumar  $m$ . Estos  $m$  valores propios son los pesos y cada uno de los principales puede ser descrito como una combinación lineal de los otros

## 6.2.3 Método basado en las covarianzas

El objetivo es transformar un conjunto dado de datos de dimensión  $n \times m$  a otro conjunto de datos de menor dimensión  $n \times l$  con la menor pérdida de información útil posible utilizando para ello la matriz de covarianza ( $l \leq \min(n, m)$ )

Escalar los datos: restándoles la media y dividiendo por su desviación típica. Luego calculamos la matriz de covarianzas:

$$\text{cov}(X, Y) = \frac{\sum (X_i - \hat{X})(Y_i - \hat{Y})}{n}$$

La matriz se calcula para toda pareja de atributos válida.

Después calculamos los valores propios y vectores propios (eigenvectors, eigenvalues). Estos se calculan usando los multiplicadores de Lagrange para obtener los valores máximos de los componentes seleccionados.

Después, derivar las características de los componentes principales tomando dot product del vector propio y las columnas estandarizadas

```
X_standard = standar(X)
df_cov = X_standard.cov() # dataframe.cov()

from numpy.linalg import eig
eigvalues, eigvectors = eig(df_cov)

X_pca = np.dot(X_standard, eigvectors)
```

## 6.2.4 Preprocesado de los datos

Los datos deben estar en la misma escala. Lo normal es normalizar restando por la media y dividiendo por la desviación típica.

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

Replace each  $x_j^{(i)}$  with  $x_j - \mu_j$

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

## 6.2.5 SKLearn ya lo tenemos implementado.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer

data=load_breast_cancer()
df1=pd.DataFrame(data['data'],columns=data['feature_names'])

scaling=StandardScaler()

scaling.fit(df1)
Scaled_data=scaling.transform(df1)

principal=PCA(n_components=3)
principal.fit(Scaled_data)
x=principal.transform(Scaled_data)
```

## 6.2.6 Análisis del algoritmo:

- Presupone que los datos pueden ser proyectados usando una combinación lineal.
- Asume cierta correlación entre las características
- Se pierde información
- Se pierde interpretabilidad de los datos. Ahora los datos con los que se entrena el modelo de ML son atributos que están creados a partir de combinaciones lineales de los atributos originales.
- Problema si hay muchos casos atípicos (outliers)

## 6.2.7 Aplicaciones y malos usos

- Mejora el desempeño de los algoritmos de ML.
- Reducción el ruido (menos atributos y con más información)
- Mejora de visualización (más de tres atributos no puedo)

### **Malos usos:**

- No es recomendable usarlo para prevenir el overfitting, mejor usar regularización.
- No usar PCA si sin usarlo se consiguen buenos resultados. Aplicarlo solo cuando sin aplicar los resultados no son los satisfactorios.

## 6.3 Técnicas de agrupamiento o Clustering

El objetivo es agrupar los  $n$  individuos de nuestro conjunto de datos en una serie de grupos de forma que:

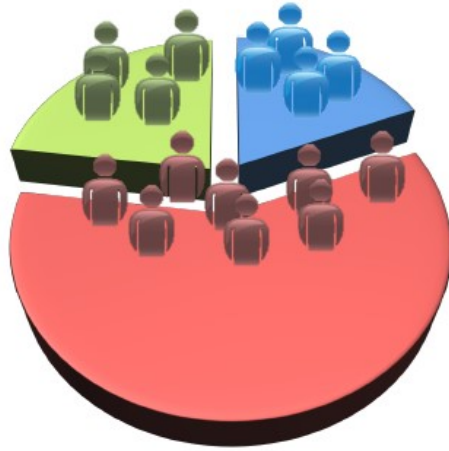
- Los individuos del mismo grupo sean lo más similares entre sí
- Los individuos de grupos diferentes sean lo más diferentes entre sí

Estos grupos nos revelarán cierta estructura de los datos. Por lo que suele ser útil hacer clustering aunque sólo sea para analizarlos.

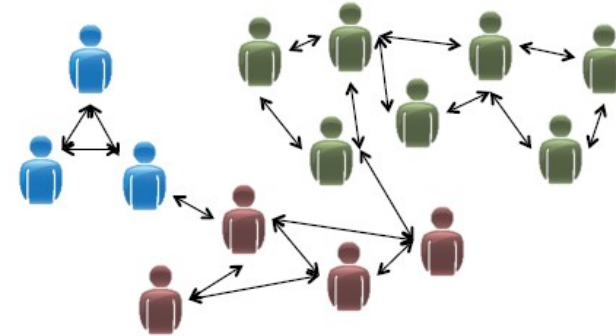
- Cuantos grupos puede y cuales son más numerosos
- Grupos más homogéneos o más dispersos, atípicos, etc.



# Applications of clustering



Market  
segmentation



Social network  
analysis



Organize computing  
clusters



Image credit: NASA/JPL-Caltech/E. Churchwell (Univ. of Wisconsin, Madison)

Astronomical data  
analysis

## 6.4 Estrategia aglomerativa

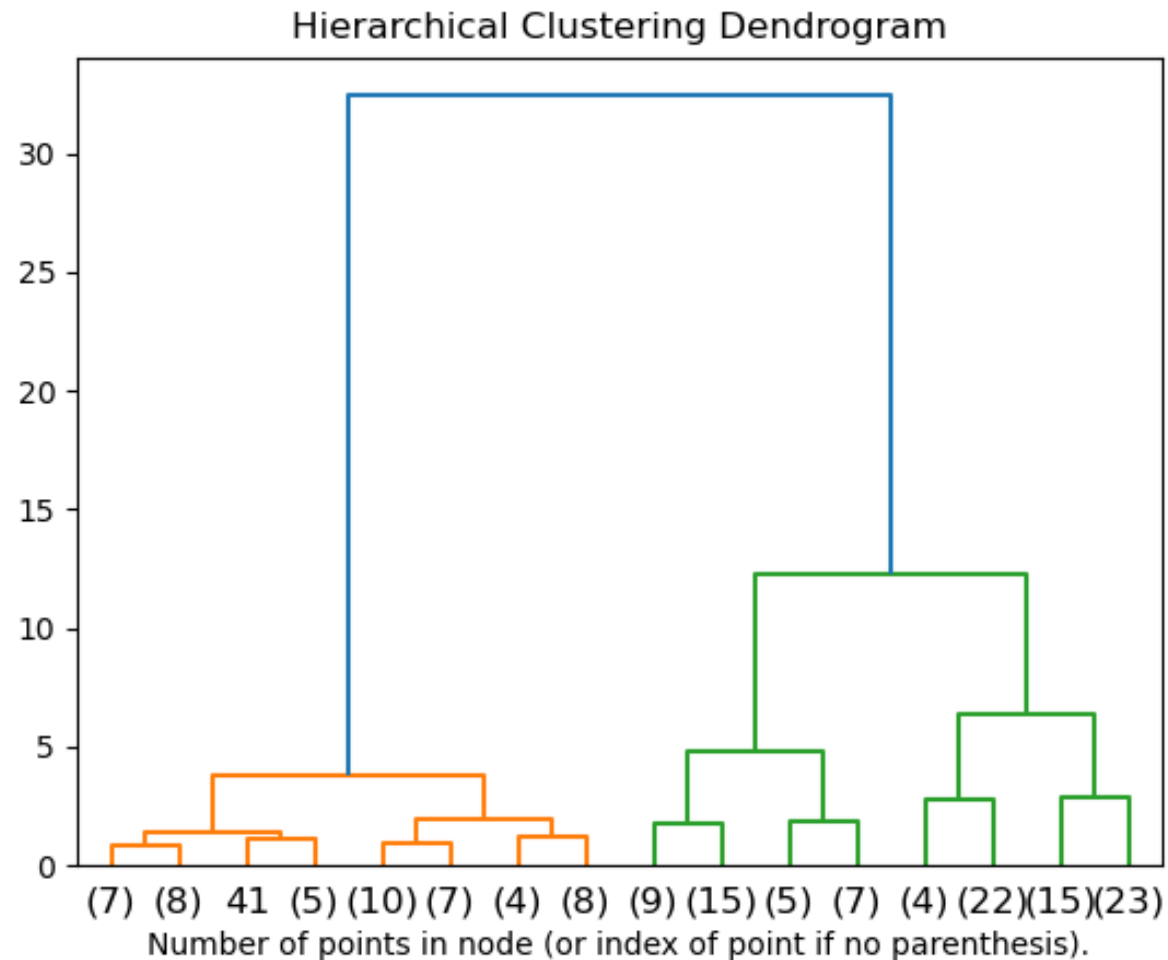
Este algoritmo inicialmente crea una clase por cada ejemplo de entrada. Después y siguiendo el criterio de similitud elegido, busca los dos conceptos más cercanos entre si y los agrupa en un concepto ficticio, sacando estos dos nodos de la lista y sustituyendolos por el nuevo concepto. Este nuevo nodo tendrá a los antiguos como hijos suyos.

el proceso se repite iterativamente hasta que no quedan más nodos que agrupar.

## 6.4.1 Algoritmo

```
function Aglomeratica(E, similitud): nodo
E: ejemplos de entrenamiento
Para todo e de E
    nodos = crearNodo(e)
Mientras nodos.length < 1
    sim1, sim2 = similitud(nodos)
    nuevo = crearNodo()
    nuevo.hijos = {sim1, sim2}
    nodos = nodos - {sim1, sim2}
    nodos = nodos + nuevo
devolver nodos
```

## 6.4.2 Ejemplo de clustering: Dendrograma



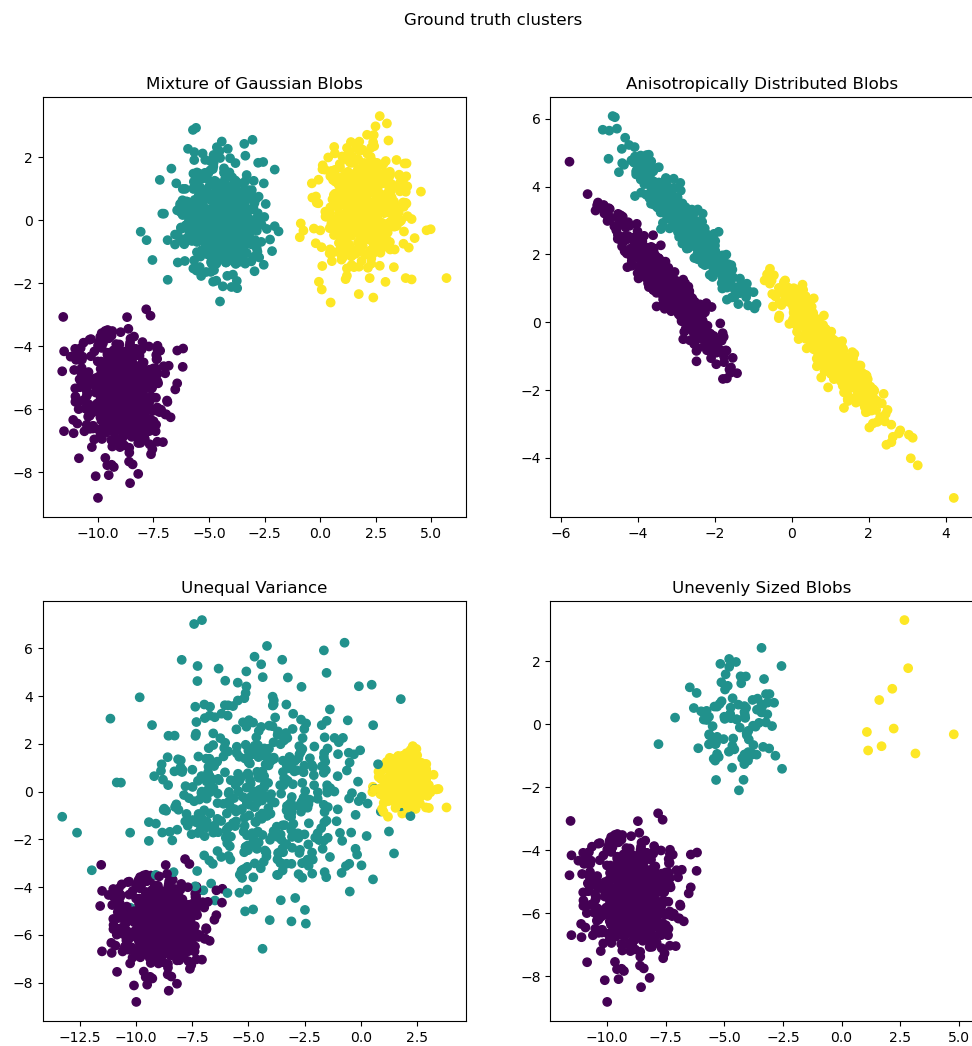
## 6.5 K-Means

Idea es maximizar la similitud entre los elementos de una clase y minimizar la similitud entre las diferentes clases. No crea jerarquías de clases si no clases de un único nivel.

El algoritmo elige un conjunto de  $k$  semillas, siendo  $k$  un parámetro dado por el usuario. Después iterativamente va agrupando los ejemplos en las  $k$  clases y recalculando el valor de dicha semilla, como el centroide de dicha clase.

Debe existir un criterio de parada o convergencia que hace que el algoritmo se detenga. Un posible criterio puede ser si durante dos ciclos no cambia la clasificación de los ejemplos.

## 6.5.1 Ejemplo de clustering con K-means



## 6.5.2 Algoritmo

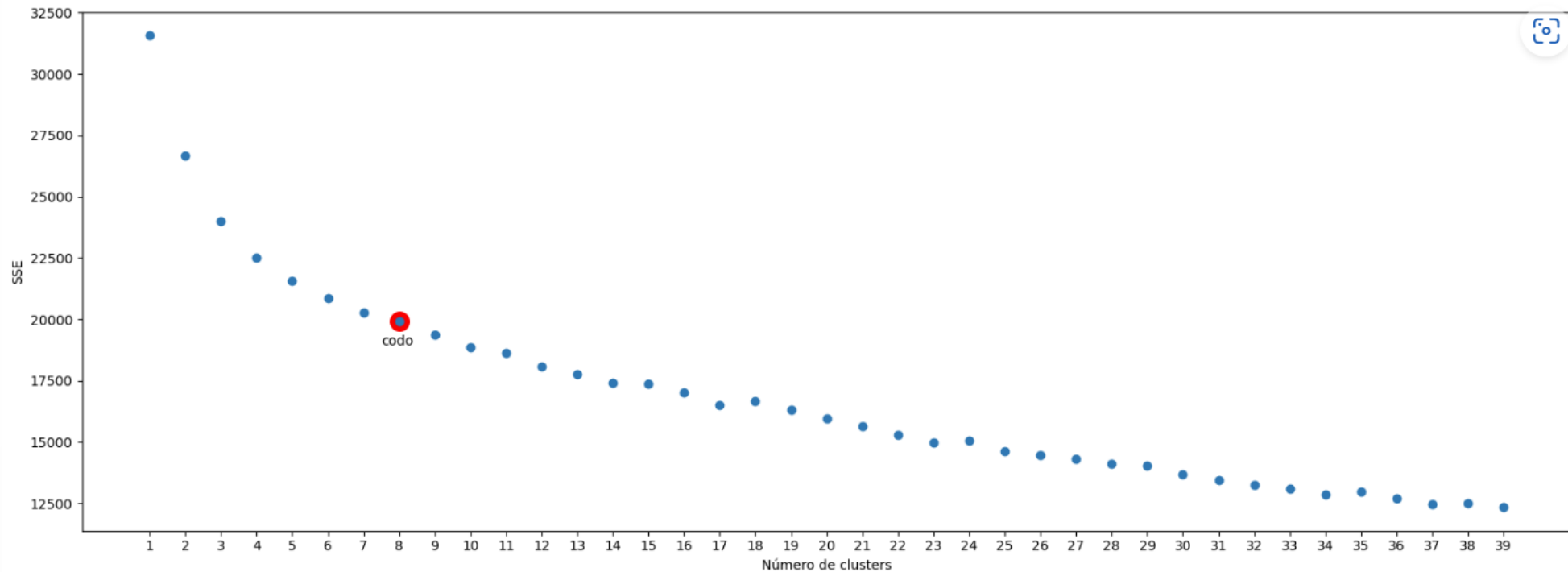
```
function Kmeans(E, similitud, K, Convergencia): clases
E: ejemplos de entrenamiento
K: clases que se desean
Para todo k de K
    c = crearClase(k)
    clase.semilla = crearSemilla(c)
    clases = clases + c
Mientras no Convergencia(clases)
    Para cada e de E
        c = claseMassimilar(clases, e)
        c.ejemplos += e
    para cada c en C
        c.semilla = CalcCentroide(c.ejemplos)
devolver clases
```

## 6.5.3 Métodos para intuir el valor óptimo de K

No sabemos cuál es el valor óptimo de K que mejor agrupa. No sabemos cuantos grupos hay realmente y por tanto no podemos saber cual es el valor óptimo de k.

si k es más grande la distancia acumulada de los ejemplos de cada cluster a los centroides será siempre menor. Por lo que, lo que debemos intentar es encontrar el valor de k que produce un punto de inflexión en la curva. A esto se le denomina **técnica del Codo**. Y se puede hacer manualmente graficando la función de "inercia" de k-means por cada k elegido.





Pero normalmente hay librerías disponibles que nos lo permiten calcularlo.

```
from kneed import KneeLocator  
kl = KneeLocator(range(1, 40), sse, curve="convex", direction="decreasing")
```

### **6.5.3.1 Coeficiente de Silhouette para medir el rendimiento**

El coeficiente de Silhouette mide qué tan cerca está una muestra a las otras muestras de su cluster y qué tan lejos está con respecto a las muestras del cluster más cercano. Este coeficiente toma valores de  $[-1,1]$ , -1 sería si los clusters están superpuestos, 0 si hay overlap y 1 que no se tocan.

El puntaje de Silhouette es el promedio de los coeficientes de Silhouette de todas las samples y se computa con la clase.

Valores bajos cercanos a 0 nos indica que la calidad de los clusters no es muy buena

## 6.5.3.2 Ejemplo en SKLearn

```
# importamos el puntaje de silhouette
from sklearn.metrics import silhouette_score

for k in range(2, 20):
    kkkmeans = KMeans(n_clusters=k)
    kkkmeans.fit(data)
    score = silhouette_score(data, kkkmeans.labels_)
    silhouette_coefficients.append(score)

fig, ax = plt.subplots(figsize = (24, 7))

ax.scatter(range(2, 20), silhouette_coefficients)
ax.set_xticks(range(2, 20))
ax.set_xlabel("Número de clusters")
ax.set_ylabel("Promedio coeficientes de Silhouette")
```

### **6.5.3.3 Se precisa de analisis del cientifico de datos**

Normalmente el método del codo puede ayudarnos a ver por donde van los datos, pero en la mayoría de ocasiones, tener intuición y conocimiento del dominio ayuda a elegir el  $k$  óptimo.

en la mayoría de casos es el investigador el que debe analizar cuál es el  $k$  que mejor clasifica y más sentido tiene.

## 6.5.4 Ejemplo de clustering con todos los conceptos que hemos contado está disponible en el campus

- Usamos sklearn
- Utilizamos cluster jerárquico aglomerativo y k-means
- Detección del codo
- Metrica silhouette.

Otros métodos de clustering usando SKLearn:

<https://scikit-learn.org/stable/modules/clustering.html>