Scikit Learn

Última actualización, 11/07/2022.

Biliografía

Principalmente documentación & vídeos.

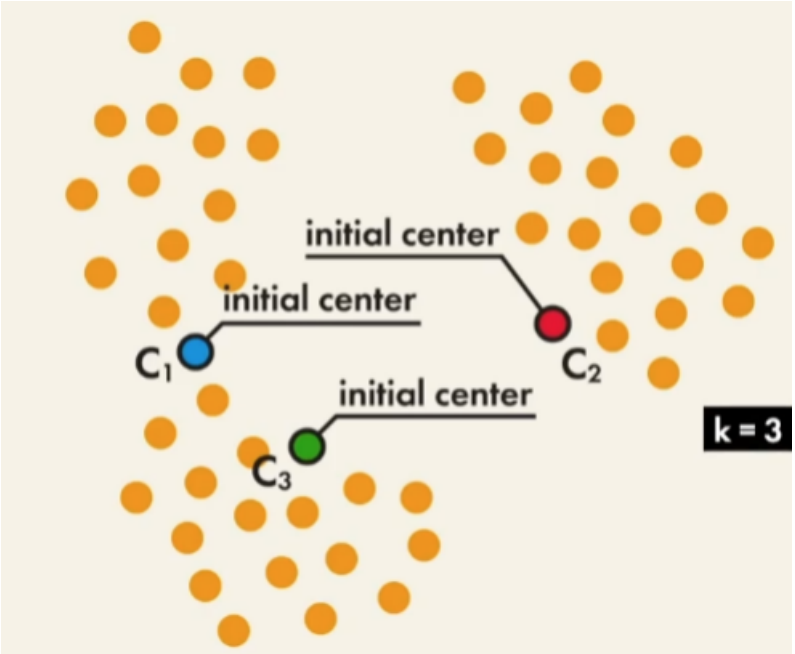
- Intuitive Machine Learning: Intuitive Machine Learning - YouTube

Índice

1. K-means clustering

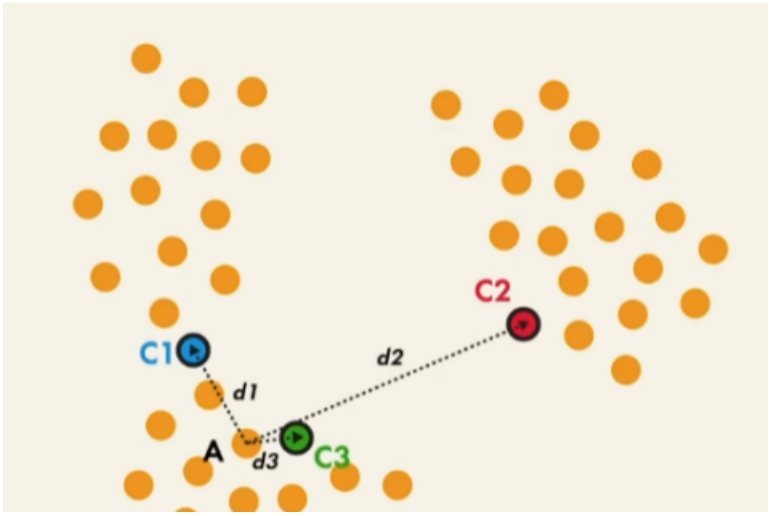
1.1. Anotaciones

- K- means es mu útil en algoritmos de clustering (agrupaciones) no supervisados (sin etiquetar previamente).
- Su función es intentar dividir el dataset en k grupos pre-definidos donde cada dato pertenece a un sólo grupo.
- Consta de 3 pasos:
    - Paso 1, consiste en elegir de manera aleatoria k puntos de datos del set de datos los cuales tomamos como el centro de los datos.

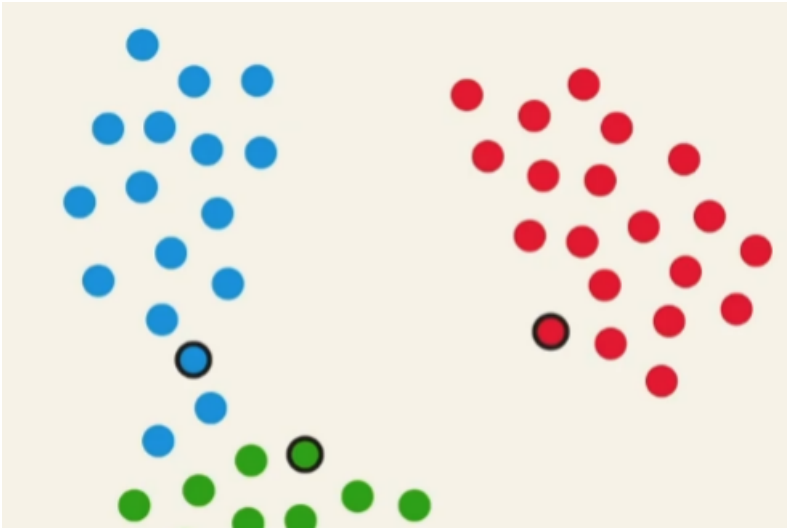


- Paso 2, calculamos la distancia del resto de puntos con respecto a los puntos tomados inicialmente como centros. La distancia la definimos como:

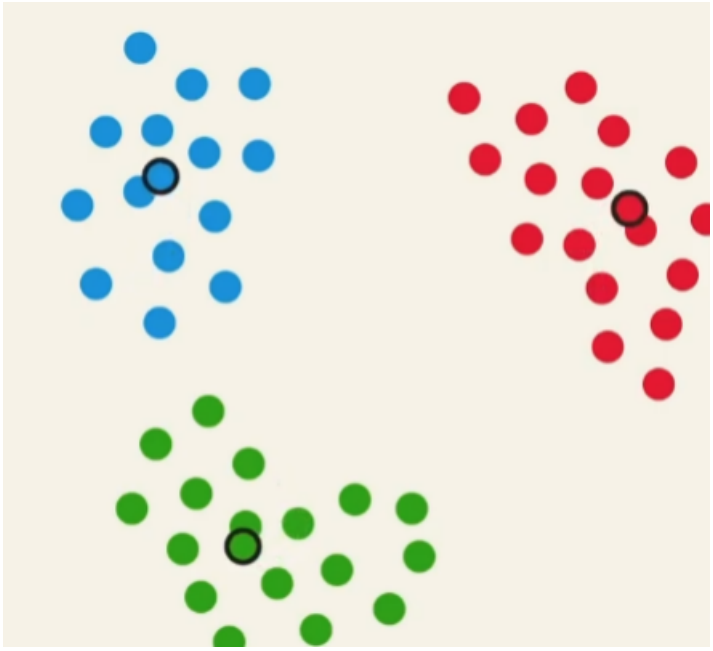
$$\sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$$



De la imagen anterior vemos que A tiene una menor distancia (d3) de C3 que con respecto a C1 y C2 por lo tanto podemos decir que el punto A pertenece al grupo C3. Realizamos este proceso para todos los puntos con el fin de que todos estén agrupados.



- Paso 3, movemos el centro, queremos obtener el centro real de cada grupo de datos, para ello, realizamos la media de todos los puntos de cada grupo y realizamos este procedimiento hasta que los centros converjan.



1.2. Código

```
# Importamos NumPy para el tratamiento de datos
import numpy as np

# Importamos make_blobs para generar manchas gaussianas isotrópicas para la
# agrupación.
# Wikipedia: La isotropía es la característica de algunos fenómenos en el espacio
# cuyas propiedades no dependen de la dirección en que son examinadas
from sklearn.datasets import make_blobs

# Importamos el modelo KMeans de Scikit Learn
from sklearn.cluster import KMeans

# Importamos la librería MatplotLib para realizar gráficos
import matplotlib.pyplot as plt

# make_blobs permite hacer una distribucion gaussiana de datos, donde tenemos que
# considerar que una desviación típica baja indica que la mayor parte de los datos
# de una muestra tienden a estar agrupados cerca de su media
# mientras que una desviación típica alta indica que los datos tienen una mayor
# dispersión de valores.
# Por lo tanto, si cambiamos el valor del cluster_std a un valor menor, veremos
# que los puntos se irán agrupando más y se encontrarán mejor diferenciados
x, y = make_blobs(
  n_samples = 200, # Numero total de puntos
  n_features = 2, # Número de características, 2 dimensiones
  centers = 3, # 3 grupos
  cluster_std = 0.5, # desviacion típica de cada distribucion gaussiana
  random_state = 0 # estado de aleatoriedad de la preparacion de los datos
)

# 'x' son 200 muestras aleatorias hay 2 columnas, cada columna representa una una
# característica para esa muestra
x

# 'y' es sólo una lista de etiquetas de grupos para cada muestra
y

# Vamos a visulizar los datos en 2D con la funcion scatter de Matplotlib
# Si tenemos más de 2 dimensiones podemos aplicar técnicas como PCA
# para la reduccion de dimensiones a 2 y verlo en 2D
plt.scatter(
  x[:, 0], x[:, 1],
```

```
  c = 'white',
  edgecolors = 'black'
)

# n_cluster = el numero de agrupaciones (clusters) que queremos formar, así
# como el número de centros que queremos generar.
# init = inicialización de los centros.
# n_init = indica el número de veces que el algoritmo de k-means
# se ejecutará con diferentes centros como estado inicial. El resultado final
# será la mejor salida de n_init.
# max_iter = número de iteraciones máximas de k-means para una ejecución.
# tol = Tolerancia relativa con respecto a la norma de Frobenius de la diferencia
# de los centros de los clusters de dos iteraciones consecutivas para declarar
# la convergencia.
kmeans = KMeans(
  n_clusters = 3, init = 'random',
  n_init = 1, max_iter = 10,
  tol = 1e-04, random_state = 2
)

y_km = kmeans.fit_predict(x)

# Graficamos los 3 grupos
plt.scatter(
  x[y_km == 0, 0], x[y_km == 0, 1],
  s=50, c='lightgreen',
  marker='s', edgecolor='black',
  label='cluster 1'
)

plt.scatter(
  x[y_km == 1, 0], x[y_km == 1, 1],
  s=50, c='orange',
  marker='o', edgecolor='black',
  label='cluster 2'
)

plt.scatter(
  x[y_km == 2, 0], x[y_km == 2, 1],
  s=50, c='lightblue',
  marker='v', edgecolor='black',
  label='cluster 3'
)

# Graficamos el centro de los grupos
plt.scatter(
  kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
  s=250, marker='*',
  c='red', edgecolor='black',
  label='centros'
)
plt.legend(scatterpoints=1
```