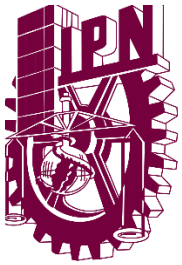


INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

MACHINE LEARNING

5BV1

Ejercicio de Laboratorio 2: K-Means 2

PROFESOR: ANDRÉS GARCÍA FLORIANO

ALUMNOS:

MIGUEL ANGEL OCAMPO PORCAYO,

GERARDO MARTINEZ AYALA

Septiembre 2024

INTRODUCCIÓN

El algoritmo K-Means es uno de los métodos más populares para realizar agrupamientos en el aprendizaje automático no supervisado. Este algoritmo organiza un conjunto de datos en diferentes grupos o *clusters*, donde cada punto pertenece al grupo con el centroide más cercano. Uno de los aspectos clave del K-Means es que su desempeño puede variar dependiendo del número de *clusters* que se defina, lo cual se debe determinar cuidadosamente para lograr buenos resultados.

Algunas de las características que hacen de K-Means una herramienta eficiente incluyen la capacidad de manejar grandes cantidades de datos y su velocidad de convergencia, especialmente cuando se utiliza el método de inicialización `k-means++`. Este método ayuda a seleccionar los centroides iniciales de manera que el proceso converja más rápido y se eviten resultados subóptimos que pueden surgir cuando el algoritmo se queda atrapado en mínimos locales.

En este trabajo, además de explicar las funcionalidades del algoritmo K-Means, se realizan dos implementaciones basadas en los ejemplos proporcionados en la documentación oficial de *scikit-learn*. En la primera, se agrupan un conjunto de puntos en dos *clusters*, mientras que en la segunda se utiliza el algoritmo para predecir la clasificación de nuevos puntos, con el objetivo de mostrar cómo este puede aplicarse a problemas reales.

IMPLEMENTACION 1: K-Means clustering on the handwritten digits data

Para esta implementación se realizan varios pasos los cuales serán explicados brevemente su funcionamiento.

Este código carga el conjunto de datos digits usando la función load_digits de scikit-learn, que contiene imágenes de dígitos escritos a mano. Los datos y las etiquetas se asignan a las variables data y labels, respectivamente. Luego, se obtiene el número de muestras (n_samples), características (n_features) y la cantidad de dígitos únicos (n_digits) en el conjunto. Finalmente, imprime la cantidad de dígitos, muestras y características que hay en los datos.

1_demo of K-Means clusterir

Code

Python 3 (ipykernel)

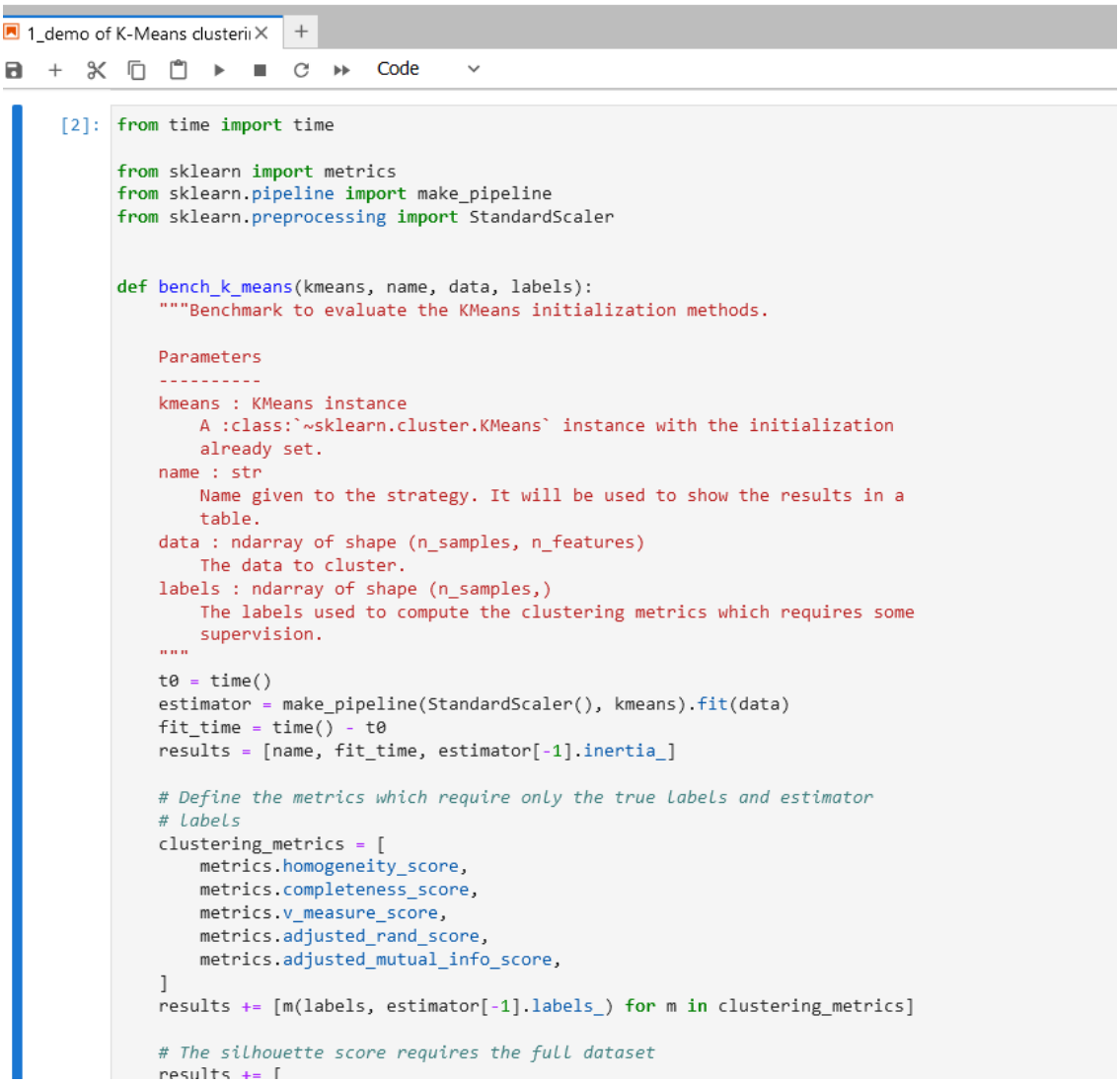
```
[1]: import numpy as np

from sklearn.datasets import load_digits

data, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data.shape, np.unique(labels).size

print(f"# digits: {n_digits}; # samples: {n_samples}; # features {n_features}")
```

```
# digits: 10; # samples: 1797; # features 64
```

Este código define una función llamada `bench_k_means`, que evalúa el rendimiento del algoritmo K-Means con diferentes métodos de inicialización. Dentro de la función, primero se mide el tiempo de ajuste del modelo utilizando un pipeline que incluye el escalado de los datos (`StandardScaler`) y el modelo K-Means. Luego, se calculan varias métricas de evaluación, como la homogeneidad, completitud y el puntaje silhouette, para comparar la calidad de los clusters generados en relación con las etiquetas verdaderas. Finalmente, imprime los resultados en una tabla formateada, mostrando el tiempo de ejecución y las métricas calculadas.

Este código define una función llamada `bench_k_means` que se utiliza para comparar distintos métodos de inicialización del algoritmo K-Means. La función toma como entrada una instancia de K-Means, un nombre para identificar el método de inicialización, los datos a agrupar y las etiquetas verdaderas para evaluar el rendimiento del clustering.

Primero, mide el tiempo que toma ajustar el modelo con los datos utilizando un pipeline que estandariza los datos antes de aplicar el K-Means. Luego, calcula varias métricas de calidad del clustering, como la homogeneidad y la puntuación silhouette, que ayudan a evaluar cuán bien se agruparon los datos. Finalmente, imprime los resultados en una tabla para facilitar la comparación.



Este código compara diferentes estrategias de inicialización para el algoritmo K-Means. Primero, imprime una cabecera para mostrar las métricas de rendimiento que se van a medir: tiempo, inercia, homogeneidad, completitud, etc.

Luego, se crean tres instancias del modelo K-Means con diferentes métodos de inicialización:

k-means++: Es el método predeterminado, que selecciona los centroides de manera eficiente para acelerar la convergencia.

random: Selecciona los centroides iniciales de manera aleatoria.

PCA-based: Usa los componentes principales generados por un análisis PCA (Análisis de Componentes Principales) para la inicialización de los centroides.

Después, cada modelo se evalúa con la función bench_k_means, que compara su desempeño en términos de las métricas de agrupamiento. Finalmente, se imprime una línea separadora para terminar la comparación.

Los resultados muestran una comparación entre tres métodos de inicialización para el algoritmo K-Means: k-means++, random y PCA-based. A continuación, un breve análisis de cada una de las métricas:

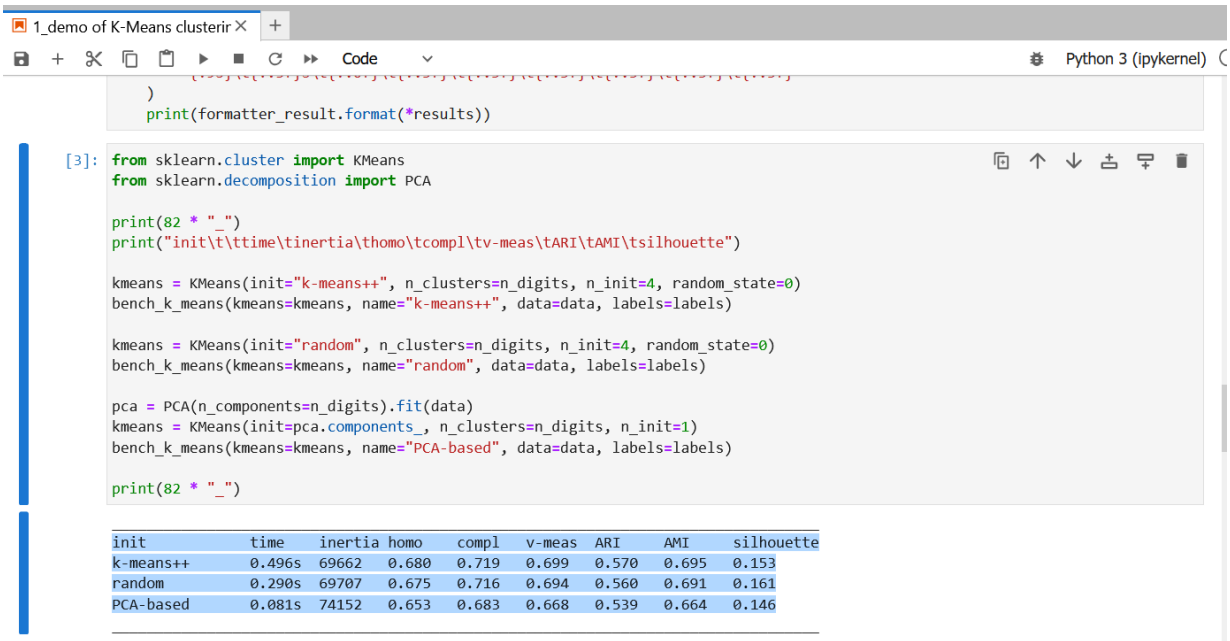
Tiempo (time): El método PCA-based es el más rápido, seguido por random, y finalmente k-means++, que tarda un poco más debido a su estrategia de inicialización optimizada.

Inercia (inertia): La inercia mide la suma de las distancias al cuadrado de los puntos a sus centroides más cercanos. El método k-means++ tiene la menor inercia (69662), lo que indica que genera clusters más ajustados, seguido por random y PCA-based, que tiene la inercia más alta (74152).

Homogeneidad, completitud y v-measure: Estas métricas son ligeramente mejores para k-means++, lo que indica que los clusters generados son más consistentes y completos en comparación con los otros métodos.

ARI (Adjusted Rand Index) y AMI (Adjusted Mutual Information): Estas métricas también favorecen a k-means++ sobre los otros métodos, indicando que su agrupamiento es más similar a las etiquetas verdaderas.

Silhouette Score: Curiosamente, random tiene un puntaje de silhouette un poco mejor (0.161), lo que podría indicar que los clusters están algo más separados en este caso.



Este código lo que hace es agrupar los datos de los dígitos escritos a mano (de 0 a 9) y mostrarlos en un gráfico simple, usando dos pasos clave: reducir los datos a algo más fácil de visualizar y luego agruparlos con el algoritmo K-Means.

Reducir los datos a 2 dimensiones: Los datos originales tienen muchas características (64), lo que hace difícil representarlos en un gráfico. Para simplificarlo, usamos PCA (que es un método para reducir las dimensiones de los datos) y dejamos solo 2 dimensiones. Esto nos permite dibujar los datos en un plano, como si estuviéramos viendo una imagen desde arriba en 2D.

Agrupar los datos con K-Means: Luego aplicamos K-Means para agrupar estos puntos en 10 grupos (ya que hay 10 dígitos, del 0 al 9). Este algoritmo agrupa los datos en clusters (grupos) y calcula los centroides (que son como los puntos centrales de cada grupo).

Crear una cuadrícula: Después, se crea una malla de puntos sobre la que el modelo de K-Means hace predicciones para ver a qué grupo pertenece cada punto de esa cuadrícula. Esto sirve para trazar las fronteras de los clusters, que es donde un punto dejaría de pertenecer a un cluster y pasaría a otro.

Dibujar el gráfico: Se pinta el gráfico, donde, as regiones coloreadas representan los diferentes clusters.

Los puntos negros son los datos originales reducidos a 2 dimensiones.

Los "X" blancos marcan el centro de cada cluster.

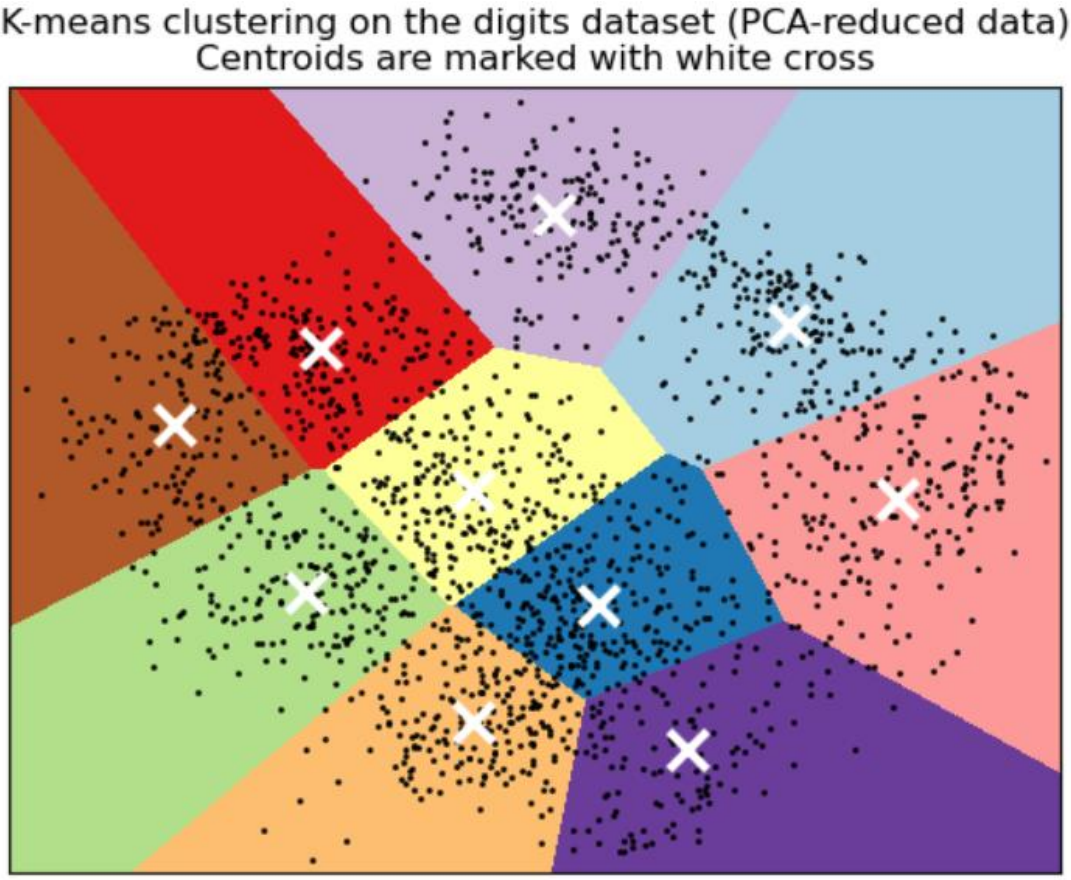
Al final, lo que vemos es un gráfico que muestra cómo se agrupan los dígitos y en qué región del gráfico cae cada cluster. Es una forma visual de entender cómo funciona K-Means y cómo se organizan los datos.

1_demo of K-Means clusterir

Code

[4]:

```
import matplotlib.pyplot as plt

reduced_data = PCA(n_components=2).fit_transform(data)
kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4)
kmeans.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.
h = 0.02  # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color to each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower",
)

plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.", markersize=2)
```



IMPLEMENTACION 2: Demonstration of k-means assumptions

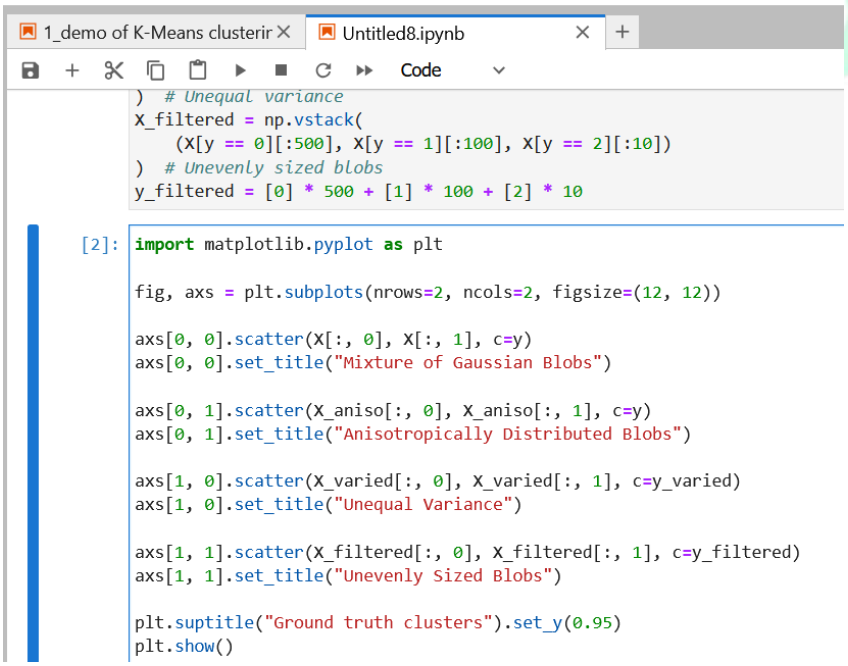
Este código genera varios conjuntos de datos usando la función make_blobs de scikit-learn, que crea agrupaciones de puntos (blobs). Primero, se crean 1500 puntos distribuidos en grupos, y se les aplica una transformación que los distorsiona, haciendo que ya no sean círculos perfectos, sino más alargados en una dirección específica. Luego, se genera otro conjunto de datos donde cada grupo tiene una dispersión diferente (algunos más compactos y otros más dispersos). Finalmente, se seleccionan subconjuntos de esos puntos para que los grupos tengan diferentes tamaños: uno con 500 puntos, otro con 100 y otro con solo 10. Este tipo de datos se puede usar para probar algoritmos de agrupamiento en distintas situaciones.

1_demo of K-Means clusterir | Untitled8.ipynb

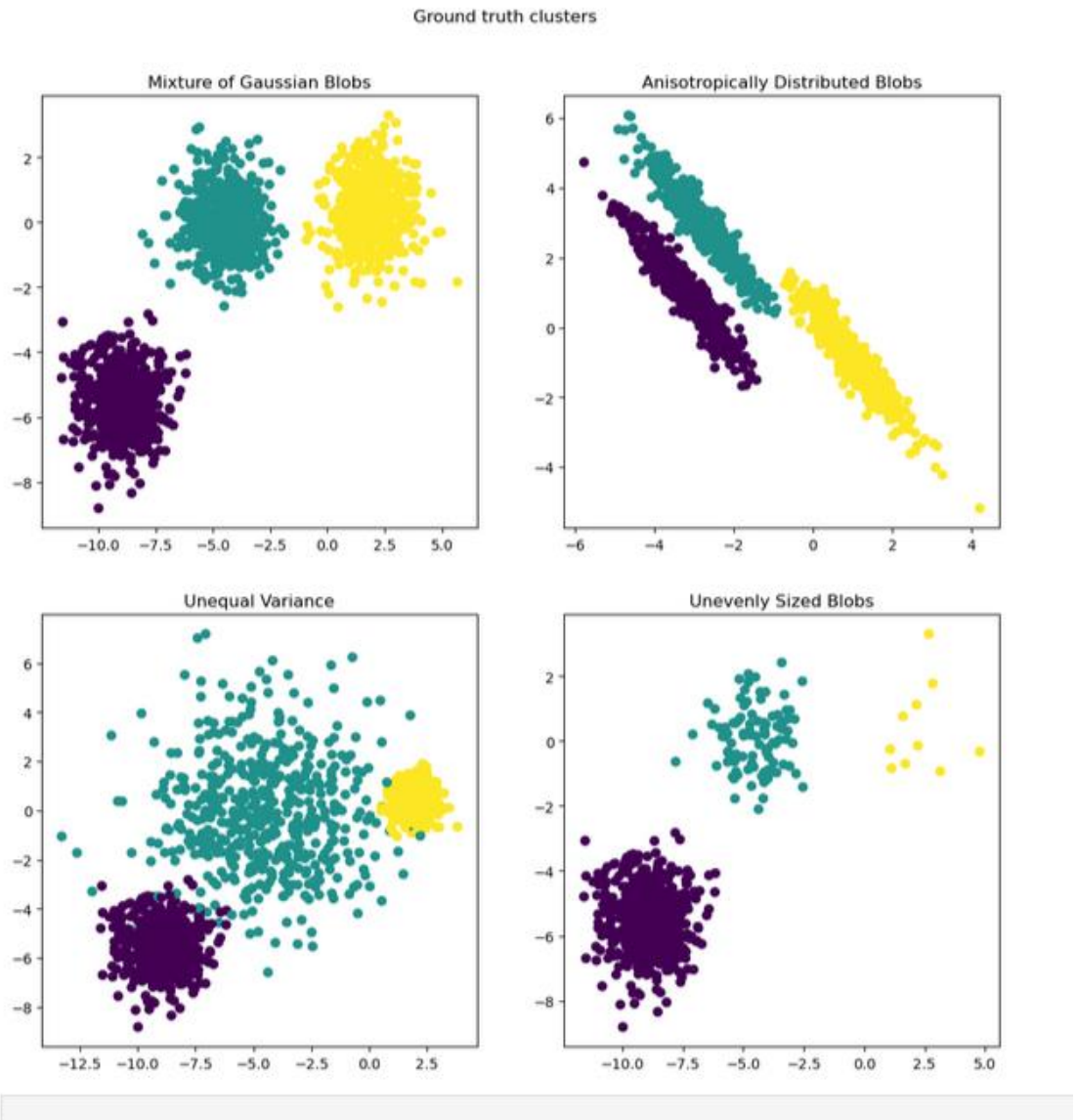
Code

[1]:
```
import numpy as np

from sklearn.datasets import make_blobs

n_samples = 1500
random_state = 170
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]

X, y = make_blobs(n_samples=n_samples, random_state=random_state)
X_aniso = np.dot(X, transformation)  # Anisotropic blobs
X_varied, y_varied = make_blobs(
    n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=random_state
)  # Unequal variance
X_filtered = np.vstack(
    (X[y == 0][:500], X[y == 1][:100], X[y == 2][:10])
)  # Unevenly sized blobs
y_filtered = [0] * 500 + [1] * 100 + [2] * 10
```

Este código dibuja cuatro gráficos que muestran diferentes tipos de agrupaciones de puntos. Utiliza `plt.subplots` para crear una figura con cuatro secciones. En cada una se dibujan los puntos de diferentes formas: en el primero, se muestran los blobs originales; en el segundo, se ven los blobs distorsionados; en el tercero, los blobs tienen diferentes tamaños de dispersión; y en el último, los blobs tienen tamaños muy desiguales. Al final, se agrega un título general que indica que estos gráficos representan los grupos originales, y luego se muestra la figura completa.



Finalmente se muestran cuatro gráficos diferentes que representan diferentes distribuciones de puntos. El primer gráfico muestra los blobs originales generados por la función `make_blobs`, con agrupaciones bien definidas y formas circulares. El segundo gráfico presenta blobs distorsionados, estirados en una dirección específica debido a la aplicación de una transformación anisotrópica. En el tercer gráfico, los blobs tienen diferentes niveles de dispersión, algunos más compactos y otros más dispersos. El último gráfico muestra blobs de tamaños desiguales, con algunos grupos mucho más grandes que otros. Estos gráficos permiten visualizar cómo diferentes ajustes en los datos afectan su distribución y agrupamiento.



IMPLEMENTACION 3: Cuantificación de color mediante K-Means

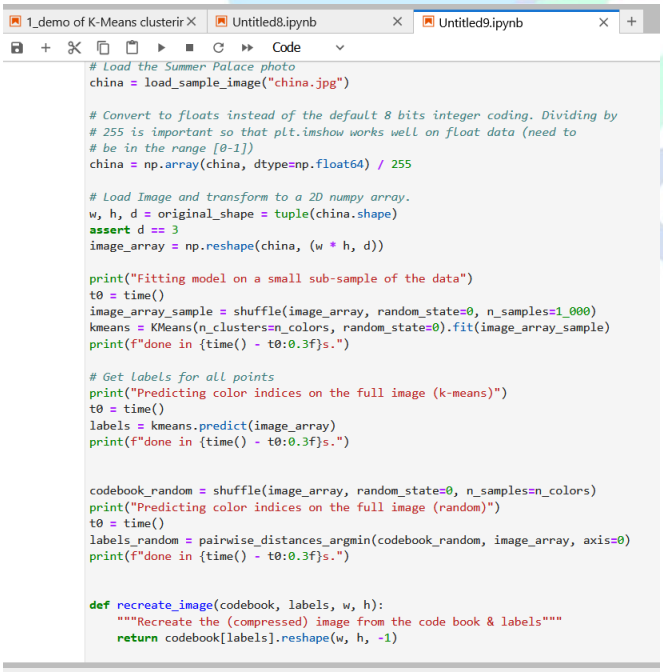
Este código toma una imagen con muchos colores y reduce su paleta a solo 64 colores utilizando el algoritmo K-Means. Luego, compara el resultado con una versión donde los colores son seleccionados de manera aleatoria. Te lo explico de forma sencilla:

Primero, se carga una imagen de muestra (en este caso, "china.jpg"), que es una foto del Summer Palace. La imagen se convierte a un formato de números flotantes (entre 0 y 1) para que las funciones de matplotlib puedan manejarla correctamente. Luego, se transforma la imagen en un arreglo 2D donde cada fila representa un píxel, y cada columna representa los valores de color (RGB).

Después, se toma una pequeña muestra de los datos de color para entrenar el modelo K-Means. El K-Means agrupa los colores en 64 clusters, y cada uno de esos clusters representa un color "promedio" en la imagen.

El código entonces utiliza K-Means para predecir los colores de toda la imagen basándose en estos 64 colores agrupados. También genera una versión de la imagen donde los colores son seleccionados de manera aleatoria para compararla con el resultado de K-Means.

Se muestran las 3 versiones de la foto, en la cual se puede comprimir la imagen, teniendo la óptima predicción de colores.




```python
# Load the Summer Palace photo
china = load_sample_image("china.jpg")

# Convert to floats instead of the default 8 bits integer coding. Dividing by
# 255 is important so that plt.imshow works well on float data (need to
# be in the range [0-1])
china = np.array(china, dtype=np.float64) / 255

# Load Image and transform to a 2D numpy array.
w, h, d = original_shape = tuple(china.shape)
assert d == 3
image_array = np.reshape(china, (w * h, d))

print("Fitting model on a small sub-sample of the data")
t0 = time()
image_array_sample = shuffle(image_array, random_state=0, n_samples=1_000)
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(image_array_sample)
print(f"done in {time() - t0:0.3f}s.")

# Get labels for all points
print("Predicting color indices on the full image (k-means)")
t0 = time()
labels = kmeans.predict(image_array)
print(f"done in {time() - t0:0.3f}s.")


codebook_random = shuffle(image_array, random_state=0, n_samples=n_colors)
print("Predicting color indices on the full image (random)")
t0 = time()
labels_random = pairwise_distances_argmin(codebook_random, image_array, axis=0)
print(f"done in {time() - t0:0.3f}s.")


def recreate_image(codebook, labels, w, h):
    """Recreate the (compressed) image from the code book & labels"""
    return codebook[labels].reshape(w, h, -1)
```

Original image (96,615 colors)

Quantized image (64 colors, K-Means)

CONCLUSION

A lo largo de este trabajo, hemos visto cómo el algoritmo K-Means se puede aplicar en diferentes situaciones, desde agrupar datos sintéticos hasta reducir los colores de una imagen. K-Means es una herramienta versátil, que permite encontrar patrones o grupos en los datos de manera eficiente. Vimos cómo podemos generar diferentes tipos de datos y cómo el algoritmo se adapta bien a ellos, permitiéndonos visualizarlos de forma clara.

También aprendimos que, aunque existen otras formas de reducir los colores de una imagen, como elegirlos de manera aleatoria, K-Means logra un mejor resultado al agrupar los colores de forma más precisa. Esto hace que la imagen mantenga una buena calidad visual aun cuando se limita a una paleta de colores más pequeña.

K-Means es un algoritmo muy útil, no solo para análisis de datos, sino también para tareas como la compresión de imágenes, siempre logrando buenos resultados de manera rápida y sencilla.

REFERENCIAS:

KMeans. (n.d.). Scikit-Learn. Retrieved September 18, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html