Ingeniería en Inteligencia Artificial,
Machine Learning
Sem: 2025-1, 5BM1, Práctica 1, Fecha: 10 de Septiembre 2024

Ejercicio de Laboratorio 2: K-Means 2

Machine Learning

Grupo: 5BM1

Profesor: Andrés Floriano García

Integrantes:
Juan Manuel Alvarado Sandoval
Alexander Iain Crombie Esquinca
Herrera Saavedra Jorge Luis
Quiñones Mayorga Rodrigo

Contents

1 Introducción al algoritmo k-means

El algoritmo *k-means* es un método de *clustering* ampliamente utilizado en el campo de la inteligencia artificial y el aprendizaje automático. Introducido por primera vez por Stuart Lloyd en 1957 y popularizado por MacQueen en 1967, su objetivo es particionar un conjunto de datos en k grupos o clusters, minimizando la varianza dentro de cada grupo. El algoritmo funciona de manera iterativa, asignando cada punto de datos al centroide más cercano y luego recalculando los centroides de cada cluster, repitiendo el proceso hasta que las asignaciones no cambien. *k-means* se ha convertido en una técnica fundamental para la agrupación no supervisada de datos debido a su simplicidad y eficiencia computacional.

2 Síntesis del algoritmo K-Means

El algoritmo *K-Means* es una técnica de *clustering* no supervisado cuyo objetivo es particionar un conjunto de datos $X = \{x_1, x_2, \ldots, x_n\}$ en k grupos o clusters $C = \{C_1, C_2, \ldots, C_k\}$, donde cada C_i contiene los puntos de datos más cercanos a su centroide correspondiente. El algoritmo minimiza la función de coste, que es la suma de las distancias cuadradas entre cada punto y su centroide asignado. Matemáticamente, el objetivo es minimizar la siguiente función de error:

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

donde μ_i es el centroide del cluster C_i y $\|x_j - \mu_i\|$ es la distancia euclidiana entre el punto de datos x_j y su centroide. El algoritmo itera en dos pasos principales: primero, asigna cada punto x_j al cluster cuyo centroide esté más cercano:

$$C_i = \{x_j : \|x_j - \mu_i\|^2 \leq \|x_j - \mu_l\|^2, \forall l = 1, 2, \ldots, k\}$$

Luego, recalcula los centroides actualizando μ_i como el promedio de los puntos asignados al cluster C_i:

$$\mu_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$$

Este proceso se repite hasta que las asignaciones de los puntos ya no cambien. Aunque el algoritmo es eficiente y fácil de implementar, puede ser sensible a la elección inicial de los centroides y al número k, que debe ser definido previamente.

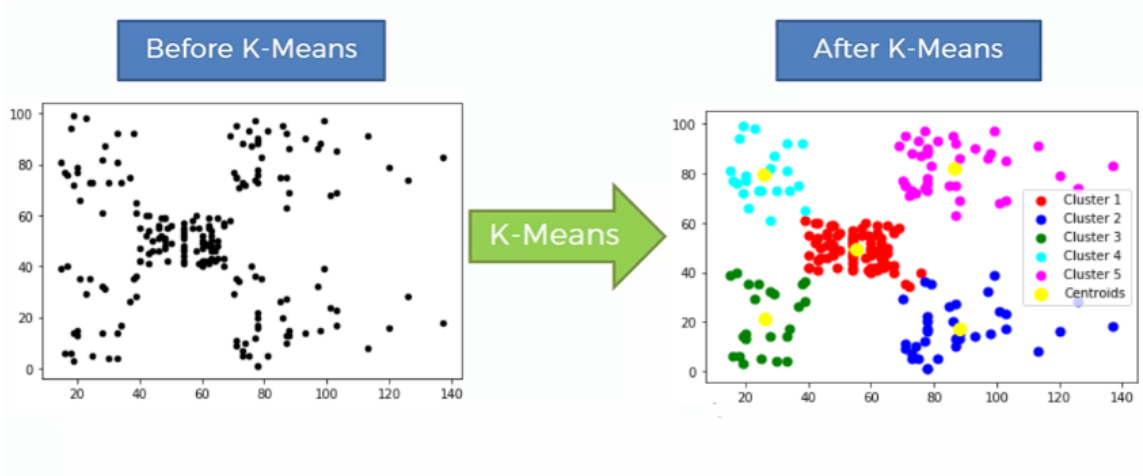


Figure 1: Ejemplo gráfico de agrupación del algoritmo K-Means.

3 Evidencia de la Práctica (pantallazos)

3.1 Ejemplo 1

Recuperado de

https://scikit-learn.org/stable/auto_examples/text/plot_document_clustering.html

In [1]:
```
# Práctica 2

# Clustering text documents using k-means

import numpy as np

from sklearn.datasets import fetch_20newsgroups

#Lista de articulos de texto
categories= [
    "alt.atheism",
    "talk.religion.misc",
    "comp.graphics",
    "sci.space",
]
```

In [3]:
```
#limpiar la info del dataset
dataset= fetch_20newsgroups(
    remove=("headers","footers","quotes"),
    subset="all",
    categories=categories,
    shuffle=True,
    random_state=42,
)


#se recupera el número de documentos y categorias del dataset
labels= dataset.target
unique_labels, category_sizes= np.unique(labels, return_counts= True)
true_k= unique_labels.shape[0]
print(f"{len(dataset.data)} documentos -{true_k} categorias")
```

```
3387 documentos -4 categorias
```

Figure 2: Carga del dataset y validación de los documentos.

In [31]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(
    max_df=0.5,
    min_df=5,
    stop_words="english",
)
t0 = time()
X_tfidf = vectorizer.fit_transform(dataset.data)

print(f"vectorizacion hecha en: {time() - t0:.3f} s")
print(f"muestras: {X_tfidf.shape[0]}, n_features: {X_tfidf.shape[1]}")
```

```
vectorizacion hecha en: 0.614 s
muestras: 3387, n_features: 7929
```

In [18]:
```
print(f"{X_tfidf.nnz / np.prod(X_tfidf.shape):.3f}")
```

```
0.007
```

In [32]:
```
from sklearn.cluster import KMeans

for seed in range(5):
    kmeans = KMeans(
        n_clusters=true_k,
        max_iter=100,
        n_init=1,
        random_state=seed,
    ).fit(X_tfidf)
    cluster_ids, cluster_sizes = np.unique(kmeans.labels_, return_counts=True)
    print(f"Numero de elementos asignados a cada cluster: {cluster_sizes}")
print()
print(
    "Número verdadero de Documentos en cada categoria según las etiquetas (labels) de clases: "
    f"{category_sizes}"
)
```

```
Numero de elementos asignados a cada cluster: [ 481  675 1785  446]
Numero de elementos asignados a cada cluster: [1689  638  480  580]
Numero de elementos asignados a cada cluster: [   1    1    1 3384]
Numero de elementos asignados a cada cluster: [1887  311  332  857]
Numero de elementos asignados a cada cluster: [ 291  673 1771  652]
```

Figure 3: Vectorización y clustering de elementos.

```
clustering hecho en:  0.11 ± 0.03 s
Homogeneity: 0.304 ± 0.066
Completeness: 0.333 ± 0.067
V-measure: 0.317 ± 0.066
Adjusted Rand-Index: 0.276 ± 0.047
Silhouette Coefficient: 0.027 ± 0.004
```

```
In [24]:
original_space_centroids = lsa[0].inverse_transform(kmeans.cluster_centers_)
order_centroids = original_space_centroids.argsort()[:, ::-1]
terms = vectorizer.get_feature_names_out()

for i in range(true_k):
    print(f"Cluster {i}: ", end="")
    for ind in order_centroids[i, :10]:
        print(f'{terms[ind]} ', end="")
    print()
```

```
Cluster 0: space nasa shuttle station sci launch program like think just
Cluster 1: thanks graphics image know files edu file does program looking
Cluster 2: god people think don say just jesus religion know believe
Cluster 3: just like orbit earth time moon years launch think mission
```

Figure 4: Reducción de dimensionalidad LSA, métricas del clustering con MiniBatchKMeans y por último imprime las palabras más representativas de cada cluster.

```
In [28]:
import matplotlib.pyplot as plt
import pandas as pd

fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(16, 6), sharey=True)

df = pd.DataFrame(evaluations[::-1]).set_index("estimator")
df_std = pd.DataFrame(evaluations_std[::-1]).set_index("estimator")

df.drop(
    ["train_time"],
    axis="columns",
).plot.barh(ax=ax0, xerr=df_std)
ax0.set_xlabel("Clustering scores")
ax0.set_ylabel("")

df["train_time"].plot.barh(ax=ax1, xerr=df_std["train_time"])
ax1.set_xlabel("Clustering time (s)")
plt.tight_layout()
```

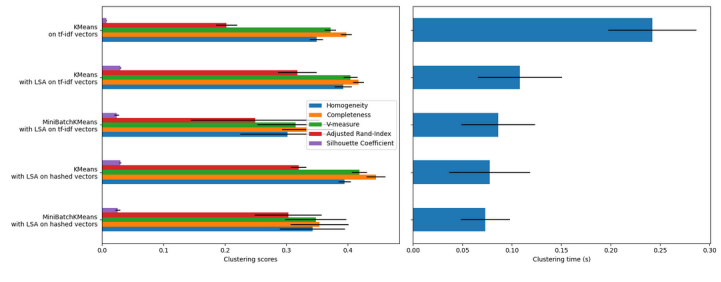


Figure 5: Gráficos para las métricas comparativas de rendimiento.

3.2 Ejemplo 2

Figure 6: Contraste imagen original con la segmentada.

Figure 7: Contraste asignación media vs colores random.

3.3 Ejemplo 3

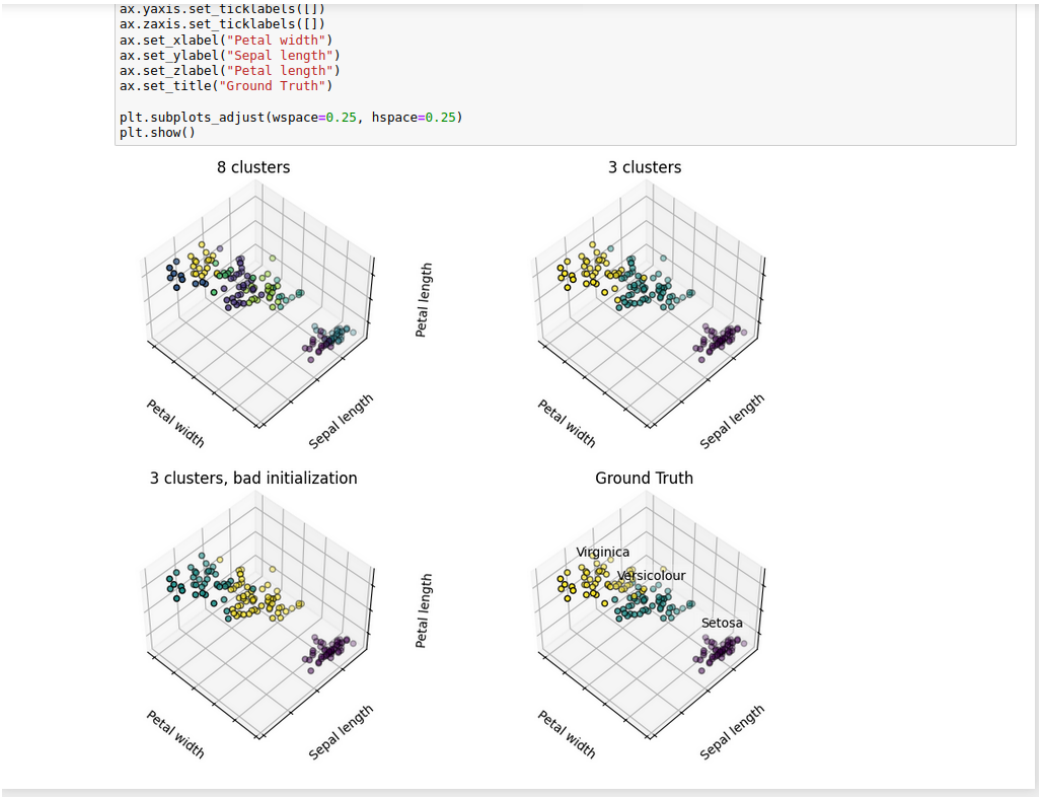


Figure 8: Este gráfico está comparando diferentes resultados de clustering, evaluando cómo afectan la cantidad de clusters, la inicialización de los centroides, y mostrando cómo los datos se agruparían si tuviéramos el conocimiento previo sobre la verdadera distribución (ground truth)..

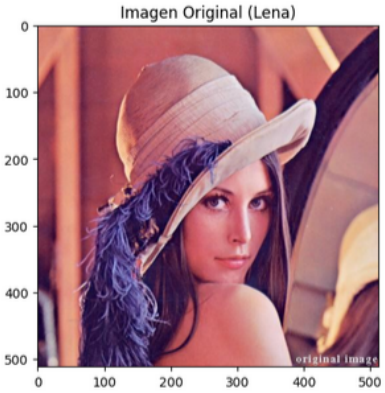
3.4 Ejemplo 4 - Extra

```
plt.hist(G.ravel(), bins=256, color='green', alpha=0.7, label='Verde', histtype='step')

# Histograma del canal azul
plt.hist(B.ravel(), bins=256, color='blue', alpha=0.7, label='Azul', histtype='step')

# Configuración del gráfico
plt.title('Histogramas de los Canales de Color')
plt.xlabel('Valor del píxel')
plt.ylabel('Frecuencia')
plt.xlim([0, 256])
plt.legend(loc='upper right')

# Mostrar los histogramas
plt.show
```




```
Out[47]: <function matplotlib.pyplot.show(close=None, block=None)>
```

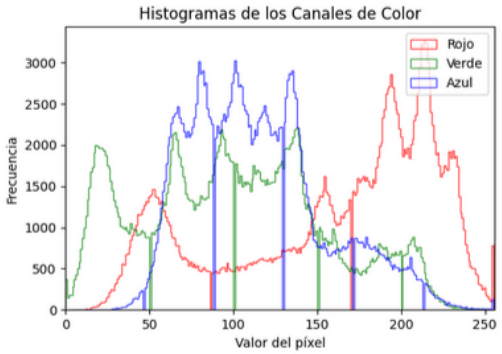


Figure 9: Imagen original procesada con histograma de planos.

In [49]:
```
#Aplicamos KMeans

#Numero de Clústers
k=5

#Aplicamos el algoritmo K-Means
k_means= KMeans(n_clusters=k, random_state=42)
k_means.fit(image_new) #Método fit para entrenar o ajustar el modelo

#Obtener las etiquetas para cada pixel (cluster de pertenencia)
imagen_segmentada= k_means.labels_
imagen_segmentada= imagen_segmentada.reshape(filas,columnas)
```

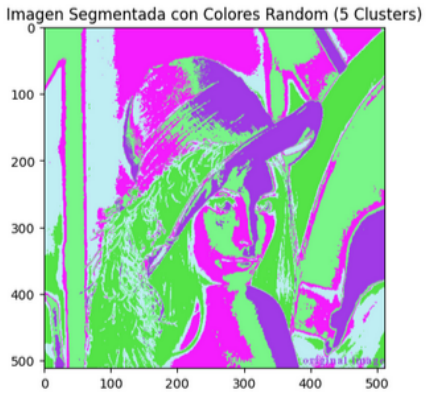
In [51]:
```
# Generar colores exóticos aleatorios para cada cluster
def generar_colores_exoticos(n_clusters):
    return np.array([[random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)] for _ in range(n_cluste

colores_exoticos_cluster = generar_colores_exoticos(k)

# Crear una nueva imagen donde cada pixel se asocie al color de su cluster
imagen_segmentada_coloreada = np.zeros_like(image_rgb)

for i in range(filas):
    for j in range(columnas):
        imagen_segmentada_coloreada[i, j] = colores_exoticos_cluster[imagen_segmentada[i, j]]

# Mostrar la imagen segmentada con colores exóticos
plt.imshow(imagen_segmentada_coloreada)
plt.title(f'Imagen Segmentada con Colores Random ({k} Clusters)')
plt.axis('on')
plt.show()
```



Otra imagen segmentada con más centroides

In [57]:
```
# Aplicamos KMeans

# Número de Clústers
```

Figure 10: Imagen segmentada con menos centroides.

Otra imagen segmentada con más centroides

In [57]:
```
# Aplicamos KMeans

# Número de Clústers
n_clusters = 200

# Aplicamos el algoritmo K-Means
kmeans_model = KMeans(n_clusters=n_clusters, random_state=42)
kmeans_model.fit(image_new)  # image_data representa la imagen preprocesada

# Obtener las etiquetas para cada pixel (cluster de pertenencia)
image_segmented_labels = kmeans_model.labels_
image_segmented = image_segmented_labels.reshape(filas, columnas)

# Generar colores exóticos para cada clúster
exotic_colors = generar_colores_exoticos(n_clusters)  # Función para generar colores llamativos

# Crear una imagen coloreada basada en los clusters
image_colored_segmented = np.zeros_like(image_rgb)

# Asignar colores a cada píxel según el clúster
for i in range(filas):
    for j in range(columnas):
        image_colored_segmented[i, j] = exotic_colors[image_segmented[i, j]]

# Mostrar la imagen segmentada con colores llamativos
plt.imshow(image_colored_segmented)
plt.title(f'Imagen Segmentada con Colores Exóticos ({n_clusters} Clusters)')
plt.axis('on')
plt.show()
```

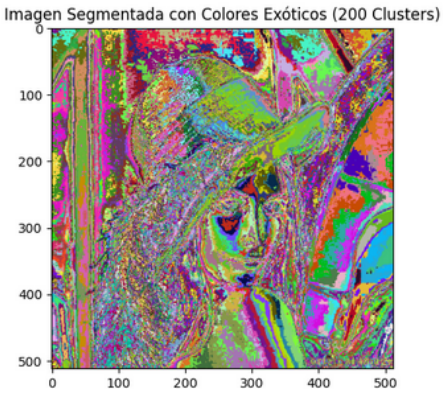


Figure 11: Imagen segmentada con más centroides.

4 Enlace al repositorio

Haz click para seguir el enlace

5 Referencias

- Scikit-learn developers. (2021). *sklearn.cluster.KMeans — scikit-learn 0.24.2 documentation*. Scikit-learn. Recuperado de: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- The Machine Learners. (2024). *K-means Clustering Algorithm*. Recuperado de: https://www.themachinelearners.com/k-means/