HOJA DE TAREA: HT-02

Crea programas con algoritmos de aprendizaje no supervisado

HOJA DE OPERACIONE:

- **HO-04** : DESCRIBE LOS TIPOS DE ALGORITMOS DE APRENDIZAJE NO SUPERVISADO
- **HO-05** : DEFINE DIFERENCIAS ENTRE ALGORITMOS DE CLASIFICACION Y AGRUPAMIENTO

OBJETIVOS:

Al finalizar la sesión, el estudiante describirá los tipos de algoritmos de aprendizaje no supervisado y definirá las diferencias entre algoritmos de clasificación y de agrupamiento, explicando sus características y aplicaciones en el análisis de datos.

Tipos de Algoritmos de Aprendizaje No Supervisado

A) Algoritmos de Agrupamiento (Clustering)

1. **K-means**:

   Este algoritmo agrupa datos en kkk grupos (clusters) basándose en la similitud. Inicialmente, selecciona kkk puntos aleatorios como centros de los clusters (centroides). Luego, asigna cada punto de datos al cluster cuyo centro es más cercano y recalcula los centroides. Este proceso se repite hasta que no hay cambios significativos en los clusters.

   K-means es un algoritmo que busca dividir un conjunto de datos en kkk grupos o clusters. Este algoritmo funciona de la siguiente manera:

   - Inicialización: Selecciona aleatoriamente kkk puntos como los centroides iniciales de los clusters.
   - Asignación: Cada punto del conjunto de datos se asigna al cluster cuyo centro (centroide) es más cercano.
   - Actualización: Se recalculan los centroides de cada cluster como el promedio de todos los puntos asignados a ese cluster.
   - Iteración: Los pasos de asignación y actualización se repiten hasta que los centroides ya no cambian significativamente o se alcanza un número máximo de iteraciones.

Ejercicio 1: K-means Clustering

Instrucciones:

1. Genera un conjunto de datos utilizando make_blobs.
2. Aplica el algoritmo K-means para agrupar los datos en 3 clusters.
3. Visualiza los resultados.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generar datos de ejemplo
X, y = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)

# Aplicar K-means
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Visualizar los resultados
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75)
plt.title('K-means Clustering')
plt.xlabel('Caracteristica 1')
plt.ylabel('Caracteristica 2')
plt.show()
```

Explicación del Código

1. **Importar las Librerías**:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

- numpy: Biblioteca para realizar cálculos numéricos en Python.
- matplotlib.pyplot: Biblioteca para crear gráficos en Python.
- make_blobs: Función de scikit-learn que genera conjuntos de datos sintéticos con características similares.
- KMeans: Clase del módulo sklearn.cluster que implementa el algoritmo K-means.

2. **Generar Datos de Ejemplo**:

```
X, y = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)
```

   - n_samples=300: Se generan 300 muestras.
   - centers=3: Se crean 3 centros (o clusters) en el espacio de datos.
   - cluster_std=0.60: Define la desviación estándar de los clusters. Un valor menor significa que los puntos estarán más cerca del centro.
   - random_state=0: Se utiliza para asegurar que los resultados sean reproducibles. Especifica una semilla aleatoria para la generación de datos.

   Este paso genera dos variables:
   - X: Contiene las coordenadas de los puntos en un espacio de 2 dimensiones.
   - y: Contiene las etiquetas de cluster (no se usa más adelante en el código).

3. **Aplicar K-means**:

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

   - KMeans(n_clusters=3): Crea un objeto KMeans configurado para encontrar 3 clusters.
   - kmeans.fit(X): Ajusta el modelo a los datos X. El algoritmo calcula los centroides iniciales y asigna los puntos a los clusters.
   - y_kmeans = kmeans.predict(X): Predice el cluster al que pertenece cada punto en X, devolviendo un array con las etiquetas de los clusters asignados.

4. **Visualizar los Resultados**:

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75)
plt.title('K-means Clustering')
plt.xlabel('Caracteristica 1')
plt.ylabel('Caracteristica 2')
plt.show()
```

   - plt.scatter(X[:, 0], X[:, 1], ...): Dibuja un gráfico de dispersión de los puntos en X, utilizando las etiquetas de cluster (y_kmeans) como colores.
     - c=y_kmeans: Los colores de los puntos se basan en los clusters asignados.
     - s=50: Tamaño de los puntos en el gráfico.
     - cmap='viridis': Mapa de colores que se utiliza para representar los clusters.

- centers = kmeans.cluster_centers_: Obtiene las coordenadas de los centroides de los clusters calculados.
- plt.scatter(centers[:, 0], centers[:, 1], ...): Dibuja los centroides en el gráfico.
  - c='red': Color de los centroides (rojo).
  - s=200: Tamaño de los centroides en el gráfico.
  - alpha=0.75: Opacidad de los puntos de los centroides.
- plt.title, plt.xlabel, plt.ylabel: Añade título y etiquetas a los ejes del gráfico.
- plt.show(): Muestra el gráfico final.

2. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**:

- Agrupa puntos basándose en la densidad de puntos en el espacio. Identifica regiones densas de puntos y las separa de áreas menos densas. También puede identificar puntos que no pertenecen a ningún grupo como "ruido".

  DBSCAN es un algoritmo de agrupamiento basado en la densidad que tiene la capacidad de encontrar grupos de puntos densos y separarlos de los puntos que no pertenecen a ningún grupo (ruido). Funciona así:

  1. **Parámetros**: Necesita dos parámetros:
     - eps: el radio de búsqueda para determinar la vecindad de un punto.
     - min_samples: el número mínimo de puntos requeridos para formar un cluster.
  2. **Agrupamiento**: Un punto se considera un núcleo si tiene al menos min_samples puntos en su vecindad (dentro del radio eps).
  3. **Expansión**: Los puntos nucleares forman un cluster, y se agregan puntos densamente conectados hasta que ya no hay más puntos en la vecindad.

Ejercicio 2: DBSCAN

Instrucciones:

1. Genera un conjunto de datos utilizando make_moons para crear una forma de luna, que es ideal para DBSCAN.
2. Aplica el algoritmo DBSCAN para identificar grupos.
3. Visualiza los resultados.

```
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

# Generar datos en forma de luna
X, _ = make_moons(n_samples=300, noise=0.1, random_state=0)

# Aplicar DBSCAN
dbscan = DBSCAN(eps=0.2, min_samples=5)
y_dbscan = dbscan.fit_predict(X)

# Visualizar los resultados
plt.scatter(X[:, 0], X[:, 1], c=y_dbscan, s=50, cmap='viridis')
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Explicación del Código

1. **Importar las Librerías**:

```
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
```

   - make_moons: Esta función de sklearn.datasets genera un conjunto de datos sintéticos que tiene la forma de dos lunas entrelazadas. Es útil para probar algoritmos de agrupamiento que pueden no funcionar bien con datos esféricos (como K-means).
   - DBSCAN: Clase del módulo sklearn.cluster que implementa el algoritmo DBSCAN para la detección de clusters basados en la densidad.

2. **Generar Datos en Forma de Luna**:

```
X, _ = make_moons(n_samples=300, noise=0.1, random_state=0)
```

   - n_samples=300: Se generan 300 muestras en total.
   - noise=0.1: Se añade ruido a los datos para hacerlos más realistas y desafiantes para el algoritmo de agrupamiento. Este parámetro controla la cantidad de dispersión de los puntos.
   - random_state=0: Asegura que los resultados sean reproducibles al establecer una semilla aleatoria.

El resultado de esta línea es un conjunto de puntos X que tienen la forma de dos lunas.

3. **Aplicar DBSCAN**:

```
dbscan = DBSCAN(eps=0.2, min_samples=5)
y_dbscan = dbscan.fit_predict(X)
```

   - DBSCAN(eps=0.2, min_samples=5): Crea un objeto DBSCAN configurado con:
     - eps=0.2: Distancia máxima entre dos puntos para que se consideren parte del mismo cluster. Un valor menor significa que se requieren puntos más cercanos para formar un cluster.
     - min_samples=5: Número mínimo de puntos requeridos para formar un cluster. Este parámetro ayuda a definir la densidad mínima que se necesita para que un grupo de puntos sea considerado un cluster.

   - y_dbscan = dbscan.fit_predict(X): Aplica el algoritmo DBSCAN a los datos X y devuelve un array con las etiquetas de los clusters asignados a cada punto.
     - Los puntos asignados a un cluster específico tendrán un número entero como etiqueta.
     - Los puntos que no pertenecen a ningún cluster (considerados "ruido") serán etiquetados como -1.

4. **Visualizar los Resultados**:

```
plt.scatter(X[:, 0], X[:, 1], c=y_dbscan, s=50, cmap='viridis')
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

   - plt.scatter(X[:, 0], X[:, 1], ...): Dibuja un gráfico de dispersión de los puntos en X, utilizando las etiquetas de cluster (y_dbscan) como colores.
     - c=y_dbscan: Los colores de los puntos se basan en los clusters asignados.
     - s=50: Tamaño de los puntos en el gráfico.
     - cmap='viridis': Mapa de colores que se utiliza para representar los clusters.
   - plt.title, plt.xlabel, plt.ylabel: Añade título y etiquetas a los ejes del gráfico.
   - plt.show(): Muestra el gráfico final.

3. **Hierarchical Clustering**:
    - Crea una jerarquía de grupos mediante la creación de un árbol de decisiones llamado dendrograma. Puede ser aglomerativo (comienza con cada punto como un cluster y los combina) o divisivo (comienza con todos los puntos en un solo cluster y los divide).
    - Este método de agrupamiento crea una jerarquía de clusters, generalmente visualizada como un dendrograma. Existen dos enfoques principales:
        1. Aglomerativo: Comienza considerando cada punto como un cluster individual y combina iterativamente los más similares.
        2. Divisivo: Comienza con un único cluster que contiene todos los puntos y los divide en clusters más pequeños.
    -
4. **Agglomerative Clustering**:
    - Es un tipo de clustering jerárquico que une grupos similares de manera ascendente. Comienza con cada punto como su propio cluster y combina iterativamente los más similares hasta que se alcanza un número predefinido de clusters.

Ejercicio 3: Agglomerative Clustering

Instrucciones:

1. Utiliza el conjunto de datos generado con make_blobs.
2. Aplica el algoritmo de agrupamiento aglomerativo.
3. Visualiza el dendrograma para ver la jerarquía de agrupamientos.

```
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Generar datos
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)

# Crear el modelo de agrupamiento aglomerativo
agglo = AgglomerativeClustering(n_clusters=3)
y_agglo = agglo.fit_predict(X)

# Visualizar los resultados
plt.scatter(X[:, 0], X[:, 1], c=y_agglo, s=50, cmap='viridis')
plt.title('Agglomerative Clustering')
plt.xlabel('Feature 1')
```

```
plt.ylabel('Feature 2')
plt.show()

# Crear el dendrograma
linked = linkage(X, 'ward')
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Dendrogram')
plt.xlabel('Index')
plt.ylabel('Distance')
plt.show()
```

Explicación del Código

1. **Importar las Librerías**:

```
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
```

   - make_blobs: Esta función genera un conjunto de datos sintéticos, ideal para probar algoritmos de agrupamiento.
   - AgglomerativeClustering: Clase de sklearn.cluster que implementa el algoritmo de agrupamiento aglomerativo.
   - dendrogram y linkage: Funciones de scipy.cluster.hierarchy que se utilizan para crear y visualizar un dendrograma, que muestra la relación jerárquica entre los clusters.

2. **Generar Datos**:

```
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)
```

   - n_samples=300: Se generan 300 muestras.
   - centers=3: Se especifica que se crearán 3 grupos o centros de clusters.
   - cluster_std=0.60: Controla la dispersión de los puntos en torno a los centros de los clusters.
   - random_state=0: Asegura que los resultados sean reproducibles.

   El resultado es un conjunto de puntos X que forman tres grupos.

3. **Crear el Modelo de Agrupamiento Aglomerativo**:

```
agglo = AgglomerativeClustering(n_clusters=3)
y_agglo = agglo.fit_predict(X)
```

   - AgglomerativeClustering(n_clusters=3): Se crea un objeto de agrupamiento aglomerativo que buscará formar 3 clusters.
   - y_agglo = agglo.fit_predict(X): Aplica el modelo a los datos X y devuelve un array con las etiquetas de los clusters asignadas a cada punto.

4. **Visualizar los Resultados**:

```
plt.scatter(X[:, 0], X[:, 1], c=y_agglo, s=50, cmap='viridis')
plt.title('Agglomerative Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

   - plt.scatter(X[:, 0], X[:, 1], ...): Dibuja un gráfico de dispersión de los puntos en X, utilizando las etiquetas de cluster (y_agglo) como colores.
     - c=y_agglo: Los colores de los puntos se basan en los clusters asignados.
     - s=50: Tamaño de los puntos en el gráfico.
     - cmap='viridis': Mapa de colores que se utiliza para representar los clusters.
   - Se añaden el título y las etiquetas a los ejes del gráfico y finalmente se muestra el gráfico.

5. **Crear el Dendrograma**:

```
linked = linkage(X, 'ward')
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Dendrogram')
plt.xlabel('Index')
plt.ylabel('Distance')
plt.show()
```

   - linked = linkage(X, 'ward'): Realiza el enlace jerárquico de los datos X utilizando el método de Ward, que busca minimizar la varianza dentro de los clusters.
   - plt.figure(figsize=(10, 7)): Establece el tamaño de la figura para el dendrograma.
   - dendrogram(linked): Dibuja el dendrograma, que muestra cómo se combinan los clusters a diferentes niveles de similitud.
   - Se añaden el título y las etiquetas a los ejes y finalmente se muestra el dendrograma.

B) Algoritmos de Reducción de Dimensionalidad

1. **PCA (Principal Component Analysis)**:
    - Reduce la dimensionalidad del conjunto de datos al proyectar los datos en un nuevo espacio, donde las nuevas dimensiones (componentes principales) capturan la mayor varianza posible. PCA es útil para visualizar datos de alta dimensión.
    - PCA es un método utilizado para reducir la dimensionalidad de un conjunto de datos mientras se conserva la mayor cantidad de varianza posible. Funciona transformando las características originales en un nuevo conjunto de variables no correlacionadas (componentes principales). Los pasos básicos son:
        1. Estandarización: Normalizar los datos para que cada característica tenga una media de cero y una desviación estándar de uno.
        2. Cálculo de la Matriz de Covarianza: Obtener la matriz de covarianza para identificar cómo varían las características en relación unas con otras.
        3. Cálculo de los Vectores y Valores Propios: Determinar los vectores propios (direcciones de máxima varianza) y los valores propios (magnitudes de varianza) a partir de la matriz de covarianza.
        4. Selección de Componentes Principales: Elegir los primeros kkk vectores propios en función de la varianza que representan.

Ejercicio 1: PCA (Análisis de Componentes Principales)

Instrucciones:

- Utiliza el conjunto de datos Iris.
- Aplica PCA para reducir la dimensionalidad a 2 dimensiones.
- Visualiza los resultados.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

# Cargar el conjunto de datos Iris
iris = load_iris()
X = iris.data
y = iris.target
```

 # Aplicar PCA

```
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```

 # Visualizar los resultados

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis')
plt.title('PCA - Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(*scatter.legend_elements(), title='Classes')
plt.show()
```

Explicación del Código

1. **Importar las Librerías**:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

   - numpy: Biblioteca utilizada para trabajar con arreglos y operaciones matemáticas.
   - matplotlib.pyplot: Biblioteca utilizada para crear gráficos y visualizaciones.
   - load_iris: Función de sklearn.datasets que carga el conjunto de datos Iris, un clásico en aprendizaje automático.
   - PCA: Clase de sklearn.decomposition que implementa el Análisis de Componentes Principales.

2. **Cargar el Conjunto de Datos Iris**:

```
iris = load_iris()
X = iris.data
y = iris.target
```

   - iris = load_iris(): Carga el conjunto de datos Iris, que contiene características de flores de iris y sus respectivas etiquetas de clase.
   - X = iris.data: X contiene las características (longitudes y anchos de los sépalos y pétalos).
   - y = iris.target: y contiene las etiquetas de las clases (tipos de iris).

3. **Aplicar PCA**:

```
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```

   - pca = PCA(n_components=2): Se crea un objeto PCA que reducirá la dimensionalidad a 2 componentes principales.
   - X_reduced = pca.fit_transform(X): Se aplica PCA al conjunto de datos X, transformando los datos a un espacio de menor dimensión (2D) y almacenando los resultados en X_reduced.

4. **Visualizar los Resultados**:

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis')
plt.title('PCA - Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(*scatter.legend_elements(), title='Classes')
plt.show()
```

   - plt.figure(figsize=(8, 6)): Establece el tamaño de la figura para la visualización.
   - scatter = plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis'): Dibuja un gráfico de dispersión utilizando los dos componentes principales. Aquí:
     - X_reduced[:, 0]: Las coordenadas del primer componente principal.
     - X_reduced[:, 1]: Las coordenadas del segundo componente principal.
     - c=y: Los colores de los puntos se basan en las clases originales.
     - cmap='viridis': Especifica el mapa de colores para la visualización.
   - Se añaden un título y etiquetas a los ejes, y la leyenda se genera automáticamente en base a los colores de las clases. Finalmente, se muestra el gráfico.

2. **t-SNE (t-distributed Stochastic Neighbor Embedding)**:

   - Una técnica de reducción de dimensionalidad especialmente buena para la visualización de datos de alta dimensión. t-SNE preserva la estructura local de los datos, haciendo que puntos similares estén cerca en el espacio reducido.

     1. t-SNE es un algoritmo de reducción de dimensionalidad que es particularmente útil para la visualización de datos de alta dimensión. Funciona transformando las similitudes de los datos en alta dimensión en distancias en un espacio de menor dimensión (generalmente 2D o 3D). Los pasos son:

2. Cálculo de Probabilidades de Similitud: Calcula las probabilidades de que los puntos sean vecinos en alta dimensión.
3. Cálculo de Probabilidades en el Espacio de Menor Dimensión: Crea una representación en un espacio de menor dimensión que preserve esas relaciones de vecindad.
4. Minimización de la Divergencia Kullback-Leibler: Ajusta la representación de manera que las distribuciones de probabilidad en ambos espacios sean lo más similares posible.

Ejercicio 2: t-SNE

Instrucciones:

1. Utiliza el conjunto de datos digits de sklearn.
2. Aplica t-SNE para reducir la dimensionalidad a 2 dimensiones.
3. Visualiza los resultados.

```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE

# Cargar el conjunto de datos de dígitos
digits = load_digits()
X_digits = digits.data
y_digits = digits.target

# Aplicar t-SNE
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X_digits)

# Visualizar los resultados
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits, cmap='viridis', s=20)
plt.title('t-SNE - Digits Dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(scatter)
plt.show()
```

Explicación del Código

1. **Importar las Librerías**:

```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE
```

   - load_digits: Función de sklearn.datasets que carga el conjunto de datos de dígitos manuscritos.
   - TSNE: Clase de sklearn.manifold que implementa el algoritmo t-SNE, utilizado para la reducción de dimensionalidad y la visualización de datos en espacios de baja dimensión.

2. **Cargar el Conjunto de Datos de Dígitos**:

```
digits = load_digits()
X_digits = digits.data
y_digits = digits.target
```

   - digits = load_digits(): Carga el conjunto de datos que contiene imágenes de dígitos manuscritos (0-9).
   - X_digits = digits.data: X_digits contiene las características (los píxeles de las imágenes de 8x8).
   - y_digits = digits.target: y_digits contiene las etiquetas de las clases (los dígitos reales que representan).

3. **Aplicar t-SNE**:

```
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X_digits)
```

   - tsne = TSNE(n_components=2, random_state=0): Se crea un objeto t-SNE que reducirá la dimensionalidad a 2 componentes principales. El parámetro random_state=0 se establece para garantizar la reproducibilidad del resultado.
   - X_tsne = tsne.fit_transform(X_digits): Se aplica t-SNE al conjunto de datos X_digits, transformando los datos a un espacio de menor dimensión (2D) y almacenando los resultados en X_tsne.

4. **Visualizar los Resultados**:

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits, cmap='viridis', s=20)
plt.title('t-SNE - Digits Dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(scatter)
plt.show()
```

- plt.figure(figsize=(8, 6)): Establece el tamaño de la figura para la visualización.
- scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits, cmap='viridis', s=20): Dibuja un gráfico de dispersión utilizando los dos componentes reducidos. Aquí:
  - X_tsne[:, 0]: Las coordenadas del primer componente t-SNE.
  - X_tsne[:, 1]: Las coordenadas del segundo componente t-SNE.
  - c=y_digits: Los colores de los puntos se basan en las etiquetas de los dígitos.
  - cmap='viridis': Especifica el mapa de colores para la visualización.
  - s=20: Establece el tamaño de los puntos en el gráfico.
- Se añaden un título y etiquetas a los ejes. Además, se incluye una barra de color (plt.colorbar(scatter)) que muestra la relación entre los colores y las clases (dígitos) representadas. Finalmente, se muestra el gráfico.

3. **UMAP (Uniform Manifold Approximation and Projection)**:

- Un algoritmo de reducción de dimensionalidad que se basa en la topología. UMAP es similar a t-SNE, pero generalmente más rápido y preserva mejor la estructura global de los datos.
- UMAP es un algoritmo de reducción de dimensionalidad que, al igual que t-SNE, es útil para la visualización, pero es más rápido y escalable a conjuntos de datos más grandes. Utiliza una representación matemática basada en la teoría de la topología. Los pasos son:
  1. Construcción de un Grafo: Crea un grafo en el que los puntos están conectados por sus vecinos más cercanos.
  2. Optimización de la Embedding: Minimiza la diferencia entre la representación en alta dimensión y la representación en baja dimensión para mantener la estructura del grafo.

Ejercicio 3: UMAP

Instrucciones:

1. Instala la librería umap-learn si no está instalada.
2. Utiliza el conjunto de datos wine de sklearn.
3. Aplica UMAP para reducir la dimensionalidad a 2 dimensiones.
4. Visualiza los resultados.

```
!pip install umap-learn

import umap
from sklearn.datasets import load_wine

# Cargar el conjunto de datos de vino
wine = load_wine()
X_wine = wine.data
y_wine = wine.target

# Aplicar UMAP
umap_model = umap.UMAP(n_components=2, random_state=42)
X_umap = umap_model.fit_transform(X_wine)

# Visualizar los resultados
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y_wine, cmap='viridis', s=50)
plt.title('UMAP - Wine Dataset')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.colorbar(scatter)
plt.show()
```

Explicación del Código

1. **Instalar la Librería UMAP**:

```
!pip install umap-learn
```

UMAP es una biblioteca de reducción de dimensionalidad que debe instalarse antes de su uso. Este comando instala el paquete umap-learn, que es necesario para aplicar el modelo.

2. **Importar las Librerías**:

```
import umap
from sklearn.datasets import load_wine
```

   - umap: Es la biblioteca que contiene el modelo UMAP para realizar la reducción de dimensionalidad.
   - load_wine: Función de sklearn.datasets que carga el conjunto de datos de vino.

3. **Cargar el Conjunto de Datos de Vino**:

```
wine = load_wine()
X_wine = wine.data
y_wine = wine.target
```

   - wine = load_wine(): Carga el conjunto de datos de vino, que contiene información química sobre diferentes variedades de vino.
   - X_wine = wine.data: Asigna las características del conjunto de datos (variables químicas) a X_wine.
   - y_wine = wine.target: Asigna las etiquetas (la clase de vino) a y_wine.

4. **Aplicar UMAP**:

```
umap_model = umap.UMAP(n_components=2, random_state=42)
X_umap = umap_model.fit_transform(X_wine)
```

   - umap_model = umap.UMAP(n_components=2, random_state=42): Se crea un objeto UMAP que reducirá la dimensionalidad a 2 componentes principales. El parámetro random_state=42 asegura la reproducibilidad de los resultados.
   - X_umap = umap_model.fit_transform(X_wine): Se aplica el modelo UMAP al conjunto de datos X_wine, generando una versión en 2D (2 componentes) de los datos, que se guarda en X_umap.

5. **Visualizar los Resultados**:

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y_wine, cmap='viridis', s=50)
plt.title('UMAP - Wine Dataset')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.colorbar(scatter)
plt.show()
```

- plt.figure(figsize=(8, 6)): Configura el tamaño de la figura para la visualización.
- scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y_wine, cmap='viridis', s=50): Dibuja un gráfico de dispersión usando los componentes 2D generados por UMAP:
  - X_umap[:, 0]: Coordenadas del primer componente de UMAP.
  - X_umap[:, 1]: Coordenadas del segundo componente de UMAP.
  - c=y_wine: Colorea los puntos basándose en la clase de vino.
  - cmap='viridis': Usa el mapa de colores 'viridis' para diferenciar las clases.
  - s=50: Ajusta el tamaño de los puntos en el gráfico.
- Se agregan un título y etiquetas de los ejes, y plt.colorbar(scatter) añade una barra de color que indica la relación entre los colores y las clases de vino.

C) Algoritmos de Detección de Anomalías

1. **Isolation Forest**:
   - Un algoritmo de detección de anomalías basado en árboles de decisión. Se construyen múltiples árboles y se aísla cada punto. Los puntos que son más fáciles de aislar se consideran anomalías.
   - Isolation Forest es un algoritmo que identifica anomalías al aislar observaciones. Funciona construyendo un conjunto de árboles de decisión (bosque) donde las anomalías son más fáciles de aislar. Los pasos básicos son:
     1. Construcción del Árbol: Para cada árbol, selecciona aleatoriamente una característica y luego un valor de división aleatorio para dividir los datos.
     2. Aislamiento: Las anomalías tienden a ser aisladas más rápidamente en un menor número de divisiones, ya que son diferentes del resto de los datos.
     3. Cálculo de Anomalía: La puntuación de anomalía se calcula como la longitud del camino hasta que se aísla un punto. Las puntuaciones más bajas indican puntos más normales, mientras que las puntuaciones más altas indican anomalías.

Ejercicio 1: Isolation Forest
Instrucciones:

1. Carga el conjunto de datos Iris.
2. Aplica Isolation Forest para detectar anomalías.
3. Visualiza los resultados, marcando las anomalías.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import IsolationForest

# Cargar el conjunto de datos Iris
iris = load_iris()
X = iris.data

# Aplicar Isolation Forest
model = IsolationForest(contamination=0.1, random_state=42)
y_pred = model.fit_predict(X)

# Visualizar los resultados
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm', edgecolor='k', s=100)
plt.title('Isolation Forest - Iris Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Explicación del Código

1. **Importación de Librerías**:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import IsolationForest
```

   - numpy: Usada para operaciones matemáticas y manejo de matrices.
   - matplotlib.pyplot: Para visualización de datos.
   - load_iris: Carga el conjunto de datos Iris, que contiene información sobre tres especies de flores.
   - IsolationForest: Algoritmo que detecta valores atípicos (anomalías) en los datos.

2. **Cargar el Conjunto de Datos Iris**:

```
iris = load_iris()
X = iris.data
```

   - iris = load_iris(): Carga el conjunto de datos Iris.
   - X = iris.data: Almacena las características del conjunto de datos (como longitud y ancho del sépalo y pétalo).

3. **Aplicar Isolation Forest**:

```
model = IsolationForest(contamination=0.1, random_state=42)
y_pred = model.fit_predict(X)
```

   - model = IsolationForest(contamination=0.1, random_state=42): Crea un modelo Isolation Forest.
     - contamination=0.1 indica que se espera que el 10% de los datos sean anomalías.
     - random_state=42 fija la semilla para obtener resultados reproducibles.
   - y_pred = model.fit_predict(X): Entrena el modelo con los datos X y obtiene predicciones y_pred, donde:
     - 1 indica que un punto es "normal".
     - -1 indica que un punto es una "anomalía".

4. **Visualización de Resultados**:

```
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm', edgecolor='k', s=100)
plt.title('Isolation Forest - Iris Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

   - plt.figure(figsize=(8, 6)): Configura el tamaño de la gráfica.
   - plt.scatter(...): Dibuja un gráfico de dispersión para visualizar las anomalías:
     - X[:, 0] y X[:, 1] son las dos primeras características del conjunto de datos.
     - c=y_pred colorea los puntos según las predicciones del modelo (1 para normal y -1 para anomalías).
     - cmap='coolwarm' establece el mapa de colores para diferenciar visualmente entre normales y anómalos.
     - edgecolor='k' agrega un borde negro a los puntos.
     - s=100 ajusta el tamaño de los puntos en el gráfico.

2. **One-Class SVM (Support Vector Machine)**:
    - Un método que busca encontrar una función de decisión que separa los datos normales de las anomalías en un espacio de alta dimensión. Se entrena solo con datos normales.
    - One-Class SVM es una variante de la máquina de vectores de soporte (SVM) diseñada para la detección de anomalías. Este algoritmo se entrena utilizando solo ejemplos de la clase normal y busca identificar puntos que no se ajusten a esta clase. Los pasos son:
        1. Entrenamiento: Se entrena el modelo utilizando solo las instancias normales.
        2. Decisión de Separación: El modelo crea un límite en el espacio de características que separa las instancias normales de las anomalías.
        3. Predicción: Las instancias que caen fuera de este límite se consideran anomalías.

Ejercicio 2: t-SNE

Instrucciones:

1. Utiliza el conjunto de datos digits de sklearn.
2. Aplica t-SNE para reducir la dimensionalidad a 2 dimensiones.
3. Visualiza los resultados.

```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE

# Cargar el conjunto de datos de dígitos
digits = load_digits()
X_digits = digits.data
y_digits = digits.target

# Aplicar t-SNE
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X_digits)

# Visualizar los resultados
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits, cmap='viridis', s=20)
plt.title('t-SNE - Digits Dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(scatter)
plt.show()
```

Explicación del Código

1. **Importación de Librerías**:

```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

   - load_digits: Carga el conjunto de datos de dígitos. Este dataset contiene imágenes de dígitos del 0 al 9 en una representación de 8x8 píxeles.
   - TSNE: Algoritmo de reducción de dimensionalidad que proyecta datos en espacios de menor dimensión para facilitar su visualización.
   - matplotlib.pyplot: Librería para graficar datos.

2. **Cargar el Conjunto de Datos de Dígitos**:

```
digits = load_digits()
X_digits = digits.data
y_digits = digits.target
```

   - digits = load_digits(): Carga el conjunto de datos de dígitos.
   - X_digits = digits.data: Contiene las características (información de los píxeles) de cada imagen de dígito.
   - y_digits = digits.target: Contiene las etiquetas o clases de cada dígito (del 0 al 9).

3. **Aplicar t-SNE**:

```
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X_digits)
```

   - TSNE(n_components=2, random_state=0): Configura el modelo t-SNE.
     - n_components=2 reduce los datos a 2 dimensiones para visualización en un plano 2D.
     - random_state=0 asegura que los resultados sean reproducibles.
   - X_tsne = tsne.fit_transform(X_digits): Ajusta el modelo t-SNE en los datos de dígitos y los transforma, proyectando los datos en un espacio 2D.

4. **Visualización de Resultados**:

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits, cmap='viridis', s=20)
plt.title('t-SNE - Digits Dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(scatter)
plt.show()
```

- plt.figure(figsize=(8, 6)): Configura el tamaño de la gráfica.
- plt.scatter(...): Dibuja un gráfico de dispersión:
  - X_tsne[:, 0] y X_tsne[:, 1] son las dos dimensiones principales obtenidas tras la reducción.
  - c=y_digits aplica colores diferentes a cada dígito según su clase, facilitando la distinción visual.
  - cmap='viridis' asigna una escala de colores al gráfico.
  - s=20 define el tamaño de cada punto.
- plt.colorbar(scatter): Agrega una barra de color para identificar las clases de dígitos.

3. **Local Outlier Factor (LOF)**:

- Un algoritmo que detecta anomalías al medir la densidad local de un punto en comparación con sus vecinos. Los puntos con densidad significativamente menor se consideran anomalías.
- Local Outlier Factor (LOF) es un algoritmo que detecta anomalías al comparar la densidad local de un punto con la de sus vecinos. Funciona de la siguiente manera:

  1. Densidad Local: Calcula la densidad de cada punto basándose en la distancia a sus vecinos más cercanos.
  2. Comparación de Densidades: Compara la densidad de un punto con la de sus vecinos. Si la densidad de un punto es significativamente menor, se considera una anomalía.
  3. Puntuación LOF: Asigna una puntuación que indica cuán anómalo es un punto en función de esta comparación de densidades.

Ejercicio 3: UMAP

Instrucciones:

1. Instala la librería umap-learn si no está instalada.
2. Utiliza el conjunto de datos wine de sklearn.
3. Aplica UMAP para reducir la dimensionalidad a 2 dimensiones.
4. Visualiza los resultados.

```
!pip install umap-learn

import umap
from sklearn.datasets import load_wine

# Cargar el conjunto de datos de vino
wine = load_wine()
X_wine = wine.data
y_wine = wine.target

# Aplicar UMAP
umap_model = umap.UMAP(n_components=2, random_state=42)
X_umap = umap_model.fit_transform(X_wine)

# Visualizar los resultados
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y_wine, cmap='viridis', s=50)
plt.title('UMAP - Wine Dataset')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.colorbar(scatter)
plt.show()
```

Explicación del Código

1. **Instalación y Carga de Librerías**:

```
!pip install umap-learn
import umap
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt
```

   - umap-learn: Librería que implementa el algoritmo UMAP para reducción de dimensionalidad.
   - load_wine: Función de sklearn que carga el conjunto de datos de vino, el cual tiene características químicas de varios tipos de vino.
   - matplotlib.pyplot: Librería para graficar los resultados.

2. **Cargar el Conjunto de Datos de Vino**:

```
wine = load_wine()
X_wine = wine.data
y_wine = wine.target
```

   - wine = load_wine(): Carga el conjunto de datos de vino.
   - X_wine = wine.data: Contiene las características de cada muestra de vino (por ejemplo, acidez, alcohol).
   - y_wine = wine.target: Contiene las etiquetas de cada tipo de vino, que permiten identificar su clase.

3. **Aplicar UMAP**:

```
umap_model = umap.UMAP(n_components=2, random_state=42)
X_umap = umap_model.fit_transform(X_wine)
```

   - umap.UMAP(n_components=2, random_state=42): Configura el modelo UMAP.
     - n_components=2 reduce los datos a 2 dimensiones, facilitando la visualización en un plano 2D.
     - random_state=42 asegura que los resultados sean reproducibles.
   - X_umap = umap_model.fit_transform(X_wine): Ajusta el modelo UMAP a los datos y transforma los datos originales en el nuevo espacio de 2D.

4. **Visualización de Resultados**:

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y_wine, cmap='viridis', s=50)
plt.title('UMAP - Wine Dataset')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.colorbar(scatter)
plt.show()
```

- plt.figure(figsize=(8, 6)): Configura el tamaño de la gráfica.
- plt.scatter(...): Dibuja un gráfico de dispersión:
  - X_umap[:, 0] y X_umap[:, 1] representan las dos dimensiones principales generadas por UMAP.
  - c=y_wine aplica diferentes colores para cada clase de vino, facilitando la distinción visual.
  - cmap='viridis' asigna una escala de color.
  - s=50 define el tamaño de los puntos.
- plt.colorbar(scatter): Añade una barra de color para identificar a qué clase corresponde cada punto.

Diferencias entre Clasificación y Agrupamiento

Aspecto	**Clasificación (Supervisado)**	**Agrupamiento (No Supervisado)**
Definición	Asigna etiquetas a los datos basándose en ejemplos etiquetados.	Agrupa datos en función de patrones o similitudes sin etiquetas.
Etiquetas	Requiere datos etiquetados (ej. entrenamiento).	No requiere etiquetas, solo datos.
Objetivo	Predecir la clase de nuevos datos.	Descubrir la estructura subyacente de los datos.
Ejemplo	Clasificar correos electrónicos como "spam" o "no spam".	Agrupar clientes similares en un negocio.