COLOMBIA POTENCIA DE LA VIDA

TIC

Algoritmos de Agrupamiento

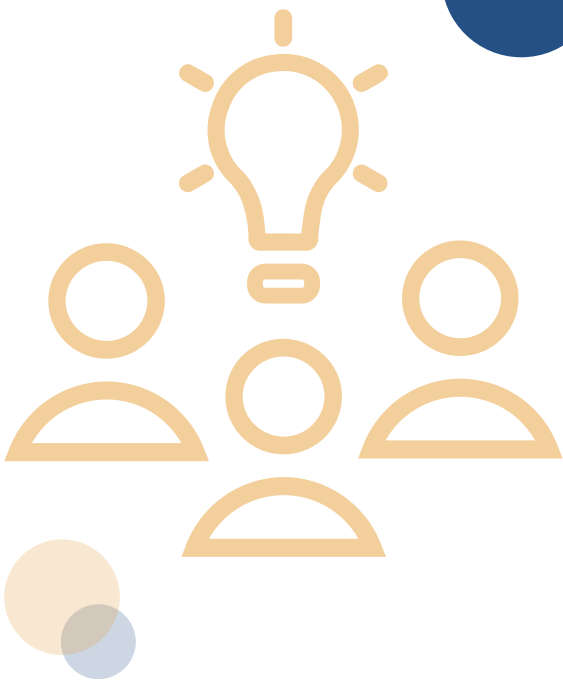
TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira

Algoritmos de Agrupamiento

El agrupamiento es una técnica de aprendizaje no supervisado que consiste en dividir un conjunto de datos en grupos o clusters, donde los elementos dentro de un mismo grupo son más similares entre sí que con los elementos de otros grupos. Esta técnica es fundamental en el aprendizaje no supervisado porque permite descubrir patrones y estructuras ocultas en los datos sin la necesidad de etiquetas previas.



Por ejemplo, en un conjunto de datos de clientes de un supermercado, el agrupamiento podría ayudar a identificar diferentes segmentos de clientes con características similares de compra.

Conceptos Básicos

Distancia Euclidiana

Es una medida de la distancia entre dos puntos en un espacio euclidiano. Se calcula como la longitud del segmento de línea recta que une los dos puntos. La distancia euclidiana es comúnmente utilizada en algoritmos de agrupamiento para medir la similitud entre puntos.

Función de Similitud

Es una función que cuantifica la similitud entre dos elementos en un conjunto de datos. Puede ser una medida de distancia, como la distancia euclidiana, o una medida de similitud, como la correlación de Pearson. La función de similitud es esencial para determinar qué tan cerca están dos puntos y, por lo tanto, para agruparlos adecuadamente.

Centroides

En el contexto del agrupamiento, un centroide es el punto representativo de un cluster, calculado como el promedio de todos los puntos en el cluster. Los centroides son utilizados por algoritmos como K-Means para definir la ubicación de los clusters.

Aplicaciones de los Algoritmos de Agrupamiento en Diferentes Campos

Los algoritmos de agrupamiento tienen una amplia gama de aplicaciones en diversos campos, incluyendo:

Marketing
Segmentación de clientes para campañas de marketing personalizadas.

Medicina
Agrupamiento de pacientes basado en datos de historias clínicas para la detección temprana de enfermedades.

Análisis de Imágenes
Agrupamiento de píxeles para la segmentación de imágenes médicas o la identificación de objetos en fotografías.

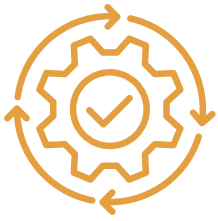
Análisis de redes sociales
Identificación de comunidades o grupos de usuarios con intereses similares en redes sociales.

Procesamiento de Lenguaje Natural:
Agrupamiento de documentos de texto para la organización y clasificación de información.

1. K-Means

K-Means es un algoritmo de agrupamiento que divide un conjunto de datos en k grupos o clusters. El proceso es el siguiente:

1. El proceso comienza con la **selección aleatoria** de **k centroides**, que son los puntos representativos de cada cluster.
2. Luego, se **asigna cada punto de datos** al centroide más cercano, formando así los clusters.
3. A continuación, se **recalculan los centroides** como el promedio de los puntos dentro de cada cluster.



Este proceso se repite iterativamente hasta que los centroides no cambian significativamente o se alcanza un número máximo de iteraciones.

Proceso de agrupamiento utilizando K-Means

Inicialización de centroides: Seleccionar k centroides al azar del conjunto de datos.

Asignación de puntos a clusters: Asignar cada punto de datos al centroide más cercano.

Actualización de centroides: Recalcular los centroides como el promedio de los puntos en cada cluster.

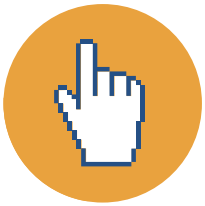
Iteración: Repetir los pasos 2 y 3 hasta que los centroides converjan o se alcance el número máximo de iteraciones.

Consideraciones sobre la inicialización de centroides y selección del número de clusters

Inicialización de centroides:

La selección inicial de centroides puede afectar los resultados del agrupamiento. Diferentes métodos de inicialización pueden conducir a resultados diferentes. Algunas técnicas comunes incluyen la inicialización aleatoria, la inicialización basada en heurísticas y la inicialización inteligente utilizando técnicas como KMeans++.

Selección del número de clusters (k):



Determinar el número óptimo de clusters es crucial para obtener resultados significativos. Se pueden utilizar técnicas como el método del codo (Elbow Method), el criterio de información bayesiano (BIC), o la validación cruzada para seleccionar el valor óptimo de k.

Generación de Datos Aleatorios:

```
import numpy as np
# Generar datos aleatorios con dos características para la demostración
np.random.seed(0)
X = np.random.rand(100, 2)
print("Datos generados aleatoriamente:")
print(X[:5])
```

Cálculo de la Distancia Euclidiana:

```
from scipy.spatial.distance import euclidean
# Calcular la distancia euclidiana entre dos puntos
point1 = [1, 2]
point2 = [4, 6]
distance = euclidean(point1, point2)
print("Distancia Euclidiana entre los puntos:", distance)
```

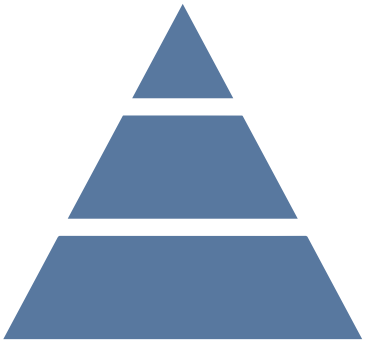
Implementación de K-Means:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Generar datos aleatorios para clustering
X = np.random.rand(100, 2)

# Inicializar y ajustar el modelo K-Means
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

# Visualizar los clusters
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
marker='x', color='red', s=200)
plt.title("Clustering con K-Means")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

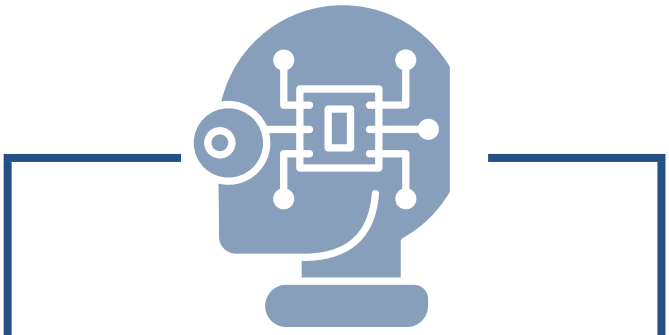
Aplicación de Clustering en un Conjunto de Datos Real:

```python
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

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

# Normalizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Reducir dimensionalidad con PCA para visualización
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Inicializar y ajustar el modelo K-Means
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_scaled)

# Visualizar los clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans.labels_, cmap='viridis')
plt.title("Clustering con K-Means en Conjunto de Datos Iris")
plt.xlabel("Componente Principal 1")
plt.ylabel("Componente Principal 2")
plt.show()
```

2. Agrupamiento Jerárquico



El agrupamiento jerárquico es un enfoque de agrupamiento en el que los datos se organizan en una estructura jerárquica de clusters. Este método no requiere que se especifique el número de clusters de antemano y puede representarse como un árbol o dendrograma. Hay dos enfoques principales en el agrupamiento jerárquico:



Aglomerativo

Comienza con cada punto de datos como un cluster individual y fusiona iterativamente los clusters más cercanos hasta que todos los puntos estén en un solo cluster.

Divisivo

Comienza con un único cluster que contiene todos los puntos de datos y divide iterativamente el cluster en clusters más pequeños hasta que cada punto esté en un cluster individual.

Métodos de enlace

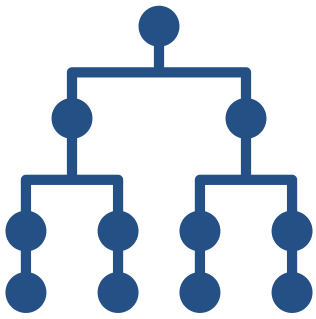
Los métodos de enlace se utilizan para calcular la distancia entre dos clusters durante el proceso de agrupamiento. Algunos de los métodos de enlace más comunes incluyen:

Single-linkage (o mínimo)	La distancia entre dos clusters se define como la distancia **mínima** entre cualquier par de puntos, uno de cada cluster.
Complete-linkage (o máximo)	La distancia entre dos clusters se define como la distancia **máxima** entre cualquier par de puntos, uno de cada cluster.
Average-linkage	La distancia entre dos clusters se define como el **promedio** de todas las distancias entre pares de puntos, uno de cada cluster.

Los métodos de enlace determinan cómo se calcula la distancia entre clusters. El método single-linkage considera la distancia mínima entre los puntos de los clusters. El complete-linkage considera la distancia máxima. El average-linkage considera el promedio de las distancias.

Visualización de dendrogramas y selección del número de clusters

Los dendrogramas son representaciones gráficas de la jerarquía de clusters generados durante el agrupamiento jerárquico. Permiten visualizar la estructura jerárquica y facilitan la selección del número óptimo de clusters al observar la altura a la que se realizan las fusiones en el dendrograma.



Implementación de agrupamiento jerárquico en Python

En Python, se puede implementar el agrupamiento jerárquico utilizando bibliotecas como **scikit-learn** o **scipy**. Estas bibliotecas ofrecen funciones y clases para realizar agrupamiento jerárquico con diferentes métodos de enlace y criterios de distancia. Una vez agrupados los datos, se pueden visualizar los dendrogramas y seleccionar el número óptimo de clusters según los requisitos del problema.

Ejercicios

Visualización de dendrogramas

Veamos el siguiente ejercicio de visualización de dendrogramas para ayudar en la selección del número de clusters.

```
# Visualización de dendrogramas
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Generar datos aleatorios
np.random.seed(0)
X = np.random.randn(20, 2)

# Calcular la matriz de enlace
Z = linkage(X, 'ward')

# Visualizar dendrograma
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title('Dendrograma')
plt.xlabel('Índices de la muestra')
plt.ylabel('Distancia')
plt.show()
```

Implementación de agrupamiento jerárquico

Muestra cómo implementar el agrupamiento jerárquico utilizando scikit-learn.

```
# Implementación de agrupamiento jerárquico en Python
from sklearn.cluster import AgglomerativeClustering

# Generar datos aleatorios
np.random.seed(0)
X = np.random.randn(20, 2)

# Crear una instancia del modelo de agrupamiento jerárquico
model = AgglomerativeClustering(n_clusters=3, linkage='ward')

# Ajustar el modelo a los datos
clusters = model.fit_predict(X)

# Imprimir resultados
print("Clusters:", clusters)
```

3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Es un algoritmo de agrupamiento que se basa en la densidad de los puntos en el espacio de características. Su funcionamiento se centra en la identificación de regiones de alta densidad, que se consideran como clusters, separados por regiones de baja densidad. Para ello, utiliza dos parámetros clave:

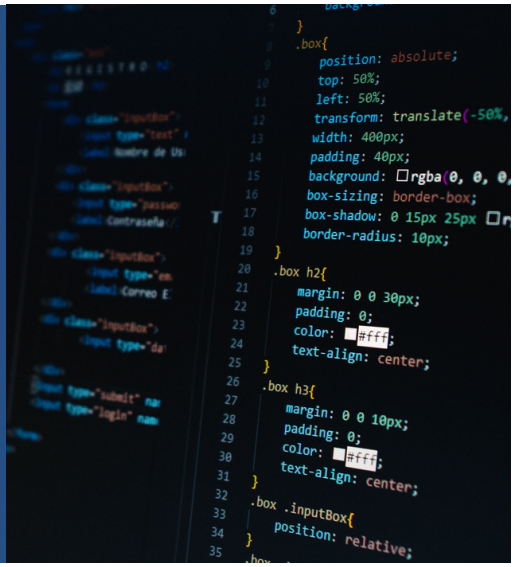
epsilon (ε)

Especifica la distancia máxima entre dos puntos para que se consideren vecinos.

minPts

Establece el número mínimo de puntos dentro de un vecindario para que un punto se considere central.

DBSCAN clasifica los puntos como centrales, de frontera o de ruido, lo que permite identificar clusters de diversas formas y tamaños, así como puntos atípicos. Su aplicación abarca desde la segmentación de datos espaciales hasta la detección de anomalías en conjuntos de datos de alta dimensionalidad.



DBSCAN es un algoritmo de agrupamiento que se basa en la densidad de los puntos en el espacio de características. Su funcionamiento se puede resumir en los siguientes pasos:

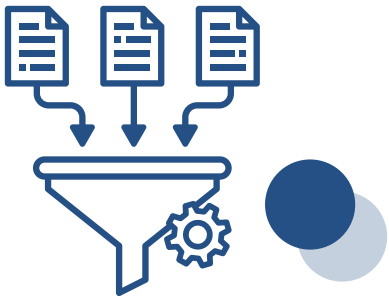
1. Principio de funcionamiento

DBSCAN divide el conjunto de datos en regiones de alta densidad separadas por regiones de baja densidad. Para ello, utiliza dos parámetros principales:

Epsilon (ε) Especifica la distancia máxima entre dos puntos para que se consideren vecinos.	**MinPts** Establece el número mínimo de puntos dentro de un vecindario para que un punto se considere central.

2. Definición de parámetros

Epsilon y MinPts son dos parámetros fundamentales en DBSCAN. Epsilon determina el tamaño del vecindario alrededor de cada punto, mientras que MinPts define la densidad mínima requerida para formar un cluster.



3. Identificación de puntos

DBSCAN clasifica los puntos en tres categorías:

Puntos centrales Son aquellos que tienen al menos MinPts puntos dentro de su vecindario de distancia ε.	**Puntos de frontera** No son puntos centrales, pero están dentro del vecindario de un punto central.	**Puntos de ruido** No son centrales ni de frontera.

4. Ejemplos de aplicación

DBSCAN se utiliza en diversas aplicaciones, como la detección de anomalías y el clustering de datos. Por ejemplo, en la detección de anomalías, los puntos de ruido pueden considerarse como anomalías, mientras que en el clustering de datos, DBSCAN puede identificar clusters de diferentes formas y tamaños de manera automática.

Ejercicios

Implementación de DBSCAN

Escribe una función en Python que implemente el algoritmo DBSCAN. La función debe aceptar los siguientes parámetros de entrada:

- **data:** La matriz de datos para agrupar.
- **epsilon:** El valor de epsilon para definir el radio del vecindario.
- **minPts:** El número mínimo de puntos dentro de un vecindario para que un punto se considere central.

La función debe devolver una lista de etiquetas de cluster para cada punto en los datos.

```
from sklearn.cluster import DBSCAN
def custom_dbscan(data, epsilon, minPts):
      dbscan = DBSCAN(eps=epsilon, min_samples=minPts)
      labels = dbscan.fit_predict(data)
      return labels
```

Identificación de Puntos

Dado un conjunto de datos y los resultados del agrupamiento utilizando DBSCAN, escribe una función para identificar y contar el número de puntos centrales, puntos de frontera y puntos de ruido.

```
def identify_points(labels):
      unique_labels = set(labels)
      core_points = sum(1 for label in labels if label != -1)
      border_points = sum(1 for label in labels if label == -1)
      noise_points = sum(1 for label in labels if label == 0)
      return core_points, border_points, noise_points

# Ejemplo de uso
labels = [0, 1, -1, 1, 2, 2, 0, -1]
core, border, noise = identify_points(labels)
print("Puntos centrales:", core)
print("Puntos de frontera:", border)
print("Puntos de ruido:", noise)
```

Ejemplo de Aplicación

Utiliza DBSCAN para agrupar un conjunto de datos sintéticos y visualiza los resultados utilizando un gráfico de dispersión.

```
from sklearn.datasets import make_moons
import numpy as np
import matplotlib.pyplot as plt

# Generar datos sintéticos
np.random.seed(0)
X, _ = make_moons(n_samples=200, noise=0.1)
```

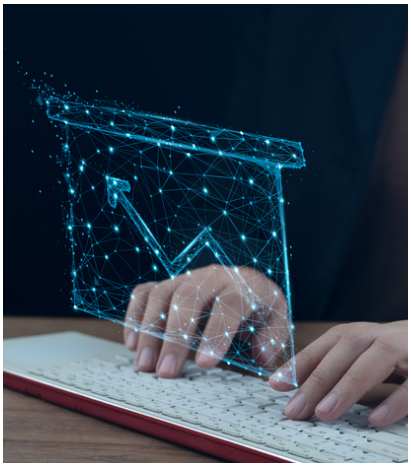
```
# Aplicar DBSCAN
epsilon = 0.2
minPts = 5
labels = custom_dbscan(X, epsilon, minPts)

# Visualizar los resultados
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('Agrupamiento con DBSCAN')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.show()
```

4. Mean Shift

Mean Shift es un algoritmo de agrupamiento no paramétrico que busca los modos de densidad en un conjunto de datos para encontrar los centroides de los clusters. Funciona mediante la definición de una función de densidad de probabilidad y moviendo iterativamente los puntos de datos hacia las regiones de mayor densidad. Esto permite que el algoritmo encuentre automáticamente el número de clusters y su forma sin necesidad de especificarlos de antemano.

Concepto de ventana de búsqueda y convergencia hacia los modos de densidad

La ventana de búsqueda es un parámetro clave en el algoritmo Mean Shift que determina el radio de la región alrededor de cada punto de datos dentro de la cual se busca el centroide del cluster. Durante cada iteración del algoritmo, los puntos de datos se desplazan hacia las regiones de mayor densidad, lo que gradualmente conduce a la convergencia hacia los modos de densidad, es decir, los centroides de los clusters.



Implementación de Mean Shift en Python y ejemplos de uso en agrupamiento de datos

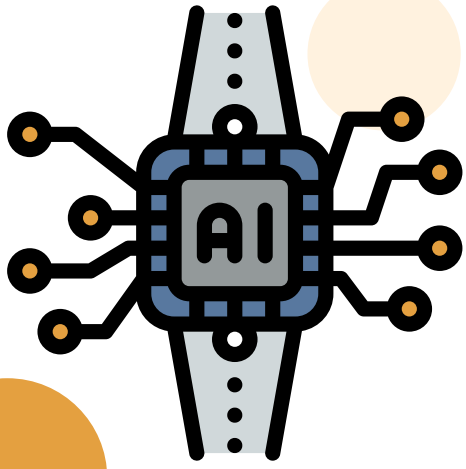
Mean Shift se puede implementar fácilmente en Python utilizando bibliotecas como scikit-learn. En la implementación, se ajusta un modelo de Mean Shift a los datos y se utilizan las etiquetas de cluster asignadas para visualizar y analizar los resultados.

Ejemplos prácticos de uso de Mean Shift incluyen la segmentación de imágenes, la detección de objetos en imágenes y la agrupación de puntos de datos en conjuntos no lineales.

```python
#Implementación de Mean Shift en Python
from sklearn.cluster import MeanShift
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generar datos de ejemplo
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)

# Aplicar Mean Shift
ms = MeanShift()
ms.fit(X)
labels = ms.labels_
cluster_centers = ms.cluster_centers_

# Visualizar los resultados
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis')
plt.scatter(cluster_centers[:,0], cluster_centers[:,1], marker='x',
color='red', s=300, linewidth=5)
plt.show()

#Ejemplo de uso en agrupamiento de datos
from sklearn.datasets import make_circles

# Generar datos ficticios en forma de anillo concéntrico
X, _ = make_circles(n_samples=1000, noise=0.05, random_state=42)
```

```
# Aplicar Mean Shift
ms = MeanShift()
ms.fit(X)
labels = ms.labels_
cluster_centers = ms.cluster_centers_

# Visualizar los resultados
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis')
plt.scatter(cluster_centers[:,0], cluster_centers[:,1], marker='x',
color='red', s=300, linewidth=5)
plt.show()
```

5. Gaussian Mixture Models (GMM) y Spectral Clustering

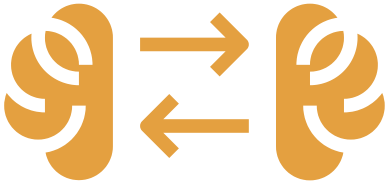


Mean Shift es un algoritmo de agrupamiento no paramétrico que busca los modos de densidad en un conjunto de datos para encontrar los centroides de los clusters. Funciona mediante la definición de una función de densidad de probabilidad y moviendo iterativamente los puntos de datos hacia las regiones de mayor densidad. Esto permite que el algoritmo encuentre automáticamente el número de clusters y su forma sin necesidad de especificarlos de antemano.

Modelos de mezclas gaussianas (GMM) y su relación con el agrupamiento

Los modelos de mezclas gaussianas son un enfoque probabilístico para el agrupamiento de datos. En GMM, se asume que los datos provienen de una mezcla de varias distribuciones gaussianas y se busca estimar los parámetros de estas distribuciones para modelar la estructura de los datos. GMM es útil cuando los datos no se agrupan claramente en grupos distintos y cuando los grupos pueden tener formas y tamaños diferentes.

Concepto de clustering espectral y su aplicación en grafos



El clustering espectral es una técnica que se basa en la representación de datos como grafos y en la exploración de las propiedades de sus matrices de afinidad.

Este algoritmo se utiliza para encontrar estructuras de grupos en datos que pueden no ser linealmente separables en el espacio de características original. El algoritmo busca los autovectores asociados con los autovalores más grandes de la matriz de afinidad para determinar los grupos.

Comparación entre GMM y Spectral Clustering

GMM

Es un enfoque probabilístico que modela la distribución de los datos mediante gaussianas.

Puede funcionar bien en datos con distribuciones complejas

VS

Spectral Clustering

se basa en la estructura del grafo de afinidad de los datos.

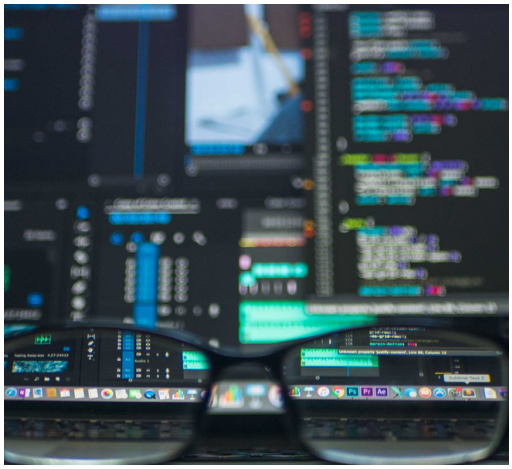
Es útil para datos no linealmente separables.

Modelos de Mezclas Gaussianas (GMM)

- Método probabilístico que modela la distribución de los datos como una combinación de distribuciones gaussianas.
- Útil para datos con formas complejas y superpuestas.
- Asigna probabilidades de pertenencia a cada clúster para cada punto de datos.

Clustering Espectral

- Basado en la teoría de grafos para encontrar estructuras de agrupamiento en datos.
- Utiliza la estructura del grafo y los autovectores de la matriz de afinidad para agrupar los datos.
- Útil para datos no linealmente separables y estructuras de clústeres no convencionales.



Ejemplos prácticos de aplicación de ambos algoritmos en Python

En Python, GMM se puede implementar utilizando bibliotecas como scikit-learn. Se puede aplicar a conjuntos de datos para modelar la distribución de los datos y realizar el agrupamiento. Por otro lado, Spectral Clustering también está disponible en scikit-learn y se puede usar para agrupar datos representados como grafos.

Introducción a GMM

- ## Paso 1. Generación de datos

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

# Generar datos sintéticos con 3 clústeres
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0,
random_state=42)

# Visualizar los datos
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.title("Datos de ejemplo")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.show()
```

- ## Paso 2: Ajuste del modelo GMM

```
from sklearn.mixture import GaussianMixture

# Crear el modelo GMM con 3 componentes
gmm = GaussianMixture(n_components=3, random_state=42)

# Ajustar el modelo a los datos
gmm.fit(X)

# Obtener las etiquetas de clúster para cada punto de datos
labels = gmm.predict(X)

# Visualizar los resultados
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.title("Agrupamiento con GMM")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.show()
```

Introducción a Spectral Clustering

- ## Paso 1. Generación de datos

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

# Generar datos sintéticos con forma de media luna
X, _ = make_moons(n_samples=300, noise=0.1, random_state=42)

# Visualizar los datos
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.title("Datos de ejemplo")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.show()
```

- ## Paso 2: Ajuste del modelo Spectral Clustering

```
from sklearn.cluster import SpectralClustering

# Crear el modelo Spectral Clustering con 2 clústeres
spectral = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
random_state=42)

# Ajustar el modelo a los datos
spectral.fit(X)

# Obtener las etiquetas de clúster para cada punto de datos
labels = spectral.labels_

# Visualizar los resultados
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.title("Agrupamiento con Spectral Clustering")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.show()
```

Ejercicio

Comparación entre GMM y Spectral Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture
from sklearn.cluster import SpectralClustering

# Generar datos sintéticos
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)

# Ajustar modelo GMM
gmm = GaussianMixture(n_components=4).fit(X)
labels_gmm = gmm.predict(X)

# Ajustar modelo Spectral Clustering
spectral = SpectralClustering(n_clusters=4, affinity='nearest_neighbors',
assign_labels='kmeans')
labels_spectral = spectral.fit_predict(X)

# Comparación visual de GMM y Spectral Clustering en los mismos datos
plt.figure(figsize=(12, 4))

# GMM
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=labels_gmm, s=50, cmap='viridis',
alpha=0.5)
plt.title("GMM")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
```

```
# Spectral Clustering
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=labels_spectral, s=50, cmap='viridis',
alpha=0.5)
plt.title("Spectral Clustering")
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")

plt.tight_layout()
plt.show()
```