PROGRAMA DE INTELIGENCIA ARTIFICIAL | IBM SkillUp 2024¶

|  |  |  |
| --- | --- | --- |

# Fundamentos de Python y el Aprendizaje Automático[¶](#gjdgxs)

Python es un lenguaje de programación versátil y ampliamente utilizado en el campo del aprendizaje automático por su simplicidad y potente ecosistema de bibliotecas. El aprendizaje automático, por su parte, es una rama de la inteligencia artificial que permite a los sistemas aprender y hacer predicciones a partir de datos.

Utilizando algoritmos específicos, estos sistemas pueden identificar patrones y tomar decisiones sin estar explícitamente programados para cada tarea. **Python**, con sus bibliotecas como **NumPy**, **Pandas**, **scikit-learn**, **TensorFlow** y **PyTorch**, proporciona herramientas sólidas para implementar modelos de aprendizaje automático en una variedad de aplicaciones, desde la clasificación y regresión hasta la detección de anomalías y el agrupamiento de datos.

El aprendizaje automático se divide principalmente en tres categorías:

* **Aprendizaje supervisado:** utiliza datos etiquetados para entrenar modelos en tareas como clasificación y regresión.
* **Aprendizaje no supervisado:** encuentra patrones ocultos en datos no etiquetados, como en el agrupamiento (**clustering**).
* **Sistemas de recomendación:** sugiere productos, servicios o contenidos personalizados a los usuarios basándose en sus preferencias y comportamientos anteriores.
* **Aprendizaje por refuerzo:** el modelo aprende a tomar decisiones mediante recompensas y penalizaciones, útil en áreas como juegos y robótica.

Aprendizaje supervisado (Supervised Learning)[¶](#30j0zll)

El **aprendizaje supervisado** es una técnica de aprendizaje automático en la que un modelo se entrena utilizando un conjunto de datos etiquetados, donde cada dato de entrada tiene una salida conocida asociada. El modelo aprende a predecir estas salidas ajustando sus parámetros internos para minimizar los errores de predicción. Este tipo de aprendizaje es fundamental para resolver problemas de clasificación y regresión.

Tipos de aprendizaje supervisado:

1. **Clasificación:** asigna una categoría o etiqueta a cada dato de entrada.
   * Ejemplos:
     + Clasificar correos electrónicos como “spam” o “no spam”.
     + Identificar imágenes de gatos vs. perros.
   * Aplicaciones: reconocimiento de voz, diagnóstico médico, clasificación de documentos.
2. **Regresión:** predice un valor continuo a partir de las entradas de datos.
   * Ejemplos:
     + Estimar el precio de una casa según sus características.
     + Predecir la temperatura futura basada en datos históricos.
   * Aplicaciones: predicción de precios, pronóstico del tiempo, análisis financiero.

Regresión lineal[¶](#1fob9te)

La regresión es un subcampo del aprendizaje automático cuyo proposito es predecir valores continuos a partir de ciertas entradas o variabl es descriptivas. Existen dos tipos de regressión, lineal (simple o múltiple) y no lineal.

La regresión es usada para resolver problemas de la vida real, por ejemplo, predecir el precio de una casa a parte de su área en metros cuadrados, numerto de cuartos, etc. Tambien para predecir el número de ventas de un producto basado en la publicidad gastada o el número de consultas realizadas por los clientes sobre ese producto.

La regresión lineal es aplicable si los datos disponibles se pueden representar mediante una línea, es decir, al mostrarlos en un gráfico tienen un comportamiento lineal. Si contamos con una variable descriptiva estamos ante una regresion lineal simple.

La ecuación de la regresión lineal simple es:

\***y = W0 (intersección) + W1 (pendiente) · X (variable independiente)**\*

* x representa a la variable independiente e "y" la variable dependiente.
* w0 y w1 son coeficientes o parámetros que debemos encontrar
* Idealmente se busca que una línea pase lo más cerca posible de todos los puntos de datos

Práctica 1: Regresión lineal con NumPy[¶](#3znysh7)

En esta práctica, aplicaremos la regresión lineal para predecir el precio de las casas basado en su área en metros cuadrados. Supondremos que tenemos datos históricos sobre el área de varias casas y sus precios de venta. Usaremos estos datos para entrenar un modelo que nos permita predecir el precio de nuevas casas en función de su área.

Basándonos en la fórmula que hemos visto arriba:

**y = W0 + W1 \* X**

* **Intersección** (W0): Es el precio base o el costo inicial de la casa antes de considerar el área. En nuestro ejemplo, supongamos que es 50,000€. Este valor podría incluir costos fijos como la ubicación, el terreno o los materiales básicos utilizados en la construcción.
* **Pendiente** (W1): Indica cuánto aumenta el precio de la casa por cada metro cuadrado adicional. Por ejemplo, si la pendiente es 3,000€, significa que el precio de la casa aumenta en 3,000€ por cada metro cuadrado adicional.

Ejemplo:

* Precio inicial de una casa con 0 m² de área: Precio = 50,000 + 3,000 \* 0 = 50,000
* Precio para una casa de 50 m²: Precio = 50,000 + 3,000 \* 50 = 200,000
* Precio para una casa de 100 m²: Precio = 50,000 + 3,000 \* 100 = 350,000

1. Generación de datos simulados[¶](#2et92p0)

En este paso, generaremos datos que simulan cómo el área de las casas afecta su precio.

In [ ]:

**import** numpy **as** np *# Librería necesaria para trabajar con datos*  
**import** matplotlib.pyplot **as** plt *# Librería necesaria para mostrar los datos*  
  
*# Fijar la semilla para obtener los mismos resultados cada vez*  
np**.**random**.**seed(42)  
  
*# Generar datos simulados: áreas de casas en metros cuadrados entre 50 y 150*  
area\_casas **=** 50 **+** 100 **\*** np**.**random**.**rand(20, 1) *# Generar 20 puntos de datos entre 50 y 150 m²*  
  
*# El precio de las casas depende del área más un poco de variabilidad*  
*# Fórmula simple: precio = 50000 + 3000 \* area + ruido*  
precio\_casas **=** 50000 **+** 3000 **\*** area\_casas **+** np**.**random**.**randn(20, 1) **\*** 10000  
  
*# Mostrar los datos en un gráfico*  
plt**.**scatter(area\_casas, precio\_casas)  
plt**.**xlabel("Área de la Casa (m²)")  
plt**.**ylabel("Precio de la Casa (€)")  
plt**.**title("Precio de las Casas en función del Área")  
plt**.**show()

2. Implementación de la Regresión Lineal[¶](#tyjcwt)

En esta etapa, calcularemos la línea de mejor ajuste que describe la relación entre el área de las casas y su precio. Esto se logra ajustando un modelo de regresión lineal simple a nuestros datos.

In [ ]:

*# Agregamos una columna de unos al área para la fórmula de la línea (facilita el cálculo al incluir un término constante)*  
area\_casas\_b **=** np**.**c\_[np**.**ones((20, 1)), area\_casas]  
  
*# Calculamos los parámetros de la línea usando la ecuación normal*  
*# theta representa un vector que contiene los parámetros de la línea de regresión*  
*# es decir, los valores de ( W\_0 ) y ( W\_1 ).*  
*# theta = (X.T \* X)^-1 \* X.T \* y*  
theta\_mejor **=** np**.**linalg**.**inv(area\_casas\_b**.**T **@** area\_casas\_b) **@** area\_casas\_b**.**T **@** precio\_casas  
  
*# Mostrar los parámetros calculados (intersección y pendiente)*  
print("Parámetros de la línea de mejor ajuste (theta):", theta\_mejor)

3. Predicciones[¶](#3dy6vkm)

Realizamos las predicciones de nuestro modelo.

In [ ]:

*# Crear algunos nuevos datos de áreas de casas para predecir sus precios*  
area\_casas\_nuevas **=** np**.**array([[50], [150]])  
  
*# Agregar la columna de unos a estos nuevos datos*  
area\_casas\_nuevas\_b **=** np**.**c\_[np**.**ones((2, 1)), area\_casas\_nuevas]  
  
*# Calcular los precios predichos usando la línea de mejor ajuste*  
precio\_predicho **=** area\_casas\_nuevas\_b **@** theta\_mejor  
  
*# Dibujar la línea de regresión sobre el gráfico*  
plt**.**plot(area\_casas\_nuevas, precio\_predicho, "r-", label**=**"Línea de regresión")  
plt**.**scatter(area\_casas, precio\_casas)  
plt**.**xlabel("Área de la casa (m²)")  
plt**.**ylabel("Precio de la casa (€)")  
plt**.**legend()  
plt**.**title("Línea de regresión: precio vs área de la casa")  
plt**.**show()

Clasificación[¶](#1t3h5sf)

La clasificación es un subcampo del aprendizaje automático enfocado en asignar etiquetas o categorías a entradas basadas en ciertas características descriptivas. Los algoritmos de clasificación son fundamentales para resolver problemas donde la salida es discreta, como decidir si un correo es spam o no, determinar si una imagen contiene un gato o un perro, o predecir si un cliente comprará un producto.

Tipos de Clasificación:

1. **Clasificación Binaria:**
   * El objetivo es clasificar las entradas en una de dos posibles categorías.
     + Ejemplos: diagnóstico médico (enfermo o sano), detección de fraude (fraudulento o no).
2. **Clasificación Multiclase:**
   * Se extiende la clasificación a más de dos categorías.
     + Ejemplos: clasificación de correos electrónicos (spam, social, promociones), reconocimiento de dígitos manuscritos (0-9).

### Regresión Logística[¶](#4d34og8)

La regresión logística es particularmente útil para la clasificación binaria, donde se desea predecir la probabilidad de que una observación pertenezca a una de dos categorías. La salida de la regresión logística es una probabilidad entre 0 y 1, que luego se convierte en una clasificación utilizando un umbral (por ejemplo, 0.5).

Ejemplos de Uso:

* Clasificar correos electronicos como "spam" o "no spam" basándose en caracteristicas como el contenido dle mensaje, el remitente y la frecuencia de ciertas palabras.
* Determinar si un paciente tiene una enfermeda basándose en síntomas y resultados de pruebas médicas.
* Predecir si un cliente es buen o mal candidato para un préstamo basándose en su historial crediticio y otras caracteristicas.

Práctica 2: Clasificación con scikit-learn: Regresión Logística[¶](#2s8eyo1)

Simulamos datos donde el nivel de glucosa en sangre se usa para predecir si un paciente tiene diabetes o no.

1. Crear y visualizar los datos[¶](#17dp8vu)

En este ejemplo, generaremos datos simples que relacionen el nivel de glucosa en sangre con el diagnóstico de diabetes (tener o no tener diabetes).

In [ ]:

*# Importar las siguientes librerias*  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** pandas **as** pd  
  
*# Crear datos simples: nivel de glucosa en sangre y diagnóstico de diabetes (0 = No tiene diabetes, 1 = Tiene diabetes)*  
*# Supongamos que estos son los datos históricos que tenemos*  
niveles\_glucosa **=** np**.**array([80, 90, 100, 110, 120, 130, 140, 150, 160, 170]) *# Niveles de glucosa en mg/dL*  
diagnostico\_diabetes **=** np**.**array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1]) *# 0 = No tiene diabetes, 1 = Tiene diabetes*  
  
*# Crear un DataFrame con estos datos*  
df **=** pd**.**DataFrame({  
 'Nivel de Glucosa': niveles\_glucosa,  
 'Diagnóstico de Diabetes': diagnostico\_diabetes  
})  
  
*# Mostrar los datos en un gráfico de dispersión*  
plt**.**scatter(niveles\_glucosa, diagnostico\_diabetes, color**=**'blue', label**=**'Datos Reales')  
plt**.**xlabel("Nivel de Glucosa (mg/dL)")  
plt**.**ylabel("Diagnóstico de diabetes (0 = No tiene diabetes, 1 = Tiene diabetes)")  
plt**.**title("Diagnóstico de diabetes según nivel de glucosa en sangre")  
plt**.**legend()  
plt**.**show()

2. Entrenar un modelo de Regresión Logística[¶](#3rdcrjn)

En esta sección, dividiremos los datos en conjuntos de entrenamiento y prueba, entrenaremos un modelo de regresión logística para predecir el diagnóstico de diabetes basado en el nivel de glucosa en sangre, y evaluaremos la precisión del modelo.

In [ ]:

*# Importar las siguientes librerias*   
**from** sklearn.model\_selection **import** train\_test\_split *# Importar la función para dividir los datos*  
**from** sklearn.linear\_model **import** LogisticRegression *# Importar el modelo de regresión logística*  
**from** sklearn.metrics **import** accuracy\_score *# Importar la métrica para evaluar la precisión*  
  
*# Dividir los datos en conjuntos de entrenamiento y prueba*  
X **=** niveles\_glucosa**.**reshape(**-**1, 1) *# Convertir niveles de glucosa en un formato de 2 dimensiones*  
y **=** diagnostico\_diabetes *# Etiquetas (0 = No tiene diabetes, 1 = Tiene diabetes)*  
  
*# Dividir los datos: 80% para entrenamiento, 20% para prueba*  
X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**42)  
  
*# Crear y entrenar el modelo de regresión logística*  
modelo\_logistico **=** LogisticRegression()  
modelo\_logistico**.**fit(X\_train, y\_train)  
  
*# Hacer predicciones con el modelo entrenado*  
y\_pred **=** modelo\_logistico**.**predict(X\_test)  
  
*# Evaluar la precisión del modelo*  
precision **=** accuracy\_score(y\_test, y\_pred)  
print("Precisión del modelo de regresión logística:", precision)

3. Visualizar el Modelo y las Predicciones[¶](#26in1rg)

En esta etapa, visualizaremos cómo el modelo clasifica los datos y la curva que separa las dos clases (tener diabetes y no tener diabetes) según los niveles de glucosa en sangre.

In [ ]:

*# Crear una secuencia de niveles de glucosa para mostrar la línea de decisión*  
niveles\_glucosa\_nuevos **=** np**.**linspace(70, 180, 100)**.**reshape(**-**1, 1) *# Crear 100 puntos entre 70 y 180 mg/dL*  
  
*# Predecir las probabilidades de tener diabetes para estos nuevos niveles de glucosa*  
probabilidades **=** modelo\_logistico**.**predict\_proba(niveles\_glucosa\_nuevos)[:, 1] *# Obtener la probabilidad de tener diabetes (clase 1)*  
  
*# Dibujar la línea de decisión (probabilidad de tener diabetes)*  
plt**.**plot(niveles\_glucosa\_nuevos, probabilidades, color**=**'red', label**=**'Línea de Decisión')  
plt**.**scatter(niveles\_glucosa, diagnostico\_diabetes, color**=**'blue', label**=**'Datos Reales')  
plt**.**xlabel("Nivel de Glucosa (mg/dL)")  
plt**.**ylabel("Probabilidad de Tener Diabetes")  
plt**.**title("Modelo de Regresión Logística: diagnóstico de Diabetes")  
plt**.**legend()  
plt**.**show()

Aprendizaje no supervisado (Unsupervised Learning)[¶](#lnxbz9)

El Aprendizaje no supervisado es una técnica de aprendizaje automático en la que un modelo se entrena con datos que no están etiquetados, es decir, sin salidas conocidas. El objetivo es descubrir patrones, estructuras ocultas o relaciones en los datos sin ninguna orientación previa. Esta técnica se utiliza principalmente para explorar datos, identificar grupos naturales, reducir la dimensionalidad y detectar anomalías.

Tipos de aprendizaje no supervisado:

1. **Agrupamiento (Clustering):** Organiza los datos en grupos o clusters basados en su similitud.

* Ejemplos:
  + **K-Means:** agrupa puntos en K clusters basándose en la cercanía a los centros de los clusters.
  + **Agrupamiento jerárquico:** crea una estructura de árbol de clusters, que puede dividirse de manera recursiva para formar una jerarquía.
  + **DBSCAN:** identifica clusters basados en la densidad de los puntos, permitiendo formas arbitrarias y detectando puntos de ruido.

1. **Reducción de Dimensionalidad:** Simplifica los datos reduciendo el número de características mientras se conserva la información importante.

Practica 3: Agrupamiento (Clustering)[¶](#35nkun2)

El clustering es una técnica fundamental de aprendizaje no supervisado que se utiliza para agrupar datos en conjuntos (clusters) que contienen puntos similares. En lugar de depender de etiquetas predefinidas, el clustering busca identificar patrones y estructuras inherentes en los datos, agrupando elementos que son más cercanos o similares entre sí según alguna métrica de distancia o similitud. Esta técnica es invaluable en la exploración de datos y en la simplificación de la complejidad al dividir grandes conjuntos de datos en grupos más manejables. Clustering se aplica en diversas áreas como segmentación de mercado, análisis de comportamiento, detección de anomalías, y compresión de datos.

K-MEANS[¶](#1ksv4uv)

K-Means es uno de los algoritmos de agrupamiento (clustering) más populares y simples en el aprendizaje automático. Su objetivo principal es dividir un conjunto de datos en K grupos (clusters), de modo que los puntos dentro de cada grupo sean más similares entre sí que con los puntos de otros grupos. Es ampliamente utilizado por su simplicidad y eficacia en la agrupación de datos en aplicaciones como segmentación de clientes, análisis de patrones y compresión de datos.

Vamos a suponer que queremos agrupar varios puntos en el espacio basándonos en su proximidad, como si estuviéramos organizando niños en diferentes grupos en un patio de recreo según dónde están parados.

1. Generamos datos simples que representan la posición de los niños en el patio de recreo.[¶](#44sinio)

Vamos a crear y visualizar la posición de los niños en el patio como puntos en un gráfico.

In [ ]:

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.cluster **import** KMeans  
  
*# Generar datos simulados (puntos) para el agrupamiento*  
*# Supongamos que estos puntos representan niños en un patio de recreo*  
*# Cada par de coordenadas entre los corchetes representa la posicion de un niño en el eje X e Y*  
*# X son las coordenadas de los puntos en el espacio*  
X **=** np**.**array([  
 [1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6],  
 [9, 11], [8, 2], [10, 2], [9, 3], [7, 8],  
 [7, 9], [8, 9], [0.5, 1], [1, 1.5], [6, 8],  
 [9, 9], [7, 3], [8, 4], [2, 2.5], [0.3, 0.5]  
])  
  
*# Mostrar los datos en un gráfico de dispersión*  
plt**.**scatter(X[:, 0], X[:, 1], s**=**100, color**=**'blue', label**=**'Niños en el Patio')  
plt**.**xlabel("Coordenada X")  
plt**.**ylabel("Coordenada Y")  
plt**.**title("Posición de los Niños en el Patio")  
plt**.**legend()  
plt**.**show()

2. Aplicando algoritmo K-MEANS[¶](#2jxsxqh)

Agruparemos puntos en 3 grupos y visualizaremos los resultados con colores y centros destacados.

In [ ]:

*# Crear un modelo K-Means para 3 grupos (clusters)*  
*# Supongamos que queremos dividir a los niños en 3 grupos basados en su proximidad*  
kmeans **=** KMeans(n\_clusters**=**3, n\_init**=**10, random\_state**=**0) *# Especificamos n\_init=10*  
  
*# Ajustar el modelo a los datos (es decir, encontrar los grupos)*  
kmeans**.**fit(X)  
  
*# Predecir el grupo al que pertenece cada punto*  
y\_kmeans **=** kmeans**.**predict(X)  
  
*# Mostrar los resultados de los grupos en un gráfico*  
plt**.**scatter(X[:, 0], X[:, 1], c**=**y\_kmeans, s**=**100, cmap**=**'viridis', label**=**'Niños Agrupados')  
centros **=** kmeans**.**cluster\_centers\_ *# Obtener los centros de los clusters*  
plt**.**scatter(centros[:, 0], centros[:, 1], c**=**'red', s**=**300, alpha**=**0.75, label**=**'Centros de los Grupos')  
plt**.**xlabel("Coordenada X")  
plt**.**ylabel("Coordenada Y")  
plt**.**title("Agrupamiento de niños en el patio con K-Means")  
plt**.**legend()  
plt**.**show()

Agrupamiento Jerárquico (Hierarchical Clustering)[¶](#z337ya)

¿Qué es el Agrupamiento Jerárquico?

Es una técnica de agrupamiento que construye una jerarquía de clusters. A diferencia de K-Means, que forma un número fijo de clusters, el agrupamiento jerárquico organiza los datos en una estructura de árbol. El Dendrograma es una representación visual de la jerarquía de clusters. Muestra cómo se agrupan los puntos en diferentes niveles de similitud.

Vamos a usar los mismos puntos de los niños y aplicar el agrupamiento jerárquico para ver cómo se agrupan en un dendrograma.

* Utilizamos los puntos X que representan la posición de los niños en el patio de recreo.
* Usamos linkage(X, method='ward') para calcular los enlaces jerárquicos. El método 'ward' se utiliza para minimizar la varianza dentro de cada grupo, lo que suele dar buenos resultados para datos que están bien distribuidos.
* Usamos dendrogram(Z) para dibujar el dendrograma.
* El dendrograma es un gráfico que muestra cómo se agrupan los puntos a diferentes niveles de similitud. Cuanto más baja sea la unión en el dendrograma, más similares son

In [ ]:

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** scipy.cluster.hierarchy **import** dendrogram, linkage *# Importar funciones para el agrupamiento jerárquico*  
  
*# Datos de los niños en el patio (mismos puntos que antes)*  
*# Cada par de coordenadas entre los corchetes representa la posicion de un niño en el eje X e Y*  
X **=** np**.**array([  
 [1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6],  
 [9, 11], [8, 2], [10, 2], [9, 3], [7, 8],  
 [7, 9], [8, 9], [0.5, 1], [1, 1.5], [6, 8],  
 [9, 9], [7, 3], [8, 4], [2, 2.5], [0.3, 0.5]  
])  
  
*# Aplicar el agrupamiento jerárquico a los datos*  
*# El método 'ward' minimiza la varianza dentro de cada grupo*  
Z **=** linkage(X, method**=**'ward')  
  
*# Visualizar el dendrograma, que muestra cómo se agrupan los puntos*  
plt**.**figure(figsize**=**(10, 7)) *# Ajustar el tamaño del gráfico*  
dendrogram(Z) *# Dibujar el dendrograma*  
plt**.**title("Dendrograma para el Agrupamiento Jerárquico") *# Título del gráfico*  
plt**.**xlabel("Índice de los Puntos") *# Etiqueta del eje X*  
plt**.**ylabel("Distancia") *# Etiqueta del eje Y*  
plt**.**show() *# Mostrar el gráfico*

Agrupamiento espacial basado en densidad de aplicaciones con ruido (DBSCAN)[¶](#3j2qqm3)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Es un algoritmo de clustering que agrupa puntos basándose en la densidad local de los datos. Es particularmente útil para detectar clusters de forma arbitraria y para identificar puntos de ruido.

Este ejemplo muestra cómo DBSCAN puede agrupar puntos en formas complejas como las lunas crecientes y menguantes, y cómo puede identificar puntos que no pertenecen a ningún grupo (ruido). Utilizamos un conjunto de datos visualmente interesante para ilustrar cómo DBSCAN funciona en la práctica.

* Usamos make\_moons para crear un conjunto de puntos que forman dos lunas crecientes. Esto crea una estructura de datos interesante donde los puntos forman dos clusters en forma de media luna.
* noise=0.1 añade un poco de variabilidad, simulando ruido natural en los datos.
* Creamos un modelo DBSCAN con eps=0.2 y min\_samples=5.

In [ ]:

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.cluster **import** DBSCAN  
**from** sklearn.datasets **import** make\_moons *# Para generar datos en forma de lunas*  
  
*# Generar datos simulados en forma de lunas*  
*# Estos datos representan puntos que forman dos lunas crecientes con algo de ruido*  
X, \_ **=** make\_moons(n\_samples**=**300, noise**=**0.1, random\_state**=**42)  
  
*# Aplicar el algoritmo DBSCAN a los datos*  
*# eps es la distancia máxima para considerar dos puntos como vecinos*  
*# min\_samples es el número mínimo de puntos necesarios para formar un cluster*  
dbscan **=** DBSCAN(eps**=**0.2, min\_samples**=**5) *# Ajustamos eps y min\_samples para estos datos*  
y\_dbscan **=** dbscan**.**fit\_predict(X) *# Ajustar el modelo y predecir los clusters para cada punto*  
  
*# Visualizar los clusters y los puntos de ruido*  
plt**.**scatter(X[:, 0], X[:, 1], c**=**y\_dbscan, s**=**50, cmap**=**'viridis', label**=**'Puntos Agrupados')  
plt**.**title("Agrupamiento DBSCAN de Puntos en Forma de Lunas Crecientes") *# Título del gráfico*  
plt**.**xlabel("Coordenada X")  
plt**.**ylabel("Coordenada Y")  
plt**.**legend()  
plt**.**show() *# Mostrar el gráfico*  
  
*# Mostrar los resultados de DBSCAN*  
print("Etiquetas de los clusters asignadas por DBSCAN (incluyendo ruido, si hay):")  
print(y\_dbscan)

## Sistemas de recomendación (Recommender Systems)[¶](#1y810tw)

Los sistemas de recomendación son una parte integral del aprendizaje automático, diseñados para sugerir productos, servicios o contenido a los usuarios basándose en sus preferencias y comportamientos previos. Utilizados ampliamente en la industria, estos sistemas personalizan la experiencia del usuario al predecir lo que podría interesarle, desde películas y música hasta productos de compra y artículos de lectura.

Existen principalmente dos enfoques en los sistemas de recomendación: filtrado colaborativo y filtrado basado en contenido.

1. **Filtrado colaborativo:** este método recomienda elementos basándose en las preferencias y comportamientos de otros usuarios similares.

* Ejemplos:
  + Netflix sugiere películas que otros usuarios con gustos similares han visto y calificado positivamente.
  + Amazon recomienda productos que otros compradores con intereses similares han comprado.

1. **Filtrado basado en contenido:** este enfoque se centra en recomendar elementos que sean similares a aquellos que el usuario ha mostrado interés en el pasado.

* Ejemplos:
  + Spotify sugiere canciones que comparten características similares a las canciones que el usuario ha escuchado anteriormente.
  + YouTube recomienda videos que son similares en tema o estilo a los videos que el usuario ha visto o buscado.

Filtrado basado en contenido[¶](#4i7ojhp)

1. Creamos un conjunto de datos simulado[¶](#2xcytpi)

Creamos un conjunto de datos con 5 películas de cada género para establecer el sistema de recomendaciones.

In [ ]:

**import** pandas **as** pd  
  
*# Crear un DataFrame con información de películas y sus géneros*  
peliculas **=** {  
 'titulo': ['La jungla de cristal', 'La guerra de las galaxias', 'El padrino', 'El caballero oscuro', 'Indiana Jones'],  
 'genero': ['Acción', 'Ciencia ficción', 'Drama', 'Acción', 'Acción']  
}  
  
df\_peliculas **=** pd**.**DataFrame(peliculas) *# Convertir el diccionario en un DataFrame*  
  
*# Mostrar el DataFrame de películas para verificar que se creó correctamente*  
df\_peliculas**.**head()

2. Función de recomendación basada en género[¶](#1ci93xb)

Desarrollaremos una función que recomienda películas basadas en el género de una película dada.

In [ ]:

**def** recomendar\_peliculas(titulo\_pelicula, df):  
 """  
 Esta función recomienda películas basadas en el género de la película dada.  
   
 titulo\_pelicula: título de la película de referencia.  
 df: DataFrame que contiene información de películas y géneros.  
 """  
 *# Obtener el género de la película dada*  
 genero\_pelicula **=** df[df['titulo'] **==** titulo\_pelicula]['genero']**.**values[0]  
   
 *# Encontrar otras películas del mismo género*  
 recomendaciones **=** df[df['genero'] **==** genero\_pelicula]['titulo']**.**tolist()  
   
 *# Remover la película de referencia de la lista de recomendaciones*  
 recomendaciones**.**remove(titulo\_pelicula)  
   
 **return** recomendaciones  
  
*# Probar la función de recomendación*  
print("Recomendaciones para 'La jungla de cristal':", recomendar\_peliculas('La jungla de cristal', df\_peliculas))

Filtrado colaborativo[¶](#3whwml4)

1. Creamos un conjunto de datos simulado de puntuaciones de usuarios[¶](#2bn6wsx)

Creamos un conjunto de datos donde los usuarios han puntuado varias películas, pero una lista de 15 de películas, con el fin de ver más claro el ejemplo.

In [865]:

**import** pandas **as** pd  
**from** sklearn.neighbors **import** NearestNeighbors  
pd**.**reset\_option('display.max\_rows')  
pd**.**reset\_option('display.max\_columns')  
  
*# Crear un diccionario con puntuaciones de usuarios para diferentes películas*  
puntuaciones **=** {  
 'Usuario': ['Usuario 1', 'Usuario 2', 'Usuario 3', 'Usuario 4', 'Usuario 5'],  
 'La jungla de cristal': [5, 4, 5, 5, 2],  
 'La guerra de las galaxias': [4, 5, 4, 4, 3],  
 'El padrino': [1, 0, 1, 1, 5],  
 'El caballero oscuro': [0, 1, 0, 0, 2],  
 'Indiana Jones': [2, 3, 2, 2, 4],  
 'Titanic': [5, 4, 5, 5, 2],  
 'Vengadores: Endgame': [0, 5, 4, 4, 5],  
 'El señor de los anillos': [3, 4, 3, 3, 5],  
 'Harry Potter': [5, 3, 5, 5, 5],  
 'Matrix': [4, 5, 4, 4, 3],  
 'Origen': [5, 4, 5, 5, 4],  
 'Gladiador': [3, 5, 3, 3, 3],  
 'La red social': [5, 4, 5, 5, 4],  
 'Cadena perpetua': [4, 5, 4, 4, 4],  
 'Forrest Gump': [5, 4, 5, 5, 5]  
}  
  
*# Convertir el diccionario en un DataFrame y establecer el índice en 'Usuario'*  
df\_puntuaciones **=** pd**.**DataFrame(puntuaciones)**.**set\_index('Usuario')  
  
*# Mostrar el DataFrame de puntuaciones para verificar que se creó correctamente*  
print("DataFrame de Puntuaciones:")  
df\_puntuaciones**.**head()

DataFrame de Puntuaciones:

Out[865]:

|  | **La jungla de cristal** | **La guerra de las galaxias** | **El padrino** | **El caballero oscuro** | **Indiana Jones** | **Titanic** | **Vengadores: Endgame** | **El señor de los anillos** | **Harry Potter** | **Matrix** | **Origen** | **Gladiador** | **La red social** | **Cadena perpetua** | **Forrest Gump** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Usuario** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Usuario 1** | 5 | 4 | 1 | 0 | 2 | 5 | 0 | 3 | 5 | 4 | 5 | 3 | 5 | 4 | 5 |
| **Usuario 2** | 4 | 5 | 0 | 1 | 3 | 4 | 5 | 4 | 3 | 5 | 4 | 5 | 4 | 5 | 4 |
| **Usuario 3** | 5 | 4 | 1 | 0 | 2 | 5 | 4 | 3 | 5 | 4 | 5 | 3 | 5 | 4 | 5 |
| **Usuario 4** | 5 | 4 | 1 | 0 | 2 | 5 | 4 | 3 | 5 | 4 | 5 | 3 | 5 | 4 | 5 |
| **Usuario 5** | 2 | 3 | 5 | 2 | 4 | 2 | 5 | 5 | 5 | 3 | 4 | 3 | 4 | 4 | 5 |

2. Filtración colaborativa con vecinos más cercanos (Collaborative Filtering with Nearest Neighbors)[¶](#qsh70q)

Utilizaremos el algoritmo de vecinos más cercanos para encontrar usuarios similares y recomendar películas.

In [897]:

**from** sklearn.neighbors **import** NearestNeighbors *# Importar la clase NearestNeighbors de scikit-learn*  
  
*# Crear el modelo de filtración colaborativa basado en vecinos más cercanos*  
modelo\_vecinos **=** NearestNeighbors(metric**=**'cosine', algorithm**=**'brute') *# Usar la distancia coseno como métrica de similitud*  
modelo\_vecinos**.**fit(df\_puntuaciones**.**values) *# Ajustar el modelo a los datos de puntuaciones de los usuarios*  
  
*# Encontrar los usuarios más cercanos a 'Usuario 1'*  
*# Extraer los datos del 'Usuario 1' como array numpy*   
datos\_usuario\_1 **=** df\_puntuaciones**.**loc[['Usuario 1']]**.**values *# .values convierte el DataFrame a array numpy*  
  
*# Encontrar los usuarios más cercanos a 'Usuario 1'*  
distancias, indices **=** modelo\_vecinos**.**kneighbors(datos\_usuario\_1, n\_neighbors**=**3) *# 3 vecinos más cercanos en gustos*  
  
*# Mostrar los usuarios más cercanos a 'Usuario 1'*  
usuarios\_cercanos **=** df\_puntuaciones**.**index[indices**.**flatten()]  
  
*# Obtener los nombres de los usuarios a partir de los índices*  
print("Usuarios más cercanos a 'Usuario 1':", list(usuarios\_cercanos)) *#hacemos un cast a una lista, para evitar imprimir la informacion del objeto*

Usuarios más cercanos a 'Usuario 1': ['Usuario 1', 'Usuario 3', 'Usuario 4']

3. Recomendación basada en vecinos cercanos (Nearest Neighbors Recommendation)[¶](#3as4poj)

Desarrollaremos una función para recomendar películas a un usuario basándonos en los usuarios más similares.

In [899]:

**def** recomendar\_basado\_en\_vecinos(usuario, df, modelo, n\_vecinos**=**3):  
 """  
 Recomienda películas a un usuario basado en los vecinos más cercanos.  
   
 usuario: usuario de referencia.  
 df: DataFrame de puntuaciones de usuarios.  
 modelo: modelo de vecinos más cercanos ajustado a los datos de puntuaciones.  
 n\_vecinos: número de vecinos a considerar.  
 """  
 *# Extraer los datos del usuario como array numpy (sin nombres de características)*  
 datos\_usuario **=** df**.**loc[[usuario]]**.**values  
   
 *# Encontrar los vecinos más cercanos al usuario especificado*  
 distancias, indices **=** modelo**.**kneighbors(datos\_usuario, n\_neighbors**=**n\_vecinos **+** 1) *# n\_vecinos + 1 para excluir al propio usuario*  
   
 *# Obtener los nombres de los vecinos más cercanos, excluyendo al propio usuario*  
 usuarios\_similares **=** df**.**index[indices**.**flatten()]  
 usuarios\_similares **=** [u **for** u **in** usuarios\_similares **if** u **!=** usuario]  
   
 *# Sumar las puntuaciones de los vecinos más cercanos para cada película*  
 puntuaciones\_usuarios\_similares **=** df**.**loc[usuarios\_similares]**.**sum(axis**=**0)  
   
 *# Borrar las películas que el usuario ya ha visto (con puntuaciones no cero)*  
 peliculas\_vistas **=** df**.**loc[usuario][df**.**loc[usuario] **>** 0]**.**index**.**tolist()  
 recomendaciones **=** puntuaciones\_usuarios\_similares**.**drop(peliculas\_vistas)**.**sort\_values(ascending**=False**)  
   
 *# Retornar las mejores recomendaciones (hasta 5 si existen)*  
 **return** recomendaciones**.**head()**.**index**.**tolist()  
  
*# Probar la función de recomendación para 'Usuario 1'*  
print("Recomendaciones para 'Usuario 1':", recomendar\_basado\_en\_vecinos('Usuario 1', df\_puntuaciones, modelo\_vecinos, n\_vecinos **=** 2))

Recomendaciones para 'Usuario 1': ['Vengadores: Endgame', 'El caballero oscuro']

In [ ]: