UNIVERSIDAD TECNOLÓGICA DE CHIHUAHUA

Tecnologías de la información

Extracción de Conocimiento en Bases de Datos

IV.1. Algoritmos de agrupación (25%)

Docente

ING. LUIS ENRIQUE MASCOTE CANO

Alumno

Erick Eduardo Maffiodo Delgado

IDGS 91N

Domingo, 30 de Noviembre del 2025

Índice

Introducción

En el campo de la extracción de conocimiento y el aprendizaje automático, dos técnicas fundamentales son el agrupamiento (clustering) y la reducción de dimensionalidad. El agrupamiento se utiliza para descubrir y segmentar grupos naturales en los datos sin supervisión, es decir, sin conocer etiquetas previas. Permite identificar patrones ocultos organizando objetos o instancias similares en categorías o clusters homogéneos. Por su parte, la reducción de dimensionalidad tiene como objetivo simplificar los datos disminuyendo la cantidad de variables o características, manteniendo al mismo tiempo la mayor parte de la información relevante. Esto ayuda a mitigar la "maldición de la dimensionalidad", facilitar la visualización de datos de alta dimensión y reducir ruido o redundancia en los datos.

En la práctica, ambas técnicas son complementarias. El clustering ayuda a descubrir estructuras o segmentos en conjuntos de datos complejos (por ejemplo, segmentar clientes según comportamientos similares), mientras que la reducción de dimensionalidad permite resumir y visualizar datos con muchísimas variables (por ejemplo, comprimir cientos de atributos en unos pocos componentes principales para graficarlos). A continuación, describiremos los principales algoritmos de agrupamiento y de reducción de dimensionalidad, detallando su funcionamiento, parámetros, ventajas, limitaciones y ejemplos de aplicación.

Algoritmos de agrupación (clustering)

A continuación se presentan tres algoritmos representativos de agrupamiento no supervisado: K-means, clustering jerárquico aglomerativo y DBSCAN. Cada uno ejemplifica un enfoque distinto (particional, jerárquico y basado en densidad, respectivamente).

K-means

Principio de funcionamiento: El algoritmo K-means (o de las K-medias) es un método de agrupamiento particional que organiza los datos en K clústeres basándose en la similitud (distancia) de los puntos a los centroides de cada clúster. Su objetivo es minimizar la suma de las distancias (al cuadrado) dentro de cada grupo; es decir, busca que cada punto esté lo más cerca posible del centroide de su clúster. El proceso iterativo de K-means consiste en alternar dos pasos: (1) Asignación – asignar cada punto de datos al centroide más cercano (según distancia Euclidiana u otra métrica), y (2) Actualización – recalcular la posición de cada centroide como la media de los puntos asignados a su clúster. Estos pasos se repiten hasta que los centroides convergen (ya no cambian significativamente) o se alcanza un número máximo de iteraciones. En esencia, K-means implementa una forma de algoritmo EM (Expectation-Maximization) simple: en la etapa E asigna puntos a clústeres dados los centroides actuales, y en la etapa M recalcula los centroides dados los puntos asignados.

Parámetros clave: El parámetro principal de K-means es K, el número de clústeres deseados, que típicamente se debe definir a priori. Adicionalmente, es importante la inicialización de centroides: por defecto se suelen elegir aleatoriamente, pero métodos como k-means++ mejoran la inicialización para lograr mejores resultados. Otros parámetros incluyen el criterio de convergencia (por ejemplo, tolerancia mínima de cambio en centroides) y el número de iteraciones máximas permitidas. También se puede escoger la métrica de distancia (Euclidiana es la habitual). En implementaciones prácticas (p. ej. sklearn), suele haber un parámetro para el número de reinicios o inicializaciones diferentes (n_init) para mitigar soluciones locales – el algoritmo siempre converge pero podría hacerlo a un óptimo local.

Ventajas: K-means es un algoritmo popular por varias razones:

- **Simple y fácil de implementar:** su lógica es sencilla y comprensible.

- **Eficiente en cómputo:** se escala bien a grandes conjuntos de datos, con complejidad aproximadamente lineal $O(n \cdot K \cdot t)$ (n datos, K clústeres, t iteraciones).

- **Convergencia garantizada:** con un número fijo de puntos y K fijo, el algoritmo siempre termina (aunque puede converger en un óptimo local).

- **Adaptable a nuevos datos:** una vez calculados los centroides, se pueden asignar rápidamente nuevas instancias al clúster más cercano (actualizando centroides de ser necesario).

- **Generalizable (con variantes):** En su forma básica asume clústeres aproximadamente esféricos de similar densidad, pero existen extensiones para permitir **formas elípticas** o distintas varianzas por clúster (por ejemplo, usando **modelos de mezcla gaussianos**, equivalentes a una generalización de K-means).

Limitaciones: A pesar de su utilidad, K-means presenta varias **limitaciones** importantes:

- **Debe definirse K manualmente:** el usuario debe saber o probar cuántos clústeres son adecuados, lo cual no siempre es evidente de antemano. Existen métodos como la "**técnica del codo**" o el **coeficiente de silueta** para ayudar a elegir K, pero sigue siendo un parámetro externo.

- **Sensibilidad a la inicialización:** los resultados dependen de la elección inicial de los centroides. Una mala inicialización puede conducir a un óptimo local pobre. Por ello es aconsejable ejecutar K-means varias veces con diferentes semillas e importar el mejor resultado (o usar inicialización k-means++ que dispersa mejor los centroides inicialmente).

- **Supone clústeres esféricos de similar tamaño/densidad:** K-means funciona bien si los grupos tienen varianzas similares; pero tiene dificultades para agrupar datos con **clústeres de tamaños o densidades muy diferentes**. Sin adaptaciones, tiende a crear clústeres de volumen similar y puede fallar en detectar estructuras de forma irregular.

- **Sensibilidad a valores atípicos:** puntos *outlier* pueden desplazar significativamente los centroides o incluso formar clústeres aislados de un solo punto. K-means no tiene un mecanismo para ignorar outliers; es común eliminarlos o reducir su impacto antes de aplicar el algoritmo.

- **Escalabilidad en alta dimensión:** conforme aumenta la dimensionalidad, las distancias euclidianas se vuelven menos discriminativas (efecto de la concentración de la distancia). En espacios de alta dimensión, todos los puntos tienden a estar a distancias similares, dificultando la formación de clústeres significativos. Por ello, suele ser necesario realizar primero una reducción de dimensionalidad (por ejemplo con **PCA**) antes de aplicar K-means en datos de dimensionalidad muy alta.

Ejemplo de aplicación (pseudocódigo): A continuación se ilustra el proceso de K-means en pasos simplificados:

1. **Inicialización:** elegir KKK y seleccionar KKK centroides iniciales (por ejemplo, puntos aleatorios del espacio o datos escogidos aleatoriamente).

2. **Asignar puntos a centroides:** para cada punto de datos, calcular su distancia a cada centroide y asignarlo al clúster cuyo centroide esté más cercano.

3. **Recalcular centroides:** actualizar la posición de cada centroide como la media (promedio) de todos los puntos asignados a ese clúster.

4. **Iterar:** repetir los pasos 2 y 3 hasta que no cambien las asignaciones de los puntos (convergencia) o se alcance el número máximo de iteraciones.

5. **Resultado:** se obtienen K clústeres definidos por los puntos agrupados y los centroides finales. Se puede opcionalmente evaluar la calidad del agrupamiento (ej. varianza intra-clúster total, **inercia** o SSE) y comparar con otras ejecuciones o valores de K.

Como ejemplo concreto, si aplicamos K-means con K=3K=3K=3 para segmentar clientes en un conjunto de datos de marketing, el algoritmo podría dividir automáticamente a los clientes en tres grupos (por ejemplo: "ahorradores", "compradores moderados" y "grandes gastadores") basándose en patrones similares de comportamiento de compra. Cada grupo estaría centrado alrededor de un **centroide** que representa el perfil medio de ese segmento de clientes.

Clustering jerárquico (aglomerativo)

Principio de funcionamiento: El **agrupamiento jerárquico** construye una jerarquía de clústeres, representada típicamente en un **dendrograma** (diagrama de árbol). Existe un enfoque **aglomerativo** (de abajo hacia arriba) y uno **divisivo** (de arriba hacia abajo). El más común es el **aglomerativo (AGNES)**: comienza considerando cada punto de datos como un clúster individual y luego, repetidamente, **fusiona los dos clústeres más similares** hasta que finalmente todos los puntos quedan en un único clúster (o hasta cierto criterio de parada). En cada paso de fusión, la definición de "más similares" depende de un **criterio de enlace (linkage)** que determina la distancia entre clústeres a partir de las distancias entre sus elementos. Por ejemplo, en **vinculación simple** la distancia entre dos clústeres es la distancia mínima entre cualquier par de puntos (uno de cada clúster), en **vinculación completa** es la distancia máxima entre puntos de ambos clústeres, y en **vinculación promedio** es la distancia promedio entre todos los pares. Otro método popular es el **método de Ward**, que fusiona los clústeres cuya unión produce el menor incremento en la varianza intra-clúster (minimiza la suma de cuadrados intra-clúster). El resultado del algoritmo aglomerativo es un dendrograma que muestra en qué orden se fusionaron los clústeres y la distancia a la que se unieron.

En contraste, el enfoque **jerárquico divisivo** (por ejemplo, algoritmo DIANA) comienza con *todos* los puntos en un único clúster y va **dividiéndolo** recursivamente en clústeres más pequeños. Este método es conceptualmente inverso al aglomerativo y suele ser más costoso, aunque a veces puede capturar mejor ciertas estructuras globales. En la práctica, el método aglomerativo es más usado por su simplicidad.

Parámetros clave: Los algoritmos jerárquicos tienen como parámetros principales la **métrica de distancia** usada para calcular la similitud entre puntos (por ejemplo Euclidiana, Manhattan, coseno, etc.) y el **criterio de enlace (linkage)** para calcular distancia entre clústeres. Estas elecciones determinan la forma del dendrograma. A diferencia de métodos particionales, **no requieren especificar de antemano el número de clústeres K**; este puede decidirse posteriormente "cortando" el dendrograma a cierta altura. Sin embargo, se puede establecer un **criterio de parada** alternativo, por ejemplo, detener las fusiones cuando la distancia mínima entre clústeres exceda un umbral, o cuando se hayan obtenido K clústeres. Otro aspecto paramétrico es el manejo de los datos: conviene estandarizar las variables si están en diferentes escalas, ya que la distancia euclidiana es sensible a la escala de las características.

Ventajas: El clustering jerárquico ofrece varias ventajas destacables:

- **No requiere elegir K a priori:** podemos generar el dendrograma completo y luego decidir el número de clústeres óptimo según la estructura observada (por ejemplo, buscando un "salto" grande en la distancia de fusión). Esta flexibilidad es útil en exploración de datos cuando no sabemos cuántos grupos esperar.

- **Resultados interpretables visualmente:** el **dendrograma** brinda una representación intuitiva de cómo se agrupan progresivamente los puntos. Podemos ver relaciones entre instancias y subgrupos anidados, lo cual ayuda a comprender la estructura de los datos a múltiples niveles de granularidad.

- **Flexibilidad en forma de clústeres:** dependiendo del criterio de enlace y la métrica de distancia elegidos, el método puede capturar diversos tipos de formas de clúster (alargados con single-linkage, compactos con complete-linkage, etc.). También es posible

utilizar distancias especializadas (por ejemplo, distancia de edición para secuencias, distancia Jaccard para conjuntos), lo que hace al método aplicable a diferentes dominios y tipos de datos.

- **Anidamiento de grupos:** a diferencia de K-means que produce una partición plana, el método jerárquico muestra **subestructura** – es decir, clústeres pueden contener sub-clústeres. Esto es útil, por ejemplo, en biología para ver subclases dentro de clases en un árbol filogenético, o en segmentación de clientes para ver subsegmentos dentro de un segmento más general.

Limitaciones: Entre las principales desventajas o desafíos del clustering jerárquico se encuentran:

- **Alta complejidad computacional:** la estrategia aglomerativa típica tiene complejidad al menos *O(n^2)* (debido al cálculo y mantenimiento de la matriz de distancias), llegando a *O(n^3)* en implementaciones ingenuas, lo que la vuelve **poco escalable para conjuntos de datos muy grandes**. En problemas con decenas de miles de puntos, puede ser impracticable sin optimizaciones.

- **Sensibilidad al ruido y outliers:** unos pocos **valores atípicos** pueden distorsionar notablemente la estructura del dendrograma. Por ejemplo, un outlier puede quedar fusionado tempranamente con algún clúster, alterando las distancias de fusión posteriores. A diferencia de métodos como DBSCAN, el clustering jerárquico estándar **no tiene un mecanismo de "ruido"**: todos los puntos acabarán en algún clúster del árbol, por lo que outliers influyen en la jerarquía (se suele preprocesar eliminando o aislando outliers antes de aplicar el algoritmo).

- **Fusiones irrevocables:** una vez que dos clústeres se fusionan, **no se puede deshacer** esa decisión (es un algoritmo codicioso). Si se comete una fusión "equivocada" temprano, no hay corrección posterior, pudiendo obtener agrupamientos subóptimos. Esto contrasta con

K-means, donde las asignaciones pueden reajustarse iterativamente.

- **Elección de corte de dendrograma:** aunque no hay que fijar K de antemano, igualmente el analista debe decidir **dónde "cortar" el dendrograma** para obtener un conjunto final de clústeres. Esta decisión puede ser subjetiva si la estructura no tiene saltos claros; se pueden usar métricas (coeficiente de silueta, inconsistencia, etc.) pero no siempre es obvio.

- **Almacenamiento de distancias:** para datos muy numerosos, mantener la **matriz de distancias** completa de tamaño n×nn \times nn×n puede ser pesado en memoria. Existen aproximaciones, pero es una consideración práctica en implementaciones.

Ejemplo de aplicación (pseudocódigo): A continuación se presenta un esquema simplificado del algoritmo aglomerativo:

1. **Inicialización:** calcular la matriz de distancias entre todos los puntos. Inicializar cada instancia como un clúster individual.

2. **Búsqueda del par más cercano:** encontrar los dos clústeres distintos AAA y BBB cuya distancia (según el criterio de enlace elegido) sea mínima en la matriz de distancias actual.

3. **Fusión:** unir AAA y BBB en un único clúster nuevo C=A∪BC = A \cup BC=A∪B. Registrar la distancia a la cual se fusionaron en el dendrograma.

4. **Actualizar distancias:** eliminar AAA y BBB de la matriz y agregar CCC. Calcular las distancias de CCC con todos los demás clústeres restantes (aplicando la fórmula del criterio de enlace correspondiente, p. ej. promedio de distancias, etc.).

5. **Iterar:** repetir pasos 2–4 hasta que todos los datos estén en un solo clúster (o hasta obtener el número deseado de clústeres).

6. **Construcción del dendrograma:** a partir del historial de fusiones, construir el dendrograma que visualiza el orden de fusiones y las distancias a cada nivel. Luego **decidir el corte** del dendrograma según la cantidad de clústeres que se quiera obtener (por ejemplo, cortar en un nivel de distancia que genere kkk grupos finales).

Como ejemplo concreto, supongamos un análisis jerárquico de *clientes* basado en similitud de patrones de compra. El algoritmo podría empezar con cada cliente separado y luego fusionar primero los dos clientes más parecidos. Gradualmente, puede fusionar pequeños grupos en grupos más grandes. El resultado quizá sea un dendrograma donde al cortar a cierta altura se distinguen, por ejemplo, **3 grandes segmentos** de clientes. Estos segmentos podrían interpretarse como *clientes de compras básicas*, *clientes de lujo* y *clientes esporádicos*, cada uno subdividido a su vez en subgrupos menores si examinamos el dendrograma con más detalle. La ventaja es que el dendrograma permite experimentar con distintos números de segmentos de forma visual e intuitiva.

DBSCAN

Principio de funcionamiento: DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de agrupamiento basado en densidad. Su idea central es identificar "**regiones densas**" de puntos en el espacio de datos y separarlas por regiones de menor densidad, formando clústeres de forma arbitraria y marcando los puntos aislados como **ruido**. A diferencia de K-means o métodos jerárquicos, DBSCAN **no requiere conocer el número de clústeres de antemano**, sino que depende de dos parámetros: una distancia ε\varepsilonε (epsilon) y un número mínimo de puntos (**MinPts**). En términos sencillos, DBSCAN clasifica los puntos en tres categorías:

- **Puntos núcleo (core points):** aquellos que tienen al menos **MinPts** vecinos dentro de un radio ε\varepsilonε. Son el centro de regiones densas.

- **Puntos frontera (border points):** no cumplen ellos mismos con la densidad mínima (menos de MinPts vecinos en ε\varepsilonε), pero están dentro del alcance ε\varepsilonε de algún punto núcleo. Forman el borde de un clúster.

- **Puntos de ruido (outliers):** puntos que no son núcleo ni frontera; quedan demasiado aislados (menos de MinPts vecinos y fuera del alcance de cualquier núcleo). Estos se consideran **ruido** y no pertenecen a ningún clúster.

El algoritmo DBSCAN procede iterando sobre los puntos no visitados: cuando encuentra un punto núcleo, crea un nuevo clúster y agrega a él todos los puntos que estén a distancia ≤ε\leq \varepsilon≤ε de ese núcleo. Luego expande recursivamente el clúster: cualquier vecino que resulte ser núcleo (suficientemente denso) a su vez incorporará a *sus* vecinos, y así sucesivamente, propagando la "alcanzabilidad por densidad". Este proceso continua hasta que no se puedan agregar más puntos al clúster actual. Luego DBSCAN pasa a otro punto no visitado y repite el proceso, formando otro clúster si es un núcleo, o marcándolo como ruido si no cumple densidad. Al final, todos los puntos habrán sido marcados ya sea como pertenecientes a algún clúster o como ruido.

Parámetros clave: Los dos parámetros fundamentales de DBSCAN son **ε (epsilon)** y **MinPts**:

- **ε (radio de vecindad):** define la distancia máxima para considerar que dos puntos son vecinos. Es básicamente el tamaño del vecindario en el espacio de características.

- **MinPts (mínima cantidad de puntos):** el número mínimo de vecinos (incluyendo el propio punto) que se requieren dentro del radio ε para considerar a un punto como núcleo (punto central de un clúster).

Elegir estos parámetros es crítico: **ε** demasiado grande puede juntar clústeres distintos en uno, mientras que demasiado pequeño puede dejar datos verdaderamente agrupados sin unir. **MinPts** suele elegirse en función de la dimensionalidad (una regla básica es MinPts ≈ 2 * dim), pero también debe reflejar cuán denso se espera que sea un clúster. Además de estos, se utiliza una

métrica de distancia (Euclidiana por defecto) para calcular distancias entre puntos. Herramientas prácticas para ajustar ε incluyen inspeccionar la gráfica *k-distances* (distancia al k-ésimo vecino más cercano) buscando el "codo" donde empieza a crecer rápidamente – ese valor suele ser un ε razonable.

Ventajas: DBSCAN ofrece varias ventajas notables, especialmente frente a algoritmos de clustering tradicionales:

- **No requiere especificar el número de clústeres:** a diferencia de K-means o aglomerativos (que eventualmente requieren elegir corte), DBSCAN **determina automáticamente cuántos clústeres** se forman según la distribución de densidades en los datos. Esto es muy útil en exploración cuando la estructura de clústeres es desconocida.

- **Detecta formas arbitrarias:** puede encontrar clústeres de **cualquier forma** – no asume convexidad ni formas esféricas. Por ejemplo, puede descubrir con facilidad conjuntos en forma de media luna, anillos, espirales, etc., donde algoritmos como K-means fracasarían.

- **Robusto a outliers:** DBSCAN tiene la capacidad de **identificar ruido** explícitamente. Los puntos aislados simplemente quedan etiquetados como ruido y no distorsionan los clústeres. Esto le permite manejar conjuntos de datos con datos atípicos sin que afecten el agrupamiento principal, una ventaja crucial en escenarios del mundo real con datos ruidosos.

- **Clústeres con densidad variable:** hasta cierto punto, DBSCAN puede manejar clústeres con diferentes densidades locales mejor que K-means (que presupone densidades similares). Por ejemplo, puede descubrir tanto un clúster muy denso como otro más disperso en el mismo conjunto de datos, siempre que se elijan parámetros apropiados.

- **Identifica regiones de interés en datos espaciales:** dado su fundamento en densidad, es muy adecuado para aplicaciones geoespaciales, visión por computadora y demás, donde

agrupar puntos próximos tiene significado (por ejemplo, encontrar “hotspots” geográficos de actividad).

Limitaciones: También es importante reconocer las limitaciones de DBSCAN:

- **Sensibilidad a los parámetros:** la efectividad de DBSCAN depende fuertemente de elegir bien ε y MinPts. Una **mala elección de parámetros** puede producir resultados pobres (por ejemplo, clúster único gigante o al contrario demasiados clústeres fragmentados). Ajustar estos valores suele requerir conocimiento del dominio o experimentación (gráfica k-distancia, etc.).

- **Dificultad en alta dimensionalidad:** en espacios de muchas dimensiones, la noción de densidad se vuelve difusa porque las distancias entre puntos tienden a homogeneizarse (maldición de la dimensionalidad). Como resultado, puede ser difícil encontrar un ε apropiado que delimite densidades significativas. DBSCAN funciona mejor en dimensiones bajas-moderadas o después de reducir la dimensionalidad.

- **Problemas con densidades muy desiguales:** aunque maneja densidades diferentes mejor que K-means, si en el *mismo* conjunto hay regiones con densidades extremadamente distintas, un único par de parámetros ε/MinPts puede no ser adecuado para todos. Áreas de muy alta densidad podrían fragmentarse o, al contrario, áreas de baja densidad podrían no detectarse como clúster sin englobar ruido. En esos casos, variantes como HDBSCAN (DBSCAN con epsilon variable) pueden rendir mejor.

- **Coste computacional en grandes bases:** DBSCAN tiene una complejidad promedio *O(n log n)* (usando estructuras espaciales eficientes) pero en el peor caso *O(n^2)*, lo cual para datasets muy grandes puede ser pesado. No es tan eficiente como K-means en datasets enormes, especialmente si la distancia de vecindad requiere cálculos costosos. La implementación puede aprovechar *k-d trees* o *ball trees*, pero sigue siendo **menos escalable que algoritmos más simples** para cientos de miles de puntos.

Ejemplo de aplicación (pseudocódigo): A continuación se describe el algoritmo DBSCAN de forma resumida:

1. **Inicializar:** definir ε y MinPts. Marcar todos los puntos como no visitados.

2. **Iterar puntos:** tomar un punto no visitado PPP. Marcar PPP como visitado. Obtener el conjunto N=N = N= vecinos de PPP que están dentro de la distancia ε.

    - Si ∣N∣<MinPts|N| < \text{MinPts}|N|<MinPts: marcar PPP como **ruido** (temporalmente; podría ser frontera si luego se asocia a un núcleo vecino). Pasar al siguiente punto.

    - Si ∣N∣≥MinPts|N| \geq \text{MinPts}|N|≥MinPts: PPP es un **punto núcleo**. Comenzar un **nuevo clúster** y agregar PPP a ese clúster. Inicializar una lista de puntos por procesar con los vecinos en NNN.

3. **Expansión del clúster:** mientras la lista de vecinos por procesar no esté vacía, tomar un punto QQQ de esa lista:

    - Marcar QQQ como visitado. Encontrar sus vecinos NQN_QNQ (puntos a distancia ≤ ε de QQQ).

    - Si ∣NQ∣≥MinPts|N_Q| \geq \text{MinPts}|NQ∣≥MinPts (o sea, QQQ es núcleo), entonces agregar a la lista todos los vecinos de QQQ que aún no hayan sido visitados **ni** ya pertenezcan al clúster. (Estos se unirán al clúster y ampliarán potencialmente su frontera).

    - Si QQQ no es núcleo pero fue alcanzado por estar en la vecindad de otro núcleo, simplemente se marca como **punto frontera** del clúster.

4. **Continuar:** repetir el paso 2 con otro punto no visitado, creando nuevos clústeres según se encuentren nuevos puntos núcleo, hasta haber visitado todos los puntos. Los puntos

marcados como ruido que resulten estar dentro de ε de algún núcleo habrán sido reclasificados como frontera durante la expansión de ese clúster; los que queden como ruido de forma definitiva no pertenecerán a ningún clúster.

5. **Resultado:** se obtiene una partición en clústeres (cada uno con sus núcleos y fronteras) y un conjunto de puntos aislados identificados como **outliers**.

Para ilustrar, imaginemos un **ejemplo** con datos geográficos: supongamos tener coordenadas de ubicaciones de restaurantes en una ciudad y queremos encontrar "zonas con alta concentración" de restaurantes (clústeres) y aislar los restaurantes muy alejados de otros (outliers). Usando DBSCAN, con un ε equivalente a 500 metros y MinPts = 5, el algoritmo detectaría grupos de al menos 5 restaurantes en radio de 500 m entre sí. Así identificaría barrios densos en oferta gastronómica (por ejemplo, una zona de muchos restaurantes juntos formaría un clúster) y marcaría como ruido aquellos locales muy aislados sin vecinos cercanos. Este enfoque se ha utilizado en *minería de datos espacial* para hallar **hotspots** urbanos, por ejemplo identificando agrupaciones de delitos en criminología o de eventos sismológicos en geología. En resumen, DBSCAN encontraría automáticamente cuántas zonas densas existen en la ciudad sin que especifiquemos ese número, y señalaría explícitamente los restaurantes fuera de cualquier zona densa.

Algoritmos de reducción de dimensionalidad

Ahora presentaremos dos técnicas comunes para la reducción de dimensionalidad: **Análisis de Componentes Principales (PCA)** y **t-SNE** (*t-distributed Stochastic Neighbor Embedding*). Aunque existen métodos lineales y no lineales (y tanto supervisados como no supervisados), hemos elegido PCA como ejemplo clásico lineal no supervisado, y t-SNE como ejemplo moderno no lineal especialmente útil para visualización. Estas técnicas buscan **proyectar los datos originales de alta dimensión en espacios de dimensión inferior** preservando cierta estructura (varianza global en PCA, estructura de proximidad local en t-SNE).

Análisis de Componentes Principales (PCA)

Fundamento conceptual: El **Análisis de Componentes Principales (ACP o PCA, por sus siglas en inglés)** es un método estadístico lineal que proyecta los datos a un nuevo sistema de coordenadas, de menor dimensión, maximizando la **varianza explicada**. En esencia, PCA encuentra **nuevas variables "componentes" ortogonales** (no correlacionadas entre sí) que son combinaciones lineales de las variables originales. El primer componente principal es la dirección (vector) en el espacio original a lo largo de la cual los datos tienen mayor varianza; el segundo componente es la dirección ortogonal a la primera con la siguiente mayor varianza, y así sucesivamente. De este modo, PCA **condensa la mayor información posible** (en términos de variabilidad de los datos) en las primeras componentes, permitiendo descartar las últimas con poca pérdida de información. Matemáticamente, los componentes principales son los **autovectores** (vectores propios) de la matriz de covarianza (o de la matriz de datos centrados) ordenados según sus **autovalores** (valores propios) de mayor a menor. Cada autovalor indica cuánta varianza del dato original captura el correspondiente autovector – por eso se suele seleccionar un número de componentes tal que la suma de sus autovalores explique un porcentaje alto (por ej. 95%) de la varianza total.

En resumen, PCA realiza una **transformación ortogonal** del espacio original a un espacio de menor dimensión, alineado con las direcciones de mayor varianza de los datos. Es un método **no supervisado** (no utiliza etiquetas de clase) y **lineal** (basado en combinaciones lineales), lo que significa que preserva bien relaciones lineales pero no captura patrones altamente no lineales.

Parámetros clave: PCA en sí no requiere parámetros de aprendizaje como tal (es un cálculo directo mediante álgebra lineal). El parámetro principal a elegir es **el número de componentes principales (m)** a retener. Uno puede decidir m de varias formas: un número fijo (por ejemplo, reducir de 100 dimensiones a 2 para visualizar), o basado en la **varianza acumulada** (ej. elegir el mínimo m tal que $\geq 90\%$ de la varianza esté explicada). En problemas de compresión suele importarnos retener suficiente varianza con pocos componentes. Otra decisión es si **estandarizar** los datos antes de PCA: dado que PCA depende de la varianza, es importante escalar las variables para que ninguna con escala mayor domine la primera componente. Por ejemplo, si una variable está medida en millones y otra en fracciones, sin escalado la primera monopolizará la varianza. Por ello, suele aplicarse escalamiento (estandarización a media 0, varianza 1) previo. Existen variantes como **Kernel PCA** (no lineal) y **PCA robusto**, pero el método clásico no tiene

más hiperparámetros. En resumen, el usuario elige cuántos componentes tomar y se asegura de preparar adecuadamente los datos de entrada.

Ventajas: PCA es una de las técnicas más utilizadas por varias razones:

- **Reduce la dimensionalidad con mínima pérdida de información:** logra resumir decenas o cientos de variables originales en unos pocos componentes que retienen la mayor parte de la variabilidad de los datos. Esto simplifica los datos, ahorra almacenamiento y computación, y puede mejorar la eficiencia de algoritmos posteriores (p. ej. clasificadores).

- **Elimina colinealidad y ruido:** al producir componentes ortogonales, PCA **resuelve problemas de multicolinealidad** (variables altamente correlacionadas) combinándolas en componentes únicas. También actúa como filtrado de ruido al descartar componentes asociadas a varianza muy baja que a menudo corresponden a ruido o detalles insignificantes. Esto puede mejorar la robustez de modelos predictivos reduciendo el sobreajuste.

- **Facilita la visualización y exploración:** es muy útil para **visualizar datos de alta dimensión** en 2D o 3D. Por ejemplo, con PCA podemos proyectar un dataset de 50 variables en los primeros 2 componentes y graficarlos para detectar patrones, agrupaciones o outliers fácilmente. De hecho, PCA es a menudo el primer paso en análisis exploratorio de datos complejos.

- **Computacionalmente factible:** el cálculo de PCA se reduce a operaciones matriciales bien estudiadas (descomposición en valores propios o SVD). Para tamaños moderados es muy rápido, y existen técnicas para grandes datasets también (por ejemplo, PCA incremental). Además, una vez calculados los componentes, proyectar nuevos datos es extremadamente rápido (multiplicación matriz-vector).

- **Aplicaciones amplias:** PCA se ha aplicado en infinidad de campos – desde compresión de imágenes (*eigenfaces* para reconocimiento facial), análisis de genética (reduciendo

miles de genes a componentes principales), hasta finanzas (análisis de factores de mercado). Siempre que se necesite **reducir complejidad** manteniendo información, PCA es una opción natural y bien fundamentada.

Limitaciones: A pesar de sus virtudes, PCA tiene algunas limitaciones importantes a considerar:

- **Asume relaciones lineales:** PCA solo capta la estructura lineal de los datos. Si las relaciones importantes entre variables son **no lineales**, PCA no las representará eficientemente. En esos casos, pueden ser más adecuados métodos no lineales (como t-SNE, UMAP, autoencoders) que preservan otras propiedades de los datos.

- **Componentes poco interpretables:** las nuevas variables (componentes) son combinaciones de las originales, a menudo involucrando todos los atributos. Esto dificulta la interpretación *directa* de su significado. Por ejemplo, la "Componente 1" podría ser 0.7*VariableA – 0.2*VariableB + 0.1*VariableC..., etc., lo cual no siempre tiene una interpretación inmediata en términos del problema. A diferencia de seleccionar un subconjunto de variables originales, PCA mezcla las características, por lo que perdemos claridad sobre qué variables originales influyen más en cada componente (aunque se puede inspeccionar los *loadings* para entenderlo parcialmente).

- **Sensibilidad a la escala y a outliers:** si no se escalan adecuadamente las variables, PCA dará más peso a las que tengan mayor varianza absoluta (que podría ser simplemente por unidades más grandes). Asimismo, PCA utiliza la varianza que es sensible a valores extremos: unos pocos outliers pueden influir en la dirección de los componentes principales. Es recomendable limpiar outliers y normalizar datos antes de aplicar PCA para obtener resultados más significativos.

- **No considera información de la variable respuesta:** PCA es no supervisado; si nuestro fin es predecir una variable objetivo (clasificación/regresión), PCA no garantiza que las componentes principales más importantes para varianza sean las más relevantes para la predicción. De hecho, podría descartar información de variabilidad baja pero crucial para

distinguir clases. En esos casos existen métodos supervisados como **LDA (Análisis Discriminante Lineal)** que optimizan la separación de clases en lugar de la varianza global.

Ejemplo sencillo (pseudocódigo): A continuación describimos el procedimiento típico para aplicar PCA a un conjunto de datos:

1. **Preprocesamiento:** obtener la matriz de datos XXX de dimensión n×pn \times pn×p (n muestras, p variables). Opcionalmente estandarizar cada columna (restar media y dividir por desviación estándar) para igualar escalas.

2. **Calcular covarianzas:** computar la matriz de covarianza S=1n−1XTXS = \frac{1}{n-1} X^T XS=n−11XTX (si los datos están centrados). Alternativamente, calcular la **descomposición en valores singulares (SVD)** de la matriz de datos.

3. **Descomposición en autovalores:** obtener los **autovalores** λ1≥λ2≥...≥λp\lambda_1 \ge \lambda_2 \ge ... \ge \lambda_pλ1≥λ2≥...≥λp y **autovectores** asociados v1,v2,...vpv_1, v_2, ... v_pv1,v2,...vp de la matriz de covarianza. Cada autovector viv_ivi es una dirección en el espacio original (combinación de variables originales) y su autovalor λi\lambda_iλi indica cuánta varianza de los datos hay en esa dirección.

4. **Seleccionar componentes principales:** elegir los primeros m autovectores correspondientes a los mayores autovalores (por ejemplo, los primeros m que suman cierto porcentaje de varianza acumulada). Estos v1,...,vmv_1,...,v_mv1,...,vm formarán la base ortonormal del subespacio reducido.

5. **Proyectar los datos:** calcular Z=X⋅[v1 v2 ... vm]Z = X \cdot [v_1 \ v_2 \ ... \ v_m]Z=X⋅[v1 v2 ... vm], obteniendo así una nueva matriz ZZZ de dimensión n×mn \times mn×m. Cada fila de ZZZ son las coordenadas de una muestra original en el espacio de componentes principales. Este es el conjunto de datos reducido.

6. **Analizar resultados:** utilizar ZZZ para los fines deseados – por ejemplo, graficar ZZZ si m=2 o 3, usar ZZZ como entrada para un algoritmo de clustering o de predicción, etc. Evaluar cuánta **varianza explicada** representan estos m componentes (relación ∑i=1mλi/∑i=1pλi\sum_{i=1}^m \lambda_i / \sum_{i=1}^p \lambda_i∑i=1mλi/∑i=1pλi). Si es baja, quizás tomar más componentes.

Como **ejemplo concreto**, supongamos el famoso *conjunto de datos Iris* (4 características de flor para 150 muestras de iris de 3 especies). Aplicando PCA, podríamos encontrar que las **primeras 2 componentes** explican ~95% de la varianza total. Proyectando las 150 muestras en estos dos componentes y graficándolas, veríamos que las especies de iris Setosa, Versicolor y Virginica aparecen agrupadas distintamente en el plano principal. Esto no solo confirma que las especies se diferencian principalmente por ciertas combinaciones de las 4 medidas originales, sino que además nos permitió **visualizar en 2D** algo que originalmente requería 4D. De hecho, PCA suele usarse así: por ejemplo, en reconocimiento facial se reducen imágenes (de miles de píxeles) a unas pocas componentes (*eigenfaces*) que capturan las variaciones principales de las caras, facilitando almacenar y clasificar las imágenes con mucha menos información.

t-SNE (t-Distributed Stochastic Neighbor Embedding)

Fundamento conceptual: t-SNE es una técnica de reducción de dimensionalidad **no lineal** y **no supervisada** especialmente diseñada para **visualizar datos de alta dimensión** en 2D o 3D. A diferencia de PCA (que es lineal y global), t-SNE se concentra en **preservar las relaciones de cercanía locales** entre puntos cuando los embebe en un espacio de menor dimensión. En términos simples, t-SNE intenta que si dos puntos eran **vecinos** (muy cercanos) en el espacio original de alta dimensión, aparezcan también cercanos en la proyección de baja dimensión; y si estaban muy alejados, trata de reflejar esa lejanía en lo posible. Técnicamente, t-SNE construye **distribuciones de probabilidad** de vecindad: primero define, para cada par de puntos en el espacio original, una probabilidad (alta si los puntos están cerca, muy baja si están lejos) usando una distribución Gaussiana. Luego define de forma similar una distribución de probabilidades para distancias en el espacio reducido, pero utilizando una **distribución t de Student** (de cola más pesada que la gaussiana). Finalmente, el algoritmo optimiza las coordenadas de los puntos

en baja dimensión para que **las probabilidades de vecindad (similitudes) en el espacio de menor dimensión reproduzcan las del espacio original**. Esta optimización se logra minimizando una divergencia de Kullback-Leibler entre ambas distribuciones mediante descenso de gradiente.

La razón de usar una t-Student en la proyección es mitigar el llamado *crowding problem*: en baja dimensión es difícil conservar simultáneamente todas las distancias; la t-Student permite que puntos que deban estar moderadamente lejanos puedan estar más dispersos. En resumen, **t-SNE preserva bien la estructura local** (quién es vecino de quién) a costa de que las distancias globales (entre grupos lejanos) no necesariamente sean a escala. Por eso t-SNE es muy bueno para visualizar **clústeres o subgrupos** existentes en los datos, aunque las posiciones relativas entre grupos no deban sobre-interpretarse.

Parámetros clave: Aunque t-SNE no requiere especificar un número de clústeres (no es clustering per se), sí tiene **hiperparámetros** importantes que afectan su resultado:

- **Dimensión de salida:** normalmente 2 (para graficar en plano) o 3 (para visualización 3D).

- **Perplejidad:** es quizás el parámetro más crítico de t-SNE. Controla el equilibrio entre atención a la estructura local vs. global. Técnicamente está relacionado con el número efectivo de vecinos que considera cada punto. Valores típicos van de 5 a 50; un valor bajo de perplejidad hace que t-SNE se enfoque en estructuras muy locales (pequeños grupos muy separados), mientras que valores altos consideran patrones más globales (riesgo de sobre-agrupación). En general, se recomienda probar varios valores.

- **Tasa de aprendizaje:** afecta la optimización por gradiente. Si es muy baja, la convergencia puede ser muy lenta; si es muy alta, puede producir resultados inestables. Suele haber un valor por defecto (por ejemplo 200) adecuado para muchos casos.

- **Iteraciones:** número de iteraciones de descenso de gradiente. Suficientes iteraciones aseguran convergencia a un estado estable (por ejemplo, 1000 o 2000 iteraciones; en

algunos casos más).

- **Inicialización:** t-SNE a menudo inicializa los puntos en baja dimensión aleatoriamente o con PCA. La inicialización aleatoria conlleva que distintas corridas puedan dar resultados ligeramente distintos (aunque cualitativamente similares), por lo que suele fijarse una semilla aleatoria para reproducibilidad.

- **Método de cálculo:** para datasets grandes, existe una variante llamada *Barnes-Hut t-SNE* que aproxima los cálculos para ser más rápida (O(nlogn)O(n \log n)O(nlogn) en lugar de O(n2)O(n^2)O(n2)). Herramientas como scikit-learn la implementan. En datasets muy extensos, aún así t-SNE puede ser lento.

Elegir la **perplejidad** es clave: por ejemplo, con perplejidad muy baja t-SNE puede fracturar un clúster grande en muchos pequeños (como si asumiera pocos vecinos por punto), mientras que con perplejidad muy alta puede fundir clústeres distintos en uno (al intentar que muchos puntos se vean próximos). Una recomendación común es empezar con perplejidad ~30 y ajustar según resultados.

Ventajas: t-SNE se ha vuelto popular por su capacidad de producir visualizaciones esclarecedoras de datos complejos:

- **Excelente visualización de estructuras complejas:** es muy eficaz para encontrar y mostrar **grupos naturales** en datos de alta dimensión al proyectarlos en 2D. Por ejemplo, en datos de imágenes de dígitos escritos a mano o en datos genómicos, t-SNE típicamente muestra clústeres bien separados correspondientes a categorías o estados distintos, revelando patrones que PCA u otras técnicas lineales podrían no distinguir.

- **Preserva estructura local:** agrupa correctamente puntos similares. Esto permite descubrir subclústeres o relaciones de vecindad. En datos etiquetados, a menudo los puntos de la misma clase aparecen juntos en el mapa t-SNE, lo cual sugiere que el

algoritmo está captando bien las relaciones intrínsecas.

- **Revela estructura en múltiples escalas:** a diferencia de métodos que se centran solo en la varianza global, t-SNE puede **revelar estructuras a diferentes escalas** en un mismo mapa. Por ejemplo, puede mostrar clústeres grandes y dentro de ellos subgrupos, etc. Es sensible a sutilezas locales sin perder totalmente la visión global (dentro de lo posible en 2D).

- **Manejo implícito de ruido:** puntos que no tenían vecinos cercanos en alta dimensión quedarán dispersos (un poco como ruido) en el mapa, mientras que los que formaban densidades se agruparán. De cierta forma, separa las "masas" significativas y deja aislados los outliers.

- **Amplio uso en exploración de datos:** se ha convertido en una herramienta estándar para visualizar **embeddings** de palabras en NLP, activaciones de capas de redes neuronales, datos de genética de poblaciones, etc. Cuando hay necesidad de "ver" datos en 2D que viven en cientos o miles de dimensiones, t-SNE suele dar mejores resultados que PCA en destacar agrupaciones no triviales.

Limitaciones: A pesar de sus fortalezas, es importante entender las limitaciones y consideraciones de t-SNE:

- **Enfoque en visualización (2D/3D):** t-SNE está principalmente diseñado para reducciones a 2 o 3 dimensiones para fines visuales. No es tan útil si uno quiere reducir a, digamos, 10 dimensiones para luego entrenar un modelo; para eso PCA u otros métodos pueden ser más apropiados.

- **Alto costo computacional:** es más pesado que PCA. La optimización por gradiente puede ser lenta para muchos puntos (miles a decenas de miles). Aunque existen aceleraciones, en datos masivos (>100k puntos) puede ser inviable directamente.

- **No hay función de mapeo explícita:** a diferencia de PCA, que proporciona una transformación lineal aplicable a nuevos datos, t-SNE no genera una función sencilla para proyectar nuevos puntos. Si llega un dato nuevo, básicamente habría que re-ejecutar (o al menos incorporarlo y refinar) el algoritmo. Esto dificulta su uso más allá de la visualización estática del conjunto original.

- **Resultado no determinista (sin fijar semilla):** dado su componente estocástico (inicialización aleatoria, etc.), diferentes ejecuciones pueden producir disposiciones algo distintas de los puntos. Se recomienda fijar random_state para reproducir. Aun con la misma semilla, cambios en hiperparámetros generan diferentes outputs.

- **Interpretación de distancias:** cuidado con interpretar distancias globales en el gráfico resultante. t-SNE no preserva escalas globales: por ejemplo, dos clústeres separados en el gráfico no necesariamente tienen la misma lejanía real que otros dos clústeres separados similar en el gráfico. En general, **la técnica distorsiona las escalas globales** para enfatizar las locales. Por ello, es correcto interpretar qué puntos forman conglomerados, pero no siempre qué tan lejos están conglomerados distintos entre sí.

- **Parámetros a afinar:** como ya se mencionó, la necesidad de calibrar la perplejidad y, a veces, la tasa de aprendizaje, significa que requiere experimentación. Malos parámetros pueden llevar a "gráficos feos" – p. ej., todos los puntos mezclados sin estructura (perplejidad demasiado alta o demasiado baja, etc.).

- **No conserva exactamente densidades o proporciones de grupos:** t-SNE enfatiza igualar distribuciones de similitud, pero en ese esfuerzo puede estirar o comprimir grupos. Por ejemplo, un grupo grande de puntos podría aparecer visualmente del mismo tamaño que uno mediano, dependiendo de cómo distribuya las distancias. La salida es más cualitativa que cuantitativa.

Visualización comparativa de técnicas de reducción de dimensionalidad: La siguiente imagen ilustra la diferencia entre PCA y t-SNE aplicados a un conjunto de datos de **dígitos escritos a**

mano (cada punto representa una imagen de un dígito 0–9, proyectada a 2D). En la proyección de PCA (derecha), los dígitos forman grupos algo superpuestos, mientras que con t-SNE (izquierda) se observan clústeres mucho más separados, agrupando claramente los dígitos por clase. Cada color/etiqueta corresponde a un dígito diferente (0–9). Vemos que t-SNE logra discernir mejor las estructuras intrínsecas, sacrificando la preservación de varianza global a cambio de mantener vecinos juntos.

Cómo funciona (pasos simplificados): El algoritmo t-SNE puede describirse en tres etapas principales:

1. **Calcular similitudes en alta dimensión:** para cada par de puntos en el espacio original de dimensión alta, calcular una medida de similitud. Esto se hace centrando, para cada punto iii, una distribución gaussiana sobre iii y calculando la probabilidad condicional pj∣ip_{j|i}pj∣i de que iii elegiría a jjj como vecino (más cerca implica probabilidad mayor). Luego se define la probabilidad conjunta pijp_{ij}pij de que iii y jjj sean vecinos. El parámetro de **perplejidad** influye en la desviación estándar de estas gaussianas, controlando el alcance del vecindario considerado.

2. **Calcular similitudes en baja dimensión:** distribuir inicial/aleatoriamente los puntos en el plano (o 3D) y definir de modo análogo una probabilidad qijq_{ij}qij de vecindad en el espacio reducido, pero usando una **distribución t-Student** en lugar de gaussiana. La t-Student, con muchos grados de libertad (vinculado a perplejidad), permite distancias relativamente grandes entre puntos conservando cierta probabilidad (evita que todos los puntos lejanos tengan probabilidad prácticamente cero de interacción, como pasaría con Gaussiana).

3. **Minimizar divergencia:** ajustar las posiciones de los puntos en la proyección de baja dimensión tratando de que qijq_{ij}qij se parezca lo más posible a pijp_{ij}pij para todos los pares. Esto se logra minimizando la **divergencia KL** ∑i≠jpijlogpijqij \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}∑i=jpijlogqijpij. Mediante descenso de gradiente, se mueven los puntos iterativamente para reducir esa divergencia. En cada iteración, puntos con qijq_{ij}qij demasiado grandes comparados a pijp_{ij}pij se separarán, y puntos con

qijq_{ij}qij demasiado pequeños (respecto a pijp_{ij}pij) se acercarán. Tras suficientes iteraciones (y quizás reduciendo la tasa de aprendizaje hacia el final), el sistema llega a un **estado estable** donde la divergencia no mejora significativamente (generalmente un mínimo local).

Al finalizar, obtenemos un conjunto de puntos en 2D/3D que podemos graficar para visualizar la estructura de los datos originales.

Ejemplo de uso: Una de las aplicaciones más conocidas de t-SNE es visualizar salidas de redes neuronales o datos de altas dimensiones como palabras embebidas o datos genómicos. Por ejemplo, t-SNE se usó para visualizar en 2D el conjunto **MNIST** de imágenes de dígitos manuscritos (cada imagen de 28×28 píxeles, es decir 784 dimensiones). El algoritmo produce un mapa donde, efectivamente, las imágenes de cada dígito (0, 1, 2, ..., 9) forman grupos bien definidos separados entre sí. Esto sugiere que incluso sin conocer las etiquetas, la estructura de los datos de pixeles tiene “clusters” correspondientes a cada número, y t-SNE pudo capturar esas relaciones locales de similitud (imágenes del mismo número son más parecidas entre sí que a las de otros números). En la imagen incluida arriba se aprecia este resultado: dígitos iguales se agrupan juntos, con muy poca superposición entre diferentes dígitos, lo que no sucede tan claramente con PCA u otros métodos lineales. Otro ejemplo práctico: en procesamiento de lenguaje natural, t-SNE se utiliza para ver en 2D **word embeddings** de alta dimensión (como Word2Vec o GloVe); palabras con significados similares suelen aparecer cercanas en el plano t-SNE, formando grupos semánticos (por ejemplo, países cerca de sus capitales, etc.), brindando una validación visual de la calidad del embedding.

En resumen, t-SNE es la herramienta preferida cuando queremos **“ver” la estructura** de datos complejos, descubrir agrupaciones ocultas o simplemente entender si los datos tienen patrones separables, especialmente en problemas donde una visualización informativa no es posible con técnicas lineales como PCA.

Comparativa y conclusiones

Clustering vs. Reducción de dimensionalidad: Aunque tanto las técnicas de agrupamiento como las de reducción de dimensionalidad forman parte del toolkit de *aprendizaje no supervisado*, cumplen **objetivos diferentes** y no excluyentes. A modo de comparación:

- El **agrupamiento (clustering)** se enfoca en la **organización de instancias**: dado un conjunto de objetos o ejemplos, intenta **separarlos en grupos** homogéneos según su similitud. Su resultado son **etiquetas de clúster** o segmentaciones – por ejemplo, dividir clientes en segmentos, agrupar imágenes similares, etc. La meta es descubrir categorías ocultas o patrones de agrupamiento en los datos. En la mayoría de los algoritmos de clustering, se trabaja aún en el **espacio original de las variables** (o en una transformación explícita de estas, como distancias kernel), pero *no reduce el número de atributos*, sino que produce una asignación de cada instancia a un grupo. Es decir, **el número de variables permanece igual**, solo que añadimos una nueva "variable" que es la etiqueta de clúster (o probabilidad de pertenencia, en métodos difusos).

- La **reducción de dimensionalidad**, por otro lado, se enfoca en la **transformación de variables**: dado un conjunto de características potencialmente muy grande o redundante, intenta **resumirlas en menos variables** informativas. Su resultado es un **nuevo conjunto de atributos** (menos numerosos que los originales) que son combinaciones o selecciones de los iniciales, conservando la estructura de los datos. La meta es simplificar la representación de los datos, eliminando redundancias, ruido y facilitando su manejo o visualización. Importante: esta transformación típicamente aplica *a todos los datos a la vez* (proyectándolos al nuevo subespacio), pero no directamente segmenta instancias ni identifica grupos por sí sola (aunque a veces la estructura baja dimensional obtenida *sugiere* agrupamientos, como vimos con PCA o t-SNE).

Otra manera de diferenciar: **Clustering** actúa en el **espacio de muestras**, agrupando las muestras entre sí. **Reducción de dimensión** actúa en el **espacio de características**, reestructurando cómo representamos cada muestra.

Es común que en un **flujo de análisis de datos**, primero se aplique reducción de dimensionalidad (para limpiar o condensar datos) y luego se aplique clustering sobre esa representación más compacta. De hecho, muchas veces **van de la mano** en problemas complejos: por ejemplo, en bioinformática, se puede usar PCA para reducir miles de genes a unas decenas de componentes relevantes y luego aplicar clustering jerárquico para encontrar tipos de células similares. O en visión, usar un autoencoder para reducir características de imágenes y luego DBSCAN para encontrar grupos de imágenes parecidas.

¿Cuándo usar uno u otro? Depende de la naturaleza del problema:

- Si el objetivo es **identificar grupos o patrones en los datos** (segmentación de clientes, agrupamiento de documentos, detección de comunidades en redes, etc.), entonces necesitamos un algoritmo de **clustering**. Este nos dará directamente la **estructura de clústeres** y asignación de cada punto a un grupo. La reducción de dimensionalidad en este caso puede servir de paso previo (para eliminar ruido o reducir complejidad) pero no entrega por sí sola grupos discretos.

- Si el objetivo es **reducir la complejidad de los datos, visualizar datos, o preparar datos para modelos supervisados**, entonces usamos **reducción de dimensionalidad**. Por ejemplo, si tenemos 1000 variables potencialmente correlacionadas en un dataset, PCA puede simplificarlas a unas pocas componentes principales interpretables o manejables. En cambio, hacer clustering no resuelve la alta dimensionalidad en sí – podría encontrar grupos pero seguiría habiendo 1000 variables que entender.

- En problemas de **preprocesamiento para aprendizaje supervisado**, la reducción de dimensionalidad (PCA, LDA, autoencoders) a veces se usa para evitar sobreajuste o mejorar velocidad, mientras que el clustering puede ser útil para generar nuevas características (por ejemplo, segmentar datos y luego usar la pertenencia a cluster como feature) o para detectar *label noise* o clusters de clases. Pero nuevamente, cumplen roles distintos.

- Hay situaciones donde **ninguno** de los dos es directamente el objetivo, pero aún así son útiles. Por ejemplo, en **detección de anomalías**: DBSCAN puede aislar outliers (ruido) y PCA puede reducir dimensionalidad para luego aplicar métodos de detección de anomalías más fácilmente en menos dimensiones.

- Un detalle: existen métodos supervisados de reducción (LDA) y también se puede pensar en clustering supervisado (clustering guiado por conocimiento), pero en general en la extracción de conocimiento usual estamos hablando de **métodos no supervisados**.

Situaciones prácticas – prioridad de uno sobre otro: Supongamos varios escenarios:

- **Exploración inicial de datos desconocidos:** suele ser útil aplicar primero una reducción de dimensionalidad (por ejemplo PCA) para **visualizar** y entender la estructura global, y a la vez eliminar ruido. Luego, sobre ese espacio reducido (o incluso en el gráfico resultante), uno podría aplicar un algoritmo de clustering o simplemente ver visualmente agrupaciones. Si los datos son muy altos en dimensión (imágenes pixeladas, texto con miles de términos, etc.), **reducción de dimensión es casi un prerrequisito** para cualquier análisis, ya que procesar clustering directamente podría ser computacionalmente costoso o dar resultados poco fiables.

- **Segmentación de clientes a partir de decenas de variables demográficas/de comportamiento:** aquí el objetivo final es clustering (encontrar segmentos de clientes). Sin embargo, si hay muchas variables altamente correlacionadas, uno podría primero usar PCA para condensar la información en ejes principales. No obstante, se debe tener cuidado de no dificultar la interpretabilidad de los segmentos resultantes: a veces es preferible usar las variables originales para clusters interpretables, otras veces PCA ayuda si había mucha colinealidad. En general, si el número de variables es manejable y tienen significado directo, se podría hacer clustering directamente; si es muy alto o ruidoso, se da prioridad a reducir dimensión primero.

- **Clasificación de imágenes o texto – features altamente dimensionales:** aquí, métodos modernos suelen aprender *embeddings* (representaciones de menor dimensión, a menudo con autoencoders o redes neuronales) y luego a partir de esas representaciones más compactas se pueden hacer tareas de agrupamiento o clasificación. Por ejemplo, con autoencoders (reducción no lineal) obtenemos codificaciones de imágenes en un espacio latente de, digamos, 32 dimensiones a partir de 1024 píxeles, y después podríamos hacer K-means en ese espacio latente para agrupar imágenes similares. Entonces la prioridad es primero reducción (aprendizaje de características) y luego agrupamiento en el espacio aprendido.

- **Detección de comunidades en redes sociales:** aquí los datos pueden ser grafos complejos; clustering puede referirse a encontrar comunidades. No hay "dimensionalidad" en el sentido tradicional de variables, así que la reducción de dimensionalidad no aplica directamente; se va directo a clustering (por ejemplo, algoritmos tipo Louvain). Este es un ejemplo donde clustering es la técnica clave y la idea de reducir dimensión solo entraría si convertimos alguna representación en vectores.

- **Análisis de factores vs. segmentación:** en investigación de mercados se distingue entre **análisis factorial** (similar a PCA, reducir ítems de encuesta a factores latentes) y **segmentación de encuestados** (clustering de personas en grupos). Si se quieren entender las **dimensiones subyacentes** en las variables, se aplica PCA primero, y luego quizás se segmenta a las personas según sus puntuaciones en esos factores. Ambos se usan, pero si la prioridad es encontrar segmentos de personas, el clustering tiene el foco final, usando la reducción como paso auxiliar.

En muchos casos, **no se trata de elegir permanentemente uno sobre otro**, sino de saber en qué orden o combinación usarlos. Por ejemplo, **primero reducir dimensionalidad para eliminar ruido, luego clusterizar** para hallar grupos; o **clusterizar por variables para reducir variables agrupándolas en factores (clustering de variables)** y luego crear nuevas variables resumen.

En conclusión, **clustering y reducción de dimensionalidad son técnicas complementarias**: el clustering descubre **agrupaciones ocultas de instancias**, mientras la reducción de dimensionalidad descubre **estructuras ocultas en las variables**. Ambas contribuyen a extraer conocimiento de los datos sin supervisión.

Conclusiones generales

En este reporte hemos revisado algunos de los principales algoritmos de agrupamiento (K-means, jerárquico, DBSCAN) y de reducción de dimensionalidad (PCA, t-SNE), detallando sus fundamentos, fortalezas y debilidades. Cada algoritmo tiene supuestos y objetivos distintos:

- **K-means** es sencillo y eficiente para particionar datos en grupos esféricos, aunque sensible a K, inicialización y outliers.

- **Clustering jerárquico** ofrece una visión completa multiescala de la estructura de clústeres, útil para conjuntos pequeños/medianos donde la interpretabilidad es clave, aunque su costo computacional lo limita en datos grandes.

- **DBSCAN** brilla en detectar clústeres de formas arbitrarias y separar ruido, siendo valioso en datos espaciales y con outliers, siempre que podamos calibrar bien sus parámetros de densidad.

- **PCA** es la técnica estándar para condensar variables, reducir redundancia y preparar datos para modelado o visualización, siempre que las relaciones predominantes sean aproximadamente lineales; de lo contrario, métodos no lineales podrían capturar más.

- **t-SNE** ha revolucionado la exploración visual de datos complejos, permitiendo ver agrupaciones que antes permanecían ocultas en tablas de números, aunque es esencialmente una herramienta de análisis exploratorio más que un método para producción (dada su falta de transformada explícita y costo).

En la práctica de ciencia de datos, a menudo se usan **en conjunto**: por ejemplo, aplicar PCA para reducir ruido y luego DBSCAN para clusterizar en el espacio reducido; o usar un clustering para entender la estructura luego de que PCA ha bajado dimensionalidad para poder graficar. La elección depende del problema: **no existe un "mejor" algoritmo universal**, sino aquel que se ajusta a la distribución de los datos y a la pregunta que queremos responder.

Finalmente, es importante mencionar que tanto clustering como reducción de dimensionalidad requieren *criterios de validación*. En clustering se suelen evaluar medidas como el coeficiente de silueta, homogeneidad, comparación contra etiquetas conocidas si las hay, etc., para juzgar si los grupos tienen sentido. En reducción de dimensionalidad se evalúa la varianza explicada (en PCA) o la calidad de la visualización (en t-SNE, cualitativamente). El analista debe combinar estos métodos con su conocimiento del dominio para **extraer información útil**. Cuando se emplean adecuadamente, estas técnicas permiten **descubrir patrones significativos** y **simplificar complejidades**, cumpliendo roles fundamentales en la extracción de conocimiento a partir de datos brutos.

Referencias

- Austin Chia (DataCamp, 2023). *Agrupación jerárquica: Concepto general con ejemplos*. DataCamp Tutorial. datacamp.comdatacamp.com

- Google Developers (2022). *Ventajas y desventajas de los k-medios*. Curso de Machine Learning (Agrupamiento en clústeres). developers.google.comdevelopers.google.com

- IBM (s.f.). *¿Qué es la agrupación en clústeres k-means?* Think Blog, IBM Knowledge Center. ibm.comibm.com

- IBM (s.f.). *¿Qué es el análisis de componentes principales (PCA)?* Think Blog, IBM Knowledge Center. ibm.comibm.com

- DataCamp (2023). *Guía del algoritmo de agrupación DBSCAN*. DataCamp Tutorial. datacamp.comdatacamp.com

- DataCamp (2023). *Introducción al t-SNE: Reducción no lineal de la dimensionalidad*. DataCamp Tutorial. datacamp.comdatacamp.com