UNIVERSIDAD TECNOLÓGICA DE CHIHUAHUA

TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

IV.1. ALGORITMOS DE AGRUPACIÓN

MATERIA: Extracción de Conocimiento en Bases de Datos

MAESTR@: Enrique Mascote

ALUMNO: Carlos Adrián Mata Nevárez

Matricula: 1123250015

FECHA: 29/11/2025

ÍNDICE

INTRODUCCIÓN

Contextualización: La Extracción de Conocimiento (Knowledge Discovery) busca patrones valiosos a partir de grandes volúmenes de datos. Los métodos no supervisados son esenciales aquí porque trabajan con datos sin etiquetas previas.

Clustering (Agrupación): Su objetivo es dividir un conjunto de datos en grupos (clusters) de tal forma que los puntos dentro de un mismo grupo sean más similares entre sí que con respecto a los puntos en otros grupos. Sirve para segmentación de mercado, detección de anomalías o clasificación automática.

Reducción de Dimensionalidad: Su propósito es reducir el número de características (dimensiones) de los datos mientras se preserva la mayor cantidad de información o varianza posible. Sirve para visualización de datos de alta dimensión, eliminar ruido o acelerar los algoritmos de Machine Learning.

ALGORITMOS DE AGRUPACIÓN

K-means

K-means es un algoritmo de agrupación basado en centroides que inicializa k centroides aleatoriamente y asigna cada punto de datos al centroide más cercano usando distancia euclidiana. Luego recalcula los centroides como la media de los puntos asignados y repite hasta convergencia o máximo de iteraciones, minimizando la suma de distancias intra-cluster (Arvai, 2025).

Parámetros Clave

- k: Número de clusters preespecificado, seleccionado vía método del codo analizando inertia_ (suma de cuadrados intra-cluster).
- init: Método de inicialización como 'k-means++' para evitar mínimos locales o 'random'.
- max_iter: Límite de iteraciones por defecto 300 para controlar convergencia.

Ventajas y Limitaciones

Ventajas:

- Rápido y escalable a grandes datasets mediante optimizaciones como mini-batch.
- Fácil implementación en scikit-learn con KMeans(n_clusters=k).fit(X).
- Eficiente para clusters esféricos bien separados (Rogel-Salazar, 2025).

Limitaciones:

- Requiere especificar k заранее, sensible a outliers que distorsionan centroides.
- Asume clusters esféricos de tamaño similar; falla con formas irregulares.
- Inicialización aleatoria puede llevar a resultados inconsistentes sin k-means++ (geeksforgeeks, 2025).

Ejemplo en Python (Segmentación de Clientes)

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Escalado necesario para distancias euclidianas
scaler = StandardScaler()
X_scaled = scaler.fit_transform(clientes_df)

# Ajuste K-means con k=3
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Visualización
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=300, c='red', marker='x')
plt.title('Clusters K-means')
u
```

Este código segmenta clientes por features escaladas, visualizando centroides rojos.

Clustering Jerárquico Aglomerativo

El clustering jerárquico aglomerativo inicia cada punto como cluster individual y fusiona iterativamente los dos clusters más cercanos hasta formar uno solo, generando un dendrograma que visualiza la jerarquía de fusiones. La distancia entre clusters se actualiza tras cada fusión según el criterio elegido, permitiendo "cortar" el dendrograma para obtener k clusters deseados (ibm, 2025).

Parámetros Clave

- linkage: Criterio como 'ward' (minimiza varianza intra-cluster), 'average' (promedio de distancias entre puntos) o 'complete' (máxima distancia).

- metric: Distancia entre puntos, típicamente 'euclidean' para datos continuos.

- n_clusters o distance_threshold: Define el corte del dendrograma para número final de clusters (datacamp, 2025).

Ventajas y Limitaciones

Ventajas:

- No requiere especificar k заранее; dendrograma revela estructura natural de datos.
- Útil para análisis exploratorio y datasets pequeños donde la jerarquía importa.
- Maneja formas irregulares mejor que K-means.

Limitaciones:

- Complejidad $O(n^3)$ o $O(n^2)$ limita escalabilidad a grandes datasets.
- Fusiones irrevocables; sensible a ruido y outliers que distorsionan etapas tempranas.
- Interpretación subjetiva del corte en dendrograma (ibm, 2025).

Ejemplo en Python (scikit-learn)

```python
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Escalado y linkage matrix
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
Z = linkage(X_scaled, method='ward')

# Dendrograma
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title('Dendrograma Jerárquico Aglomerativo')
plt.xlabel('Índices de Muestras'); plt.ylabel('Distancia')

# Clustering con k=3
model = AgglomerativeClustering(n_clusters=3, linkage='ward')
clusters = model.fit_predict(X_scaled)
```

Este código genera dendrograma y asigna clusters cortando a 3 grupos.

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

DBSCAN define clusters como regiones de alta densidad separadas por áreas de baja densidad, clasificando puntos en núcleo (≥ $MinPts$ vecinos en radio ϵ), frontera (dentro de ϵ de núcleo, pero sin suficientes vecinos) o ruido. Expande clusters desde puntos núcleo conectando puntos densidad-alcanzables, identificando automáticamente outliers sin asumir formas esféricas (datacamp, 2025).

Parámetros Clave

- ϵ (eps): Radio máximo de vecindad; se elige analizando k-distancia graph para detectar "codos".
- $MinPts$: Mínimo puntos para región densa, comúnmente $\sqrt{n}$ o 4-10 según dimensionalidad. (towardsdatascience, 2025).

Ventajas y Limitaciones

Ventajas:

- Descubre clusters arbitrarios y detecta ruido explícitamente sin especificar k.
- Robusto a formas irregulares y variando densidades moderadas.
- Escalable con complejidad $O(n\log\ n)$ usando estructuras espaciales.

Limitaciones:

- Sensible a ϵ y $MinPts$; falla con densidades muy variables entre clusters.
- Estricto con métricas de distancia; alto-dimensional curse afecta rendimiento.
- No apto para datasets muy grandes sin optimizaciones.

Ejemplo en Python (Sismos)

```python
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

# Datos de sismos (lat, lon -> km)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(sismos[['lat', 'lon']])

# DBSCAN: eps=0.1 (~5km escalado), min_samples=10
dbscan = DBSCAN(eps=0.1, min_samples=10)
clusters = dbscan.fit_predict(X_scaled)

# Visualizar: -1=ruido, 0,1,...=clusters
plt.scatter(X_scaled[:,0], X_scaled[:,1], c=clusters, cmap='viridis')
plt.title('Clusters DBSCAN Sismos')
```

Identifica zonas sísmicas densas (clusters) y eventos aislados (ruido).

ALGORITMOS DE REDUCCIÓN DE DIMENSIONALIDAD

Análisis de Componentes Principales (PCA)

PCA transforma datos originales en componentes principales ortogonales mediante descomposición en autovalores/autovectores de la matriz de covarianza, centrando datos en media cero para capturar máxima varianza. El primer PC maximiza varianza proyectada: $PC_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$, con autovector asociado al mayor autovalor; subsiguientes son ortogonales y maximizan varianza residual (ibm, 2025).

Parámetros Clave

- n_components: Número de dimensiones finales (ej. 2 para visualización) o fracción de varianza (ej. 0.95 para retener 95% información).
- Escalado previo con StandardScaler esencial para variables con escalas dispares (wikipedia, 2025).

Ventajas y Limitaciones

Ventajas:

- Reduce dimensionalidad preservando varianza, eliminando ruido y multicolinealidad.
- Rápido ($O(p^3)$) y descorrelaciona features automáticamente.
- Ideal para visualización y preprocesamiento ML.

Limitaciones:

- Lineal: falla con no-linealidades (usa t-SNE/UMAP alternas).
- Componentes pierden interpretabilidad original.
- Sensible a escalado y outliers.

Ejemplo en Python (Reducción a 2D)

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

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y)
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%} var)')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%} var)')
plt.title('Datos en Espacio PCA 2D')
print(f'Varianza total explicada: {sum(pca.explained_variance_ratio_):.1%}')
```

Proyecta 10D a 2D reteniendo mayor varianza; gráfica muestra separación en nuevo espacio.

t-SNE

t-SNE convierte distancias euclidianas de alta dimensión en probabilidades condicionales Gaussiana ($p_{j|i}$) para puntos cercanos, luego mapea a baja dimensión (2D/3D) usando distribución t-Student para similitudes ($q_{j|i}$), minimizando divergencia KL: $\sum_i \ KL(P_i \ || \ Q_i)$ vía descenso gradiente. Preserva estructuras locales (vecinos cercanos permanecen próximos) mejor que globales, ideal para visualización de clusters no lineales (datacamp, 2024).

Parámetros Clave

- n_components: Dimensiones salida (2 o 3 para gráficos).
- perplexity: Balance local/global (5-50; $\approx 3 \times MinPts$); analiza gráfico perplexity vs KL para óptimo.
- n_iter: Iteraciones optimización (1000 por defecto) (interactivechaos, 2025).

Ventajas y Limitaciones

Ventajas:

- Revela clusters complejos no lineales en visualizaciones 2D superiores a PCA.
- No asume linealidad; destaca manifolds locales.

Limitaciones:

- Computacionalmente costoso ($O(n^2)$); no determinístico (random_state fijo).
- Distancias inter-cluster en 2D no preservan distancias reales; solo agrupaciones locales válidas.
- Sensible a perplexity; datasets >50k necesitan pre-PCA.

Ejemplo en Python (MNIST)

```python
from sklearn.manifold import TSNE
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# MNIST: 784D -> 2D
mnist = fetch_openml('mnist_784')
X_scaled = StandardScaler().fit_transform(mnist.data)

tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X_scaled[:1000])  # Submuestra

plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=mnist.target[:1000].astype(int), cmap='tab10')
plt.title('t-SNE MNIST: Dígitos en 2D')
plt.colorbar(label='Dígito')
```

Muestra clusters separados por dígito; usa submuestra para velocidad.

COMPARATIVA Y CONCLUSIONES

Característica	Agrupación (Clustering)	Reducción de Dimensionalidad
Objetivo Principal	**Segmentación:** Encontrar subgrupos intrínsecos en los datos.	**Proyección/Compresión:** Reducir el número de características.
Salida	Etiquetas de cluster (ej. C_1, C_2) para cada punto.	Un nuevo conjunto de datos con menos columnas (dimensiones).
Uso Típico	Segmentación de mercado, análisis exploratorio, detección de anomalías.	Preprocesamiento para ML, visualización de datos, compresión de datos.
Preserva	La proximidad para definir grupos.	La mayor varianza (PCA) o la estructura local (t-SNE).

Conclusiones

El Clustering es prioritario cuando el objetivo es encontrar grupos o patrones ocultos para la toma de decisiones, como segmentar clientes para marketing. Su función es agrupar datos similares sin etiquetas previas.

La Reducción de Dimensionalidad es la prioridad cuando se manejan conjuntos de datos con una cantidad excesiva de variables. Su objetivo es comprimir la información para facilitar la visualización (por ejemplo, reducir 500 características para poder graficar) o para acelerar algoritmos de Machine Learning al eliminar el ruido.

En la práctica, ambos métodos suelen complementarse: se reduce la dimensionalidad primero para limpiar y optimizar los datos, y luego se aplica el clustering para realizar la segmentación final.

REFERENCIAS

(21 de Agosto de 2024). Obtenido de datacamp: https://www.datacamp.com/es/tutorial/introduction-t-sne

(23 de Jul de 2025). Obtenido de geeksforgeeks: https://www.geeksforgeeks.org/machine-learning/k-means-clustering-on-the-handwritten-digits-data-using-scikit-learn-in-python/

(29 de Nov de 2025). Obtenido de datacamp: https://www.datacamp.com/es/tutorial/hierarchical-clustering-R

(29 de Nov de 2025). Obtenido de datacamp: https://www.datacamp.com/tutorial/dbscan-clustering-algorithm

(29 de Nov de 2025). Obtenido de towardsdatascience: https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556/

(29 de Nov de 2025). Obtenido de ibm: https://www.ibm.com/mx-es/think/topics/principal-component-analysis

(29 de Nov de 2025). Obtenido de wikipedia: https://es.wikipedia.org/wiki/Análisis_de_componentes_principales

(29 de Nov de 2025). Obtenido de interactivechaos: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/t-sne

Arvai, K. (29 de Nov de 2025). *realpython*. Obtenido de https://realpython.com/k-means-clustering-python/

ibm. (29 de Nov de 2025). Obtenido de https://www.ibm.com/es-es/think/topics/hierarchical-clustering

Rogel-Salazar, D. J. (23 de Junio de 2025). Obtenido de domino: https://domino.ai/blog/getting-started-with-k-means-clustering-in-python