UNIVERSIDAD ANDINA DEL CUSCO
FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

DISEÑO ARQUITECTÓNICO
VISTAS ARQUITECTÓNICAS

Asignatura: Inteligencia Artificial

Docente: Hugo Espetia Huamanga

MyA:

JEAN CARLOS HUANCA PILARES

Jamil Zamata Sotec

Del Castillo Molina Gino Galois

SEBASTIAN RODRIGO ANAYA RODRIGUEZ

Zavaleta Handa Fernando Hiroshi

CUSCO – PERÚ

2024

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

DBSCAN es un algoritmo de agrupamiento basado en densidad, que identifica clusters en grandes conjuntos de datos según la densidad de puntos en el espacio. A diferencia de otros métodos de clustering (como K-means), no necesita que se especifique el número de clusters con anticipación.

¿Cómo funciona DBSCAN?

- Epsilon (ε): Radio que define una región alrededor de cada punto. Si otros puntos están dentro de este radio, se consideran vecinos.
- MinPts: Número mínimo de puntos que debe tener una región para considerarse un cluster.
- **Puntos Core, Border y Noise:**
    - Core Points: Puntos que tienen al menos MinPts vecinos dentro de su radio ε.
    - Border Points: Puntos que están dentro del radio ε de un punto core pero que tienen menos de MinPts vecinos.
    - Noise Points: Puntos que no pertenecen a ningún cluster (no son puntos core ni border).

Características clave:

- Identificación de ruido: DBSCAN marca los puntos que no encajan bien en ningún cluster como ruido, permitiendo una mejor identificación de patrones.
- Clusters de forma arbitraria: Es eficaz para detectar clusters de cualquier forma, ya que se basa en la densidad de puntos.
- No requiere especificar el número de clusters. A diferencia de K-means, el número de clusters se deriva del parámetro ε.

Ventajas:

- Flexibilidad en formas de clusters: DBSCAN detecta clusters de formas arbitrarias, como clusters no esféricos o con distribuciones complejas.
- Detección automática de ruido.
- No requiere predefinir el número de clusters.

Desventajas:

- Elección sensible de parámetros: Elegir un valor inapropiado para ε o MinPts puede afectar significativamente el rendimiento del algoritmo.
- Problemas en datos de alta dimensión: En espacios de alta dimensión, puede ser difícil encontrar una buena distancia ε debido a la "maldición de la dimensionalidad."

Aplicaciones:

- Detección de anomalías en datos espaciales.
- Análisis de agrupamiento en datos geoespaciales.
- Agrupamiento de datos de redes sociales para identificar comunidades.

Modelos de Asociación (Algoritmo Apriori o FP-Growth)

Los algoritmos de reglas de asociación, como **FP-Growth**, son técnicas que buscan descubrir relaciones interesantes entre variables en grandes bases de datos. Son especialmente útiles para análisis de transacciones, como en supermercados, donde se busca encontrar qué productos suelen comprarse juntos (ej. el análisis de "market basket").

¿Cómo funciona FP-Growth?

- **FP-Growth** Construye una estructura de árbol compacta (FP-tree) para evitar el recuento reiterado de itemsets frecuentes, mejorando la eficiencia en grandes bases de datos.

Características clave:

- **Análisis exploratorio de grandes bases de datos:** Proporciona reglas que pueden ayudar a comprender mejor los patrones ocultos.
- **Eficiencia en bases de datos grandes:** FP-Growth es más eficiente manejando grandes bases de datos.

Ventajas:

Mejora la eficiencia en bases de datos grandes, ya que evita la generación innecesaria de conjuntos de ítems.

Desventajas:

Aunque más rápido, su implementación es más compleja y requiere más memoria.

Aplicaciones:

- **Análisis de ventas en retail:** Descubre qué productos suelen comprarse juntos, para promociones o disposición de productos en tiendas.
- **Optimización de catálogos:** Ayuda a comprender qué productos tienden a ser solicitados juntos para mejorar el stock y las ofertas.
- **Medicina:** Descubre correlaciones entre síntomas y enfermedades o entre tratamientos y resultados.

Uso de FP-growth con los datos sobre casas y departamentos en cusco

Ejemplo de los datos

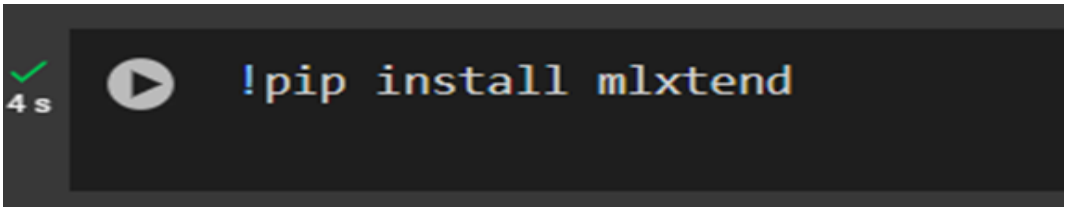
Precio,Locacion,Aream2,NroDormitorios,NroBanos,NroPisos,Zona
289311,San Sebastian,125,2,1,6,Urbanizacion
529492,San Jeronimo,150,2,1,6,Urbanizacion
529820,Saylla,111,2,1,1,Institucion
612894,Santiago,249,3,3,8,Ãrea residencial
778810,Wanchaq,242,5,5,3,Ãrea residencial
966957,Wanchaq,248,3,4,6,Avenida
447051,Wanchaq,114,2,5,6,Avenida
543573,San Jeronimo,161,4,3,4,Urbanizacion
1346776,Wanchaq,214,5,4,9,Parque
436136,Wanchaq,100,3,3,4,Pasaje
525578,Cusco,101,2,3,7,Barrio
1271855,Saylla,86,3,4,7,Institucion
845550,Cusco,161,4,5,3,Condominio
608027,Cusco,63,5,2,6,Condominio
872259,Saylla,188,2,5,1,Residencial
947899,San Jeronimo,69,2,1,8,Residencial
698662.Savlla.248.4.4.3.Institucion

el CSV que estamos usando tiene más de 1000 datos con los cuales estamos trabajando

pasos para evaluar y trabajar dichos datos con FP: Growth en colab

Instalar las librerías necesarias

Usaremos la librería mlxtend que tiene implementaciones útiles para minería de patrones frecuentes, incluido FP-Growth. Además, trabajaremos con pandas para cargar y manipular los datos



2. Importar las librerías

```
[2] import pandas as pd
    from mlxtend.frequent_patterns import fpgrowth, association_rules
```

3. Cargar los archivos que vamos a usar

```
import pandas as pd

# Listar archivos en la ruta actual de Colab (opcional, para verificar si tu arch
!ls /content/

# Cargar el archivo al DataFrame
df = pd.read_csv('/content/datos_casas_cusco.csv')

# Evaluar la estructura del DataFrame: mostrar primeras 5 filas
print(df.head())
```

```
Datos  datos_casas_cusco.csv  sample_data
   Precio       Locacion  Aream2  NroDormitorios  NroBanos  NroPisos  \
0  289311  San Sebastian     125               2         1         6
1  529492   San Jeronimo     150               2         1         6
2  529820         Saylla     111               2         1         1
3  612894       Santiago     249               3         3         8
4  778810        Wanchaq     242               5         5         3

              Zona
0     Urbanizacion
1     Urbanizacion
2       Institucion
3  Área residencial
4  Área residencial
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWar
  and should_run_async(code)
```

Cargamos los archivos y verificamos que estén correctamente cargados haciendo en print de los primeros 4 datos de la tabla

Convertir los datos a formato binario

Como estamos trabajando con categorías (e.g., Locación, Zona) y algunos valores numéricos, podemos transformar estas columnas en categorías binarias, lo que permitirá aplicar FP-Growth.

```
[6] # Convertir columnas categóricas (Locacion y Zona) a formato binario
    df_binarizado = pd.get_dummies(df[['Locacion', 'Zona']])

    # Opcional: Binarizar los datos numéricos en rangos específicos
    df_binarizado['Area_Grande'] = (df['Aream2'] > 150).astype(int)
    df_binarizado['Muchos_Dormitorios'] = (df['NroDormitorios'] >= 3).astype(int)

    # Visualizamos los primeros datos binarizados
    print(df_binarizado.head())
```

```
   Locacion_Cusco  Locacion_San Jeronimo  Locacion_San Sebastian  \
0           False                  False                    True
1           False                   True                   False
2           False                  False                   False
3           False                  False                   False
4           False                  False                   False

   Locacion_Santiago  Locacion_Saylla  Locacion_Wanchaq  Zona_Avenida  \
0              False            False             False         False
1              False            False             False         False
2              False             True             False         False
3               True            False             False         False
4              False            False              True         False

   Zona_Barrio  Zona_Calle  Zona_Condominio  Zona_Institucion  Zona_Paradero  \
0        False       False            False             False          False
1        False       False            False             False          False
2        False       False            False              True          False
3        False       False            False             False          False
4        False       False            False             False          False

   Zona_Parque  Zona_Pasaje  Zona_Residencial  Zona_Urbanizacion  \
```

5. Aplicar FP-Growth

```
[7] from mlxtend.frequent_patterns import fpgrowth, association_rules

    # Ejecutar FP-Growth con soporte mínimo del 20% (0.2)
    patrones_frecuentes = fpgrowth(df_binarizado, min_support=0.2, use_colnames=True)

    # Mostrar los patrones frecuentes encontrados
    print(patrones_frecuentes)

    # Generar reglas de asociación con un umbral de confianza del 50%
    reglas = association_rules(patrones_frecuentes, metric="confidence", min_threshold=

    # Mostrar las reglas de asociación
    print(reglas)
```

```
    support                         itemsets
0    0.7485             (Muchos_Dormitorios)
1    0.4905                    (Area_Grande)
2    0.3675  (Area_Grande, Muchos_Dormitorios)
     antecedents           consequents  antecedent support  \
0  (Area_Grande)  (Muchos_Dormitorios)              0.4905

   consequent support  support  confidence      lift  leverage  conviction  \
0              0.7485   0.3675    0.749235  1.000983  0.000361    1.002933

   zhangs_metric
0       0.001927
```

6. Generación de gráficos para explicar más detalladamente
   Gráficos de barras
   Muestra el soporte de cada patrón frecuente para ver cuáles combinaciones de ítems ocurren más.

```
# Generar un gráfico de barras de los soportes de los patrones frecuentes
plt.figure(figsize=(10, 6))
patrones_frecuentes.plot(kind='bar', x='itemsets', y='support', legend=False, ax=pl
plt.title('Patrones Frecuentes - Gráfico de Barras')
plt.xlabel('Conjuntos de Ítems')
plt.ylabel('Soporte')
plt.xticks(rotation=45, ha='right')
plt.show()
```

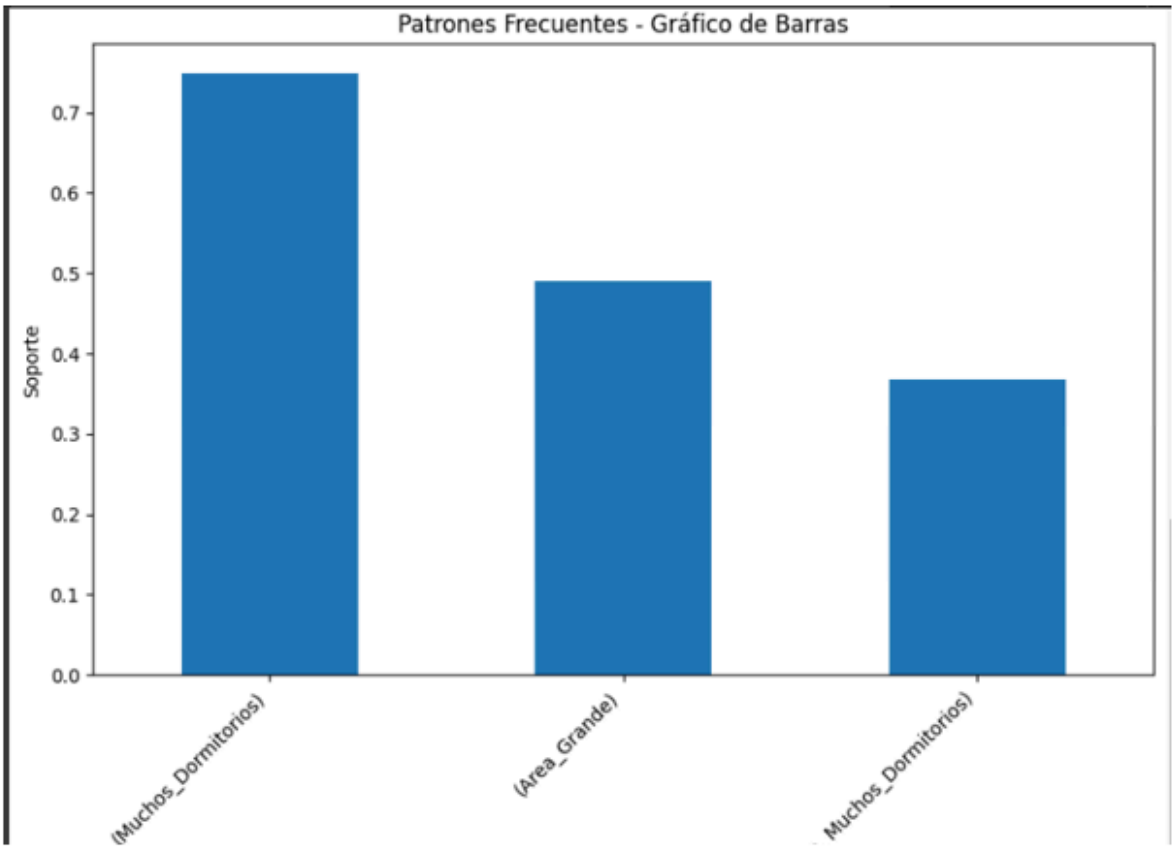


Gráfico de pastel

Representa la distribución del soporte de los patrones frecuentes como proporciones del total.

```
# Gráfico de pastel para mostrar distribución de soportes
plt.figure(figsize=(8, 8))
plt.pie(patrones_frecuentes['support'], labels=patrones_frecuentes['itemsets'], aut
plt.title('Distribución de Soporte de los Patrones Frecuentes')
plt.show()
```

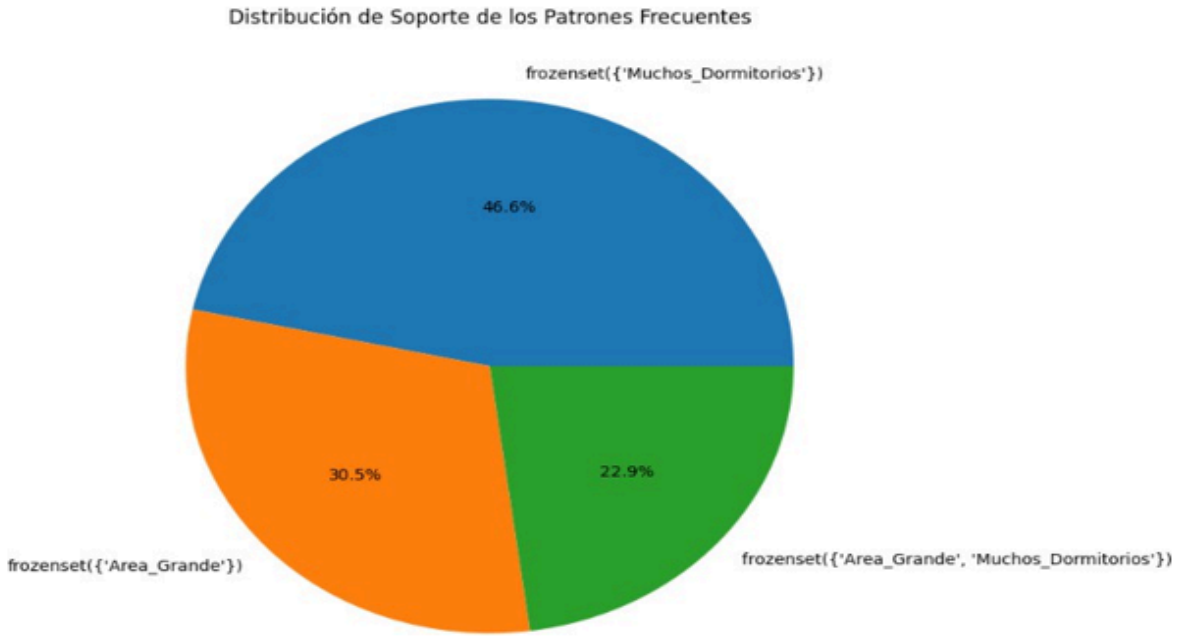
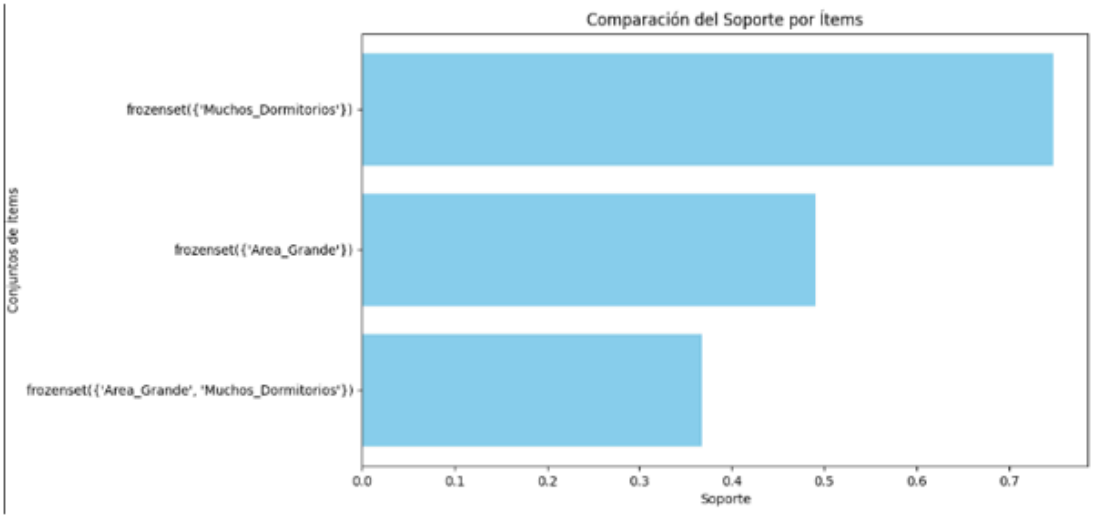


Gráfico comparativo

Compara los ítems más frecuentes para identificar cuáles tienen mayor peso en los patrones.

```
# Gráfico comparativo entre los ítems y su soporte
plt.figure(figsize=(10, 6))
plt.barh(patrones_frecuentes['itemsets'].astype(str), patrones_frecuentes['support'
plt.title('Comparación del Soporte por Ítems')
plt.xlabel('Soporte')
plt.ylabel('Conjuntos de Ítems')
plt.gca().invert_yaxis()  # Invertir el eje para una mejor lectura
plt.show()
```



Estos gráficos nosA permiten:

Identificar patrones frecuentes en los datos (cuáles ocurren más).
Visualizar la distribución del soporte (cuáles dominan y cuáles son minoritarios).
Comparar patrones de forma clara para destacar combinaciones relevantes.