REGLAS DE ASOCIACIÓN

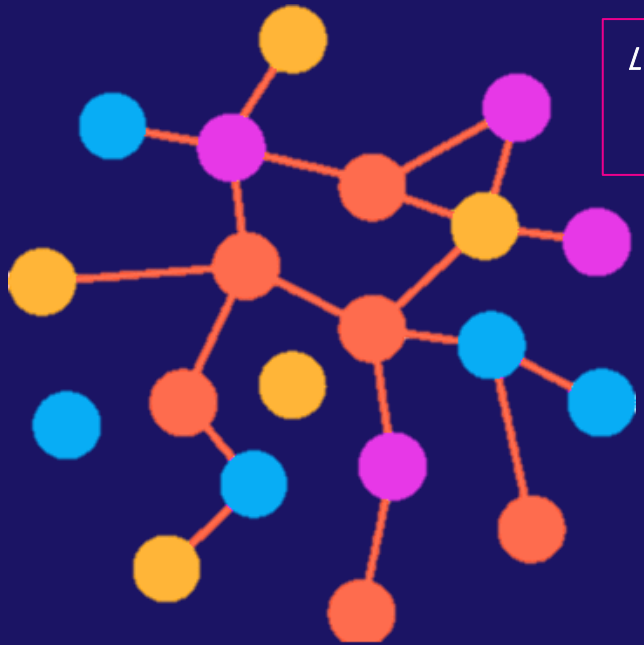
Mara Anahí Martínez Cervantes 1798584
Sebastián Canizales Melchor 1842108
Katia del Carmen Ortiz Martínez 1799830
Lluvia Elizabeth Avilés Zuñiga 1821636

01

¿Qué son?

Estudian las ocurrencias de un evento dentro de un conjunto de datos, dadas las diferentes medidas de interés (ítems), por ello es importante conocer la estructura en que viene la base de datos, puesto que el método de aprendizaje que presentan las reglas deben ser coherente con el objeto de estudio.

Un articulo llamado “REGLAS DE ASOCIACIÓN APLICADAS A LA DETECCIÓN DE FRAUDE CON TARJETAS DE CRÉDITOS” dice que :



Las reglas de asociación son implicaciones que relacionan la presencia de ítem en las transacciones. Las transacciones son la estructura básica desde donde las reglas de asociación son obtenidas.

La definición por la IBM:

Las reglas de asociación relacionan una determinada conclusión con un conjunto de condiciones.

VENTAJAS

Las asociaciones pueden darse entre cualquiera de los atributos

Al tener muchas reglas se pueden obtener muchas conclusiones

Al estar sujetas a un número de restricciones, la comprobación y fiabilidad es alta.

DESVENTAJAS

Los algoritmos de asociación tratan de encontrar patrones en un espacio de búsqueda potencialmente muy amplio y, por tanto, pueden necesitar mucho más tiempo de ejecución que un algoritmo de árbol de decisión.

02

ALGORITMOS DE REGLAS DE ASOCIACIÓN

CONCEPTOS PRINCIPALES

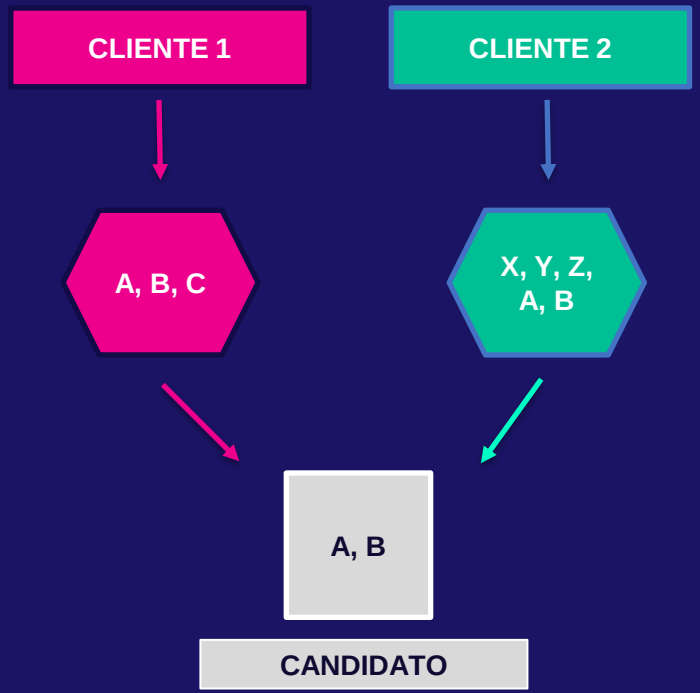
Itemset: Colección de uno o más items (ej. {A, B, C})
Support: Frecuencia relativa en la que un itemset aparece dentro de los datos.
Frequent itemset: Un itemset el cual aparece más veces que un determinado umbral.
Confidence: Relación de dos secesos con la cantidad de veces que ocurre uno de ellos.
Lift: Indica la probabilidad de que se compre B después de comprar A.

APRIORI

No analiza patrones, sino que desarrolla todas las opciones de relación candidatas de los datos.

Para cada combinación:

- Support (A) = (# de transacciones donde está A) / (# de transacciones)
- Confidence (A⟶B) = (# de transacciones donde están A y B) / (# de transacciones donde está A)
- Lift (A⟶B) = Confidence (A⟶B) / Support (B)

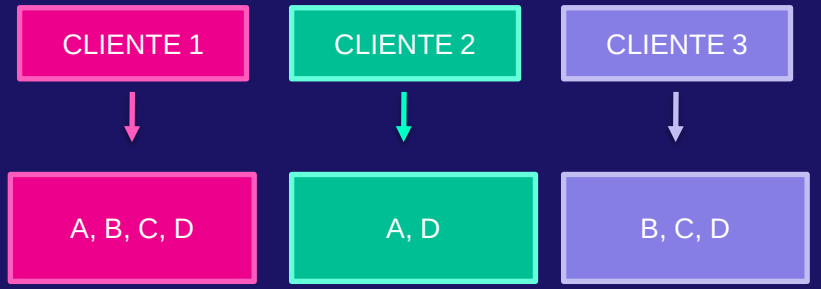


ECLAT

Similar a Apriori, pero más rápido. Analiza las apariciones de cada elemento.

Solo utilizamos la definición de Support:

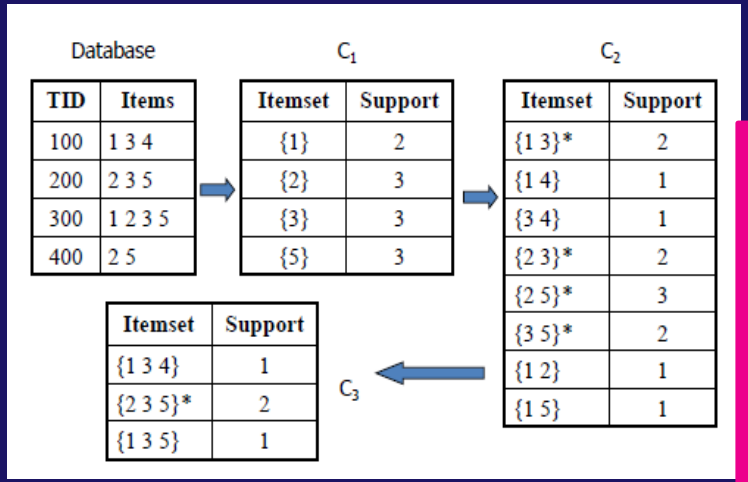
- Support (A) = (# de transacciones donde está A) / (# de transacciones)



A	**CLIENTE 1, CLIENTE 2**
B	**CLIENTE 1, CLIENTE 3**
C	**CLIENTE 1, CLIENTE 3**
D	**CLIENTE 1, CLIENTE 2, CLIENTE 3**

AIS

- Los conjuntos de elementos (itemset) encontrados se crean y cuentan en tiempo real mientras se lee la base de datos.
- Para cada nueva transacción, se determina cuál de los principales (mayores) conjuntos anteriormente generados ya contienen a esta transacción.



TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

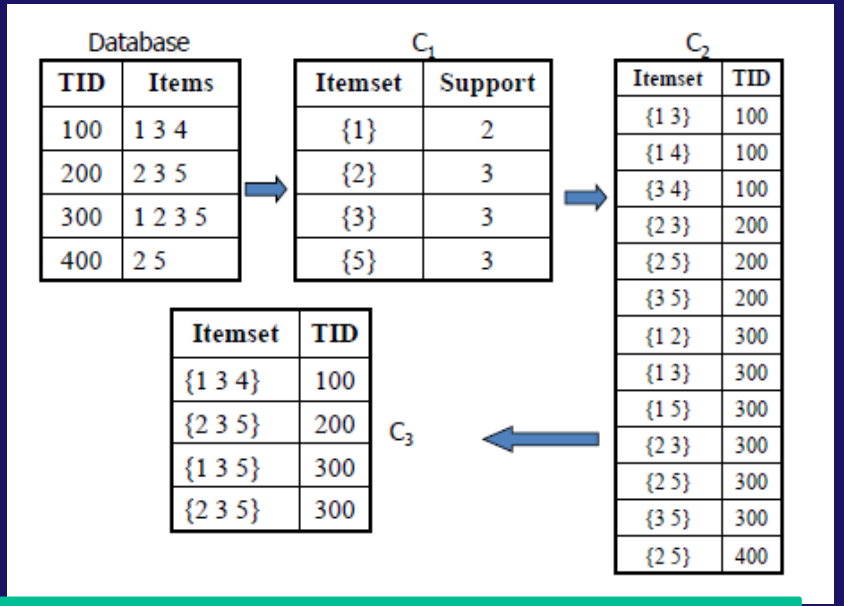
Itemset	Support
{1}	2
{2}	3
{3}	3
{5}	3

Itemset	Support
{1 3}*	2
{1 4}	1
{3 4}	1
{2 3}*	2
{2 5}*	3
{3 5}*	2
{1 2}	1
{1 5}	1

Itemset	Support
{1 3 4}	1
{2 3 5}*	2
{1 3 5}	1

SETM

- Los conjuntos de elementos se generan en el momento mientras se procesa la base de datos, pero se cuentan al finalizar la pasada.
- Los nuevos conjuntos se generan de igual forma que el algoritmo AIS, pero el TID de la transacción generada se guarda junto con el conjunto candidato en una estructura secuencial.



03

APLICACIONES

APLICACIONES

- Diagnosticar pacientes
- Patrones de compra
- Optimización
- Cross Marketing
- Patrones de investigación

04

EJEMPLO EN CÓDIGO PAYTHON

Librerías Utilizadas:

```
[26] pip install mlxtend
```

```
Requirement already satisfied: mlxtend in /usr/local/lib/python3.7/dist-packages (0.14.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from mlxtend) (57.4.0)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (0.22.2.post1)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.4.1)
Requirement already satisfied: matplotlib>=1.5.1 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (3.2.2)
Requirement already satisfied: pandas>=0.17.1 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.1.5)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.19.5)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.1->mlxtend) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.1->mlxtend) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.1->mlxtend) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.1->mlxtend) (1.3.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib>=1.5.1->mlxtend) (1.15.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.17.1->mlxtend) (2018.9)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.18->mlxtend) (1.0.1)
```

```
[1]  pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.1.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.19.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
```

Creación de una lista de datos:

```
[52] lista_datos = [["pelicula1","pelicula2","pelicula3"],
                    ["pelicula1","pelicula2"],
                    ["pelicula1","pelicula3"],
                    ["pelicula2","pelicula4"]]
```

```
[53] print(lista_datos)
```

```
[['pelicula1', 'pelicula2', 'pelicula3'], ['pelicula1', 'pelicula2'], ['pelicula1', 'pelicula3'], ['pelicula2', 'pelicula4']]
```

Transformar la lista de datos a un Data Frame, con valores booleanos:

```
[54] import pandas as pd
     from mlxtend.preprocessing import TransactionEncoder
```

```
[55] te = TransactionEncoder()
     te_array = te.fit(lista_datos).transform(lista_datos)
     datos = pd.DataFrame(te_array, columns=te.columns_)
```

```
[56] datos
```

	pelicula1	pelicula2	pelicula3	pelicula4
0	True	True	True	False
1	True	True	False	False
2	True	False	True	False
3	False	True	False	True

Encontrar la frecuencia de ocurrencias de cada item:

```
[114] from mlxtend.frequent_patterns import association_rules

      reglas = association_rules(frecuencia_items, metric="confidence")
```

```
[115] reglas
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(pelicula3)	(pelicula1)	0.50	0.75	0.50	1.0	1.333333	0.1250	inf
1	(pelicula4)	(pelicula2)	0.25	0.75	0.25	1.0	1.333333	0.0625	inf
2	(pelicula3, pelicula2)	(pelicula1)	0.25	0.75	0.25	1.0	1.333333	0.0625	inf

FUENTES DE CONSULTA

https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=nodes-association-rules
https://www.bing.com/videos/search?q=reglas+de+asociacion&&view=detail&mid=5F81996C36A18CA227CE5F81996C36A18CA227CE&rvsmid=8D4BE47A03803F4E42778D4BE47A03803F4E4277&FORM=VDMCNR
https://www.bing.com/videos/search?q=reglas+de+asociacion&docid=6080392032125754008&mid=8D4BE47A03803F4E42778D4BE47A03803F4E4277&view=detail&FORM=VIRE