Minería de patrones frecuentes y reglas de asociación

Máster Online en Ciencia de Datos

UCO ONLINE

Dr. José Raúl Romero

Profesor Titular de la Universidad de Córdoba y Doctor en Ingeniería Informática por la Universidad de Málaga. Sus líneas actuales de trabajo se centran en la democratización de la ciencia de datos (*Automated ML* y *Explainable Artificial Intelligence*), aprendizaje automático evolutivo y analítica de software (aplicación de aprendizaje y optimización a la mejora del proceso de desarrollo de software).

Miembro del Consejo de Administración de la *European Association for Data Science*, e investigador senior del Instituto de Investigación Andaluz de *Data Science and Computational Intelligence*.

Director del **Máster Online en Ciencia de Datos** de la Universidad de Córdoba.

Algoritmo Apriori

Introducción

UCO ONLINE

- Desarrollado por Agrawal y Srikant (IBM Research), 1994
- Fue una forma innovadora de encontrar asociaciones a gran escala, permitiendo obtener implicaciones que contienen más de un elemento
- Algoritmo basado en un umbral de soporte mínimo (ya utilizado en un algoritmo previo, AIS 1993)
- Múltiples versiones del algoritmo:
  - Apriori (version básica), más rápido en las primeras iteraciones
  - AprioriTid, más rápido en las últimas iteraciones
  - AprioriHybrid, cambia de Apriori a AprioriTid después de las primeras iteraciones

Accesible: http://www.vldb.org/conf/1994/P487.PDF

Conceptos necesarios:

- **K-itemset**: Itemset que contiene *k* elementos
- **Soporte** o **Frecuencia**: Número de transacciones que contienen un determinado itemset
- **Itemset frecuente**: Un itemset que satisface el soporte mínimo *minsup*

  Se denota por $\mathbf{L_k}$ para un k-itemset
- **Operación JOIN** (*unión*): C_k, el conjunto de k-itemsets candidatos es generado uniendo L_{k-1} consigo mismo

  (L_1: frequent 1-itemset, L_k: frequent k-itemset)
- **Operación PRUNE** (*poda*): L_k, el conjunto de k-itemsets frecuentes se extrae de C_k mediante poda – eliminando todos los k-itemsets no frecuentes en C_k
  - Enfoque iterativo: k-itemsets utilizados para explorar (k+1)-itemsets
  - El algoritmo Apriori encuentra k-itemsets frecuentes
- **Propiedad Apriori**: *Todos los subconjuntos no vacíos de un itemset frecuente deben ser frecuentes*

Algoritmo Apriori

Algoritmo y pseudocódigo

UCO ONLINE

Se inicia con el proceso de **creación de itemsets frecuentes**

- Se definen:
  - C_k – itemset candidato de tamaño *k*
  - L_k – itemset frecuente de tamaño *k*
- Los principales **pasos de la iteración** son:
  1. Encontrar el conjunto frecuente L_{k-1}
  2. Paso **JOIN**: C_k es generado uniendo L_{k-1} consigo mismo (producto cartesiano, L_{k-1} **x** L_{k-1})
  3. Paso **PRUNE** (**Propiedad Apriori**): Cualquier itemset de tamaño (k − 1) que no es frecuente NO puede ser un subconjunto de un itemset frecuente de tamaño k -> debe ser eliminado
  4. Se consigue el conjunto frecuente L_k

- Apriori utiliza una **búsqueda en amplitud** y una **estructura de árbol hash** para crear los itemsets candidatos de forma eficiente
- Se contabiliza la **frecuencia de ocurrencia para cada itemset candidato**

  Un subconjunto de un itemset frecuente también debe ser frecuente

  Si {AB} es un itemset frecuente, tanto {A} como {B} deben ser frecuentes
- Aquellos itemsets candidatos que tienen mayor frecuencia que el umbral de soporte mínimo ***minsup*** son cualificados como **itemsets frecuentes**

Pseudocódigo

Paso 1: descubrimiento de los *k*-itemsets

*Inicio: Base de datos transaccional **D** y umbral de soporte mínimo **minsup** definido por usuario*

$L_1 = \{large\ 1\text{-}itemsets\}$

For $(k = 2; L_{k-1} \neq \phi; k++)$ do begin

  C_k = apriori-gen(L_{k-1});

  forall transactions $t \in D$ do begin

    C_t = subset(C_k, t)

    forall candidates $c \in C_t$ do

      $c.count++$;

    end

  end

  $L_k = \{c \in C_k | c.count \geq minsup\}$

end

$Answer = \bigcup_k L_k$;

Contar ocurrencias de items

Generar nuevos k-itemsets candidatos

Candidatos C_k en la transacción

Calcular el soporte de todos los candidatos

Tomar sólo aquellos con soporte por encima de ***minsup***

Condición de parada: No hay más grandes itemsets

Pseudocódigo

Paso 2: Generación de candidatos – **apriori-gen()**

apriori-gen

- Paso JOIN

insert into C_k
select $p.item_1, p.item_2, p.item_{k-1}, q.item_{k-1}$
from $L_{k-1}p, L_{k-1}q$
where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$

p y **q** son (k-1)-itemsets idénticos para todos sus (k-2) primeros items

No es = para prevenir duplicados

UNIR añadiendo el último item de **q** a **p**

- Paso PRUNE

forall *itemsets* $c \in C_k$ do
  forall *(k-1)-subsets s of c* do
    if $(s \notin L_{k-1})$ then
      delete *c from* C_k

Comprobar todos los subconjuntos

$$C_k \subseteq L_k$$

Pseudocódigo

Paso 2: Generación de candidatos – **`apriori-gen()`** / ejemplo

- L_3 = { {1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4} }

- Después de Paso JOIN:

  { {1 2 3 4}, {1 3 4 5} }

- Después de Paso PRUNE:

  {1 2 3 4} ← {1 4 5} y {3 4 5} no están en L_3

Pseudocódigo

$L_1 = \{large\ 1\text{-}itemsets\}$

For $(k = 2;\ L_{k-1} \neq \phi\ ;\ k++)$ do begin

  C_k = apriori-gen(L_{k-1});

  forall transactions $t \in D$ do begin

    C_t = subset(C_k, t)

    forall candidates $c \in C_t$ do

      $c.count++$;

    end

  end

  $L_k = \{ c \in C_k | c.count \geq minsup \}$

end

$Answer = \bigcup_k L_k$;

- Los itemsets candidatos $\mathbf{C_k}$ se almacenan en un *hash-tree*
- Encuentra en un tiempo **O(k)** si un *k-itemset* candidato está contenido en la transacción **t**
- El tiempo total es **O(max(k,size(t))**

Limitaciones del algoritmo

- Requiere recorrer en varias ocasiones todo el conjunto de datos
- Asume que todo el *dataset* está en memoria
- Utiliza un **umbral de soporte mínimo uniforme**
- Tiene dificultades para encontrar eventos que ocurren raramente:
  - Métodos alternativos (distintos a Apriori) pueden resolver este asunto utilizando un umbral de soporte no uniforme
  - Algunas otras alternativas se enfocan en la partición (*partition*) y muestreo (*sampling*)

Algoritmo Apriori

Ejemplo

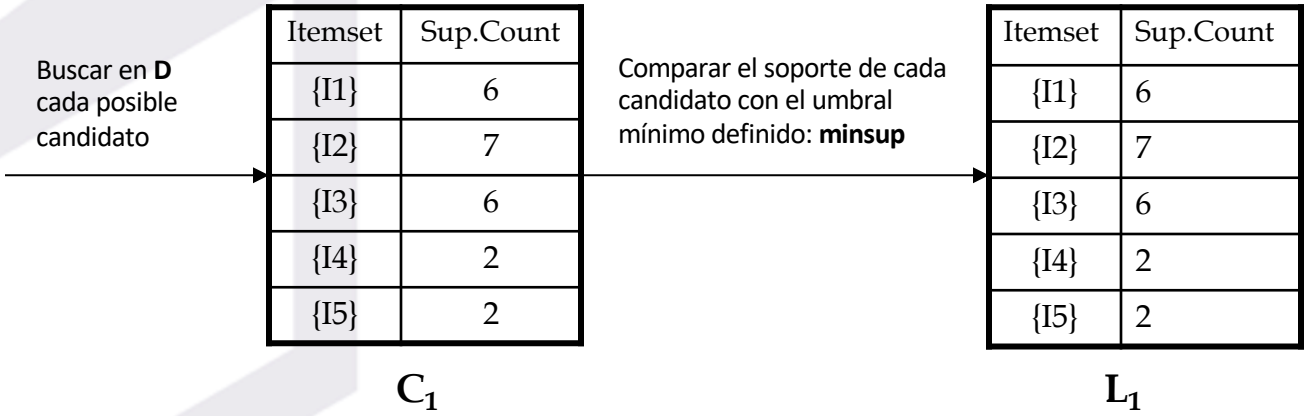
UCO ONLINE

Enunciado del ejemplo

TID	Lista de Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2 ,I3, I5
T108	I1, I2, I3

- La base de datos **D** contiene 9 transacciones
- Se supone que el soporte mínimo requerido es 2 (***minsup*** = 2/9 = 22%)
- Se establece un umbral mínimo de confianza del 70%

Paso 1: Generación de 1-itemset frecuentes



Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_1

Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

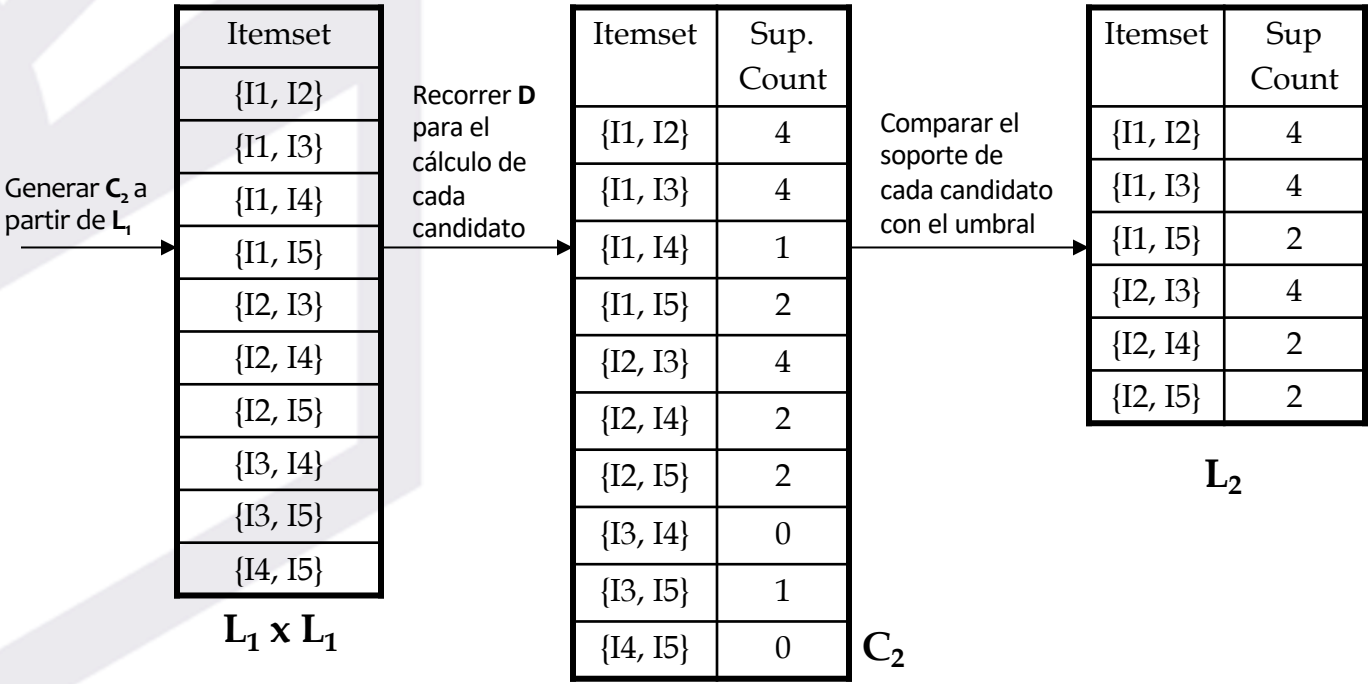
L_1

- En la primera iteración del algoritmo, cada item es un miembro del conjunto de candidatos, C_1
- El conjunto de 1-itemsets frecuentes, L_1, consiste en los 1-itemsets candidatos que satisfaces el umbral mínimo de soporte

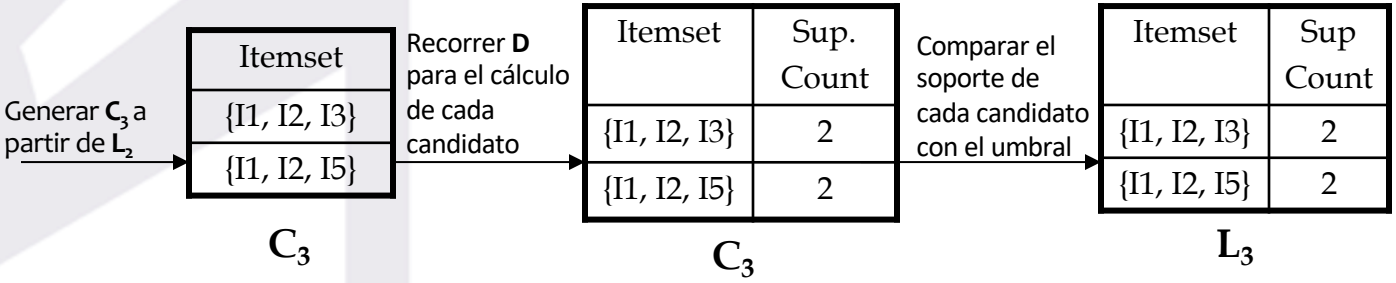
Paso 2: Generación de 2-itemset frecuentes

- Para descubrir el conjunto de 2-itemsets frecuentes, $\mathbf{L_2}$, el algoritmo utiliza $\boldsymbol{L_1}$ ***JOIN*** $\boldsymbol{L_1}$ para generar los conjuntos de $\mathbf{C_2}$ de 2-itemsets candidatos
- Después, las transacciones en **D** son escaneadas y se calcula el soporte para cada candidato $\mathbf{C_2}$
- El conjunto de 2-itemsets frecuentes, $\mathbf{L_2}$, viene dado por aquellos 2-itemsets candidatos en $\mathbf{C_2}$ que satisfacen el soporte mínimo
- **Nota**: Aún no hemos aplicado la **propiedad Apriori**

Paso 2: Generación de 2-itemset frecuentes



Paso 3: Generación de 3-itemset frecuentes



- La generación del conjunto de 3-itemsets candidatos, C_3 , implica el **uso de la propiedad Apriori**
- Para encontrar $\mathbf{C_3}$, se calcula L_2 ***JOIN*** L_2
- C_3 = **L2** ***Join*** **L2** = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}
- Después del paso JOIN, se utiliza el **paso PRUNE**, que reducirá el tamaño de $\mathbf{C_3}$ y, por tanto, el coste computacional del cálculo de C_k.

Paso 3: Generación de 3-itemset frecuentes

Tomando la **propiedad Apriori**, por la que *todos los subconjuntos de un itemset frecuente deben ser también frecuentes*, determinamos que hay varios itemset en $\mathbf{C_3}$ que no pueden ser frecuentes.
¿Cómo?

- Por ejemplo, tomemos **{I1, I2, I3}**:
  - Sus 2-item-subsets son **{I1, I2}, {I1, I3}, {I2, I3}**
  - Puesto que todos los 2-item-subsets de **{I1, I2, I3}** son miembros de $\mathbf{L_2}$, lo mantenemos en **{I1, I2, I3}** en $\mathbf{C_3}$
- Tomemos otro ejemplo, **{I2, I3, I5}**:
  - Sus 2-item-subsets son **{I2, I3}, {I2, I5}, {I3,I5}**
  - PERO, **{I3, I5}** NO es miembro de $\mathbf{L_2}$, por lo que no es frecuente (violación de la propiedad Apriori)
  - **{I2, I3, I5}** se debe eliminar de $\mathbf{C_3}$
- Ahora, las transacciones en **D** se recorren para determinar $\mathbf{L_3}$, que contiene *los 3-itemsets candidatos en* $\boldsymbol{C_3}$ *que superan el soporte mínimo*

Paso 4: Generación de 4-itemset frecuentes

- El algoritmo utiliza **L_3 JOIN L_3** para generar el conjunto candidato de 4-itemsets, **C_4**
- La **operación JOIN** resulta en {{I1, I2, I3, I5}}, al que se aplica **PRUNE** y resulta que el subconjunto {{I2, I3, I5}} no es frecuente
- Por tanto, **C_4 = ø** , y **el algoritmo termina**, habiendo encontrado todos los elementos frecuentes

Apriori como cuello de botella

- La generación de candidatos puede resultar en conjuntos de candidatos enormes:
  - 10^4 1-itemsets frecuentes generan 10^7 2-itemsets candidatos
  - Para descubrir un patrón frecuente de tamaño 100, p.ej. $\{a_1, a_2, \ldots, a_{100}\}$, se necesitará generar $2^{100} \sim 10^{30}$ candidatos

- Multitud de pases sobre la base de datos:
  - Se requieren (n + 1) pases, siendo **n** la longitud del patrón más largo

Algoritmo Apriori

Uso de herramientas para la ejecución de Apriori

UCO ONLINE

¡Gracias!

UCO ONLINE