

UNIDAD 2: MINERÍA DE ELEMENTOS FRECUENTES

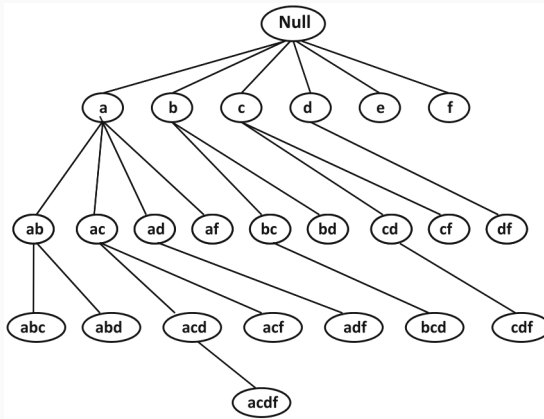
ALGORITMOS DE ÁRBOLES DE ENUMERACIÓN

Blanca Vázquez y Gibran Fuentes-Pineda

12 de septiembre de 2022

- Son algoritmos de minería de elementos frecuentes
- Los conjuntos de elementos candidatos se generan en una estructura de árbol.
- El crecimiento del árbol puede compensar diferentes estrategias entre almacenamiento, acceso a disco y eficiencia computacional.

EJEMPLO DE UN ALGORITMO DE ÁRBOLES DE ENUMERACIÓN



A esta estructura, también se le conoce como: árbol lexicográfico
($a < b, b < c \dots$)

El algoritmo de árbol de enumeración se define sobre los conjuntos de elementos frecuentes de la siguiente manera:

- Existe un nodo en el árbol correspondiente a cada conjunto de elementos frecuentes.
- La raíz del árbol corresponde al conjunto de elementos vacío.

Sea $I = \{i_1, \dots, i_k\}$ un conjunto de elementos frecuentes, donde i_1, i_2, \dots, i_k son listados en orden lexicográfico.

- El padre del nodo I es el conjunto $\{i_1, \dots, i_{k-1}\}$, por lo tanto el hijo de un nodo solamente se puede extender con elementos que aparecen lexicográficamente después de todos los elementos que ocurren en ese nodo.

- Los patrones candidatos son generados a medida que crece el árbol lexicográfico.
- Proporciona una representación abstracta jerárquica de los conjuntos de elementos.
- Esta representación, es aprovechada para explorar de forma sistemática los patrones candidatos (evitando repeticiones).
- La salida del algoritmo son los conjuntos de elementos frecuentes definidos en la estructura de árbol.

¿CÓMO CRECEN LOS ÁRBOLES DE ENUMERACIÓN?

Dependerá del algoritmo selección.

- Los algoritmos indicarán el orden y la estructura para el descubrimiento de conjuntos de elementos frecuentes
- Una extensión frecuente del árbol es: un elemento que se usa para extender un nodo.

De forma general:

- El nodo de un árbol se extiende al encontrar los elementos frecuentes en 1 (nodos candidatos).
- Los nodos candidatos, son extendidos al encontrar los elementos frecuentes en 1
- y así sucesivamente.

¿CÓMO GENERAR NODOS CANDIDATOS A PARTIR DE LOS NODOS FRECUENTES?

Para que un elemento i sea considerado un nodo candidato para extender un nodo frecuente P , entonces i debe ser una extensión frecuente del padre Q de P .

- Esto se debe a la propiedad de cierre descendente:
Para definir los nodos candidatos de un nodo P , primero se deben definir los nodos frecuentes de su padre Q

ALGORITMO DE ÁRBOL DE ENUMERACIÓN GENÉRICO (PARTE 1)

- Seleccionamos uno o más nodo P en el árbol
- Determinamos las extensiones candidatas $C(P)$ para cada nodo P
- Calculamos el soporte de los candidatos
- Añadimos los elementos frecuentes al árbol

ALGORITMO DE ÁRBOL DE ENUMERACIÓN GENÉRICO (PARTE 2)

función $\text{GET}(\mathcal{T}, \text{minsup})$

inicializamos el ET con un nodo *nulo*

mientras cualquier nodo en ET no haya sido examinado **hacer**

selecciona uno o más nodos P no examinados del ET

genera extensiones candidatas $C(P)$ para cada nodo P

determina las extensiones frecuentes $F(P) \subseteq C(P)$ para cada P con soporte

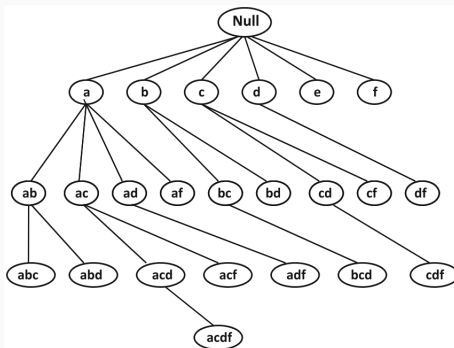
extiende cada nodo P en ET con sus extensiones frecuentes en $F(P)$

fin de mientras

devolver ET

fin de función

EJEMPLO DE UN ALGORITMO DE ÁRBOLES DE ENUMERACIÓN



- El valor de $F(P)$ cuando $P = ab$ es $\{c,d\}$
- El valor de $C(P)$ cuando $P = ab$ es $\{c,d,f\}$
- La relación $F(P) \subseteq C(P) \subset F(Q)$ siempre se cumple

MÉTODO RECURSIVO DE CRECIMIENTO DE PATRONES BASADOS EN SUFIJOS

- Es un caso especial del algoritmo de árboles de enumeración.
- Un ejemplo son los árboles de patrones frecuentes.

- El árbol de patrones frecuentes (*Frequent-Pattern tree*, *FP-tree*) se basa en la metodología *divide y vencerás*.
- Para maximizar el uso de recursos: evita generar conjuntos de elementos candidatos.
- Busca comprimir una base de datos hacia un FP-tree.

1. Escanear la base de datos y calcular la frecuencia de cada elemento
2. Ordenar los elementos por frecuencia (orden descendente) y que cumplan con un *minSup*
3. Por cada transacción, se construye una rama en el *FP-tree*
 - . Si existe una ruta con prefijo común:
 - 3.1 Incrementar la frecuencia de los nodos en esa ruta y agregar el sufijo
 - 3.2 En caso contrario, crear una nueva rama

EJEMPLO DE FP-TREE

Usando la siguiente base de datos calcular el algoritmo FP-tree (soporte mínimo igual a 0.5).

TID	Elementos
1	{f, a, c, d, g, i, m, p}
2	{a, b, f, c, l, m, o}
3	{b, f, h, j, o}
4	{b, c, k, s, p}
5	{a, f, c, e, l, m, p, n}

- Calcular la base del patrón condicional
- Construir el árbol de patrones frecuentes
- Identificar las reglas de asociación

- Completitud
 - No rompe un patrón en ninguna transacción
 - Conserva la información más relevante
- Compacto
 - Reduce información irrelevante.
 - El orden descendente ayuda a que es más probable que los elementos más frecuentes se compartan.
 - El árbol no será más grande que la base original.

EJERCICIO

A partir de la BD de transacciones, generar el FP-tree ($minSup = 0.3$)

TID	Elementos
1	{leche, pan, mantequilla, pollo, huevos}
2	{huevos, leche, tocino, yogurt}
3	{pan, aguacates, cebolla, cereal}
4	{leche, yogurt, huevos, azúcar}
5	{huevos, aguacates, jamón, té, cereal, pan}
6	{pan, mantequilla, camarones, chocolate, leche}
7	{pasta, leche, pollo, yogurt, huevos}
8	{pan, jamón, cereal, leche, pollo}
9	{camarones, fresas, té, almendras, cereal, leche}
10	{mantequilla, leche, yogurt, jugo, pollo, huevos}

- El descubrimiento de patrones de manera temprana es útil para la eficiencia computacional en la minería de patrones.
- También la estrategia de primero profundidad ayuda en la gestión de la memoria de algoritmos basados en proyecciones.

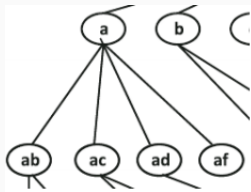
En cuanto a las estrategias de conteo, estas tienden a ser costosas, por lo tanto, las estrategias de crecimiento tratan de optimizar el conteo mientras recorren el árbol.

- El algoritmo termina hasta que ninguno de los nodos pueda extenderse más.
- La mayoría de los algoritmos de minería de patrones pueden ser vistos como extensiones del árbol de enumeración.
- Existen diferentes estrategias para el crecimiento del árbol y metodologías de conteo:
 - Eficiencia, requerimientos, costos y acceso a disco.

Estrategia de primero amplitud	Estrategia de primero profundidad
El conjunto de nodos P en una iteración, corresponde a todos los nodos en el 1er nivel del árbol.	Selecciona un solo nodo en el nivel más profundo para crear P .
Uso: en bases de datos residentes en disco, debido a que todos los nodos en un solo nivel del árbol pueden ser extendidos durante una pasada en la base de datos.	Uso: descubrimiento de patrones, debido a que la estrategia tiene una mejor capacidad de explorar el árbol en profundidad.

- El algoritmo apriori construye el *ET* usando la estrategia “primero amplitud”
- El algoritmo Apriori genera los conjuntos candidatos $(k + 1)$ mediante la fusión de 2 elementos frecuentes de los cuales los primeros $k - 1$ elementos son los mismos.

INTERPRETACIÓN DEL ALGORITMO APRIORI BASADO EN ÁRBOLES



Ejemplo: la extensión de $\{ab\}$, se obtiene con $\{c,d,f\} \subseteq F(a)$

Recordemos que: las $C(P)$ para todos los nodos P en un nivel del ET , pueden generarse de manera exhaustiva y no repetidas mediante combinaciones entre todos los pares de hermanos en el mismo nivel.

¿Y CÓMO SE PODA EL ÁRBOL?

- El algoritmo apriori descarta algunos nodos del árbol cuando no garantizan ser elementos frecuentes.
- El algoritmo termina cuando el árbol no puede crecer más.

- A este método se le conoce como: **intersección recursiva de listas**
- Fue introducida por los algoritmos de *Monet* y *Partition*.
- Cada elemento está asociado con una lista de sus identificadores de transacciones *tids*
- De forma general, este método es una transpuesta de una matriz binaria

EJEMPLO DE CONTEO VERTICAL

TID	Elementos	Representación binaria
1	{pan, mantequilla, leche}	110010
2	{huevos, leche, yogurt}	000111
3	{pan, queso, huevos, leche}	101110
4	{huevos, leche, yogurt}	000111
5	{queso, leche, yogurt}	001011

- El $\text{sup}(\{\text{leche}, \text{yogurt}\}) = 3/5 = 0.6$
- El $\text{sup}(\{\text{leche}, \text{huevos}, \text{yogurt}\}) = 2/5 = 0.4$

EJEMPLO DE CONTEO VERTICAL

Elemento	Conjunto de tids	Representación binaria
Pan	{1,3}	10100
Mantequilla	{1}	10000
Queso	{3,5}	00101
Huevos	{2,3,4}	01110
Leche	{1,2,3,4,5}	11111
Yogurt	{2,4,5}	01011

Para calcular el soporte, se hace la intersección de *tids*

- El $\text{sup}(\{\text{pan}, \text{huevos}\}) =$
- El $\text{sup}(\{\text{huevos}, \text{leche}, \text{queso}\}) =$
- El $\text{sup}(\{\text{queso}, \text{leche}\}) =$

- El método de representación vertical es más eficiente al momento de buscar elementos frecuentes, sin embargo, requiere mucha memoria debido a que va almacenando las listas de *tids*.
- Para solucionar este problema, se ha propuesto dividir la DB en pequeños fragmentos los cuales se procesan de forma independiente (*partitioned ensemble*)

- El algoritmo FP-Growth (Frequent Pattern growth) fue propuesto por Jiawei Han y otros en el 2000.
- Se deriva del algoritmo Apriori pero se caracteriza por su eficiencia y escalabilidad.
- Es uno de los algoritmos más populares.

¿CÓMO FUNCIONA EL FP-GROWTH?

- En la primera iteración, recorre la BD identificando la frecuencia de cada *item* (los cuales almacena).
- Los *items* son ordenados en forma descendiente en función a su frecuencia.
- En la siguiente iteración, construye el árbol.

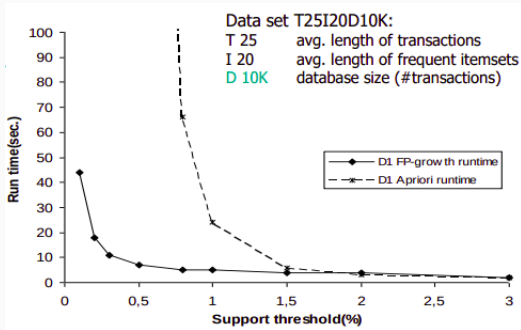
- Una de las ventajas es que optimiza el espacio en memoria.
- Una limitación es la complejidad del procesamiento (recursividad).
- Otra limitación es que si la BD cambia es necesario reconstruir el árbol completo.

- FP-Growth*: emplea una estructura llamada *FP-array* (*Frequent Pair Array*)
 - Mejora el rendimiento en BD dispersas
 - El 80 % del tiempo de procesamiento en el FP-growth se desperdicia en los recorridos en la FP-Tree.
 - Mejora el tiempo de procesamiento, aunque la construcción de matrices implica mayor demanda de memoria.

- SimpleARM: elimina por completo la construcción del FP-tree.
 - Realiza un solo escaneo a la BD.
 - Por cada *item* va construyendo una matriz con la información recolectada.
 - Es efectivo para extraer conjuntos de *items* de gran tamaño.

COMPARATIVO

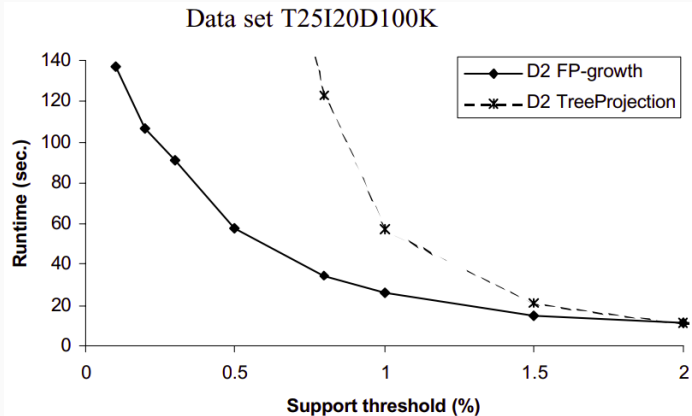
En el 2000, Han, Pei & Yin realizaron un estudio para comparar el rendimiento del algoritmo Apriori vs el FP-Tree



FP es más rápido que Apriori, incluso más rápido que TreeProjection.

COMPARATIVO

En el 2000, Han, Pei & Yin realizaron un estudio para comparar el rendimiento del algoritmo Apriori vs el FP-Tree



- El algoritmo FP no genera elementos candidatos
- El Algoritmo apriori va avanzando en primero amplitud
- FP usa una estructura compacta
- En FP, al escanear la BD, elimina elementos repetidos
- En FP, las operaciones básicas son conteo y la construcción del árbol.