Grado en Ingeniería Informática

Escuela Politécnica Superior de Jaén

UNIVERSIDAD DE JAÉN
Escuela Politécnica
Superior Jaén

Trabajo Fin de Grado

FP(2)-GROWTH. MINERÍA DE DATOS DIFUSOS EN PARALELO

Alumno: Rafael Gómez Cañizares

Tutor: Jose María Serrano Chica
Dpto: Informática

Septiembre , 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don JOSÉ MARÍA SERRANO CHICA , tutor del Proyecto Fin de Carrera titulado: FP(2)-GROWTH: MINERÍA DE DATOS DIFUSOS EN PARALELO, que presenta RAFAEL GÓMEZ CAÑIZARES, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2021

El alumno: Los tutores:

JOSÉ MARÍA SERRANO CHICA

RAFAEL GÓMEZ CAÑIZARES

Índice

1. Introducción

Entre las fases de un proceso de extracción del conocimiento en bases de datos (KDD, Knowledge discovery on databases) se encuentra la minería de datos (DM, data mining), que nos permite tallar patrones o tendencias en la información previamente desconocidas, no triviales y potencialmente útiles.

Muchos son los algoritmos y metodologías existentes hoy en día aplicados a la resolución de este tipo de problemas. En particular, uno de los algoritmos más extendidos reconocidos es FP-Growth, que nos permite obtener resultados en tiempos razonables y de manera eficiente, debido a la particularidad de que reduce el número de accesos a la base de datos con respecto a las anteriores propuestas.

Uno de los problemas derivados de la minería de datos está no sólo en la gran cantidad y variabilidad de la información sobre la que se puede aplicar, sino en el formato y en la naturaleza de la misma. Pueden aparecer factores de imprecisión o vaguedad que precisen del uso de herramientas adicionales para manejar adecuadamente dicha información. Un caso típico es el uso de la teoría de subconjuntos difusos, que permite tratar con información incierta o imprecisa mediante la definición de grados de pertenencia de un elemento a un conjunto. Una alternativa reciente es la representación por niveles (RL, Representation Levels), mediante la cual se descompone el concepto de conjunto difuso como una colección de diferentes niveles, donde los elementos pertenecen con total certeza.

1.1. Propósito General y Objetivos

Los objetivos buscados con el presente TFG son los siguientes:

- Permitir la formación del alumno en aspectos referidos a la investigación y a la transferencia de resultados
- Realizar un estudio teórico de las herramientas y técnicas de representación ya existentes con respecto al problema de partida

- Estudiar la viabilidad de la propuesta de solución al problema
- Desarrollar una herramienta informática, robusta y escalable, que permita resolver el mismo
- Analizar los resultados obtenidos

1.2.Metodología a Desarrollar

Los pasos recomendados para llevar a cabo el presente Trabajo Fin de Grado se pueden resumir en los siguientes puntos:

- Estudio del problema a tratar, de las técnicas de representación y resolución existentes, y de las herramientas disponibles.
- Análisis de la solución propuesta y diseño e implementación de una herramienta informática.
- Diseño de casos de experimentación y análisis de los resultados obtenidos.
- Todas estas tareas se documentarán durante su desarrollo, para conformar una memoria final del trabajo realizado.

1.3.Planificación del Proyecto.

1.3.1. Planificación de tareas

En la tabla 1 se muestra la planificación de tareas asociada al proyecto:

Tarea	Duración (horas)
Búsqueda de información sobre algoritmos de búsqueda de patrones frecuentes y reglas de asociación.	25
Estudio de algoritmos de búsqueda de patrones frecuentes.	20
Implementación del algoritmo FP-Growth	70
Adaptación del algoritmo FP-Growth al algoritmo FP(2)-Growth propuesto en este	80

trabajo.	
Búsqueda y/o creación de conjuntos de datos adecuados y pruebas del algoritmo.	20
Corrección de errores.	40
Redacción de la memoria.	40
Total	300

Tabla 1.1

1.3.2. Diagrama de Gantt

El diagrama de Gantt es una herramienta de gestión de proyectos en la que se recoge la planificación de un proyecto. Normalmente tiene dos secciones: en la parte izquierda se incluye una lista de tareas y, en la derecha, un cronograma con barras que representan el trabajo. Los diagramas de Gantt también pueden incluir las fechas de inicio y de finalización de las tareas, los hitos, las dependencias entre tareas y las personas asignadas.

En la ilustración 1 se presenta el diagrama de Gantt asociado a la planificación de mi proyecto.

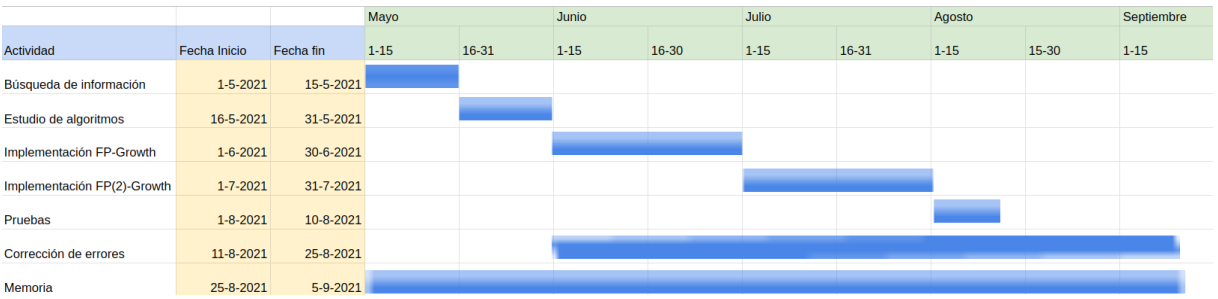

Ilustración 1

2. Extracción de reglas de asociación.

2.1. Búsqueda de patrones frecuentes

El descubrimiento de patrones frecuentes consiste en encontrar patrones frecuentes y relevantes en conjuntos de datos grandes. Los patrones comunes se definen como subconjuntos de un conjunto de datos cuya frecuencia es no menor que una frecuencia determinada por el usuario o generada automáticamente. [1]

Existen muchas técnicas para descubrir patrones frecuentes, a continuación explico brevemente algunos de ellos:

- **Clustering**: Consiste en analizar un conjunto de objetos de forma que los objetos que están en el mismo grupo (llamado cluster) son más similares entre ellos que a objetos de otros grupos.
- **Clasificación**: Trata de asignar objetos a distintas categorías diferenciadas según sus parámetros. Habitualmente se organizan en árboles.
- **Sistemas de recomendación**: Es un tipo de sistema de filtrado de información que trata de predecir las preferencias que tiene un usuario respecto a un ítem. Se suele utilizar en servicios de streaming de música y video.

Sin embargo, el descubrimiento de patrones frecuentes se puede realizar utilizando técnicas de aprendizaje de reglas de asociación. Algunos algoritmos destacados de este tipo son Apriori, Eclat y FP-Growth.

El concepto de reglas de asociación[12] se popularizó en 1993 debido a un artículo de Agrawal et al.[3] que ha adquirido más de 23.790 citaciones según Google Scholar y es uno de los artículos más citados en el ámbito de la minería de datos. Sin embargo, lo que conocemos ahora como reglas de asociación se introdujo en un artículo en 1966 [4].

2.1.1. La historia del pañal y la cerveza.

Una anécdota habitual utilizada comúnmente como ejemplo sobre cómo se pueden encontrar reglas de asociación incluso en los datos más mundanos, y recalcar la importancia de la minería de reglas de asociación es la historia de “los pañales y la cerveza”, en la que una encuesta realizada por un supermercado

descubrió que los clientes (probablemente hombres jóvenes) que compraban pañales tenían mayor tendencia a comprar cerveza. Aunque hay algunas discrepancias sobre si esta historia es cierta, el supermercado podría haber hecho uso de esta información para aumentar sus ventas (como colocar ambos productos más cerca o crear promociones sobre estos dos productos).

2.2. Revisión de trabajos anteriores

2.2.1. Apriori

2.2.1.1. Contexto

El algoritmo Apriori [5] es un algoritmo propuesto en 1994 por Rakesh Agrawal y Ramakrishnan Srikant para operar sobre bases de datos relacionales que contienen transacciones. Se trata de uno de los algoritmos más antiguos pero aún sigue siendo relevante incluso si se conocen algoritmos más eficientes, en parte por su importancia histórica.

2.2.1.2. Explicación

Cada transacción es vista como un conjunto de ítems. Dado un umbral C, el algoritmo Apriori identifica todos los conjuntos de items que son subconjuntos de al menos C transacciones en la base de datos. A priori utiliza un enfoque “bottom up”, donde subconjuntos frecuentes son ampliados por un ítem a la vez y grupos de candidatos son validados contra los datos.

Se genera un conjunto con todos los ítems cuya frecuencia es mayor o igual a un valor establecido por el usuario o calculado automáticamente. Una vez obtenido este grupo utilizaremos un valor k que iremos incrementando en 1 cada vez que realicemos una iteración. En cada iteración crearemos conjuntos de ítems candidatos de tamaño k a partir de conjuntos de items de tamaño k-1. Un conjunto de tamaño k sólo se puede obtener de dos conjuntos de items de tamaño k-1 si coinciden k-2 elementos entre ellos. Los conjuntos candidatos cuya frecuencia sea menor que el valor establecido se descarta. Los conjuntos candidatos que contienen subconjuntos que han sido descartados también se descartan. El algoritmo se

ejecuta hasta que no se puede crear un grupo de tamaño k a partir de conjuntos de tamaño k-1

En la tabla 2.1 se muestra ahora un ejemplo: [6]

TID	Ítems
T1	I1,I2,I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

Tabla 2.1

Dada esta tabla de transacciones , estableciendo además el soporte mínimo como 2 y la confianza mínima como 0.6:

1. Paso 1: k=1

Se crea una tabla que contiene los soportes de cada ítem presente en el dataset. Después se compara el soporte de cada candidato con el mínimo soporte establecido (en este caso 2). Si el soporte del candidato es menor que el soporte mínimo se elimina el candidato.

La tabla 2.2 muestra los resultados.

Itemset	Sup_count
I1	6
I2	7

I3	6
I4	2
I5	2

Tabla 2.2

2. Paso 2: k=2

Se genera el segundo set de candidatos. El segundo set de candidatos se compone de todos los subconjuntos de tamaño k de candidatos que no han sido eliminados en el anterior paso. Después se comprueban los soportes de cada subconjunto y nuevamente, si son menores que el soporte mínimo se descartan. La Tabla 2.3 muestra el resultado antes de realizar los descartes y la Tabla 2.4 muestra el resultado tras realizar los descartes.

Itemset	**Sup_count**
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

Tabla 2.3

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

Tabla 2.4

3. Paso 3

Se utiliza el subconjunto anterior para generar subconjuntos de 3 elementos. Nuevamente se calculan los soportes y se descartan subconjuntos cuyo soporte sea menor que el mínimo. Para unir dos subconjuntos es necesario que tengan k-2 elementos en común. Por ejemplo, podemos unir {i1,i2} y {i1,i3} porque k=3 y tienen un elemento en común. La tabla 2.5 muestra los resultados obtenidos tras generar los subconjuntos y eliminar los que tienen soportes menor que el mínimo.

Itemset	Sup_count
I1,I2,I3	2
I1,I2,I5	2

Tabla 2.5

4. Paso 4

Se trata de generar un nuevo conjunto de candidatos de 4 elementos. En este caso podemos generar el subconjunto {i1,i2,i3,i5}, pero contiene el subconjunto {i1,i3,i5} que no es común así que detenemos aquí la generación de subconjuntos.

5. Paso 5

En condiciones normales, el algoritmo se detiene en el paso 4, pero se puede ampliar su funcionalidad para incluir reglas de asociación de la siguiente forma:

Una vez descubiertos todos los itemsets frecuentes podemos generar reglas de asociación. Para ello calculamos la confianza de cada regla.

$$Confianza(A \text{ -> } B) = Soporte(A \cup B)/Soporte(A)$$

Tomando como ejemplo el itemset frecuente {i1,i2,i3} podríamos generar las reglas:

- $Confianza([I1, I2] \text{ -> } [I3]) = Soporte([I1, I2, I3])/Soporte([I1, I2]) = 0.5$
- $Confianza([I1, I3] \text{ -> } [I2]) = Soporte([I1, I2, I3])/Soporte([I2, I3]) = 0.5$
- $Confianza([I2, I3] \text{ -> } [I1]) = Soporte([I1, I2, I3])/Soporte([I2, I3]) = 0.5$
- $Confianza([I1] \text{ -> } [I2, I3]) = Soporte([I1, I2, I3])/Soporte([I1]) = 0.33$
- $Confianza([I2] \text{ -> } [I1, I3]) = Soporte([I1, I2, I3])/Soporte([I2]) = 0.28$
- $Confianza([I3] \text{ -> } [I1, I2]) = Soporte([I1, I2, I3])/Soporte([I3]) = 0.33$

Dado que hemos establecido un nivel de confianza mínimo de 0.6, ninguna de estas reglas pueden considerarse como reglas de asociación fuertes.

2.2.1.3. Explicación

A pesar de su relevancia histórica, el algoritmo Apriori sufre problemas que han provocado la aparición de nuevos algoritmos más eficientes con el tiempo. Estos problemas son:

- La generación de candidatos genera una lista de subconjuntos demasiado grande. Esto es lento y **muy** exigente en la memoria en conjuntos de datos grandes.
- El algoritmo escanea la base de datos con demasiada frecuencia, lo que reduce drásticamente el rendimiento.
- La complejidad del algoritmo es demasiado alta: O(2^n). Siendo N el número de ítems presentes en la base de datos.

2.2.2. Eclat

2.2.2.1. Contexto

El algoritmo ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal) se trata de una versión más eficiente y escalable del algoritmo Apriori. Mientras que el algoritmo Apriori imita una búsqueda en anchura leyendo las transacciones horizontalmente, el algoritmo ECLAT imita una búsqueda en profundidad leyendo las transacciones verticalmente.

2.2.2.2. Funcionamiento

ECLAT trata de utilizar conjuntos de identificadores de transacción (tidsets) para calcular el soporte de un candidato y evitar la generación de subconjuntos que no son frecuentes. Dicho de otra forma: en lugar de utilizar listas de transacciones que contienen ítems utilizamos listas de ítems que contienen identificadores de transacciones. La tabla 3.1 muestra un ejemplo[7].

ÍTEM	TIDSET
Pan	{T1,T4,T5,T7}
Mantequilla	{T1,T2,T3,T6}
Leche	{T3,T5,T6,T7}

Tabla 3.1

Al igual que el algoritmo Apriori, se establece un nivel k que determina el tamaño de los subconjuntos y se itera sobre el valor de k hasta no poder generar ningún nuevo subconjunto de candidatos.

En la primera llamada de la función se utilizan todos los ítems frecuentes junto a sus tidsets. Después la función se llama recursivamente y, en cada llamada, cada pareja de ítem-tidset se verifica y combina con otras parejas de ítem-tidset. El proceso continúa hasta que no queden candidatos.

2.2.2.3. Ejemplo

- Paso 1: Sobre el conjunto de transacciones se obtiene la lista inicial ítem-tidset, como la obtenida en la Tabla 3.1

- Paso 2: Construimos la nueva tabla para k=2 llamando al método de forma recursiva hasta no poder generar más pares de ítem-tidset. El resultado se muestra en la tabla 3.2

Ítem	Tidset
{Bread,Butter}	{T1,T4,T8,T9}
{Bread,Milk}	{T5,T7,T8,T9}
{Bread,Coke}	{T4}
{Bread,Jam}	{T1,T8}
{Butter,Milk}	{T3,T6,T8,T9}
{Butter,Coke}	{T2,T4}

{Butter,Jam}	{T1,T8}
{Milk,Jam}	{T8}

Tabla 3.2

- Paso 3: Repetimos para k=3 tal como se muestra en la tabla 3.3

Ítem	Tidset
{Bread,Butter,Milk}	{T8,T9}
{Bread,Butter,Jam}	{T1,T8}

Tabla 3.3

- Paso 4: Repetimos para k=4 tal como se muestra en la tabla 3.4

Ítem	Tidset
{Bread,Butter,Milk,Jam}	{T8}

Tabla 3.4

2.2.2.4. Ventajas sobre Apriori

- Memoria: Dado que ECLAT utiliza un método de búsqueda en profundidad utiliza menos memoria que apriori.
- Velocidad: El algoritmo ECLAT es típicamente más rápido que Apriori.
- Computación: El algoritmo ECLAT no requiere leer los datos repetidas veces para realizar sus cálculos, lo cual aumenta más su rendimiento con respecto Apriori.

2.2.3. FP-Growth

2.2.3.1. Contexto

El algoritmo FP-Growth[8] es un algoritmo propuesto por Jiawei Han, Hong Cheng, Dong Xin y Xifeng Yan en 2006 que trata de resolver los problemas del algoritmo Apriori.

Este algoritmo mejora el rendimiento comparado con el algoritmo Apriori. Los patrones frecuentes se encuentran sin necesidad de generar candidatos, ahorrando comprobaciones innecesarias (que aún se hacían en ECLAT). Además, se representa la base de datos en forma de árbol llamado FP-Tree, que organiza los datos de forma fácilmente accesible, ahorrando lecturas innecesarias de la base de datos.

2.2.3.2. Explicación

El algoritmo FP-Growth lee 2 veces la base de datos: La primera vez cuenta cuántas veces se repite un ítem en el conjunto de transacciones. En el segundo barrido construye la estructura FP-Tree para acceder fácilmente a los datos.

2.2.3.3. Header Table

En el primer barrido se comprueban cuántos ítems distintos se producen entre todas las transacciones y se crea una estructura *Header Table*. Si el número de veces que aparece un ítem entre todas las transacciones es mayor o igual al soporte mínimo especificado por el usuario se almacena en una lista perteneciente a la Header Table el id del nodo, su contador y un puntero que apuntará al primer nodo con dicho id cuando construyamos el árbol. Se muestra a continuación en la tabla 4.2 la Header Table asociada al conjunto de datos de la tabla 4.1 suponiendo una frecuencia mínima para ser considerado frecuente de 2.

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a.d.e}
5	{a,b,c}

6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Tabla 4.1

ID	Contador	Siguiente
a	8	Null
b	6	Null
c	6	Null
d	5	Null
e	3	Null

Tabla 4.2: Header Table asociada a la tabla 4.1

En este caso todos los ítems se almacenan en la tabla pero no hubiera sido así si hubiéramos escogido como frecuencia mínima, por ejemplo, 6. En cuyo caso sólo hubieran sido añadidos en la tabla los ID {a,b,c}. Inicialmente todos los nodos apuntan a nulo porque el árbol no ha sido construido todavía. Al insertar nuevos nodos en el árbol se comprueba la Header Table y se actualizan los enlaces al siguiente nodo.

2.2.3.4. Estructura FP-Tree

La estructura FP-Tree se trata de un árbol no binario en el cual cada nodo forma parte de una lista enlazada si y solo si hay más de un nodo que almacena el mismo item. El nodo raíz no tiene ningún valor y sólo se utiliza para almacenar a sus hijos. Cada nodo que no sea raíz almacena el identificador del ítem, un contador de cuántas veces se ha pasado por este nodo al crear el FP-Tree, sus nodos hijos y opcionalmente el siguiente nodo en la lista enlazada a la que pertenece. En la ilustración 2 se muestra ahora un ejemplo [9].

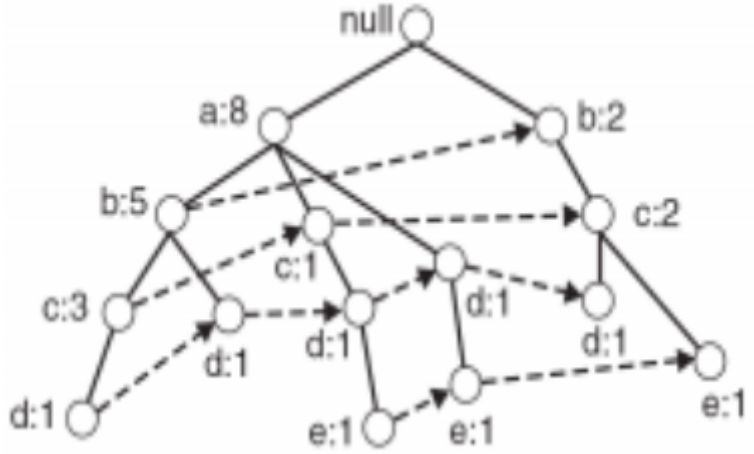

Ilustración 2.1: Representación de un FP-Tree asociado a la tabla

Para construir el FP-Tree se lee la tabla de transacciones transacción a transacción. Para cada ítem en la transacción, comprobamos si el nodo actual tiene algún hijo con el mismo id que el nombre del ítem. Si lo tiene sumamos 1 a su contador y si no lo tiene creamos un nuevo nodo. Después actualizamos el nodo actual al nuevo nodo que hemos creado o al nodo cuyo contador hemos actualizado.

Si se ha insertado un nuevo nodo se debe comprobar si hay ya algún nodo existente utilizando la header table. Si ya existe un nodo con el mismo dato se actualiza la lista enlazada.

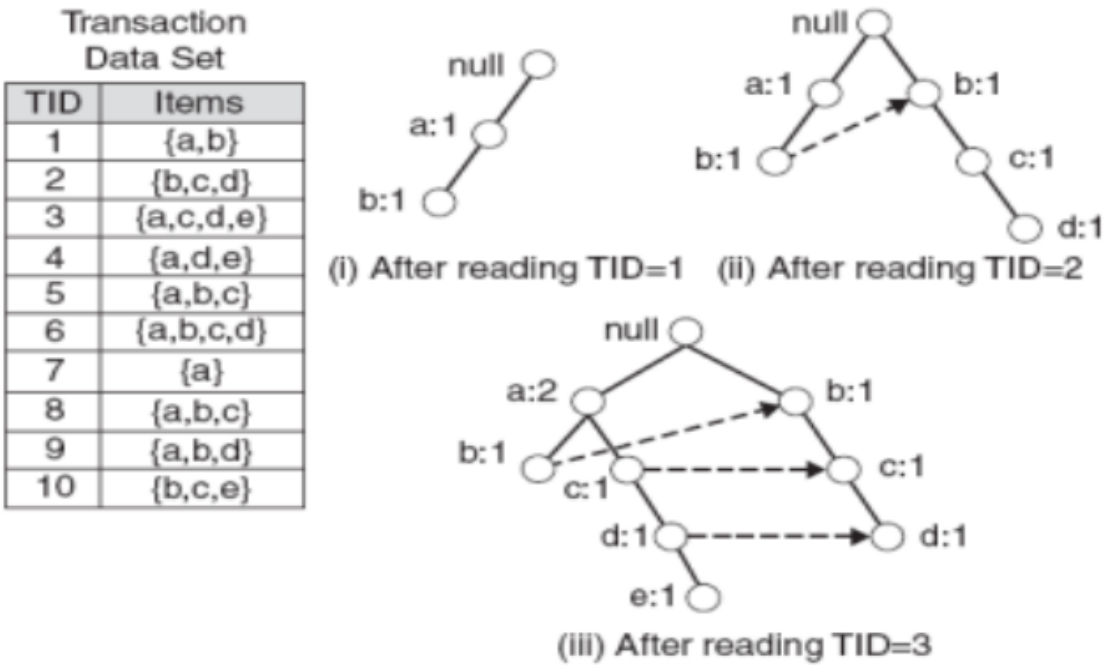

Ilustración 3: Caminos obtenidos leyendo el árbol de abajo hacia arriba.

Observando la ilustración 3 podemos observar cómo construye el árbol: Lee la primera transacción {a,b} y comprueba si hay un nodo con valor a que sea hijo del nodo raíz. Como no lo hay añade un nuevo nodo con valor a y comprueba si el ID a está almacenado en la Header Table y si lo está recorre la lista enlazada asociada a ese ID y añade el nodo al final. Después comprueba si hay algún nodo con valor b que sea hijo del nodo a. Tampoco lo hay así que lo añade y realiza el mismo procedimiento comprobando la Header Table y añadiendo el nodo al final si aplica.

Después lee la segunda transacción{b,c,d} y comprueba si hay algún nodo con valor b que sea hijo del nodo raíz. Como no lo hay añade un nodo con valor b como hijo del nodo raíz y comprueba si hay algún nodo con valor c que sea hijo de b. Inserta el nodo c y realiza la misma comprobación con d, que es también insertado. Para los 3 nodos se comprueban si su ID está en la Header Table y se añaden al final si aplica.

A continuación lee la tercera transacción {a,c,d,e} y comprueba si hay algún nodo con valor a que sea hijo del nodo raíz. Como sí existe le suma 1 al contador de frecuencias de ese nodo. Como no se ha añadido un nodo nuevo no se realiza

ninguna comprobación en la Header Table. Ahora comprueba si ese nodo a tiene algún hijo con valor c, y como no tiene lo añade. Comprueba si el nodo c tiene algún hijo con valor d y lo añade como hijo y potencialmente a la lista enlazada y comprueba si el nodo d tiene algún hijo con valor e y lo añade como hijo y potencialmente a la lista enlazada..

El tamaño del árbol depende en gran parte del orden en el que están los ítems en cada transacción. En ocasiones es posible crear árboles más pequeños si los ítems están ordenados según su soporte.

2.2.3.5. Algoritmo

Al igual que el algoritmo Apriori, este algoritmo recibe un parámetro soporte mínimo.

1. Para cada item cuyo soporte sea mayor o igual al soporte mínimo (referenciar header table) se localizan los nodos que almacenan el item y se lee el árbol hacia arriba para encontrar los caminos que llevan a este nodo (el nodo raíz no forma parte del camino)
2. Para los nodos que forman dichos caminos (excepto el último), se suman los distintos contadores asociados a distintos items y se obtienen todos los subgrupos de los ítems pertenecientes al camino cuyo soporte mínimo sea mayor o igual al soporte mínimo que recibe el algoritmo (el soporte mínimo de un subgrupo es el soporte del elemento con menor soporte).
3. El algoritmo devuelve todos los subconjuntos de items frecuentes.

A continuación se muestra un ejemplo.[10]

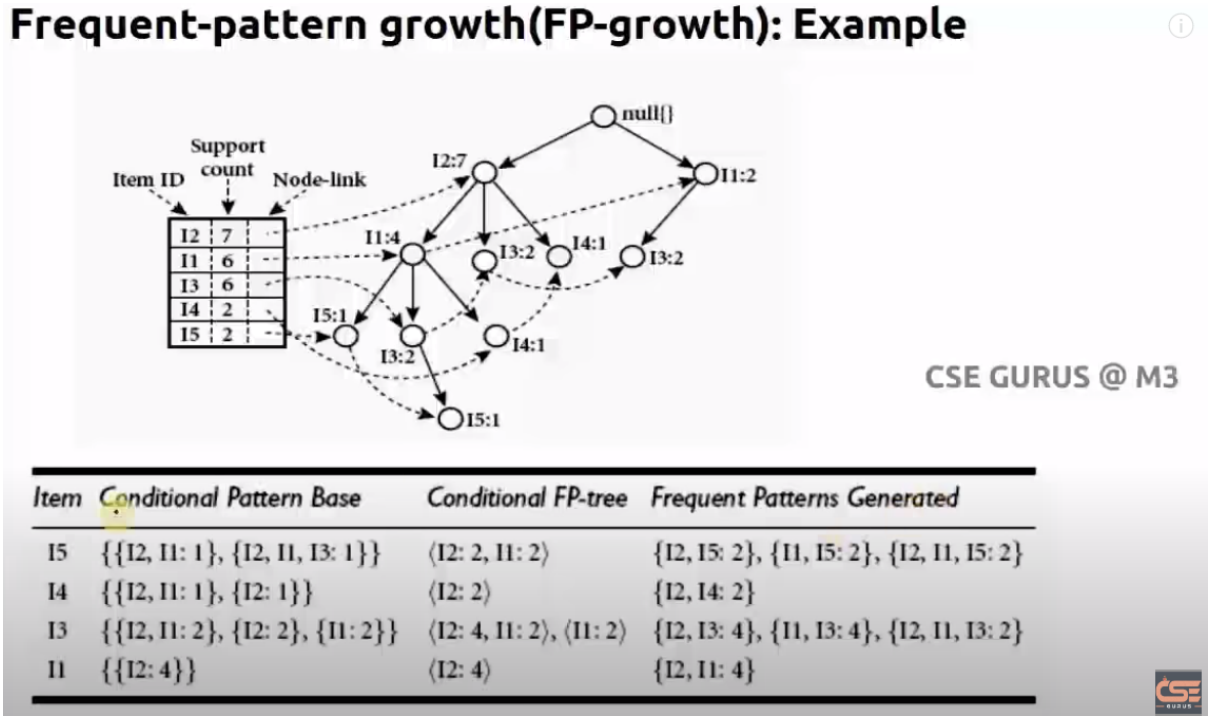

Ilustración 4: Ejemplo visual del algoritmo FP-Growth.

Observando en la ilustración 4, por ejemplo, el id **i5**, podemos observar que hay dos nodos con este valor, cada uno con su propio camino. Leyendo desde abajo hacia arriba (utilizando la header table para empezar a leer desde abajo) vemos que los dos caminos son {i2,i1} y {i2,i1,i3}. I2 y I1 aparecen 2 veces, pero I3 aparece una vez. Si utilizamos un soporte mínimo de 2 descartamos el ID I3. Para los ID restantes se generan todos los subconjuntos posibles que contienen **i5** (en este caso {i1,i5},{i2,i5},{i1,i2,i5}). el soporte de cada subconjunto es el menor de los soportes de los elementos que los componen. En este caso todos los soportes son 2 así que cada subgrupo tiene un soporte de 2. Todos los subconjuntos se consideran frecuentes.

Incluyo, a continuación, pseudocódigo asociado a este algoritmo.

```
Lista<Lista<String>> FPGrowth(Lista<Transacciones>, int minSoporte) {
    // Recorre la lista de transacciones por primera vez para generar la header table.
    Frecuencia[] frecuencias;
```

```
for(transaccion in transacciones){
    for(item in transaccion){
        frecuencias[item.id]++;
    }
}

// Compruebo la frecuencia asociada a cada item distinto y construyo la header table
for(frecuencia in frecuencias){
    if(frecuencia >= minSoporte) {
        // Añade a la headertable un nodo con el id y frecuencia correspondientes.
        headerTable.add(i,frecuencia,null);
    }
}

// Construye el árbol (segunda lectura)
FPTree arbol = construirFPTree(Lista<Transacciones>);

Lista<Lista<Nodo>> solucion;

for(elemento in headerTable){
    Lista<Nodo> camino;
    nodo=elemento.siguienteNodo;
    while(nodo.siguienteNodo!=null){
        // Obtener camino hacia la raiz
        nodoarriba=nodo;
        // El nodo raíz es nulo así que no se lee
        while(nodo.padre != null) {
            camino.add(nodoarriba);
            nodoarriba=nodoarriba.padre;
        }
```

```
                    for(subconjunto in camino.getSubconjuntos()){
                        if(subconjunto.getMenorFrecuencia() > soporteMin)
                            solucion.add(subconjunto);
                    }
                    nodo=nodo.siguienteNodo;
                }
            }

            return solucion;

}

FPTree construirFPTree(Lista<Transacciones>){
        Nodo raiz=Nodo(null);
        for(transaccion in Transacciones){
            Nodo nodoactual = raiz;
            for(item in transaccion){
                // Si no hay ningun nodo hijo con el id de transaccion
                // Se añade uno.
                if(!nodoactual.hijos.containsID(item)){
                    Nodo hijo = new Nodo(item);
                    nodoactual.hijos.insert(hijo);
                    // Se añade a la headertable si la frecuencia
                    // es lo bastante alta
                    if(headertable.containsID(item)){
                        // Lo inserta al final de la lista enlazada
                        headerTable.insertarFinal(New Nodo(item));
                    }
                    nodoactual=hijo;
                }
                // Si se localiza un nodo se suma 1 a su soporte
                else{
```

```
                        nodoactual.hijos.getHijo(item).soporte++;
                        nodoactual=nodoactual.hijos.getHijo(item);
                    }
                    // En cualquier caso se ha actualizado nodoActual.
                }
            }
}
```

2.2.3.6. Ventajas y desventajas

Ventajas del algoritmo FP-Growth:

- El algoritmo sólo necesita recorrer el dataset 2 veces.
- La estructura FP-Tree permite desplazarse rápidamente por los datos.
- No se generan candidatos (más rápido que Apriori y Eclat).

Desventajas del algoritmo FP-Growth:

- La estructura FP-Tree puede ser muy exigente en memoria en datasets grandes.
- Construir el FP-Tree es costoso, sobre todo cuando se tienen muchos items distintos.

2.3. Mejoras propuestas sobre trabajos existentes.

Todos los algoritmos expuestos anteriormente tienen un problema en común:

No pueden trabajar con valores imprecisos o inciertos. Estudiando una cesta de la compra podemos afirmar que una barra de pan está o no está, pero estudiando si en un conjunto de días hace calor nos encontramos con un atributo vago, impreciso e inconsistente. Puede hacer mucho valor, o poco calor. El día puede estar en un punto en el que no hace calor pero tampoco frío.

Para trabajar con este tipo de datos imprecisos se utiliza la lógica difusa[11]. La lógica difusa (también llamada lógica borrosa) es una lógica paraconsistente que identifica fracciones de valores verdaderos entre 0 y 1 de forma gradual, siendo 1

“completamente verdadero” y siendo 0 “completamente falso”. Fue formulada por el matemático e ingeniero Lotfi A. Zadeh[12].

La lógica difusa (fuzzy logic, en inglés) permite tomar decisiones más o menos intensas en función de grados intermedios de cumplimiento de una premisa; se adapta mejor al mundo real en el que vivimos, e incluso puede comprender y funcionar con nuestras expresiones, del tipo «hace mucho calor», «no es muy alto», «el ritmo del corazón está un poco acelerado», etc.

Toma dos valores aleatorios, pero contextualizados y referidos entre sí. Por ejemplo, una persona que mida dos metros es claramente una persona alta, si previamente se ha tomado el valor de persona baja y se ha establecido en un metro. Ambos valores están contextualizados a personas y referidos a una medida métrica lineal.

La clave de esta adaptación al lenguaje se basa en comprender los cuantificadores de cualidad para nuestras inferencias (en los ejemplos de arriba, «mucho», «muy» y «un poco»).

Si un día hace “mucho” calor, podríamos representarlo en una transacción con un valor 0.8, mientras que si hace “poco” valor podríamos representarlo en una transacción con valor 0.4.

2.3.1. α-cortes y niveles de restricción

Un α-corte [13] es un valor entre 0 y 1 que se utilizará para transformar valores de lógica difusa en lógica clásica. Dada una serie de transacciones con valores comprendidos entre 0 y 1 transformamos los valores mayores o iguales que el valor del α-corte en 1 y el resto en 0. Por ejemplo, dada la siguiente transacción mostrada en la tabla 5.1 y un α-corte=0.75 obtenemos la tabla de transacciones mostrada en la tabla 5.2.

0.75	0	1
0.8	0.2	0.5
0.5	0.95	0.75

Tabla 5.1: Ejemplo de transacciones con lógica difusa.

La transacción se transforma y adquiere los siguientes valores:

1	0	1
1	0	0
0	1	1

Tabla 5.2: Tabla de transacciones obtenida al aplicar un α-corte=0.75 sobre la tabla 5.1

Mediante el uso de α-cortes podemos introducir lógica difusa en los algoritmos Apriori, ECLAT y FP-Growth y ampliar su funcionalidad.

El problema de utilizar un α-corte para tratar con valores difusos es que se trata de una aproximación. Se pierde cierta cantidad de información cada vez que realizas el corte. Además es necesario conocer cuál es el mejor punto para realizar el corte perdiendo el mínimo de información posible.

Dado un conjunto de transacciones difusas, sus niveles de restricción(RL)[10] son un conjunto de α-cortes equivalentes a cada uno de los valores distintos que puede tomar cada ítem en cada transacción (excluido el valor 0). Por ejemplo, dada la penúltima tabla los niveles de restricción obtenidos son {1,0.95,0.8,0.75,0.5,0.2}

2.3.2. Soporte, confianza y grado de certidumbre.

Dados un conjunto de transacciones, el soporte de una regla de asociación es el ratio de transacciones en el que aparecen todos los ítems asociados a la regla juntos. Mientras que la confianza es el ratio de transacciones en las que cuando se manifiesta el lado izquierdo de la regla de asociación también se manifiesta el derecho.

La confianza viene dada por la siguiente fórmula:

$$Conf(A -> C) = Supp(A \cup C)/Supp(A)$$

Siendo Supp el soporte del conjunto de ítems.

Una medida más elaborada de la calidad de una regla de asociación que vamos a utilizar es el grado de certidumbre, cuya formula es:

$CF(A -> C) = (Conf(A - C) - Supp(C))/(1 - Supp(C))$ Si $Conf(A - C)$ >= $Supp(C)$

ó

$CF(A -> C) = (Conf(A - C) - Supp(C))/(1 - Supp(C))$ Si $Conf(A - C)$ < $Supp(C)$

3. Algoritmo FP$^{(2)}$-Growth

El algoritmo FP$^{(2)}$-Growth es nuestra mejora propuesta sobre el algoritmo FP-Growth, que haciendo uso de un conjunto de α-cortes determinado los niveles de restricción asociados al conjunto de datos permite trabajar con valores difusos.

El propósito de los valores de restricción es limitar el impacto que tiene en la precisión de nuestro tratamiento de datos el utilizar un α-corte como método aproximado. Mediante el uso de múltiples α-cortes determinados por los niveles de restricción y combinando los resultados podemos obtener resultados muy próximos a los resultados reales con un costo computacional barato.

3.1. Mejoras propuestas sobre trabajos existentes

Además de lógica difusa, podemos utilizar este método para transformar valores reales en lógica difusa recorriendo los valores de cada ítem y expresándolos en tanto por 1 como se muestra en la tabla 6.1.

i1	i2	i3		i1	i2	i3
1	2	0.5	=>	0.33	1	0.5
1.5	2	0.3		0.5	1	0.3
3	1	1		1	0.5	1

Tabla 6.1: Se muestra la transformación de una tabla de transacciones de números reales a números en tanto por uno.

1. Una vez obtenida esta matriz, se leen todos los datos distintos de esta matriz de transacciones y se almacena cada valor distinto como nivel de restricción. Se calcula una distribución de probabilidad determinada por $m(Y)=RL_k + RL_{k+1}$. Cada valor de m(Y) está asociado a su valor de RL_K.

2. A continuación, se lanzan tantos hilos en paralelo como el número de niveles de restricción que hemos obtenido, recibiendo cada hilo la matriz, un nivel de restricción distinto y el soporte mínimo para que un itemset sea frecuente.

3. Cada hilo utiliza el nivel de restricción que recibe como α-corte y genera una matriz “crisp” cuyos valores son 0 o 1 y ejecuta el algoritmo FP-Growth sobre la matriz obtenida, generando su propio FP-Tree y comprobando un soporte mínimo especificado por el usuario.

Por supuesto, una matriz de 0 y 1 puede representarse como una tabla de transacciones como se muestra en las tablas 6.2 y 6.3.

RL=0.5	i1	i2	i3
T1	0	1	1
T2	1	1	0
T3	1	1	1

Tabla 6.2: Matriz “crisp” de valores 0 y 1.

RL=5	Ítems
T1	i2,i3
T2	i1,i2
T3	i1,i2,i3

Tabla 6.3:Lista de transacciones asociada a la tabla 6.2.

Una vez ejecutados todos los hilos recibiremos una serie de subconjuntos asociados al nivel de RL para el cual hemos ejecutado el algoritmo FP-Growth, tal y como se aprecia en la tabla 6.4.

Resultados para RL 1.0	{}
Resultados para RL 0.95	{[i1,i2],[i1,i3],[i2,i3],[i1,i2,i3]}
Resultados para RL 0.8	{[i1,i2],[i1,i3],[i2,i3],[i1,i2,i3]}
Resultados para RL 0.3	{[i1,i2],[i1,i3],[i2,i3],[i1,i2,i3]}

Tabla 6.4: Resultados obtenidos tras ejecutar FP-Growth en paralelo.

4. Para cada subconjunto de resultados se calcula el producto cartesiano y se comprueba si alguno de estos productos genera una regla de asociación fuerte utilizando como medida el grado de certidumbre[10]. Por ejemplo, para el subconjunto {I1,I2,I3} se generan las reglas de asociación:

- {i1,i2}->{i3}
- {i1,i3}->{i2}
- {i2,i3}->{i1}
- {i1}->{i2,i3}
- {i2}->{i1,i3}
- {i3}->{i2,i1}

5. El grado de certidumbre se calcula a través del grado de confianza que a su vez se calcula a través del soporte de un ítem dadas las siguientes formulas, donde RL indica el nivel de restricción, Supp() el soporte de una regla de asociación, Conf() la confianza de una regla y CF() el factor de certidumbre de una regla. Dado que estamos operando dentro de un nivel de restricción, las fórmulas se ven modificadas para ajustarse a nuestra aproximación como se puede apreciar en la ilustración 5[14].

Definition 5: The support of the fuzzy association rule $A \Rightarrow C$, $RL - Supp(A \Rightarrow C)$, is given by

$$RL - Supp(A \Rightarrow C) = \sum_{(A \Rightarrow C)_\alpha \in \Omega_{A \Rightarrow C}} m_{A \Rightarrow C}((A \Rightarrow C)_\alpha) \cdot Supp_\alpha(A \Rightarrow C) \quad (13)$$

Definition 6: The confidence of the fuzzy association rule $A \Rightarrow C$, $RL - Conf(A \Rightarrow C)$, is given by

$$RL - Conf(A \Rightarrow C) = \sum_{(A \Rightarrow C)_\alpha \in \Omega_{A \Rightarrow C}} m_{A \Rightarrow C}((A \Rightarrow C)_\alpha) \cdot Conf_\alpha(A \Rightarrow C) \quad (14)$$

Definition 7: The certainty factor of the fuzzy association rule $A \Rightarrow C$, $RL - CF(A \Rightarrow C)$, is given by

$$RL - CF(A \Rightarrow C) = \sum_{(A \Rightarrow C)_\alpha \in \Omega_{A \Rightarrow C}} m_{A \Rightarrow C}((A \Rightarrow C)_\alpha) \cdot CF_\alpha(A \Rightarrow C) \quad (15)$$

Ilustración 5: Adaptación de las fórmulas para obtener el soporte, confianza y grado de certidumbre de una regla de asociación adaptada para obtener mejores resultados dentro de un nivel de restricción.

6. Las reglas de asociación con un grado de certidumbre mayor a 0’7 se guardan y el resto se descartan. Finalmente contrastamos el soporte de cada regla con la matriz difusa y descartamos aquellas reglas de asociación con un soporte menor a 0’5.

3.2. Pseudocódigo

```
// Recibe las transacciones en forma de matriz de valores reales entre 1 y 0.
FP2Growth(Lista<<Lista<Float>>>matrizDifusa){
	Lista<Float> nivelesRestriccion=matrizDifusa.getDatosDistintos();
	Lista<Float> distribucionM;

	// Calcula la función de masa de probabilidad
	for(int i = 0 ; i < nivelesRestriccion.size()-1;i++){
		distribucionM.add(nivelesRestriccion.get(i)+nivelesRestriccion.get(i+1));
	}
	// El último caso es especial
	distribucionM.add(nivelesRestriccion.get(nivelesRestriccion.size()));

	for(int i = 0 ; i < nivelesRestriccion.size;i++){
	// Se lanza hilo por cada nivel de restriccion
	hilo = new Hilo(nivelesRestriccion.get(i),matrizDifusa,
distribucionM,nivelesRestriccion);
	hilo.lanzar();
	}

	esperarHilos();

	Lista<solucion> soluciones;
	for(hilo in Hilos){
		for(solucion in hilo.getSolucion()){
			if(solucion.calculaSoporte() > 0.5)
```

```
                    soluciones.add(solucion);
                }
        }

}

Lista<Lista<Lista<int>>> hilo.lanzar(){
        // Calcula la matriz de 0 y 1.
        Lista<Lista<int>> matrizCrisp=calcularMatrizCrisp(nivelRestriccion, matrizDifusa);

        Lista<Lista<Integer>> subconjuntos = FPGrowth(matrizCrisp, minSoporte);

        Lista<Lista<Lista<int>>> soluciones;
        // Cada producto cartesiano no repetido es una regla de asociación.
        for(producto in subconjuntos.productoCartesiano()){
                if(calcularCF(producto,RL,distribucionM,listaRL)> 0.7){
                        Soluciones.add(producto)
                }
        }
}

// CF se calcula a partir de la confianzaRL, que a su vez se calcula a través de soporteRL. Las fórmulas son iguales así que no escribiré todas.
calcularCF(Lista<Lista<int>> producto, float RL, Lista<Float> distribucionM, lista<Float> listaRL){
        float CFRL=RL-(CF*distribucionM[RL]);
}
```

4. Implementación del algoritmo

Para implementar este algoritmo he decidido crear una aplicación de terminal utilizando Java 16. Esta aplicación recibe 2 parámetros como argumentos: La ruta del dataset y el soporte mínimo necesario para considerar un dataset como frecuente. Esta aplicación se ejecutará en el terminal para que pueda ejecutarse como parte de una cadena de órdenes en el tratamiento de los datos.

La aplicación utilizará, además, el mismo formato que los conjuntos de datos utilizados en Weka, la herramienta de minería de datos de software libre. Esto quiere decir que:

- Las líneas que comienzan por @attribute indican datos relacionados con cada ítem.
- Las líneas que comienzan por @data contienen transacciones.
- Las líneas que comienzan por % indican comentarios.
- El resto de líneas y los comentarios serán ignoradas.

Primero, el algoritmo necesita leer los datos. Este trabajo se relega a la clase **LecturaArchivo**, que obtiene el nombre de los ítems y las transacciones. Los nombres se devuelven en forma de lista y las transacciones en forma de matriz.

Después se invoca a la clase **MatrizYNiveles**, que recibe la matriz de datos, transforma sus valores a tanto por uno y calcula sus niveles de restricción. **Para reducir el coste computacional, cuando se genera la matriz en tanto por uno se redondearán todos los valores a 2 decimales.**

A continuación se invoca en paralelo a la clase **EjecucionParalelo**, con tantos hilos como sea el tamaño de la lista de niveles de restricción. Cada objeto de la clase recibe la matriz en tanto por uno, el nivel de restricción que se le ha asignado y el soporte mínimo necesario para que un conjunto de ítems se considere

frecuente, introducido por el usuario. Se calcula la matriz de ceros y unos correspondiente al nivel de restricción actual y con dicha matriz se crea la estructura FPTree y se llama al método FP-Growth clásico. Una vez recibe los subconjuntos de ítems frecuentes realiza la operación de producto cartesiano en cada subconjunto, obteniendo todas las posibles reglas de asociación en cada subconjunto. Para evaluar la calidad de cada regla de asociación calcula el grado de certidumbre teniendo en cuenta su nivel de restricción y descarta todas las reglas con un grado de certidumbre menor a 0'7. Finalmente devuelve al método principal todas las reglas restantes.

La clase **FPTree** recibe de EjecuciónParalelo el soporte mínimo y la matriz de unos y ceros y desde el constructor llama al método *crearFPTree()* y rellena él mismo la header table y el árbol.

La clase **FPGrowth** recibe el árbol y se ejecuta de la misma forma que hemos explicado anteriormente: Para cada nodo frecuente obtiene los caminos y de dichos caminos devuelve los subconjuntos frecuentes.

Para implementar la paralelización del algoritmo he utilizado el marco Executor. La clase FPGrowth le devuelve una lista de subconjuntos a los hilos de EjecucionParalelo, que a su vez los devuelve al método principal mediante la interfaz *Future*, garantizando de esta forma que se cumplen las propiedades de seguridad y vivacidad.

Finalmente, en el nodo principal se reúnen todos las reglas de asociación que tienen un soporte mayor al 50%.

Incluyo los resultados obtenidos al ejecutar el algoritmo sobre un dataset real:

Iniciando programa. Ruta del archivo: datasets/baskball.arff, soporte mínimo: 5
La ejecución en paralelo ha terminado.
Resultados para RL 1.0: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.99: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.98: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.97: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.95: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.94: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.93: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.92: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.91: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.9: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.89: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.88: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.87: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.86: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.85: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.84: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.83: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.82: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.81: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.8: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.79: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.78: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.77: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.76: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.75: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.74: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.73: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.72: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.71: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.7: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.69: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.68: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.67: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.66: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.65: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.64: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.63: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.62: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.61: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.6: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.59: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.58: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.57: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.56: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.55: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.54: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.53: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.52: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.51: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.5: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.49: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.48: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.47: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.46: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.45: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.44: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.43: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.42: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.41: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.4: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.39: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.38: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.37: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.36: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.35: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.34: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.33: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.32: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.31: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.3: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.29: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.28: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.27: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.26: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.25: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.23: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.22: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.21: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.19: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.17: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.16: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.15: [[[1], [4]], [[4], [1]]]

Resultados para RL 0.14: [[[1], [4]], [[4], [1]]]

Soluciones únicas confidence factor mayor que 0.7 y con soporte mayor a 0.5:

height -> points_per_minute

points_per_minute -> height

Num niveles de restricción:83

Tiempo tomado para ejecutar el algoritmo (en ms):168

5. Análisis de resultados

He ejecutado el algoritmo sobre datasets de distintos tamaños para estudiar cuánto tiempo tarda en ejecutarse y la diferencia que hace incrementar el número de atributos. Los datasets han sido obtenidos directamente de Weka a través del siguiente enlace:

https://sourceforge.net/projects/weka/files/datasets/UCI%20and%20StatLib/uci-20070111.tar.gz/download?use_mirror=deac-ams

El algoritmo está preparado para trabajar con conjuntos de datos cuyos atributos son todos numéricos positivos, ya sean números enteros o decimales. Todos los datasets que he utilizado para realizar pruebas cumplen esta condición.

Dataset 1: veteran.arff	Atributos: 8	10624 ms
	Transacciones: 136	
	RL: 85	
Dataset 2: vineyard.arff	Atributos: 4	130 ms
	Transacciones: 51	
	RL: 62	
Dataset 3: baskball.arff	Atributos: 5	168ms
	Transacciones: 95	
	RL: 83	
Dataset 4: detroit.arff	Atributos:13	4 minutos 37 segundos.
	Transacciones: 12	
	RL: 70	
Dataset 5: mbagrade.arff	Atributos: 3	182ms
	Transacciones: 182	

	RL: 37	

Todas las pruebas se han realizado sobre el mismo ordenador, con las siguientes especificaciones:

- Procesador Intel(R) Core(TM) i3-3240 CPU @ 3.40GHz
- Memoria RAM: 8GiB DIMM 1333 MHz
- Disco duro SSD: 250GB CT250MX500SSD1

Se ha utilizado como soporte mínimo el equivalente al 25% de las transacciones.

Podemos observar como en los casos en los que hay pocos atributos el algoritmo es muy rápido incluso en los casos en los que hay muchas transacciones y muchos niveles de restricción, pero el tiempo que tarda en ejecutarse el algoritmo aumenta exponencialmente relacionado con el número de atributos del dataset. Esto se debe al aumento en complejidad que tiene construir estructuras FP-Tree. El uso de memoria por parte del algoritmo también se incrementa drásticamente en conjuntos de datos con muchos ítems distintos, llegando incluso a utilizar toda mi memoria RAM en conjuntos de datos con más de 15 atributos.

6. Conclusiones y líneas futuras.

El algoritmo FP$^{(2)}$-Growth presenta un alto rendimiento gracias a su combinación de los niveles de aproximación que nos permiten los métodos de búsqueda de reglas asociación difusas mediante niveles de restricción, el aprovechamiento extra de los recursos del ordenador obtenido mediante la programación paralela y el rendimiento que ofrece el algoritmo FP-Growth comparado con otros algoritmos comunes de búsqueda de patrones frecuentes.

Sin embargo, este método tiene un alto costo de memoria debido a la gran cantidad de estructuras FP-Tree que deben construirse. Al igual que otros algoritmos comunes, se pierde considerable rendimiento conforme aumenta el número de ítems, pero en menor medida que si utilizáramos otros métodos como Apriori o ECLAT.

Recordando los objetivos de este trabajo de fin de grado, se han cumplido de la siguiente manera:

- Permitir la formación del alumno en aspectos referidos a la investigación y a la transferencia de resultados: Se han utilizado distintos artículos de investigación para desarrollar este algoritmo.
- Realizar un estudio teórico de las herramientas y técnicas de representación ya existentes con respecto al problema de partida: Se han estudiado los algoritmos Apriori, Eclat y FP-Growth.
- Estudiar la viabilidad de la propuesta de solución al problema: Se ha estudiado la viabilidad la solución al problema mediante α-cortes y niveles de restricción y se ha determinado que es viable debido a su proximidad a una solución real y el costo de recursos.
- Desarrollar una herramienta informática, robusta y escalable, que permita resolver el mismo: Se ha desarrollado una herramienta informática en Java

que permite resolver el problema. Se han planteado ideas para escalar la herramienta.

- Analizar los resultados obtenidos: Se ha ejecutado la herramienta sobre distintos conjuntos de datos y se ha estudiado su rendimiento relacionado con distintos palabras. Se han planteado sus problemas y algunas mejoras que el algoritmo puede recibir.

En el futuro se puede mejorar este algoritmo de algunas de las siguientes formas:

- mediante su inclusión en alguna plataforma de minería de datos como Weka
- añadiendo funcionalidad para poder trabajar con datasets con huecos libres, ya sea deduciendo posibles valores para esos huecos, asignándole el valor más común u otro método.
- Detectando los posibles intervalos de los valores de un ítem si está indicado en su declaración (las líneas @attribute).
- Incorporación de otras posibles medidas de calidad de una regla de asociación.

7. Guía de uso

Dado que esta herramienta se trata de una aplicación de terminal debe ejecutarse desde la línea de comandos. Ya que está hecho en Java se puede ejecutar en cualquier sistema operativo sin necesidad de recompilar. Una vez posicionados en la carpeta en la que está nuestro ejecutable JAR lo ejecutamos utilizando el formato:

java -jar FP2-Growth.jar ruta soporteMinimo

Por ejemplo:
java -jar FP2-Growth.jar ../datasets/vineyard.arff 5

Si el programa no recibe exactamente 2 parámetros por pantalla, siendo uno una ruta y el otro un número entero positivo dará error. Si se invoca correctamente el programa se ejecutará y nos mostrará los resultados por pantalla.

El algoritmo, además, no funcionará con datasets que contengan datos negativos, espacios en blanco o atributos que no sean numéricos (como etiquetas).

8. Material Complementario

Junto a esta memoria se incluye el proyecto de NetBeans asociado a la implementación del algoritmo, un archivo ejecutable .jar que ejecuta el algoritmo y un conjunto de datasets con formato adecuado para el algoritmo.

Bibliografía

[1] https://en.wikipedia.org/wiki/Frequent_pattern_discovery
[2]https://en.wikipedia.org/wiki/Association_rule_learning

[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data (SIGMOD '93). Association for Computing Machinery, New York, NY, USA, 207–216. DOI:https://doi.org/10.1145/170035.170072
[4] Hájek, P., Havel, I. & Chytil, M. The GUHA method of automatic hypotheses determination. Computing 1, 293–308 (1966). https://doi.org/10.1007/BF02345483
[5] Rakesh Agrawal and Ramakrishnan Srikant Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994. URL: http://www.vldb.org/conf/1994/P487.PDF
[6] URL: https://www.geeksforgeeks.org/apriori-algorithm/

[7]https://www.geeksforgeeks.org/ml-eclat-algorithm/

[8] Jiawei Han; Hong Cheng; Dong Xin; Xifeng Yan (2007). "Frequent pattern mining: current status and future directions" (PDF). Data Mining and Knowledge Discovery. 15: 55–86. URL: https://sites.cs.ucsb.edu/~xyan/papers/dmkd07_frequentpattern.pdf
[9] Vasiljevic Vladica. vv113314m@student.etf.rs

[10] URL: https://www.youtube.com/watch?v=VB8KWm8MXss

[11]https://en.wikipedia.org/wiki/Fuzzy_logic

[12]https://www.elmundo.es/elmundo/2013/01/15/ciencia/1358256898.html

[13] Hassanzadeh, Reza & Mahdavi, Iraj & Tajdin, Ali & Mahdavi-Amiri, Nezam. (2018). An α-cut approach for fuzzy product and its use in computing solutions of fully fuzzy linear systems. International Journal of Mathematics in Operational Research. 12. 167. 10.1504/IJMOR.2018.10010214.

[14] Molina, Carlos & Sánchez, Daniel & Serrano, Jose & Vila, M.. (2009). Finding Fuzzy Association Rules via Restriction Levels. IEEE International Conference on Fuzzy Systems. 1157-1162. 10.1109/FUZZY.2009.5277100.