REDUCCIÓN DEL ESPACIO DE ALMACENAMIENTO EN MEMORIA PARA CALCULAR ITEMSETS FRECUENTES.

Hugo Humberto Morales P.[*], Jesús Alexander Aranda B.[†], María Patricia Trujillo U. [‡]

18 de septiembre de 2000

Resumen

Se utiliza una forma de prepocesamiento para el algoritmo Apriori con el propósito de reducir el espacio de almacenamiento de los itemsets candidatos. La propuesta aplicable a cualquier base de datos, consiste en lo siguiente: a) se considera un atributo específico como una variable discreta y se toman las diferentes instancias del mismo como los posibles valores de esta variable; b) se transforma la base de datos en una tabla en donde las columnas representan atributos, y las hileras tuplas. De acuerdo con las pruebas realizadas, esta propuesta permite generar menos itemsets candidatos de tamaño dos y ahorro de tiempo en el cálculo de itemsets frecuentes.

Palabras Claves: Reglas de asociación, ítemsets frecuentes, k-ítemsets candidato simple, k-ítemsets candidato conflictivo, base de transacciones simples, base de transacciones conflictivas, soporte.

1 INTRODUCCIÓN

Se han propuesto diferentes algoritmos para el descubrimiento de reglas de asociación: Apriori, Apriori Tid, y Apriori Híbrido [AMSTV96], Partition [SON95], DIC [BMUT97] y desde el punto de vista de la implementación la mayoría de los algoritmos se han centrado en la optimización de estructuras de datos que mejoren su ejecución [AMSTV96], [SA96] y [PCY95]. Para el algoritmo Apriori se ha propuesto en particular la implementación sobre Sistemas de Manipulación de Bases de Datos (SMBD) mediante SQL92 y SQL extendido objeto_relacional [STA98] y [RIC97].

La utilidad de las reglas de asociacion no se limita al campo de las asociaciones de productos en supermercados; se utiliza tambien para encontrar asociaciones en otros campos de accion como: pronosticos financieros, diseño de catalogos, detección de fraudes, entre otros.

[*]Estudiante de Ingenieria de Sistemas,Universidad del Valle, Cali Colombia. huhumor@calvin.univalle.edu.co

[†]Estudiante de Ingenieria de Sistemas, Universidad del Valle, Cali Colombia. jesarana@calvin.univalle.edu.co

[‡]Departamento de Produccion e Investigacion de Operaciones (PINO), Universidad del Valle, Cali Colombia. mapatru@pino.univalle.edu.co

Además de lo anterior, recientemente se ha propuesto su utilizacion dentro de procesos del KDD: clustering [KMO99], data cleaning y específicamente en la sustitucion de datos incompletos [R99].

Una etapa fundamental en el proceso de descubrimiento de reglas de asociación es el cálculo de ítemset frecuentes a partir de una base de datos. El cálculo de ítemset frecuentes requiere de la representación de la base de datos en una tabla de transacciones. La demanda en términos de espacio de almacenamiento en disco de la tabla de transacciones puede ser significativa, dependiendo de la estructura de almacenamiento utilizada. Tradicionalmente en la representación de la base de transacciones se ha utilizado una tabla con codificación binaria, en donde un registro corresponde a una transacción, se tienen tantas columnas como artículos (ítems) y el valor del atributo que representa a un ítem es igual a uno si el ítem está presente en la transacción que representa el registro, e igual a cero en los demás casos [SA96]. Para implementaciones sobre SMBD, en la representación de la base de transacciones se utiliza una tabla de transacciones con dos columnas de atributos: la primera contiene un identificador de la transacción (tid) y la segunda un identificador de ítems [STA98].

Nuestra propuesta consiste en una forma distinta de representar la base de datos: a) Se considera un atributo específico como una variable discreta y se toman las diferentes instancias del mismo como posibles valores de esta variable; b) Se transforma la base de datos en una tabla en donde las columnas representan atributos, y las hileras tuplas.

Esta propuesta tiene múltiples aplicaciones: bases de datos de ventas de supermercados, bases de datos de censos, inventarios, etc.. En el caso de bases de datos de transacciones de supermercados, la propuesta consiste en crear dos tablas de transacciones: la primera contiene las transacciones simples y la segunda las transacciones conflictivas. Las transacciones simples sólo incluyen una marca de un mismo producto; en las transacciones conflictivas, en cambio, se incluyen dos o más marcas de un mismo producto. Para otras bases de datos relacionales, se crea en cambio una sola base de datos de transacciones simples, donde las transacciones simples sólo incluyen un valor por cada atributo caracteristico de la base de datos relacional.

Nosotros estamos proponiendo un tipo de generalización a nivel de detalle, el cual es distinto al presentado en artículos - [SA95], [HF95] y [HF99] - cuyo objeto de estudio es la obtención de reglas de asociación generalizadas. Buscamos reducir el espacio de almacenamiento requerido para el cálculo de itemsets frecuentes, etapa crítica dentro del proceso de cálculo de reglas de asociación.

El contenido del artículo ha sido distribuido en cinco partes. En la sección 2 se presentan el concepto de reglas de asociación, incluyendo el algoritmo Apriori, la generación de itemsets candidatos y la generación de reglas. En la sección 3 se presenta nuestra propuesta de prepocesamiento de la base de datos requerida para el cálculo de ítemsets frecuentes y una poda. En la sección 4 se hace un análisis de los resultados obtenidos con la base de datos del censo de ingresos extraida de kdd.ics.uci.edu/databases/census-income/. En las secciones 5 y 6 se exponen algunas conclusiones generales y las referencias bibliográficas citadas.

2 REGLAS DE ASOCIACIÓN

El concepto de reglas de asociación presentado por Agrawal et al., 1993 [AIS93], esta basado en un conjunto de transacciones, donde cada transacción es un conjunto de literales - llamados ítems - y una regla de asociación es una expresión de la forma X$\Rightarrow$Y, donde X y Y son

conjuntos de ítems. Al porcentaje de transacciones que contienen simultáneamente a X y Y se le denomina soporte y al porcentaje de transacciones en las cuales cuando aparece X tambin aparece Y se le denomina confianza. Un ejemplo de regla de asociación es el siguiente: "el 30% de las transacciones que contienen galletas y mermelada también contienen leche; el 2% del total de transacciones contienen estos productos", en donde el soporte es 2% y la confianza es 30%. El cálculo de reglas de asociación puede revelar patrones útiles en la toma de decisiones en áreas como mercadeo, pronósticos financiero, predicción de fallas en sistemas, entre otras.

Acudimos a Agrawal et al [AIS93] y Agrawal et al [AMSTV96] para presentar los conceptos básicos de reglas de asociación. Sea $I=\{i_1, i_2, ..., i_m\}$ un conjunto de atributos binarios, llamados ítems. Sea *T* una base de transacciones. Cada transacción *t* es representada como un vector binario, con t[k]=1si en *t* fue comprado el ítem i_k y t[k]=0 si el ítem i_k no está en la compra. Por cada transacción hay una tupla en la base de datos. Es decir, si un supermercado tiene un millón de artículos (ítems), cada transacción tendrá un millón de posiciones, donde uno y cero indican si el ítem fue o no incluido en la venta.

El cálculo de reglas de asociación puede dividirse en dos etapas: a) cálculo de itemset frecuentes; b) cálculo de reglas de asociación a partir de los itemset frecuentes. Un itemset frecuente es aquel cuyo soporte es mayor o igual a un mínimo soporte definido por el usuario (minsop).

2.1 Algoritmo Apriori

Notación:

- k-itemsets: Un itemsets que tiene k ítems.
- L_k: Conjunto de k-ítemsets frecuentes. Cada miembro de este conjunto tiene dos campos: Itemsets; Contador de soporte.
- C_k: Conjunto de k-ítemsets candidatos (ítemsets potenciales frecuentes). Cada miembro de este conjunto tiene dos campos: a) ítemsets; b) contador de soporte.

```
1)  L1 = {large 1-itemsets};
2)  for (k=2; Lk−1 ≠0; k++) do begin
3)   Ck = apriori-gen(Lk−1) //New candidates
4)   forall transactions t∈D do begin
5)     Ct = subset(Ck, t); //Candidates contained in t
6)     forall candidates c∈Ct do
7)      c.count ++;
8)   end
9)   Lk = {c∈Ck | c.count≥minsup}
10) end
11) Answer = ∪k Lk
```

Figura 2.1 Algoritmo Apriori

En la figura 2.1 se presenta el algoritmo Apriori. En la primera pasada sobre la base de datos (base de transacciones) el algoritmo Apriori cuenta el número de ocurrencias de cada ítem para determinar los 1-itemsets frecuentes. Una pasada subsecuente, por ejemplo la pasada k, está compuesta de dos fases: la primera utiliza los ítemsets L_{k-1} frecuentes encontrados en la (k-1)ésima pasada para generar los ítemsets C_k candidatos, usando la función Apriori-gen descrita en la sección 2.2; la segunda recorre la base de datos para contar el soporte de los candidatos en C_k. Los candidatos en C_k que están contenidos en una transacción t dada pueden ser determinados eficientemente por el uso de un arbol hash.

2.2 Generación de candidatos

La función Apriori-gen recibe como argumento L_{k-1}, el conjunto de todos los (k-1)-ìtemsets frecuentes y retorna un superconjunto del conjunto de todos los k-ítemsets frecuentes. Inicia con el paso de join (mezcla), donde hace un join de L_{k-1} con L_{k-1} para obtener un superconjunto del conjunto final de candidatos C_k. La unión $p \cup q$ de los ítemsets $p, q \in L_{k-1}$ es insertada en C_k si sus primeros k-2 ítems son iguales:

1) insert into C_k
2) select p[1], p[2], ..., p[k-1], q[k-1]
3) from L_{k-1} p, L_{k-1} q
4) where p[1] = q[1], ..., p[k-2] = q[k-2], p[k-1] < q[k-1];

Continua con el paso de poda, donde borra todos los ítemsets $c \in C_k$ sí alguno de los (k-1)-itemsets de *c* no esta en L_{k-1}. Una prueba de la validez de este procedimiento de generación de candidatos es presentada por Agrawal et al [AMSTV96].

Se ilustran los pasos anteriores con el siguiente ejemplo: Sea L_3={{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}. Después del paso de join, C_4= {{1 2 3 4}, {1 3 4 5}}. El paso de poda borra el ítemset {1 3 4 5} por que el subconjunto {1 4 5} no está en L_3. En C_4 queda únicamente {1 2 3 4}.

2.3 Generación de reglas

Para cada itemsets frecuente *l*, se generan todas las reglas de la forma *a–>(l-a)*, donde *a* es un subconjunto de *l*, tal que el cociente soporte(*l*)/soporte(*a*) es al menos la confianza mínima. El soporte de algún subconjunto *˜a* de *a* debe ser más grande que el soporte de *a*. Más aun, la confianza de la regla *˜a –> (l - ˜a)* no puede ser más grande que la confianza de *a –> (l - a)*. Por lo tanto, si *a* no produce ninguna regla que involucre todos los ítems en *l* y además no la contenga, entonces, no habrá tampoco ninguna regla que contenga como antecedente algún subconjunto de *a*. De esto se sigue que para una regla *(l - a) –> a* confiable, todas las reglas de la forma *(l - ˜a) –> ˜a* deben ser confiables, siempre y cuando *˜a* no sea el subconjunto vacío de *a*. Por ejemplo, si la regla *AB –> CD* es confiable, entonces las reglas *ABC –> D* y *ABD –> C* deben ser confiables. Esta característica de la confianza de las reglas es análoga a la propiedad de los ítemsets frecuentes, donde si un ítemset es frecuente lo serán todos los subconjuntos del mismo. De un ítemsets frecuente *l*, primero se generan todas las reglas con un ítem en el consecuente. Usando las reglas confiables de un solo consecuente se generan entonces todas las posibles reglas con dos ítems en el consecuente, generadas a partir de *l*, y así consecutivamente.

3 PREPROCESAMIENTO DE LA BASE DE DATOS

La idea del preprocesamiento que presentamos en esta sección nace de la necesidad de establecer asociaciones entre atributos en una base de datos. Un atributo puede ser representado como una variable discreta y sus diferentes instancias toman valores nominales.

3.1 Preprocesamiento

Los requerimientos para realizar el preprocesamiento son diferentes dependiendo de si se trata o no de una base de datos de supermercados. En el caso de ventas de supermercados se requiere de: a)Una tabla de transacciones, donde los atributos presentes deben ser de tipo categórico o numérico; b)Una tabla con la descripción de la taxonomía o jerarquía de los productos del supermercado con su correspondientes categorías.

En el caso de bases de datos relacionales en general, se requiere disponer de un metadato (diccionario de datos) o de un catálogo del sistema que nos de información relativa a la semántica de las tablas dentro de la base de datos. Con base en la tabla con la descripción de la taxonomía de los productos se establece el tipo de producto y la marca. El catálogo del sistema brinda la información relacionada con el tipo de atributo y los posibles valores que toma.

Para bases de datos de supermercados se crean dos bases de transacciones: la primera denominada base de transacciones simple y la segunda base de transacciones conflictiva. Se utiliza la primera cuando la transacción se limita una sóla marca por cada producto; se utiliza la segunda en caso de transacciones que incluyen diferentes marcas de un mismo producto. En cambio en bases de datos relacionales en general se crea únicamente una base de datos de transacciones simples.

A continuación se presenta la forma de representar la base de datos ilustrando el caso de ventas de supermercados, siendo esta representación aplicable a bases de datos relacionales donde el producto es equivalente a atributo y las marcas son esquivalente a las diferentes instancias del atributo.

La representación de las bases de transacciones se hace con base en las siguientes definiciones: Sea I=I_1, I_2, ..., I_m, un conjunto de ítems(artículos); sea P_i es el conjunto de todas las marcas disponibles del producto i; sea t = $P_1 \cup P_2 \cup ... \cup P_n$ una transacción, n es el número de productos en los cuales se clasifican los artículos; cada transacción puede ser clasificada en una de las dos bases de transacciones:

- $t \in$BTS (Base de Transacciones Simple) si $\forall i$ $|P_i| \leq 1$, i = 1, 2, ..., n.
- $t \in$BTC (Base de Transacciones Conflictiva) si $\exists i$ $|Pi| \geq 2$, i = 1, 2, ..., n.

Si una transacción $t \in$BTS, t es representada como un vector de n posiciones, con $t[p] = (p * 100) + j$, si en t fue vendida la marca j del producto p, $1 \leq j \leq k$, k es el número de marcas que hay para el producto p. Si no se vendió ninguna marca del producto p, entonces $t[p] = 0$.

Si una transacción $t \in$BTC, t es representada como un vector de n estructuras, cada estructura está compuesta por un vector de k posiciones, donde k representa el número de marcas que hay del producto. La estructura para el producto que se utiliza en las transacciones de la BTC es:

Tabla 1: Base de Transacciones Simples(BTS)

#Transacción Original	Quesos	Panes	Arroces
2	102	201	301
3	103	203	303
4	0	202	302
5	101	201	300
8	103	202	302
9	101	203	301

Tabla 2: Base de Transacciones Conflictivas(BTC)

#Transacción Original	Quesos	Panes	Arroces
1	101, 0, 103	0, 202, 203	0, 0, 303
6	101, 102, 0	201, 0, 0	301, 302, 303
7	101, 102, 103	0, 0, 203	0, 302, 303
10	0, 102, 103	0, 202, 0	301, 302, 303

```
estructura producto {
    entero_corto    vector_marcas[k];
    }
```

Vector_estructuras_producto[p].vector_marcas[j] = $(100 * p) + j$, si en *t* fue vendida la marca *j* del producto *p* ($1 \leq j \leq k$, *k* es el número de marcas del producto *p*), de otro modo Vector_estructuras_producto[p].vector_marcas[j] = 0.

Un ejemplo de una transacción de la BTC es:

transacción_conflictiva = [<0,0,0>, <0,202,0,0>, <301,0,303,304>,<401>] donde: P_1 = <0, 0, 0>, P_2 = <0, 202, 0, 0>, P_3 = <301, 0, 303, 304> y P_4 = <401>.

En el ejemplo anterior puede apreciarse que el número de ítems de la transacción es 12, que el número de productos en los cuales se clasifican los 12 items es 4 (*n*=4), y que las cardinalidades para los P_i (es decir la cantidad de marcas diferentes que hay para el producto *i*) son: $|P_1|$=3, $|P_2|$=4, $|P_3|$=4, $|P_4|$=1. Además, la forma en que se clasifican los items por productos es: P_1={a_1, a_2, a_3}, P_2={a_4, a_5, a_6, a_7}, P_3={a_8, a_9, a_{10}, a_{11}}, P_4={a_{12}}, con lo cual se aprecia que la intersección de todos los P_i es vacía y que la suma de las cardinalidades de los P_i es igual a 12. .

Un ejemplo de representación de las transacciones simples y conflictivas son las tablas 1 y 2.

Donde:

101 = queso Aljuada, 102 = queso Alpina, 103 = queso Nestlé, 201 = pan La Gitana, 202 = pan Bimbo, 203 = pan Canelia, 301 = arroz Blanquita, 302= arroz Roa , 303 = arroz Nácar y 0 = No se vendió ninguna marca del producto.

En el ejemplo anterior podemos observar que el primer dígito identifica el producto y los dos últimos dígitos identifican la marca.

3.2 Definición de ítem y de itemsets para el preprocesamiento

En nuestra propuesta de preprocesamiento un ítem puede verse como una estructura de un componente:

```
estructura ítem {
   entero    atributo_instancia;
  }
```

Un *k-itemsets* es un ítemset que tiene *k* ítems, una variable que indica si este es conflictivo o no y una variable para su soporte:

k-ítemsets = {ítem_1, ítem_2, ..., ítem_k, conflictividad, soporte}.

Un *k-ítemsets candidato simple* es un conjunto de ítems generado de la misma forma como se generan los *k*-itemsets candidatos en el algoritmo Apriori tradicional, es decir, a partir de dos ítemsets soportados de tamaño *k*-1, donde todos los ítemsets hasta la posición *k*-2 son iguales, y además se cumple la propiedad de tener todos los subconjuntos de tamaño *k*-1 soportados. La característica principal de un ítemsets candidato simple, es que todos sus ítems corresponden a diferentes productos.

Un *k-itemsets candidato conflictivo* es un conjunto de ítems generado de la misma forma que el *k-itemsets candidato simple*, con la diferencia de que dos o más de sus ítems son del mismo producto.

De lo anterior se puede deducir que:

1. Para contar el soporte de los *itemsets candidatos simples*, se tiene que examinar tanto la BTS como la BTC, puesto que un *itemsets candidato simple* puede encontrarse en cualquiera de las dos bases de transacciones, además, un itemsets candidato simple es un caso especial de una transacción de la BTC donde la cantidad de marcas vendidas para cada producto es como mínimo uno.

2. Para contar el soporte de los *itemsets candidatos conflictivos*, únicamente necesitamos examinar la BTC, puesto que nunca se encontrará un *itemsets candidato conflictivo* en la BTS.

Por lo anterior, tenemos que los *ítemsets soportados simples* son todos aquellos *ítemsets candidatos simples* que superan el soporte mínimo. De la misma forma, todos los *itemsets soportados conflictivos* son todos aquellos ítemsets candidatos conflictivos que superan el soporte mínimo.

3.3 Proceso de generación de los k-itemsets candidatos

Presentamos unas consideraciones relativas al proceso de generación de los *k-ítemsets* candidatos:

1. De dos *k-itemsets soportados conflictivos* se puede obtener unicamente un *(k+1)-itemsets candidato conflictivo*, la característica de conflictividad se hereda de los *ítemsets soportados padres* a los itemsets candidatos hijos.

2. De un *k-itemsets soportado simple* y otro *conflictivo*, solo se puede obtener un *(k+1)-itemsets candidato conflictivo*, puesto que la característica de conflictividad se hereda cuando al menos uno de los *ítemsets* padres es conflictivo.

3. De dos *k-itemsets soportados simples* se puede obtener:

- Un *(k+1)-itemsets candidato simple*, siempre y cuando el producto del ítem en la posición *k* de ambos *itemsets soportados simples* sea diferente.

- Un *(k+1)-itemset candidato conflictivo*, siempre y cuando el producto del ítem en la posición *k* en ambos *itemsets soportados simples* sea el mismo.

Tomando en cuenta las consideraciones anteriores se mezclan todos los *itemsets soportados simples y conflictivos* utilizando el principio lexicográfico para generar correctamente todos los candidatos posibles. Cuando se genera un nuevo *itemsets candidato* de inmediato se clasifica como *simple o conflictivo*, para optimizar el proceso de contar soportes en las dos bases de transacciones (BTS, BTC).

3.4 Poda por soporte en tiempo de conteo

Se emplea una poda propuesta por Mannila et al [MTV94]. Cuando se esta recorriendo la base de datos de transacciones para contar el soporte de los itemset candidatos de tamaño *n*, se eliminan los itemset candidatos si la suma del soporte alcanzado en ese momento y del total de transacciones que faltan por recorrer en la base de datos (suponiendo que el itemset candidato está presente en todos ellos) no supera el soporte mínimo.

3.5 Algoritmo AprioriDPBT(Diferente Preprocesamiento en la Base de Transacciones)

A continuación se presenta el algoritmo para cálcular ítemsets frecuentes basado en el preprocesamiento que proponemos para bases de transacciones de ventas de supermercados :

El significado para algunas siglas que aparecen en el algoritmo es el siguiente:
- SS_k = Soportados Simples de tamaño *k*.
- SC_k = Soportados Conflictivos de tamaño *k*.
- BTS = Base de Transacciones Simples.
- BTC = Base de Transacciones Conflictivas.
- CT_k = Candidatos Totales tamaño *k*.
- CS_k = Candidatos Simples tamaño *k*.
- CC_k = Candidatos Conflictivos tamaño *k*.
- Z = Es un itemset.
- S_z = Soporte del itemsets.
- t_s = Transacción Simple.
- t_c = Transacción Conflictiva.
- C_t = Conjunto de itemset Candidatos que estan en la transacción.
- minSop = Mínimo Soporte.

```
AprioriDPBT(BTS, BTC, minSop)
1)  SS1={(Z,Sz):Z∈Soportados, |Z|=1, Sz=sop(Z, BTS, BTC)}
2)  SC1={}
3)  k←2
4)  while(SSk−1≠0 || SCk−1≠0)
5)   do CTk←generarTodosLosCandidatos(SSk−1 ∪SCk−1)
6)     CSk←escogerCandidatosSimples(CTk)
7)     CCk←escogerCandidatosConflictivos(CTk)
8)     for ts∈BTS
9)       do Ct <– {(Z,Sz)∈CSk : Z⊆ts}
10)       for (Z, Sz)∈Ct
11)        do Sz←Sz + 1
12)     end
13)     for tc∈BTC
14)       do if(porcentaje_adecuado)
15)         do CSk←podaPorSoporteEnTiempoDeConteo(CSk)
16)           CCk←podaPorSoporteEnTiempoDeConteo(CCk)
17)      Ct←{(Z,Sz)∈(CSk∪CCk):Z⊆tc}
18)      for (Z, Sz)∈Ct
19)        do Sz←Sz + 1
20)    end
21)    SSk←{(Z,Sz)∈CSk: Sz≥minSop}
22)    SCk←{(Z,Sz)∈CCk: Sz≥minSop}
23)    k←k + 1
24) end
25) answer ∪k (SSk∪SCk)
```

Se puede observar en el algoritmo que se cuenta el soporte de los *itemset candidatos* en la *base de transacciones simples* y luego este valor se utiliza para podar *itemset candidatos* en el momento de trabajar con la *base de transacciones conflictivas*, evitando trabajo innecesario.

3.6 Algoritmo AprioriDPBD (Diferente Preprocesamiento en la Base de Datos).

En la figura 1 se presenta el algoritmo para calcular ítemsets frecuentes en Bases de Datos Relacionales Discretizadas.

La función *generarTodosLosCandidatosSimples*, difiere de la función Apriori-Gen (sección 2.2), en el momento de generar el superconjunto del conjunto de todos los k-ítemsets frecuentes, verificando en forma adicional que los items, de los k-itemsets frecuentes, en la posición *k* pertenezcan a atributos diferentes.

4 RESULTADOS EXPERIMENTALES

Nosotros utilizamos la base de datos del censo de ingresos, años 94 y 95, disponibles en kdd.ics.uci.edu/databases/census-income. Esta base de datos contiene 99.762 registros y 41

Figura 1: AprioriDPBD

AprioriDPBD(BTS, minSop)

1) SS_1={(Z,S_z):Z∈Soportados, |Z|=1, S_z=sop(Z, BTS)}
2) k←2
3) while(SS_{k-1}≠0)
4) do CS_k←generarTodosLosCandidatosSimples(SS_{k-1})
5) for t_s∈BTS
6) do if(porcentaje_adecuado)
7) do CS_k←podaPorSoporteEnTiempoDeConteo(CS_k)
8) C_t <– {(Z,S_z)∈CS_k : Z⊆t_s}
9) for (Z, S_z)∈C_t
10) do S_z←S_z + 1
11) end
12) SS_k←{(Z,S_z)∈CS_k: S_z≥minSop}
13) k←k + 1
14) end
15) answer $\cup_k$ (SS_k)

Tabla 3: Resultados de pruebas sobre una fracción de la base de datos del censo.

	Apriori	ApriDPBD	Apriori	ApriDPBD	Apriori	ApriDPBD	Apriori	ApriDPBD	Apriori	ApriDPBD
Soporte	50%	50%	40%	40%	30%	30%	20%	20%	10%	10%
#Candidatos tam 2	253	253	701	741	1118	1176	1691	1770	2175	2278
#Candidatos tam 3	1353	1353	3770	3770	7705	7705	13181	13181	26061	26061
#LLamados poda sub.			4804	4932	9239	9814	16598	17461	29509	31538
#itemsets tamaño 3	1288	1288	3528	3528	7146	7146	12089	12089	23305	23305
Tiempo en segundos	272	301	462	529	692	846	999	1329	1796	2007

atributos -8 continuos y 33 nominales-. Para el preprocesamiento se discretizaron los atributos continuos. Para utilizar el algoritmo Apriori la base de datos fue binarizada, creando 582 items. Para utilizar el algoritmo aprioriDPBD la base de datos discretizada se preproceso con nuestra propuesta, conservando una estructura rígida de 41 posiciones.

Para el cálculo de ítemsets frecuentes se implemento el Apriori y el AprioriDPBD en lenguaje C, utilizando las mismas estructuras de datos; los ítemsets candidatos, ítemsets soportados y la base de datos se almacenaron en disco. Los programas se ejecutaron en un computador con procesador Pentium II, 64 MB en memoria principal y sistema operativo Linux Red Hat version 6.1. Se tomo un fracción de las 440 primeras tuplas de la base de datos y se realizaron pruebas con diferentes niveles de soporte, en un computador Pentium a 150 Mhz., 32 MB en RAM y sistema operativo Linux Mandrake 7.0.

En las tablas 3 y 4 se presentan algunos resultados.

En las pruebas realizadas sobre una fracción de la base de datos del censo podemos observar

Tabla 4: Resultados del cálculo de itemset frecuentes en la base de datos del censo.

	soporte	#candidatos tam 2	#candidatos tam 3	#llamados funcion subcon	#itemsets tam 3	tiempo seg
ApriDPBD	80%	136	401	457	330	12791
Apriori	80%	136	401	457	330	16267

que el número de llamadas a la función de poda dentro del Apriori-gen es menor en el algoritmo AprioriDPBD, lo cual se ve reflejado en el tiempo de ejecución.

Se obtuvieron mejores resultados respecto al tiempo de ejecución del algoritmo AprioriDPBD con respecto al algoritmo Apriori en el cálculo de ítemsets frecuentes en la base de datos del censo de ingreso.

En pruebas realizadas con tablas de transacciones sintéticas de ventas en supermercado, se apreciaron mejores resultados en tiempo de ejecución en Apriori frente al tiempo de ejecución del algoritmo AprioriDPBT.

5 CONCLUSIONES

El tiempo de ejecución para el cálculo de ítemsets frecuentes mediante el algoritmo AprioriDPBD es menor que el tiempo de ejecución para el algoritmo Apriori, debido fundamentalmente a: a) Reducción en la cantidad de candidatos de tamaño 2 generados por AprioriDPBD comparado con la cantidad de candidatos que genera Apriori. b) La reducción en el tamaño del superconjunto de posibles candidatos de tamaño 3 por parte de la función que genera candidatos en AprioriDPBD en comparación con función Apriori-Gen dentro del algoritmo de Apriori.

En la medida en que el número de valores por atributo se incrementa, se hace más eficiente el algoritmo AprioriDPBD frente al algoritmo Apriori, debido fundamentalmente a que Apriori genera más candidatos de tamaño 2 sin validez. Nos referimos a los candidatos que presentan dos instancias de valores distintos para un mismo atributo que el AprioriDPBD no genera.

No encontramos reducciones significativas en el espacio de almacenamiento requerido para las bases de transacciones, esta reducción sólo se da cuando se utiliza una estructura de datos rigida para almacenar las bases de transacciones binarizadas.

El preprocesamiento propuesto puede ser utilizado para el cálculo de reglas de asociación generalizadas.

AGRADECIMIENTOS

Agradecemos a la Dra. Marta Millán por su colaboración y apoyo. También agradecemos a lisp.vse.cz/pkdd99challenge/chall-data.phtml, kdd.ics.uci.edu/databases/census-income/ por hacer disponibles sus bases de datos.

Referencias

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. 1993. "Mining association rules between sets of items in large database". In Proc. of the ACM SIGMOD Conf. on

Management of Data, Pages 207-216.

[AMSTV96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. 1996. "Fast Discovery of Association Rules". In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, edits., Advances in Knowledge Discovery and Data Minig. AAAI/MIT Press, Chapter 12.

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. 1997. "Dynamic itemset counting and implication rules for market basket data". In Proceedings of the ACM SIGMOD Conference on Management of Data.

[HF95] Jiawei Han and Yongjian Fu. 1995. "Discovery of Multiple-Level from Large Databases". In Proc. of the 21st VLDB Conference. Zurich, Swizerland.

[HF99] Jiawei Han and Yongjian Fu. 1999. "Mining Multiple-Level Assocation Rules in Large Databases". In IEEE Transactions On Knowledge and Data Engineering, Vol 11, No5, September/Octuber

[KMO99] Walter A. Kosters,Elena Marchiori, and Ard A.J. Oerlemans .1999." Mining Clusters with Association Rules". In IDA 99, pag 41-50.

[MTV94] H. Mannila, H. Toivonen, and A.I. Verkamo. 1994. "Efficient Algoritnms for Discovering Association Rules". In KDD94: AAAI Workshop on Knowledge Discovery Databases. Pages 181-192.

[PCY95] J. S. Park, M. Chen, and P. S. Yu. 1995. "An effective hash based algorithm for mining association rules". In Proc. of the ACM SIGMOD Conf. on Management of Data. San Jose, California.

[R99] Arnauld Ragel.1999."Preprocessing of Missing Values Using Robust Association Rules". Conference European of KDD.

[RIC97] K. Rajamani, B. Iyer, and A. Chaddha. 1997. "Using DB/2's object relational extensions for mining associations rules". In Technical report TR 03,690., Santa Teresa Laboratory, IBM Corporation.

[SA95] R. Srikant, and R. Agrawal.1995. "Mining Generalized Assocation Rules". In Proc. of the 21st VLDB Conference. Zurich, Swizerland.

[SA96] R. Srikant, and R. Agrawal.1996. "Mining Quantitative Association Rules in Large Relational Tables". In Proc. of the ACM SIGMOD Conference on Management of Data. Montreal, Canada.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. 1995. "An efficient algoritmo for Mining Association Rules in Large Databases". In 21st proceedings of the Very Large Data Base Conference.

[STA98] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. 1998. "Integrating Association Rules Mining with Relational Database Systems: Alternatives and Implications". In Proc. of the ACM SIGMOD int'l Conference on Management of Data. Seattle, Washington.