Universidad de las Ciencias Informáticas

Facultad 3

TÍTULO: Implementación de algoritmo para la generación de reglas de asociación representativas.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAs

AUTOR(A): Yesenia Pulsant Limonta.

TUTOR: Lic. Reyder Cruz De La Osa

Ciudad de la Habana
2007

“No estudio por saber más, sino por ignorar menos”

Sor Juana Inés de la Cruz

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo a la Universidad de Ciencias Informáticas de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Autor Tutor

_____ _____

AGRADECIMIENTOS

Primeramente quiero agradecerle a mis padres, por su apoyo incondicional y su total confianza en mí, y a mi hermana, por estar siempre ahí para mi, son lo mas grande que tengo.

A toda mi familia, por depositar toda su confianza en mí.

A mi novio Leyser por estar ahí cada vez que necesitaba.

A mis amigos de siempre, Dinella, Janier, Yoxsnel, Marle, Marlon, Zady, Nestor, Danaysa, por compartir conmigo los buenos y malos momentos.

A mi amiga de las luchas, Yeimi Carmenate, por su preocupación constante.

A Pascual Verdecia, gracias por tus consejos y tus exigencias.

A Dagnalia y a Lisandra, por sus ayudas y consejos.

A Ramsés, por tantos apuros que me sacó.

A la Revolución y a la Universidad de las Ciencias Informáticas por permitirme formarme profesionalmente.

A todos aquellos que de una forma u otra han ayudado a mi formación profesional y personal y que no pongo por espacio.

…gracias, sin ustedes no hubiera sido posible…

…a mis padres…

…a mi hermana…

RESUMEN

El surgimiento de la inteligencia artificial fue un gran logro para la solución a problemas generales siendo la minería de datos una de las principales áreas a la cual esta encaminada, donde las bases de datos son los principales candidatos a la hora de extraer conocimientos. El descubrimiento de reglas de asociación es un importante problema en el área de la minería de datos, donde los algoritmos para el minado de los datos son las principales soluciones a este problema. Este trabajo va encaminado a la implementación de algoritmos para la generación de reglas de asociación proponiendo una herramienta para la extracción de las mismas en las bases de datos de la Universidad.

Palabras Clave

Inteligencia Artificial, Minería de Datos, Bases de Datos, Reglas de Asociación, Algoritmos para el minado de Datos.

TABLA DE CONTENIDOS

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA. 5

1.1 INTRODUCCIÓN AL CAPÍTULO 5
1.2 LA INTELIGENCIA ARTIFICIAL COMO UNA NUEVA DISCIPLINA CIENTÍFICA. 5
1.2.1 Definiciones sobre Inteligencia Artificial: 6
1.2.2 Características de la Inteligencia Artificial. 7
1.2.3 Objetivos de la Investigación en Inteligencia Artificial. 8
1.2.4 Aplicaciones de la inteligencia artificial 8
1.3 LA MINERÍA DE DATOS DENTRO DE LA INTELIGENCIA ARTIFICIAL. 9
1.3.1 Fases de un proyecto de minería de datos 10
1.3.2 Ejemplos de uso de la minería de datos 11
1.3.3 Problemas de la minería de datos. 11
1.4 REGLAS DE ASOCIACIÓN. 12
1.4.1 Reglas de Asociación en bases de datos. 13
1.5 ANÁLISIS DE LOS REQUERIMIENTOS DE PROBLEMA. 14
1.5.1 Los problemas y sus Dominios. 15
1.5.2 Definición del problema 15
1.5.3 Definición formal del proceso de extracción de RA en BD. 18
1.5.4 Análisis de los requerimientos en KDD 21
1.6 PRINCIPALES APROXIMACIONES PARA SOLUCIONAR EL PROBLEMA 22
1.6.1 El problema de la generación de conjuntos de ítems frecuentes 23
1.7 CONCLUSIONES DEL CAPÍTULO. 30

CAPÍTULO 2: GENERACIÓN DE LAS REGLAS DE ASOCIACIÓN A PARTIR DE LOS PATRONES FRECUENTES. 31

2.1 INTRODUCCIÓN AL CAPÍTULO 31
2.2 GENERALIZACIONES AL PROBLEMA DEL DESCUBRIMIENTO DE REGLAS DE ASOCIACIÓN 32
2.2.1 Reglas de Asociación Generalizadas 32
2.2.1.1 Algoritmo básico 35
2.2.1.2 Algoritmo Cumulate 36
2.2.1.3 Algoritmo EstMerge 37
2.2.1.4 Prutax 40
2.2.2 Reglas de asociación de mínima condición y máxima consecuencia 41
2.2.3 Reglas de asociación representativas (RAR) 42
2.3 ALGUNOS OTROS TRABAJOS RELACIONADOS 44
2.4 LIMITACIONES ACTUALES 44
2.4.1 Limitaciones de las soluciones generales 44
2.4.2 Limitaciones de los algoritmos específicos 46
2.4.3 Limitaciones de las generalizaciones estudiadas 47
2.5 CONCLUSIONES DEL CAPÍTULO. 48

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DE LA HERRAMIENTA. 50

3.1 INTRODUCCIÓN 50
3.2 ¿POR QUÉ ESCOGER A_PRIORI PARA LA GENERACIÓN DE ÍTEMS FRECUENTES? 50
3.2.1 Parámetros: soporte y confianza. 51
3.3 ¿POR QUÉ LAS REGLAS DE ASOCIACIÓN REPRESENTATIVAS? 51
3.4 METODOLOGÍA Y HERRAMIENTAS EMPLEADAS EN LA IMPLEMENTACIÓN. 52
3.4.1 Modelo de diseño. 53
3.4.1.1 Módulo Aprioriapp. 54
3.4.1.2 Módulo Accdatos. 55
3.4.1.3 Módulo Combinaciones. 56
3.4.1.4 Módulo GenerarReglas. 57

3.4.1.5 Módulo Interfaces. *58*
3.4.2 Implementación. *59*
3.4.3 Resultados obtenidos con el algoritmo. *64*
3.4.3.1 Validación de los resultados. *67*
3.5 CONCLUSIONES DEL CAPÍTULO. 71
CONCLUSIONES **73**
RECOMENDACIONES **74**
REFERENCIAS BIBLIOGRÁFICAS **75**
ANEXOS **77**
ANEXO 1: ALGORITMOS PARA LA GENERACIÓN DE CONJUNTOS DE ÍTEMS FRECUENTES 77
ANEXOS 2: ALGORITMOS DE GENERACIÓN PARA RA EN BD. 81
ANEXO 3: ALGORITMOS DE RA EN COLECCIONES DE TEXTOS 85
GLOSARIO DE TÉRMINOS **88**

INTRODUCCIÓN

La lingüística computacional es la ciencia que trata de la aplicación de los métodos computacionales en el estudio del lenguaje natural. Esta ciencia es una combinación de dos ciencias más grandes; la lingüística, que estudia las leyes del lenguaje humano, y la inteligencia artificial (IA), que investiga los métodos computacionales para el manejo de sistemas complejos.

La IA comenzó como el resultado de la investigación en psicología cognitiva y lógica matemática. Se ha enfocado sobre la explicación del trabajo mental y construcción de algoritmos de solución a problemas de propósito general. Punto de vista que favorece la abstracción y la generalidad.

La era actual de la informática no limita a las bibliotecas electrónicas del pequeño círculo de investigación donde se busca el desarrollo. Se han abierto las puertas a una buena parte del conocimiento de la humanidad con acceso prácticamente ilimitado a través de las páginas Web de Internet y otros medios electrónicos. La facilidad de capturar y almacenar información en medios magnéticos, el crecimiento acelerado de la información almacenada y la necesidad de descubrimiento de nuevos conocimientos (deducidos o inducidos) constituyen los incentivos fundamentales en el desarrollo de la minería de datos. Aunque los formatos de almacenamiento de información son disímiles, es vital hurgar dentro de ellos y encontrar el conocimiento necesario.

La minería de datos es un mecanismo de explotación, consistente en la búsqueda de información valiosa en grandes volúmenes de datos. Está muy ligada a las bodegas de datos que proporcionan la información histórica con la cual los algoritmos de minería de datos tienen la información necesaria para la toma de decisiones. (Tirso, 2002)

La minería de datos puede ser dividida en:

- Minería de datos predictiva (MDP): usa primordialmente técnicas estadísticas.
- Minería de datos para descubrimiento de conocimiento (MDDC): usa principalmente técnicas de inteligencia artificial.

Las características de la información contenida en bases de datos hacen de ésta un primer candidato cuando se desea obtener nuevos conocimientos. Las aplicaciones para el descubrimiento de conocimiento en base de datos es la herramienta recomendada, donde uno de los problemas más importantes es el descubrimiento de reglas de asociación.

En la Universidad de Ciencias Informáticas (UCI), donde por sus características el volumen de información crece exponencialmente, seria de mucha utilidad una herramienta que permita establecer asociaciones entre los elementos de las Bases de datos existentes en la universidad, siendo esta la ***situación problémica*** existente.

Después de haber analizado esta problemática, se definió como ***problema científico,*** la ausencia de implementaciones de algoritmos que permitan la generación de reglas de asociación en la Universidad.

Siendo el ***objetivo de la investigación*** desarrollar una biblioteca de clases en C++ donde se implementen algoritmos de reglas de asociación representativas.

Por consiguiente, el ***objeto de investigación*** para dar solución a este problema estará enfocado al campo de la minería de datos como área identificada en el descubrimiento de las Reglas de Asociación.

Tomando como ***campo de acción*** los algoritmos de generación de Reglas de Asociación en bases de datos.

Siguiendo la ***hipótesis*** de que: Si se implementa un algoritmo para generar reglas de asociación representativas, se podrá obtener una herramienta que extraiga dichas reglas en distintas BD de la Universidad.

Para obtener resultados satisfactorios en esta investigación, a continuación se proponen una serie de ***tareas investigativas*** como son:

1. Búsqueda de trabajos realizados sobre el tema como material de apoyo.
2. Entrevistar a especialistas en el tema.
3. Analizar algoritmos propuestos por los investigadores en el tema.
4. Implementar un algoritmo para obtener los elementos frecuentes en una base de datos.
5. Implementar un segundo algoritmo que extraiga reglas de asociación representativas de los elementos obtenidos.
6. Evaluar las respuestas del algoritmo.

Como ***resultado de la investigación*** se obtendrá una herramienta que permita obtener de las BD asociaciones entre los elementos frecuentes que en ella se encuentren.

El trabajo está compuesto por tres capítulos que incluyen todo lo relacionado con el trabajo de investigación y la propuesta de un algoritmo capaz de solucionar el problema tratado.

Capítulo 1: En este capítulo se verá todo lo relacionado con el surgimiento de la Inteligencia Artificial, dándole paso a la minería de datos como un campo importante dentro de la misma, y así a las reglas de asociación en bases de datos, también se da una breve introducción al problema que dio paso al la investigación y desarrollo de las reglas de asociación en bases de datos. También se explican varios trabajos (algoritmos) desarrollados en la búsqueda de ítems (elementos) frecuentes las bases de datos.

Capítulo 2: En este capítulo se comienza a profundizar en el tema del minado de reglas de asociación en bases de datos, tratando los principales trabajos (algoritmos) desarrollados para resolver el problema a partir del definido en 1993 por R. Agrawal. También se analizan las distintas limitaciones que tienen dichos trabajos en cuanto al procesamiento o resultado de los mismos.

Capítulo 3: En este capítulo se hace una pequeña descripción de los algoritmos escogidos y la implementación de la biblioteca de clases. También se hace una evaluación de los resultados.

CAPÍTULO 1: Fundamentación Teórica.

1.1 Introducción al capítulo

La necesidad de almacenar información ha motivado históricamente el desarrollo de sistemas más eficientes, con mayor capacidad y más baratos a la hora del almacenamiento. Para esto, existen dos tipos de Bases de datos (BD):

- Bases de datos relacionales.
- DBMS (Data Base Management Systems) y repositorios de información:
  - – Bases de datos orientadas a objetos y objeto-relacionales.
  - – Bases de datos espaciales (geográficas).
  - – Bases de datos de texto y multimedia.
  - – World Wide Web (WWW).

En este capítulo se hace referencia un poco que a la historia del surgimiento de la Inteligencia Artificial como una nueva disciplina, donde el campo específico a tratar será la minería de datos y dentro del mismo, las reglas de asociación como solución a algunos problemas.

También, se hace referencia al estudio de distintas investigaciones que se han llevado a cabo desde los inicios de la minería de datos hasta la actualidad, la evolución de las distintas reglas para asociar elementos tanto en distintas bases de datos, o simplemente en una sola. También, se da a conocer el problema que dio impulso a los distintos científicos a tratar de darle solución buscando asociaciones entre los disímiles elementos que conforman una base de datos.

1.2 La Inteligencia Artificial como una nueva disciplina científica.

Como se había dicho anteriormente, la IA comenzó como el resultado de la investigación en psicología cognitiva y lógica matemática. Se ha enfocado sobre la explicación del trabajo mental y construcción de algoritmos de solución a problemas de propósito general, es una combinación de la ciencia del computador, fisiología y filosofía, tan general y amplio como eso, es que reúne varios campos, todos los cuales tienen en común la creación de máquinas que pueden "pensar".

Algunos ejemplos se encuentran en el área de control de sistemas, planificación automática, la habilidad de responder a diagnósticos y a consultas de los consumidores, reconocimiento de escritura, reconocimiento del habla y reconocimiento de patrones. De este modo, se ha convertido en una disciplina científica, enfocada en proveer soluciones a problemas de la vida diaria. Los sistemas de IA actualmente son parte de la rutina en campos como economía, medicina, ingeniería y la milicia, y se ha usado en gran variedad de aplicaciones de software, juegos de estrategia como ajedrez en la computadora y otros videojuegos.

La idea de construir una máquina que pueda ejecutar tareas percibidas como requerimientos de inteligencia humana es un atractivo. Las tareas que han sido estudiadas desde este punto de vista incluyen juegos, traducción de idiomas, comprensión de idiomas, diagnostico de fallas, robótica, suministro de asesoría experta en diversos temas.

Es así como los sistemas de administración de base de datos cada vez más sofisticados, la estructura de datos y el desarrollo de algoritmos de inserción, borrado y locación de datos, así como el intento de crear máquinas capaces de realizar tareas que son pensadas como típicas del ámbito de la inteligencia humana, acuñaron el término Inteligencia Artificial en 1956. (Agrawal, 1993)

1.2.1 Definiciones sobre Inteligencia Artificial:

Las definiciones de Inteligencia Artificial son muchas, pero podría decirse que son programas que realizan tareas que si fueran hechas por humanos se considerarían inteligentes.

- Disciplina científico-técnica que trata de crear sistemas artificiales capaces de comportamientos que, de ser realizados por seres humanos, se diría que requieren inteligencia.
- Estudio de los mecanismos de la inteligencia y las tecnologías que lo sustentan.

- Intento de reproducir (modelar) la manera en que las personas identifican, estructuran y resuelven problemas difíciles.
- Son ciertas herramientas de programación, entendiendo por herramientas:
    - Lenguajes: LISP, PROLOG.
    - Entornos de desarrollo: shells.
    - Arquitecturas de alto nivel: nodo y arco, sistemas de producciones.

1.2.2 Características de la Inteligencia Artificial.

1. Una característica fundamental que distingue a los métodos de IA de los métodos numéricos, es el uso de símbolos no matemáticos, aunque no es suficiente para distinguirlo completamente. Otros tipos de programas como los compiladores y sistemas de bases de datos, también procesan símbolos y no se considera que usen técnicas de IA.
2. El comportamiento de los programas no es descrito explícitamente por el algoritmo. La secuencia de pasos seguidos por el programa es influenciado por el problema particular presente. El programa especifica cómo encontrar la secuencia de pasos necesarios para resolver un problema dado.

   Las conclusiones de un programa declarativo no son fijas y son determinadas parcialmente por las conclusiones intermedias alcanzadas durante las consideraciones al problema específico. Los lenguajes orientados al objeto comparten esta propiedad y se han caracterizado por su afinidad con la IA.
3. El razonamiento basado en el conocimiento, implica que estos programas incorporan factores y relaciones del mundo real y del ámbito del conocimiento en que ellos operan. Los programas de IA pueden distinguir entre el programa de razonamiento o motor de inferencia y base de conocimientos dándole la capacidad de explicar discrepancias entre ellas.
4. Aplicabilidad a datos y problemas mal estructurados, sin las técnicas de IA los programas no pueden trabajar con este tipo de problemas. Un ejemplo es la

resolución de conflictos en tareas orientadas a metas como en planificación, o el diagnóstico de tareas en un sistema del mundo real: con poca información, con una solución cercana y no necesariamente exacta.

1.2.3 Objetivos de la Investigación en Inteligencia Artificial.

Uno de los principales objetivos de los investigadores en inteligencia artificial es la reproducción automática del razonamiento humano.

Los investigadores en inteligencia artificial se concentran principalmente en los sistemas expertos, la resolución de problemas, el control automático, las bases de datos inteligentes y la ingeniería del software.

Otros investigadores están trabajando en el reto del reconocimiento de patrones donde se espera un rápido progreso en este campo que abarca la comprensión y la síntesis del habla, el proceso de imágenes y la visión artificial.

Finalmente, la fundamental investigación sobre la representación del conocimiento, la conceptualización cognoscitiva y la comprensión del lenguaje natural.

1.2.4 Aplicaciones de la inteligencia artificial

- ✓ Lingüística Computacional.
- ✓ Minería de Datos (Data Mining).
- ✓ Mundos Virtuales.
- ✓ Procesamiento de Lenguaje Natural (Natural Language Processing)
- ✓ Robótica.
- ✓ Sistemas de apoyo de la decisión.
- ✓ Videojuegos.
- ✓ Prototipos Informáticos.

1.3 La minería de datos dentro de la Inteligencia Artificial.

Bajo el nombre de minería de datos se engloban un conjunto de técnicas encaminadas a la extracción de "conocimiento" procesable implícito en las bases de datos. Las bases de la minería de datos se encuentran en la IA y en el análisis estadístico. Mediante los modelos extraídos utilizando técnicas de minería de datos se aborda la solución a problemas de predicción, clasificación y segmentación. Un proceso típico de minería de datos parte de la selección del conjunto de datos, tanto en lo que se refiere a las variables dependientes, como a las variables objetivo, como posiblemente al muestreo de los registros disponibles. Como consecuencia de este análisis, al conjunto de datos de entrada se le aplican una serie de transformaciones con el objetivo de prepararlo para aplicar la técnica de minería de datos que mejor se adapte a los datos y al problema. Finalmente se selecciona la técnica de minería, se construye el modelo predictivo, de clasificación o segmentación, y se evalúan los resultados contrastando con un conjunto de datos previamente reservado para validar la generalidad del modelo.

Tradicionalmente, las técnicas de minería de datos se aplicaban sobre información contenida en almacenes de datos. De hecho, muchas grandes empresas e instituciones han creado y alimentan bases de datos especialmente diseñadas para proyectos de minería de datos en las que centralizan información potencialmente útil de todas sus áreas de negocio, etc. No obstante, actualmente está cobrando una importancia cada vez mayor la minería de datos desestructurados como información contenida en ficheros de texto, en Internet, etc. (Boulicaut, Bykowski, 2000)

Por lo tanto, la minería de datos es fundamental en la investigación científica y técnica, como herramienta de análisis y descubrimiento de conocimiento a partir de datos de observación o de resultados de experimentos.

1.3.1 Fases de un proyecto de minería de datos

Los pasos a seguir para la realización de un proyecto de minería de datos son siempre los mismos, independientemente de la técnica específica de extracción de conocimiento usada.

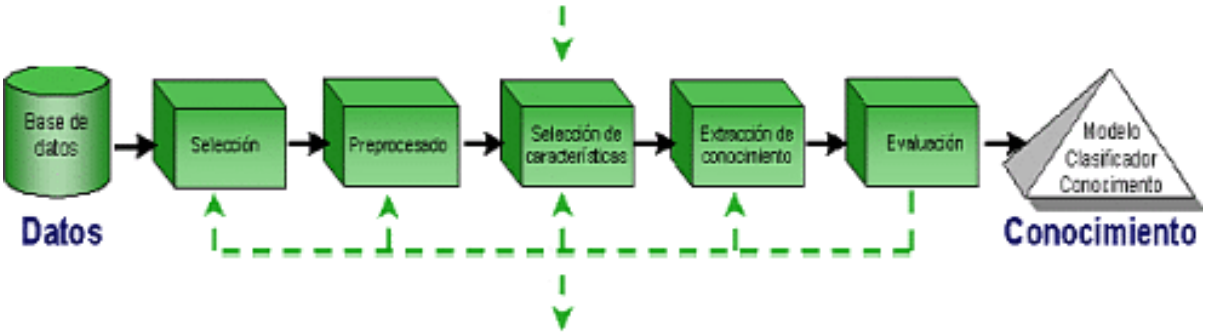


Fig.1.1 Fases de la minería de datos.

El proceso de minería de datos pasa por las siguientes fases:

- ✓ Filtrado de datos: El formato de los datos contenidos en la fuente de datos nunca es el idóneo, y la mayoría de las veces no es posible ni siquiera utilizar ningún algoritmo de minería sobre los datos "en bruto". Mediante el pre procesado, se filtran los datos, se obtienen muestras de los mismos o se reducen el número de valores posibles.
- ✓ Selección de variables: Aún después de haber sido pre procesados, en la mayoría de los casos se tiene una cantidad ingente de datos. La selección de características reduce el tamaño de los datos eligiendo las variables más influyentes en el problema, sin apenas sacrificar la calidad del modelo de conocimiento obtenido del proceso de minería.
- ✓ Algoritmos: Mediante una técnica de minería de datos, se obtiene un modelo de conocimiento, que representa patrones de comportamiento observados en los valores de las variables del problema o relaciones de asociación entre dichas variables. También pueden usarse varias técnicas a la vez para generar distintos modelos, aunque generalmente cada técnica obliga a un pre procesado diferente de los datos.

- ✓ Interpretación y evaluación: Una vez obtenido el modelo, se debe proceder a su validación, comprobando que las conclusiones que arroja son válidas y suficientemente satisfactorias. En el caso de haber obtenido varios modelos mediante el uso de distintas técnicas, se deben comparar los modelos en busca de aquel que se ajuste mejor al problema. Si ninguno de los modelos alcanza los resultados esperados, debe alterarse alguno de los pasos anteriores para generar nuevos modelos.
  - ✓ Casos de Estudio: Ejemplos de aplicación de la minería de datos. (Agrawal, Mannila, Srikant, y otros, 1996)

1.3.2 Ejemplos de uso de la minería de datos

Existen innumerables ejemplos de aplicaciones de la minería de datos en distintos sectores de investigación, algunos de ellos son:

- El ejemplo clásico de aplicación de la minería de datos tiene que ver con la detección de hábitos de compra en supermercados.
- Un ejemplo más habitual es el de la detección de patrones de fuga. En muchas industrias existe un comprensible interés en detectar cuanto antes aquellos clientes que puedan estar pensando en rescindir sus contratos para, posiblemente, pasarse a la competencia.
- Un caso análogo es el de la detección de transacciones de blanqueo de dinero o de fraude en el uso de tarjetas de crédito o de servicios de telefonía móvil e, incluso, en la relación de los contribuyentes con el fisco.
- Otro ejemplo es el análisis del comportamiento de los visitantes en una página de Internet, o la utilización de la información sobre ellos para ofrecerles propaganda adaptada específicamente a su perfil.

1.3.3 Problemas de la minería de datos.

- Reglas de asociación.
- Jerarquías de clasificación.
- Patrones secuenciales.

- Patrones en serie de tiempo, categorización y segmentación.
- Descubrimiento de patrones secuenciales.
- Descubrimiento de patrones de series de tiempo.
- Descubrimiento de reglas de clasificación, regresión.
- Redes neuronales.
- Algoritmos genéticos, agrupación y segmentación.
- Aplicaciones y herramientas comerciales de la minería de datos.

Como se había mencionado anteriormente, uno de los principales problemas dentro de la minería de datos es el descubrimiento de reglas de asociación.

1.4 Reglas de Asociación.

El descubrimiento de reglas de asociación (RA) es un importante problema en el área de la minería de datos en general. Las RA son aquellas reglas formadas por un lado izquierdo al que se le llamará implicativo y un lado derecho llamado implicado, cada lado de la regla esta formado por un conjunto de atributos.

Analizando profundamente el concepto de RA y su significado, se tiene que, como se dijo anteriormente, una RA es una relación de la forma $A \Rightarrow C$ (s, c) donde s representa el soporte y c la confidencia. Un valor elevado de confidencia indica que hay una gran probabilidad de que la aparición de A implique también la aparición de C y además si el valor del soporte es grande entonces esto quiere decir que A y C son elementos muy abundantes y por tanto influyentes a la hora de caracterizar los elementos que forman la BD que se analiza. Las RA se basan en medidas de confianza y soporte, consideran cualquier conjunto de atributos con cualquier otro conjunto de atributos; funcionan con atributos discretos.

La elegancia de expresar de forma clara y compacta los conocimientos descubiertos ha hecho de las RA una herramienta poderosa. Tanto es así, que en los últimos años, con el auge de la minería de textos, se ha intentado aplicar conceptos similares para expresar patrones descubiertos en colecciones de documentos, pero este trabajo se

centra en la extracción de las reglas de asociación representativas solamente en base de datos.

1.4.1 Reglas de Asociación en bases de datos.

Las características de la información contenida en BD hacen de ésta un primer candidato cuando se desea obtener nuevos conocimientos. Por ejemplo, una BD relacional que almacene la información sobre las compras de los clientes de una determinada cadena de tiendas. Para el gerente de dicha tienda, resulta interesante analizar información como: tipos de productos más comprados, marcas y precios más buscados, relacionados éstos con el tipo de cliente que realiza las compras; posibles combinaciones y relaciones (en precio, marca, tipo, entre otros.) entre dichos productos; detección de los cambios en las tendencias de sus clientes y sus empresas adversarias; realizar predicciones en el comportamiento del mercado ante cambios o lanzamientos de algunos productos. Analizar todos estos parámetros contando solamente con las facturas y algunos informes, es una tarea compleja e inaccesible para él; requiere por tanto, métodos automáticos para manipular el vasto volumen de información con el cuenta. Las aplicaciones para el descubrimiento de conocimiento en BD vienen a ser, en este caso, la herramienta recomendada, donde uno de los problemas más importantes es el descubrimiento de RA.

Por ejemplo, en el caso de una BD como la anterior se pudiera extraer la RA leche$\Rightarrow$pan (80%), que expresa que 4 de cada 5 consumidores si compran leche también compran pan. El 80% representa la probabilidad con la que se puede esperar que una persona compre pan, si ya ha comprado leche. Además de este valor de probabilidad, es necesaria otra magnitud que permita medir con qué frecuencia se compran ambos productos, es decir, en cuantas transacciones efectivamente se han comprado los productos relacionados, lo cual mide cuán verdaderamente importante es este descubrimiento. Reglas de este tipo son importantes en tareas de decisión, en el ejemplo anterior, podría ayudar a decidir precios de los productos, su ubicación y necesidades de promoción.

Si bien es interesante encontrar las RA en una BD, lo que es equivalente en bases de datos relacionales a encontrar relaciones entre los ítems de una relación que contiene referencias a otras tablas, más significativo es encontrar relaciones entre BD diferentes. En este caso el minero se ve obligado a encontrar los posibles nexos entre entidades de bases de datos disjuntas; la fecha y localización en muchos casos podría constituir el nexo de unión entre la información de dos BD. Por ejemplo, en una situación donde se tienen dos BD, una que contiene información sobre los consumidores de un establecimiento y otra que contiene información sobre obras de construcción vial, un minero experto podría encontrar una relación entre el poco consumo en el establecimiento y el arreglo de sus calles colindantes en un período determinado.

En casos como los descritos anteriormente es relativamente cómodo interpretar y encontrar reglas entre objetos, en el sentido que dichos objetos ya han sido analizados, por tanto, se cuenta con un conjunto de datos bien estructurados. El hecho de encontrar las RA entre la información almacenada en las BD, se traduce en encontrar los conjuntos de objetos mencionados con más frecuencia en las tablas de las BD, y verificar la posible formación de las RA con dichos conjuntos de datos.

Como se puede apreciar, las RA constituyen un mecanismo poderoso para expresar relaciones entre conjuntos de objetos de forma compacta y comprensible, pero además uniforme e independiente de la fuente de información, sobretodo si se le añade determinada semántica.

1.5 Análisis de los Requerimientos de Problema.

La definición formal del problema fue realizada en 1993 por R. Agrawal, con el objetivo de resolver el problema de encontrar relaciones entre las compras de los diferentes productos de un supermercado, considerando sólo la presencia de cada uno de ellos en las diferentes facturas. Este problema, aún después de 15 años es uno de los más investigados en el campo de la minería de datos. Múltiples soluciones han sido

desarrolladas, y la intención de mejorar la comprensión y el valor de uso de dichas reglas ha propiciado también importantes generalizaciones a este problema.

1.5.1 Los problemas y sus Dominios.

Si se posee una base de datos sobre un dominio la cualquiera pudiera ser aprovechada para resolver problemas como los enunciados a continuación, para ello, se extrae la información de la BD, esta información puede ser explícita (conocimiento almacenado), e implícita (conocimiento inferido del almacenado).

Para realizar esta recuperación o extracción se necesitan resolver ciertos problemas que son independientes de cualquier clase de problemas y sus dominios, son clases abstractas de problemas de gestión de la información.

Problemas	**Dominio**
Diagnóstico	Medicina, plomería, etc.
Interpretación de Imágenes	Medicina, geología, etc.
Surgimiento de Tendencias	Política, economía, etc.
Detección de nuevos Eventos	Cualquier dominio

Tabla1.1 Problemas y dominios

1.5.2 Definición del problema

U. Fayyad, en 1996, en la segunda Conferencia Internacional sobre el descubrimiento de patrones y el minado de datos, explica las definiciones relacionadas con la minería de datos, su relación con el descubrimiento de conocimiento en bases de datos (KDD, por sus siglas en inglés) y con otros campos relacionados, así como parte del fundamento teórico del proceso de KDD, y los algoritmos básicos para realizar la minería de datos; convirtiéndose de esta forma en el primer paso hacia la unificación de

la teoría del KDD.

Se define KDD como el proceso no trivial de identificar patrones válidos, nuevos, potencialmente útiles y comprensibles en los datos; los datos son un conjunto de hechos y los patrones son expresiones en algún lenguaje que describe un subconjunto de datos o un modelo aplicable a éstos, como son las relaciones, sucesos, tendencias y que pueden manipular no sólo regularidades sino además excepciones**.**

Las reglas para KDD son definidas por el interesado en usar el sistema. Así, se puede distinguir dos tipos de éstas: verificación, donde el sistema se limita a verificar las hipótesis del usuario, y de descubrimiento, donde el sistema de forma autónoma encuentra nuevos patrones; ésta última puede subdividirse en: descubrimiento de predicciones (para pronosticar posibles comportamientos futuros) y de descripciones (para descubrir patrones de manera comprensible para el usuario). Tiende a ser más importante la descripción que la predicción.

Por otro lado, los métodos de la minería de datos pueden dividirse en:

- Clasificación, que permite la identificación de una función que asocia un dato en una de varias clases predefinidas.
- Regresión, cuando se intenta el aprendizaje de una función que asocia un dato para conocer el valor real de las variables de predicción y descubre las relaciones funcionales entre las variables.
- Clustering, si se identifica un conjunto de categorías o clases para describir los datos.
- Resumen, cuando se encuentra una descripción compacta para un subconjunto de datos.
- Modelo de dependencias, que permite encontrar un modelo para describir las dependencias significativas entre las variables.
- Detección de cambios y desviaciones, que descubren la mayoría de los cambios significativos en los datos desde medidas previas o valores usuales.

Y están formados por tres componentes fundamentales:

- Modelo de representación, el lenguaje a emplear para describir los patrones descubiertos.
- Modelo del criterio de evaluación, una medida que cuantifica cuán bien un patrón particular se ajusta a las reglas del proceso KDD.
- Método de búsqueda, forma en que serán ajustados los parámetros y modelos de manera que optimicen el criterio de evaluación.

Uno de los modelos de representación más utilizados es las RA, o sea, el descubrimiento de relaciones entre conjuntos de objetos. Visto de este modo, es mucho más general la motivación original de la búsqueda de RA, es decir, la que devino de la necesidad de analizar el llamado conjunto de datos de las cestas de los supermercados (supermarket basket data), o sea, examinar el comportamiento de los consumidores de los productos de un supermercado. (Agrawal, 1993)

La simplicidad de las RA ha hecho que su extracción sea una de los más importantes mecanismos para el reconocimiento de patrones en bases de datos, pues puede expresar dependencias y resumir características de un subconjunto de datos.

Es necesario considerar que las RA extraídas en BD, para que realmente puedan ser manipulables deben ser lo más generales posibles, esto es, que su cantidad debe ser razonablemente pequeña, y además cubrir todas las RA que pudieran deducirse.

En la literatura no aparecen definidos los problemas de la búsqueda de RA mezclados como un único problema, sin embargo, presentan tantas similitudes que permiten definirlos como un único problema.

1.5.3 Definición formal del proceso de extracción de RA en BD.

Definición 1: Sea O = { O1, O2, ..., Om } un conjunto de literales, se llamará conjunto de objetivos o ítems presentes, el complemento del conjunto de objetivos O es el conjunto C(O) = { ¬O1, ¬O2, ..., ¬Om }, un conjunto de literales asociados a O, que denota la ausencia de los objetivos O1, O2, ..., Om respectivamente. Se llamará a $X \subseteq O$ conjunto de objetivos presentes; $X' \subseteq C(O)$, conjunto de objetivos ausentes y $X'' \subseteq O \cup C(O)$, conjunto de objetivos.

Definición 2: Sea $H \subseteq O$ un hecho o evento que se conoce ocurre. Se dirá que H contiene a $X \subseteq O$ si $X \subseteq H$ y también contiene a $X' \subseteq C(O \backslash X)$ si $X' \subseteq C(H)$ y se denota como: $X \cup X' \prec H$.

Es decir, un conjunto de objetivos $X'' = X \cup X'$ están presentes en un hecho si el conjunto de hechos presentes X pertenecen al hecho y el conjunto de objetivos ausentes es subconjunto del complemento del conjunto de objetivos presentes en el hecho. (Brin, Motwani, Ullman, Tsur, 1997)

Definición 3: Sea CH una lista de todos los hechos H que ocurren, almacenados éstos en alguna estructura como una base de datos, se dirá que $|CH|$ es la cantidad de elementos de la lista.

Definición 4: Sea $X \subseteq O \cup C(O)$, se dice que su soporte es la fracción de hechos en las que X está contenido, o sea, $soporte(X) = \frac{|\{H / H \in C_H, X \prec H\}|}{|C_H|}$. Si $soporte(X) \geq s$ entonces X es frecuente, y puede llamarse k-itemset frecuente cuando k = $|X|$.

Notar que cuanto mayor es el valor del soporte de un conjunto de objetivos, mayor es la cantidad de hechos en la lista CH que contienen a dicho conjunto de objetivos, por tanto es más importante en la muestra de hechos.

Definición 5: Una RA con soporte s y confianza c es una implicación de la forma $A \overset{S_A,S_C}{\Rightarrow} C(s,c)$ donde A, C $\subseteq$ O $\cup$ C(O), Oi$\in$A$\Rightarrow\neg$Oi $\notin$A, Oi$\in$C$\Rightarrow\neg$Oi $\notin$ C, siendo $s = soporte(A \cup C)$, $c = \frac{soporte(A \cup C)}{soporte(A)}$.

Dicha regla se interpreta como sigue: Si es aplicable al antecedente A de la regla, o parte izquierda, la semántica SA, entonces también le es aplicable al consecuente C de la regla, o parte derecha, la semántica Sc, con un soporte s y confianza c.

Hay que destacar que la confianza de una RA muestra la relación entre el soporte de todos los objetivos involucrados en la regla y el de los involucrados sólo en el antecedente de dicha regla. Mientras mayor es la confianza de una regla mayor es la dependencia del consecuente respecto a un antecedente. (Fayyad, 1995)

Definición 6: Sea VA*(SA, Sc, s, c) el conjunto de todas las RA con semántica en el antecedente SA, en el consecuente Sc, con soporte mayor o igual a s y confianza mayor o igual a c.

Sea

$$VA^*(S_A, S_c, s, c) = \{r : A \overset{S_A,S_C}{\Rightarrow} C(s,c) / \neg\exists r' : A' \overset{S_A,S_C}{\Rightarrow} C'(s,c) \in VA(S_A, S_c, s, c), r \neq r', A' \subseteq A, C \subseteq C'\}$$

o sea, el conjunto de RA con mínimo antecedente y máximo consecuente.

El problema del minado de las RA consiste en encontrar todas las RA presentes en VA*(SA, Sc, s, c). (Walczak, 1998)

Cuando se trabaja con BD, los objetivos son los objetos o ítems de la transacción sobre la que se realiza el minado de las RA, y los hechos, cada una de las transacciones que forman la relación.

La búsqueda de asociación entre objetos permite enriquecer las funcionalidades de los Sistemas de Manipulación de Datos para el procesamiento de demandas como:

- Encontrar todas las RA interesantes que tengan como antecedente el conjunto de objetos A.
- Encontrar todas las RA interesantes que tengan como consecuente el conjunto de objetos C.
- Encontrar todas las RA interesantes que tengan como antecedente el conjunto de objetos A y como consecuente el conjunto de objetos C.
- Encontrar todo el conjunto de RA interesantes.
- Encontrar las mejores k reglas que cumplan con alguna de las restricciones anteriores.

Esto implica, además, restricciones al menos en dos sentidos:

- El porcentaje de hechos (transacciones, en bases de datos) que contiene a todos los objetivos relacionados en la regla, el llamado soporte de la regla y la confianza de la regla.
- El porcentaje de hechos que garantizan que cuando aparezca el antecedente de la regla, también aparecerá su consecuente.

No debe confundirse entre estos dos términos, el primero se corresponde con la significación estadística de la regla, en tanto el segundo mide su fortaleza.

Pero además las reglas extraídas deben contener un número mínimo de antecedentes, máximo de consecuentes, y su semántica pueda ser extraída, del propio contexto del problema de aplicación.

Es importante hacer hincapié en que las RA no sólo son el método más empleado para el descubrimiento de relaciones entre conjuntos de objetos, sino también un mecanismo poderoso para la representación del conocimiento inicial de forma compacta y comprensible. La inferencia implicada entre un antecedente y un consecuente puede ser enriquecida más allá de una simple coexistencia de dos objetos, sino además puede expresar relaciones del tipo causa-efecto, reconocimiento de excepciones (si se considera además reglas del tipo Si A entonces $\neg$B), secuencias de sucesos, identificación de categorías o clases entre datos, etc.

1.5.4 Análisis de los requerimientos en descubrimiento de conocimiento en bases de datos

El hecho de RA en los sentidos que anteriormente se explicaron, obliga a tener como entrada al sistema el conjunto de n objetos a considerar y los llamados soporte y confianza con los que se espera se recuperen reglas suficientemente interesantes. La búsqueda de estas reglas no es en lo absoluto trivial, pues no puede reducirse a la "simple inspección" par a par de los 2n subconjuntos que pudieran formar las reglas de asociación, para verificar si cumplen con los valores de soporte y confianza predefinidos, por el coste computacional que ello supone. Además esta "simple inspección" requiere recorridos a través de la BD (hay que contar cuantas transacciones, contienen todos los elementos pertenecientes de la regla y cuantas los antecedentes).

Cuando se intenta buscar RA en BD relacionales, se está considerando que el cardinal del conjunto de hechos sobre el que se procesa es considerablemente grande. Así, un primer desafío para resolver el problema es precisamente la búsqueda de algoritmos cuyo tiempo de ejecución no dependa de este valor.

En muchos casos es necesario considerar cómo mezclar las informaciones provenientes de entornos diferentes, ejemplo, la relación entre el consumo en un establecimiento y el arreglo de calles donde éste se encuentra situado. Debe realizarse un pre procesamiento de manera que se mezcle adecuadamente toda la información

disponible, manteniéndose las propiedades que permitan caracterizar la pertenencia de los objetivos a los nuevos hechos generados a partir de la mezcla.

Es importante considerar que la búsqueda de un número reducido de RA significativamente importantes implica necesariamente un estudio y especificación de los requerimientos de las RA extraídas. En la mayoría de los casos, los usuarios del sistema KDD no tienen idea de las posibles asociaciones a encontrar. Si se pudiera acotar la cantidad de RA, con especificaciones superfluas y poco condicionadas a los problemas (como el número de elementos en la parte izquierda y derecha de la regla) pudieran también eliminarse relaciones con un alto valor de interés en la aplicación práctica. Entonces, hay que considerar elementos de otra índole para acotar la cantidad de RA extraídas. (Piatetsky, 1991)

Resumiendo, se podría decir que para resolver este problema se necesitan algoritmos que minimicen el recorrido por la BD o el conjunto de documentos y poden eficientemente el gran espacio de búsqueda de los conjuntos de ítems frecuentes, pues el número de reglas crece exponencialmente con el número de ítems considerados; que obtengan reglas verdaderamente interesantes en la investigación del usuario del sistema KDD y en un número manipulable por él, pudiéndole éste ayudar con ciertas especificaciones que el experto en el área de aplicación pudiera incluir; por último, los términos u objetos a considerar para la formación de las reglas deben ser consecuencia de un estudio de los que realmente puede ser interesantes, de manera que la cantidad de entidades relacionadas sea mínimo y su valor máximo. (Amir, 1997)

1.6 Principales aproximaciones para solucionar el problema

La tarea de minar reglas de asociación ha recibido una gran atención. Hoy en día el minado de tales reglas es el método más popular para el descubrimiento de conocimiento.

Pero como se mencionó anteriormente, el número de reglas crece exponencialmente con respecto al tamaño del conjunto de ítems tratados. Esta restricción obliga a dividir el problema en dos partes separadas:

1. Encontrar los conjuntos de ítems frecuentes, cuando el soporte de éstos es mayor que el valor del soporte mínimo especificado por el usuario.
2. Para cada conjunto de ítems frecuente encontrado, comprobar la confidencia de todas las reglas y eliminar aquellas que no alcancen el valor mínimo de confidencia.

Como dice Toivonen en su artículo sobre el descubrimiento de patrones frecuentes en grandes colecciones de datos, (Toivonen, 1996) “la parte laboriosa de este problema es la tarea de encontrar los conjuntos frecuentes”.

1.6.1 El problema de la generación de conjuntos de ítems frecuentes

La solución trivial de este problema (reconocer entre todos los subconjuntos de O los que satisfacen el soporte mínimo) es computacionalmente intratable, por ello es necesario atravesar el espacio de búsqueda de forma eficiente, guiados para ello de la propiedad de que todos los subconjuntos de un conjunto de ítems frecuente son también frecuentes y todos los súper conjuntos de los conjuntos de ítems no frecuentes son también infrecuentes.

Para la búsqueda de los conjuntos de ítems frecuentes se emplean las dos formas más comunes de búsqueda en árbol: primero a lo ancho (BFS, por sus siglas en inglés) y primero en profundidad (DFS, por sus siglas en inglés).

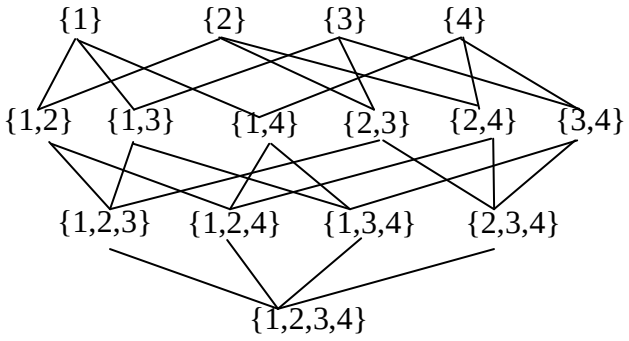


Fig.1.2. Árbol usado en los algoritmos con estrategia BFS

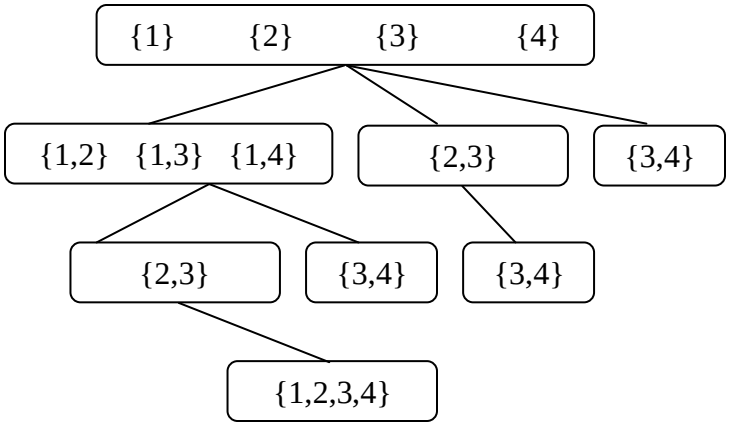


Fig.1.3. Árbol usado en los algoritmos con estrategia DFS

Todos estos algoritmos trabajan por lo general de la siguiente manera: buscan un conjunto C_k de k-itemsets con gran probabilidad de ser frecuentes, estos serán los candidatos, comenzando por k=1. Se comprueba cuáles son frecuentes y a partir de éstos se genera nuevamente un conjunto de candidatos de tamaño k+1, C_{k+1}. Todo este proceso se repite hasta que no pueda generarse un nuevo conjunto candidato, lo que ocurre en el caso de no haber encontrado en una iteración conjuntos frecuentes. Esta estrategia garantiza que sean visitados todos los itemsets frecuentes y al mismo tiempo reduce el número de itemsets infrecuentes visitados.

Con la estrategia BFS el valor del soporte de los (k-1) itemsets son determinados antes de contar el soporte de todos los k-itemsets, ello le permite utilizar la propiedad arriba enunciada. Con la estrategia DFS, no son conocidos todos los (k-1) itemsets, pero sí los necesarios (k-1) itemsets cuando se generan cada uno de los k-itemsets, pues trabaja recursivamente descendiendo por el árbol siguiendo la estructura del segundo de los árboles anteriormente descritos.

Para contar el soporte de todos los conjuntos de ítems se emplean también por lo general dos mecanismos:

1. Determinar el valor del soporte contando directamente sus ocurrencias en la BD.
2. Determinar el soporte empleando la intersección entre conjuntos.

Los algoritmos más comunes para el cálculo de los conjuntos de ítems frecuentes se muestran en la siguiente figura:

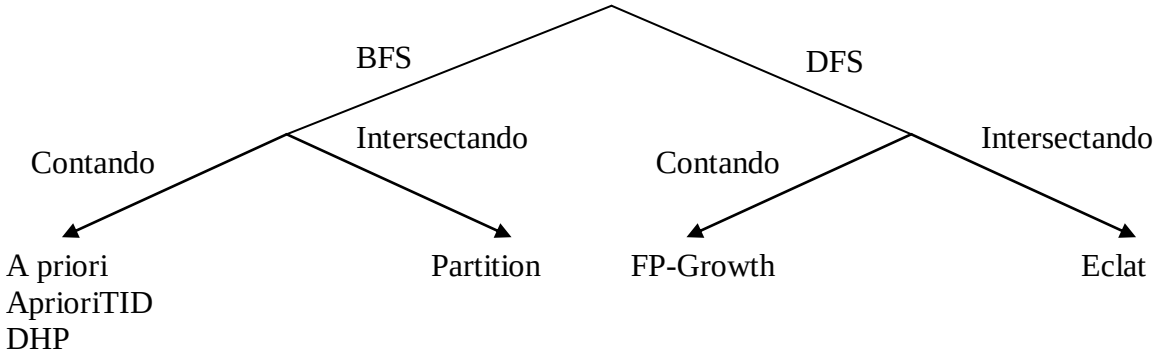


Fig.1.4. Clasificación de los algoritmos según las figuras anteriores.

Cuando se usa la estrategia DFS, los conjuntos candidatos constan solamente de conjuntos de ítems de uno de los nodos anteriormente visto. Atravesar la BD para cada nodo es extremadamente costoso y la simple combinación de estos mecanismos no tiene mucha relevancia práctica.

Antes de explicar los diferentes algoritmos, hay que hacer la siguiente salvedad: algunos algoritmos emplean el valor del soporte sin dividir por el total de transacciones, no se unificará este criterio para respetar la notación de cada algoritmo, aunque ello, por supuesto, no altera ningún resultado.

1.6.1.1 Algoritmo Apriori

El algoritmo A priori trabaja de la siguiente manera: primero busca todos los conjuntos frecuentes unitarios (contando sus ocurrencias directamente en la BD), se mezclan éstos para formar los conjuntos de ítems candidatos de dos elementos y seleccionan entre ellos los frecuentes. Considerando la propiedad de los conjuntos

de ítems frecuentes, vuelve a mezclar éstos últimos y selecciona los frecuentes, así sucesivamente se repite el proceso hasta que en una iteración no se obtengan conjuntos frecuentes. Este algoritmo asume un orden lexicográfico entre los ítems.

Apriori cuenta todos los candidatos de cardinalidad k juntos mediante una pasada a través de la BD. La parte crítica es buscar los candidatos de cada una de las transacciones. Para este propósito se genera una estructura de árbol, donde los ítems de cada transacción son usados para descender en el árbol. Cuando se llega a un nodo hoja se ha encontrado un conjunto de candidatos que tiene un prefijo común contenido en la transacción y en este caso, a cada uno de dichos candidatos se le incrementa el valor del contador de transacciones que lo contiene. Esta estrategia permite reducir la búsqueda de los candidatos de cada transacción a un recorrido del árbol de conjuntos candidatos de tamaño k, que es calculado en cada pasada. (Agrawal, 1994)

1.6.1.2 Algoritmo AprioriTID

Este algoritmo es una extensión del Apriori, pero en lugar de emplear los registros de la BD, AprioriTID representa internamente cada transacción por los actuales candidatos que este contiene, o sea, en cada pasada genera una tabla booleana que expresa la presencia de cada candidato en cada transacción. Si el total de tuplas con valor verdadero en una columna (hablando en términos de SQL) es mayor al soporte mínimo especificado por el usuario, entonces el conjunto es frecuente. Conocidos todos los conjuntos frecuentes se genera ahora una nueva tabla para asociar cada transacción con los conjuntos candidatos de dos ítems, formados por la combinación de los conjuntos frecuentes unitarios y se vuelven a reconocer los conjuntos frecuentes pasando a través de esta tabla. Este proceso se repite hasta que no aparezca ningún otro conjunto frecuente. (Agrawal, 1994)

1.6.1.3 DHP (Poda y hashing directa)

En el algoritmo de poda y hashing directa (DHP, Direct hashing and Pruning) se emplea una técnica de hash para eliminar todos los conjuntos de ítems innecesarios para la generación del próximo conjunto de ítems. Cada (k+1)-itemset es añadido a una tabla

hash en un valor hash dependiente de las ocurrencias en la BD de los conjuntos candidatos de k elementos que lo formaron, o sea, dependiente del soporte de los conjuntos candidatos de k elementos. Estas ocurrencias son contadas explorando en las transacciones de la BD. Si el soporte asociado a un valor hash es menor que el soporte mínimo entonces todos los conjuntos de ítems de k+1 elementos con este valor hash no serán incluidos entre los candidatos de k+1 elementos en la próxima pasada. (Park, 1997) (Holt, 1999)

1.6.1.4 Partition

La ventaja de mantener la BD en memoria y evitar las operaciones de entrada/salida (E/S) en disco, motivaron los trabajos de Savasare y Navathe**.** Como principal idea, propusieron particionar la BD en tantas partes como fueran necesarias para que todas las transacciones en cada partición cupieran en la memoria operativa. En contraste con los algoritmos vistos hasta el momento este algoritmo recorre la BD sólo dos veces. La primera vez, cada partición es minada independientemente para encontrar todos los conjuntos de ítems frecuentes en la partición y luego se mezclan éstos para generar el conjunto de los conjuntos de ítems candidatos. Muchos de éstos pueden ser falsos positivos, pero ninguno falso negativo. En la segunda pasada se cuenta la ocurrencia de cada candidato, aquellos cuyo soporte es mayor que el mínimo soporte especificado se retienen como conjuntos frecuentes. Hay que recordar que este algoritmo emplea el mecanismo de intersección entre conjuntos para determinar el soporte de dichos conjuntos, en este caso cada ítem en una partición mantiene la lista de los identificadores de las transacciones que contienen a dicho ítem.

Como se explicaba anteriormente, este algoritmo en la primera fase lee cada partición y calcula sus conjuntos frecuentes, mediante llamadas al algoritmo GetItemsets que toma una partición y el soporte mínimo requerido y devuelve todos los conjuntos frecuentes de la partición, mezcla estos conjuntos, formando el conjunto de candidatos C. En la segunda fase, se vuelve a leer cada partición, y se va acumulando para cada conjunto candidato el soporte en cada partición, mediante un llamado al procedimiento AcumularSoporte. (Savasare, Navathe, 1999)

1.6.1.5 Eclat

Los algoritmos del tipo Eclat fueron introducidos en 1997 (Zaki, 1997)**,** estos algoritmos, al igual que el Partition, reducen la cantidad de operaciones de E/S, aunque esta vez atravesando la BD sólo una vez. Se basa en realizar un agrupamiento (clustering) entre los ítems para aproximarse al conjunto de ítems frecuentes maximales y luego emplean algoritmos eficientes para generar los ítems frecuentes contenidos en cada grupo. Para el agrupamiento proponen dos métodos que son empleados después de descubrir los conjuntos frecuentes de dos elementos:

1. Por clases de equivalencia: esta técnica agrupa los itemsets que tienen el primer ítem igual.
2. Por la búsqueda de cliques maximales: se genera un grafo de equivalencia cuyos nodos son los ítems y los arcos conectan los ítems de los 2-itemsets frecuentes. Se agrupan los ítems por aquellos que forman cliques maximales.

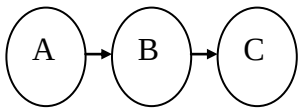
Los algoritmos del tipo Eclat son definidos por combinación de los métodos de agrupamiento y búsqueda de conjuntos frecuentes empleados, a saber:

- Eclat: Usa clases de equivalencia y buttom-up
- MaxEclat: Usa clases de equivalencia y esquema híbrido top-down/buttom-up
- Clique: Usa clique maximal y buttom-up
- MaxClique: Usa clique maximal y esquema híbrido top-down/buttom-up

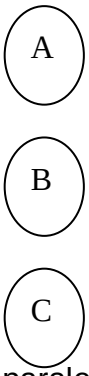
Los resultados experimentales muestran un mejor comportamiento de dichos algoritmos en el siguiente orden: MaxClique, Clique, MaxEclat, Eclat. Mejorando en un 40 por ciento el primero de éstos a Apriori y en un 20 a Partition.

1.6.1.6 Los episodios como patrones frecuentes

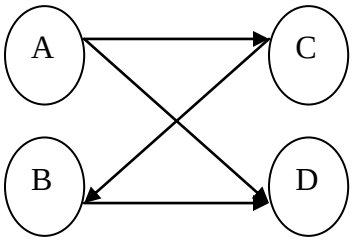
La mayor parte de los trabajos sobre aprendizaje automatizado y las técnicas de minería de datos están diseñadas para analizar colecciones desordenadas de datos, sin embargo existen importantes aplicaciones cuando los datos analizados tienen una estructura secuencial inherente, lo que se conoce como episodios de eventos. Por otro lado, también resulta importante conocer los cambios históricos del comportamiento de los datos, lo que se conoce como tendencia y también es considerado como un patrón reconocible. Los episodios en general, son conjuntos parcialmente ordenados de conjuntos de eventos. Un episodio serial se define por la ocurrencia de eventos en un determinado orden; en un episodio paralelo no existen restricciones en cuanto al orden de ocurrencia de dichos eventos. Estos tipos pueden ser descritos como un grafo dirigido acíclico de la siguiente forma:



Episodio serial, donde los eventos A, B y C ocurren en ese orden



Episodio paralelo, donde no existen restricciones en cuanto al orden entre los eventos A, B y C



Episodio que ocurre en secuencia cuando las ocurrencias de A, B (en cualquier orden) preceden las de C, D (también en cualquier orden)

Fig.1.5. Los episodios como patrones frecuentes.

El usuario de un sistema que identifica episodios, define cuán cerrado puede considerarse el ancho de tiempo (ventana) donde un episodio puede ocurrir y cuántas veces debe ocurrir para ser considerado frecuente. El análisis de una secuencia de eventos, el usuario está interesado en descubrir todos los episodios frecuentes de un tipo

determinado (serial, paralelo o con orden parcial). Conocidos los episodios, estos pueden usarse para generar RA para la predicción.

En (Mannila., 1995) se expone un algoritmo para descubrir todos los episodios frecuentes dada una secuencia de eventos, una clase de episodio y un umbral de frecuencia, y que trabaja con la misma filosofía de los algoritmos BFS: iterativamente alterna entre la búsqueda de todos los candidatos de k elementos y el reconocimiento entre éstos de los frecuentes, para volver a formar nuevos mezclando los frecuentes de máximo cardinal que hasta el momento se han encontrado.

1.7 Conclusiones del capítulo.

En este capítulo se trataron algunos aspectos necesarios para comprender el objeto de estudio que abarca la investigación, cumpliéndose así los objetivos trazados en este primer capítulo:

- ✓ Se trataron aspectos importantes para el tema el surgimiento de la inteligencia artificial como una nueva disciplina científica para la creación de maquinas que pudieran “pensar”.
- ✓ Se trató la minería de datos como una aplicación de la inteligencia artificial, aplicación a tratar en el transcurso del trabajo.
- ✓ Se dio una pequeña introducción al concepto de Reglas de Asociación y se realizó una descripción formal de la descripción de la extracción de las reglas en las bases de datos.
- ✓ Se describieron los trabajos (algoritmos) fundamentales para la extracción de elementos frecuentes en Bases de Datos como aproximaciones para mejorar el problema.

Aportando todos los epígrafes desarrollados un conocimiento previo sobre el tema y las principales definiciones para la continuidad del trabajo.

El algoritmo escogido en esta sección para la extracción de ítems frecuentes fue el Apriori, por ser uno de los más básicos y sencillo de implementar, además, los resultados que arroja son bastante claros y comprensibles para los usuarios, ya que la lógica que sigue es bastante comprensible.

CAPÍTULO 2: Generación de las Reglas de Asociación a partir de los patrones frecuentes.

2.1 Introducción al capítulo.

En este capítulo se describen los principales trabajos (algoritmos) de extracción de RA en BD, además de tratar las diferentes limitaciones que presentan dichos trabajos. En este capítulo también se define los algoritmos escogidos para su posterior implementación. La solución a este problema es trivial, y se reduce a verificar la confidencia de todas las posibles reglas que pueden generase a partir de cada conjunto frecuente X algorítmicamente se describe por:

Algoritmo Generación_Reglas_De_Asociacion

Entrada: F: Conjunto de itemsets frecuentes, cada uno con su soporte

MinConf: Mínima confidencia especificada por el usuario

Salida: R: Conjunto de reglas de asociación generadas

Método:

$R = \varnothing$

Para todo conjunto frecuente $X \in F$ hacer

Para todo $Y \subseteq X$ con $Y \neq \varnothing$ hacer

Si X.soporte /(X\Y).soporte $\geq$ MinConf entonces

$R = R \cup \{X \backslash Y \Rightarrow Y\}$

(Agrawal, 1993)

2.2 Generalizaciones al problema del descubrimiento de reglas de asociación

Algunas incursiones en este terreno fueron hechas con el objetivo de disminuir la cantidad de reglas descubiertas, desde dos puntos de vista:

1. Disminuyendo la cantidad de ítems relacionados en las reglas (ya sea en su parte izquierda o derecha), lo que dio lugar a las llamadas Reglas de Asociación Representativas (RAR) y las de mínima condición – máxima consecuencia (Krys, 1998).
2. Empleando una taxonomía de conceptos que relacionan los ítems de la BD, lo que permite generar reglas que contemplan conceptos a diferentes niveles de la taxonomía, lo que se conoce como Reglas de Asociación Generalizadas (RAG) (Srikant, 1995) (Hipp,2000).

2.2.1 Reglas de Asociación Generalizadas

(Srikant, 1995) introduce el problema del minado de Reglas Asociación Generalizadas (RAG), el que consiste en que dada una BD de transacciones (donde cada transacción es un conjunto de ítems) y una jerarquía de conceptos (una taxonomía) para los ítems, se encuentran RA entre los ítems a cualquier nivel de la taxonomía.

La mayoría de los trabajos sobre RA no consideran la presencia de taxonomía y restringen las RA a los ítems hojas de una taxonomía. Sin embargo su búsqueda puede ser importante porque los ítems hojas de una taxonomía pueden no tener soporte mayor que un valor mínimo especificado, por tanto las relaciones significativas se encuentran en niveles superiores de la jerarquía. Este tipo de RA, además puede servir para podar reglas redundantes o poco interesantes.

En lugar de emplear árboles para representar una taxonomía, se utilizan grafos dirigidos acíclicos, lo cual permite considerar múltiples jerarquías.

Sea $o \in O$, se dirá que $\hat{o}$ es ancestro de o (y que o es un descendiente de $\hat{o}$) si existe una arista desde $\hat{o}$ a o en la clausura transitiva de la taxonomía T. Notar que un nodo no es ancestro de sí mismo (por ser el grafo acíclico).

Una RAG es una implicación de la forma $X \Rightarrow Y$ donde $X,Y \subset O$, $X \cap Y = \varnothing$ y no hay ítem en Y que sea ancestro de algún ítem en X (para que no existan reglas de la forma $x \Rightarrow \hat{x}$ que es trivialmente verdadera con 100% de confianza y por tanto redundante).

El problema de minar las RAG no puede reducirse al descubrimiento de las RA que tengan como soporte y confidencia un valor mayor que ciertos valores mínimos especificados por el usuario porque se encontrarían muchas reglas redundantes. La definición formal del problema según (Srikant,1995) del conjunto de construcciones que se relacionan a continuación:

- Sea $X \subseteq O$, Pr(X) es la probabilidad para la cual todos los ítems de X estén contenidos en una transacción. Entonces el soporte de una regla $X \Rightarrow Y$, es $Pr(X \cup Y)$, o sea, $sop(X \Rightarrow Y) = Pr(X \cup Y)$ y su confianza, $conf(X \Rightarrow Y) = Pr(Y|X)$. Si un conjunto {x, y | x, $y \in O$} tiene un mínimo soporte entonces {x, $\hat{y}$}, {$\hat{x}$, y} y {$\hat{x}$, $\hat{y}$} también lo tienen. Por tanto, si la regla $x \Rightarrow y$ tienen soporte y confidencia mínima, entonces la regla $x \Rightarrow \hat{y}$ también tendrá soporte y confidencia mínima; en tanto las reglas $\hat{x} \Rightarrow y$ y $\hat{x} \Rightarrow \hat{y}$ tendrán soporte mínimo pero pueden no tener confidencia mínima.
- Sea Z, X, $Y \subseteq O$, se dirá que $\hat{Z} \subseteq O$ es ancestro de Z, si se puede obtener $\hat{Z}$ a partir de Z reemplazando uno o más ítems en Z con sus ancestros, y $|Z| = |\hat{Z}|$. Las reglas $X \Rightarrow \hat{Y}$, $\hat{X} \Rightarrow Y$, $\hat{X} \Rightarrow \hat{Y}$ son ancestros de $X \Rightarrow Y$. Dado un conjunto de reglas $\hat{X} \Rightarrow \hat{Y}$ es un ancestro cerrado de $X \Rightarrow Y$ si $\neg\exists$ X' $\Rightarrow$Y' tal que X' $\Rightarrow$Y' es ancestro de $X \Rightarrow Y$ y $\hat{X} \Rightarrow \hat{Y}$ es ancestro de X' $\Rightarrow$Y'(definiciones similares se aplican a $X \Rightarrow \hat{Y}$, $\hat{X} \Rightarrow Y$)

- Sea $E_{\hat{Z}}[Pr(Z)]$ el valor esperado de Pr(Z) dado $Pr(\hat{Z})$, donde $\hat{Z}$ es un ancestro de Z. Sea Z ={x_1, ..., x_n} y $\hat{Z} = \{\hat{x}_1,...,\hat{x}_j, x_{j+1},...,x_n\}$ 1≤ j ≤ n, donde $\hat{x}_i$ es un ancestro de x_i. Se definirá: $E_{\hat{Z}}[Pr(Z)] = \frac{Pr(x_1)}{Pr(\hat{x}_1)} \times ... \times \frac{Pr(x_j)}{Pr(\hat{x}_j)} \times Pr(\hat{Z})$ como el valor esperado de Pr(Z) dado un conjunto de ítems $\hat{Z}$.

Hay que notar que el valor esperado de un ancestro $\hat{Z}$ de Z expresa un estimado de la disminución de la probabilidad de $\hat{Z}$ cuando no se generalizan los primeros j-ésimos ítems de Z.

Similarmente se define $E_{\hat{X}\Rightarrow\hat{Y}}[Pr(Y|X)]$ como la notación para la confidencia esperada de la regla X⇒Y dada la regla $\hat{X} \Rightarrow \hat{Y}$ definida por:

$$E_{\hat{X}\Rightarrow\hat{Y}}[Pr(Y|X)] = \frac{Pr(y_1)}{Pr(\hat{y}_1)} \times ... \times \frac{Pr(y_j)}{Pr(\hat{y}_j)} \times Pr(\hat{Y}|\hat{X})$$

donde Y = {y1,..., y_n} y $Y = \{\hat{y}_1,..., \hat{y}_j, y_{j+1},..., y_n\}$, 1 ≤ j ≤ n, siendo $\hat{y}_i$ es ancestro de y_i.

- Se llamará a una regla X⇒Y R-interesante con respecto a un ancestro $\hat{X} \Rightarrow \hat{Y}$ si el soporte de la regla X⇒Y es R veces el soporte esperado basado en $\hat{X} \Rightarrow \hat{Y}$ o la confidencia es R veces la confidencia esperada basado en $\hat{X} \Rightarrow \hat{Y}$.
- Dado un conjunto de reglas S y un valor mínimo de interés R, una regla X⇒Y es interesante (en S) si no tiene ancestros o si es R-interesante con respecto a sus ancestros cerrados entre sus ancestros interesantes. Entonces, una regla es parcialmente interesante (en S) si no tiene ancestros o es R-interesante con respecto al menos a un ancestro cerrado entre sus ancestros interesantes. (Srikant, 1995)

Hay que notar que una regla será interesante si alguna regla más general que ésta es interesante y se diferencia de ella en al menos un valor mínimo dado o si es la más general posible entre los conjuntos de ítems relacionados.

- Dado un conjunto de transacciones D y un valor de interés mínimo, el problema de minar las RA con taxonomía es el de encontrar todas las RA interesantes que tengan un soporte y confidencia mayor que un valor de soporte y confidencia mínimo dado por el usuario. Notar que en algunas aplicaciones puede ser interesante encontrar las reglas parcialmente interesantes y no simplemente las interesantes, y además si el mínimo valor de interés es 0, entonces serán encontradas todas las RA.

El problema de descubrir las RAG puede descomponerse en tres partes:

1. Encontrar todos los conjuntos de ítems cuyo soporte es mayor que un valor mínimo especificado.
2. Usar los conjuntos de ítems frecuentes para generar las reglas deseadas.
3. Borrar todas las reglas no interesantes del conjunto, empleando los conceptos anteriormente vistos.

Los algoritmos en (Agrawal,1993,1994) pueden usarse para generar las reglas dados los conjuntos frecuentes. Para el cálculo de estos últimos en (Srikant,1995) se brindan tres algoritmos:

2.2.1.1 Algoritmo básico

Añade todos los ancestros de cada ítem de una transacción a dicha transacción y ejecuta entonces cualquiera de los algoritmos de minado para estas transacciones extendidas.

2.2.1.2 Algoritmo Cumulate

Este algoritmo añade al anterior las siguientes optimizaciones:

- Filtrar los ancestros que serán añadidos a las transacciones: serán añadidos sólo aquellos que estén al menos en un conjunto de ítems candidato de la iteración actual. Recordar que los algoritmos de búsqueda de los conjuntos candidatos por lo general buscan conjuntos candidatos de k elementos a partir de conjuntos frecuentes de k-1 candidatos. De esta forma se están eliminando los ancestros para todos aquellos ítems que no pertenecen a ningún conjunto candidato. Además, si un ítem original no se encuentra en ningún conjunto de ítems frecuente puede ser también eliminado.
- Realizar un pre-cálculo de los ancestros de cada ítem: de manera que no se recalculen cada vez que se requiera conocer los ancestros de cada ítem atravesando el grafo de taxonomías, y no se consideren aquellos que no están presentes en ninguna de las transacciones.
- Podar los conjuntos de ítems que contengan un ítem y su ancestro: pues el soporte de un conjunto de ítem X que contiene tanto a x como a su ancestro $\hat{x}$ será igual al del conjunto X-$\hat{x}$ y además, si F_k, el conjunto de los conjuntos de ítems frecuentes de k-elementos, no incluye a ningún conjunto de ítem que contiene tanto a él como a su ancestro, entonces el conjunto candidato C_{k+1} no incluirá a ningún conjunto de ítems que contenga tanto a él como a su ancestro, pues son podados durante la generación de los conjuntos candidatos cuando se descartan aquellos conjuntos que contienen subconjuntos infrecuentes.

2.2.1.3 Algoritmo EstMerge

Primeramente, para emplear eficientemente la información contenida en la taxonomía se define para cada conjunto de ítems un valor de su profundidad según:

$$profundidad(x) = \begin{cases} 0, & si\ no\ existen\ ancestros\ para\ X \\ \max(profundidad(\hat{X}) / \hat{X}\ es\ un\ ancestro\ de\ X) + 1, & e.o.c. \end{cases}$$

Para visitar todos los conjuntos de ítems de un tamaño específico, se podrían contar entonces primero los de profundidad 0, y los conjuntos de profundidad p se calculan como los descendientes de los conjuntos de profundidad p-1, p ≥ 1, o sea, para cada conjunto de ítems de tamaño k, hay que descender en la taxonomía mientras el soporte de dichos conjuntos excedan el soporte mínimo especificado por el usuario. Realizar el proceso anterior puede demorar mucho tiempo atravesando múltiples veces por la BD. En su lugar, este algoritmo para el cálculo de los conjuntos candidatos de k ítems, genera primero los conjuntos de k elementos de profundidad 0, luego, los de profundidad 1 y así sucesivamente hasta obtener al menos el 20% de todos los posibles candidatos de k elementos, lo que es una heurística determinada empíricamente para detener el proceso. Además, con el objetivo de disminuir los recorridos por toda la BD, este algoritmo emplea un muestreo de dicha base en su lugar, para estimar el soporte de los candidatos. Para ello, se extraen de C_k los conjuntos $C_k^{'}$ y $C_k^{''}$, los conjuntos de ítems candidatos de tamaño k que se espera que ellos o sus padres tengan soporte mínimo (los que tienen soporte mínimo respecto a la base de muestreo) y los conjuntos de ítems se espera no superen el soporte mínimo, (los que no tienen soporte mínimo respecto a la base de muestreo). Se añade entonces al conjunto de conjuntos frecuentes de tamaño k (F_k) todos los candidatos en $C_k^{'}$ para los que se comprueba que efectivamente tienen soporte mayor que el valor mínimo especificado (usando la BD completa) y en F_{k-1} aquellos candidatos que en la iteración anterior se esperaba no tuvieran mínimo soporte,

C''_{k-1}, y en la presente se comprueba que tienen el valor mínimo especificado. Hay que notar que en una misma pasada por toda la BD, se calculan los soportes de los conjuntos en C'_k y C''_{k-1}. Ello evita tener que realizar un pase extra por toda la BD una vez conocido que al menos un conjunto en C'_k tiene soporte mínimo, pues ello indicaría que se necesita una comprobación para los descendientes de los conjuntos en C'_k (esto no se logra con el cálculo de los conjuntos candidatos de la próxima iteración, pues en ésta se calculan conjuntos de k+1 elementos, no de k, generados a partir de los descendientes de conjuntos de k elementos), aunque, por supuesto, siempre se necesita el cálculo del soporte de todos los conjuntos candidatos, sea en la iteración actual o en la siguiente.

El tamaño de la muestra de la BD a emplear es estimado usando las aproximaciones de Chernoff (Hangerup., 1990) Considerando que *s* es una variable aleatoria con distribución binomial de *n* pruebas que mide el número de transacciones en la muestra que contendrán a un conjunto de ítems X, cada una teniendo probabilidad de ocurrencia *p*, entonces la probabilidad del soporte en la muestra sea al menos parecida a *k* es dado por:

$\Pr[s \succeq an] \leq \left[\left(\frac{p}{a}\right)^a \left(\frac{1-p}{1-a}\right)^{1-a}\right]^n$ donde $s \succeq k$ ("*s* es al menos parecida a *k*") se define como:

$$s \succeq k \Leftrightarrow \begin{cases} x \geq k & si\ k \geq p \cdot n \\ x \leq k, & e.o.c. \end{cases}$$

La ecuación anterior sugiere que:

- el tamaño de la muestra debe incrementarse con el decremento del valor del soporte mínimo y
- la estimación hecha resulta ser una fracción del soporte real y depende solamente

del tamaño de la muestra y no del de la base de datos.

Ello implica que las reglas con baja ocurrencia implican tener un tamaño de la muestra grande, por lo tanto en cada iteración se recorrerá prácticamente dos veces.

Hay que notar que los tres algoritmos trabajan con la misma filosofía del algoritmo Apriori, introduciendo únicamente el problema de la generación de conjuntos candidatos que mezclen ítems de cualquier profundidad dada la taxonomía que los caracteriza. Sin embargo, el algoritmo básico descrito arriba evidentemente cuenta muchos conjuntos de ítems que no son necesarios contar si se considera la información de los conceptos involucrados en la taxonomía; el segundo algoritmo, aunque hace tres importantes optimizaciones teniendo en cuenta lo anterior, requiere en una pasada contar el soporte de todos los conjuntos candidatos de k elementos, imposibilitando eliminar algunos candidatos dado el valor de su profundidad; el tercer algoritmo tiene en cuenta lo anterior (genera conjuntos de ítems comenzando por los niveles superiores de la jerarquía), pero como máximo un 20% de los candidatos no son tomados en consideración, por lo que algunas RA no podrían descubrirse. Los resultados experimentales, como era de esperar, arrojan que el de mejores resultados (en tiempo) es el EstMerge, seguido por Cumulate y con una gran diferencia respecto al algoritmo básico.

En (Hipp,1998) se describe un nuevo algoritmo para encontrar las reglas RAG de forma rápida, que a diferencia de los algoritmos en (Srikant,1995) realizan una búsqueda primero en profundidad de los candidatos que se forman.

El recorrido realizado por los algoritmos de tipo Apriori asegura que todos los conjuntos de ítems frecuentes son visitados y el número de conjuntos infrecuentes visitados se reduce. En tanto, los algoritmos introducidos en (Zaki,2000) que usan búsqueda primero en profundidad para atravesar el árbol, restringen minar sólo conjuntos frecuentes de k elementos, $k \geq 3$, y además, un recorrido arbitrario primero en profundidad no garantiza que los subconjuntos infrecuentes de (k-1) elementos de un candidato C sean conocidos en el momento que el soporte de C se determina y por tanto

los candidatos teniendo subconjuntos infrecuentes no pueden eliminarse.

Por otro lado contar el soporte de los candidatos como en Apriori es costoso, debido a que el número de subconjuntos frecuentes en cada candidato es cada vez mayor y el número de niveles en el árbol de candidatos se incrementa. Usando una implementación como la de Partition o Eclat, (empleando el cardinal de la intersección de los subconjuntos frecuentes) es relativamente más eficiente, aunque tiene dos problemas: el coste en memoria es mayor, pues requiere almacenar para cada ítem el conjunto de transacciones que lo contiene; y además el cálculo del soporte de un candidato obtenido por la unión de dos conjuntos frecuentes obliga a intersectar los dos conjuntos con el consabido coste computacional que representa.

2.2.1.4 Prutax

El algoritmo Prutax que se expone en (Hipp,2000) se basa en mezclar las dos características anteriormente citadas: búsqueda primero en profundidad y la intersección de los conjuntos frecuentes para el cálculo de los soportes. Además permite eliminar candidatos usando la información de la taxonomía, basándose en tres optimizaciones:

- Eliminar los conjuntos de ítems no frecuentes: para realizar una búsqueda primero en profundidad, mezclando conjuntos de ítems con (k-1) elementos iguales, de acuerdo a un orden lexicográfico, donde se garantiza conocer todos los subconjuntos de $|C|$-1 elementos cuando se determina el soporte de un candidato C, es necesario realizarlo también en el orden primero el de más a la derecha.
- Podar los conjuntos de ítems que contengan un ítem y su ancestro: (igual que para (Srikant,1995)).
- Podar por profundidad en la taxonomía: conocido que pueden eliminarse los candidatos cuyos ancestros son infrecuentes y que además puede usarse el mismo orden sugerido por EstMerge de acuerdo con la profundidad de los conjuntos de ítems, entonces determinar el soporte de todos los candidatos C con profundidad i, antes de contar un candidato C' con profundidad i+1 asegura

que todos los ancestros no frecuentes de un candidato son conocidos cuando su soporte es contado.

Trabaja de la siguiente manera:

Se toma un primer conjunto frecuente entre los conjuntos pasados por parámetro H_i. Se mezcla éste con cada uno de los restantes conjuntos frecuentes pasados por parámetro, asignándole en cada paso esta unión al conjunto C. Si su cardinal es 2 y el segundo de los ítems es ancestro del primero, entonces no consideró a este conjunto(optimización 2); sino, si su cardinal es distinto de 2 y alguno de los conjuntos frecuentes de $|C|$-1 elementos no pertenece a F, entonces tampoco puede considerarse(optimización 1), sino, si todos los ancestros de C están contenidos en F (garantizando el orden de la optimización 3), entonces se verificará si su soporte es mayor que el mínimo permitido, y en ese caso insertémoslo en los conjuntos de ítems frecuentes, F, y en los conjuntos frecuentes generados por la unión de dos conjuntos frecuentes del llamado actual, E. Para mezclar los conjuntos frecuentes resultantes de las mezclas anteriores, se llama nuevamente al algoritmo para aquellos conjuntos frecuentes que se formaron al mezclar dos de los conjuntos que fueron pasados por parámetro. Esto se repetiría hasta que E contenga a uno o a ningún conjunto frecuente, o sea, hasta que no pueda formarse ningún otro conjunto que contenga a H_i. Para buscar a los restantes conjuntos frecuentes habría que realizar nuevamente este proceso fijando en cada paso un nuevo conjunto H_i (de los pasados inicialmente como parámetro).

Para emplear este algoritmo basta realizar una llamada a él con los conjuntos 1-itemsets frecuentes.

2.2.2 Reglas de asociación de mínima condición y máxima consecuencia

Un subconjunto de RA que permitan predecir tanto como sea posible con la mínima cantidad de hechos es generalmente de gran interés en las aplicaciones. Tales reglas son llamadas reglas de asociación de mínima condición y máxima consecuencia (RMM). Son una generalización de las RAR, pero además de reducir la cantidad de hechos en la

parte izquierda de la regla, maximizan la cantidad de consecuentes. Formalmente el conjunto de reglas de asociación de mínima condición máxima consecuencia, con soporte mínimo MinSop y confidencia MinConf se definen como:

RMM (MinSop, MinConf) = { r: $X \Rightarrow Y \in$ RAR(MinSop, MinConf) |

$\neg\exists$ r': $X' \Rightarrow Y' \in$ RA(MinSop,MinConf), $r' \neq r$ y $X' \subseteq X$, $Y \subseteq Y'$ }

En (Krys,1998) se demuestra que RMM(MinSop, MinConf) $\subseteq$ RAR(MinSop, MinConf). El cálculo de las RMM se basa en:

- Calcular el conjunto RAR(MinSop, MinConf)
- Verificar de acuerdo a la definición anterior si son de mínimo antecedente y máximo consecuente.

2.2.3 Reglas de asociación representativas (RAR)

El número de RA generadas por los algunos algoritmos estudiados puede ser extremadamente grande, para aliviar este problema en (Krys,1998) fue introducida la noción de RAR. El conjunto de éstas es el menor conjunto de reglas que cubren todas las RA, el resto pueden ser generadas empleando el operador de cubrimiento, sin necesidad de acceder nuevamente a la BD.

Se llamará RA(MinSop, MinConf) al conjunto de las RA con soporte y confidencia mayor que MinSop y MinConf, respectivamente.

El cubrimiento C de una regla $X \Rightarrow Y$, $X,Y \neq \varnothing$, $X,Y \subset O$, siendo O el conjunto de objetivos presentes, es definida por: $C(X \Rightarrow Y) = \{ X \cup Z \Rightarrow V \setminus Z, V, Z \subseteq Y , Z \cap V = \varnothing \}$, o sea, el cubrimiento de una regla r' es el conjunto de reglas r cuyo antecedente es la unión del antecedente de r' y algún subconjunto del consecuente de r'; y su consecuente, un subconjunto de los ítems del consecuente de r' que no se encuentran en el antecedente de r.

De la definición anterior se deriva la siguiente propiedad:

Sea r: $X \Rightarrow Y$ y r': $X' \Rightarrow Y'$ reglas de asociación, entonces $r \in C(r')$ si y solamente si $X \subseteq X' \cup Y'$ y $X' \subseteq X$.

Las RAR se definen como:

$$RAR(MinSop, MinConf) = \{ r \in RA(MinSop, MinConf) \mid \neg\exists\, r' \in RA(MinSop, MinConf), r' \neq r \text{ y } r \in C(r') \}$$

o sea, las RAR son las de asociación que no pertenecen al cubrimiento de ninguna otra RA. (Kryszkiewicz, 2001)

Una RA es RAR si no existe ninguna otra RA que involucre al mismo conjunto de ítems, y cuyo antecedente sea menor que el de dicha RAR. Como se puede observar, éstas tratan de minimizar la cantidad de elementos en el antecedente.

El algoritmo FastAllGenRepresentatives toma como entrada todos los conjuntos frecuentes generados por Apriori y calcula todas las RA para cada conjunto frecuente de k elementos, $k \geq 2$. Para ello recorre cada uno de los conjuntos frecuentes, Z, de algún Fk, a partir del cual solamente k reglas pueden ser generadas. Primero, se calcula MaxSup como el máximo de los soportes de los conjuntos frecuentes en Fk+1 que sean superconjuntos de Z. Si no hay superconjunto de Z, entonces MaxSup = 0. Si el soporte de Z es igual a MaxSup, ninguna RAR puede ser generada a partir de Z (existe algún conjunto en Fk+1 que genere una regla más general que la que podría encontrarse a partir de Z). (Saquer, Deogun, 2000)

En otro caso, puede generarse el conjunto A_1 de antecedentes, formado por los conjuntos unitarios de los elementos en Z y comienza a iterarse en el ciclo 1 cuya i-ésima iteración puede describirse como: se considera cada regla candidata de la forma: $X \Rightarrow Z \backslash X$, donde $X \subset Z$ es algún conjunto de ítems de i-elementos perteneciente a A_i. Como Z es frecuente, entonces X, también lo es. Para comprobar si $X \Rightarrow Z \backslash X$ es una RA, se

determina su confianza y si su valor es mayor que la confianza mínima exigida se verifica si existirá alguna otra regla que se forme con algún conjunto frecuente en F_{k+1} que cubra a X⇒Z\X. Si MaxSup/XCount ≥ MinSop tal regla no existe, de otra manera X⇒Z\X es representativa. No es necesario comprobar si otra RA X'⇒Z\X', X'⊂X que cubra a X⇒Z\X pues cada vez que se encuentra un conjunto de A_i que es antecedente de una regla de asociación generada por los conjuntos de Z se elimina de los posibles antecedentes y se aplica nuevamente APrioriGen para generar los nuevos antecedentes de i+1 elementos a partir de los conjuntos de ítems mantenidos en A_i. (Kryszkiewicz, 2001)

2.3 Algunos otros trabajos relacionados

Muchos otros trabajos interesantes se han desarrollado a la par de los aquí discutidos. Por ejemplo en (Amir,1997) (Rougel,1998) (Han,2001) (Toivonen, 1996) donde se discuten temas relacionados con el mantenimiento y la actualización en entornos reales de las RA extraídas; en (Toivonen, 1996) se introduce el problema del descubrimiento de reglas booleanas y es retomado el tema en (Amir,1997) restringiéndolas en este caso a las llamadas reglas de exclusión.

2.4 Limitaciones actuales

En el epígrafe anterior se realizó un análisis de las principales soluciones ofrecidas al problema del minado de reglas en BD. En el presente epígrafe se repasarán estos trabajos desde el punto de vista de las principales limitaciones prácticas de cada uno de ellos.

2.4.1 Limitaciones de las soluciones generales

Para acceder a los conjuntos de ítems se emplean las estrategias BFS y DFS, en árboles de conjuntos itemsets. Cuando se emplea una estrategia de búsqueda BFS como es realizado por Apriori y Partition, todos los conjuntos frecuentes de k-1 elementos son conocidos cuando se generan los candidatos de k elementos, por lo que estos algoritmos mejoran el rendimiento al podar aquellos ítems que contienen subconjuntos infrecuentes,

aunque posteriormente al mezclar pares de conjuntos frecuentes con k-2 elementos iguales, hay que verificar si todos los subconjuntos de k-1 elementos pertenecen al conjunto de itemsets frecuentes. Esta estrategia garantiza que todos los conjuntos frecuentes sean visitados, en tanto se reduce la cantidad de conjuntos infrecuentes visitados, no garantiza que los conjuntos de candidatos generados sean frecuentes, por lo que después de la generación de éstos hay que volver a contar el soporte de dichos conjuntos.

Una búsqueda DFS arbitraria no garantiza siquiera el conocimiento de todos los k-1 itemsets cuando se generan los k-itemsets. Por tanto los candidatos conteniendo conjuntos infrecuentes no pueden ser eliminados y su soporte debe ser posteriormente contado, con el coste computacional que ello supone.

El cálculo del soporte de los candidatos, supone el uso de dos estrategias fundamentales: por la cuenta directa en la BD y el empleo de los llamados tidlist de los conjuntos de ítems.

Cuando se emplea el primero de los métodos, como es realizado por Apriori, recae en el empleo de una estructura de árbol hash, donde se describen todos los conjuntos candidatos de una iteración dada, el conteo del soporte de los candidatos. Pero, en la medida que el número de niveles en el árbol se incrementa, esta estrategia se vuelve más y más costosa, incluso cuando se cuentan candidatos que ocurren infrecuentemente; tengamos en cuenta, además, que por cada transacción de la base de datos hay que recorrer el árbol tantas veces como sea necesario para encontrar todos los candidatos que lo forman, por lo que no sólo es costosa la solución en memoria, sino además en tiempo. (Pasquier, 2000)

Calcular el soporte de los candidatos intersectando conjuntos, como en Partition y Eclat, requiere mantener los llamados tidlist para cada itmset, por lo que convergen dos problemas: el coste computacional de cada intersección es de al menos el mínimo valor del cardinal de los conjuntos que se intersectan y además el uso de la memoria se vuelve

crítico, aunque para solucionar este problema pudieran emplearse estrategias como la propuesta en (Zaki,2000).

2.4.2 Limitaciones de los algoritmos específicos

Los problemas de Apriori por el empleo del árbol hash, se intentaron solucionar en AprioriTID (empleando en lugar de la BD original una tabla booleana que contiene los conjuntos candidatos contenidos en cada transacción), de manera que no se necesite recorrer el árbol hash para descubrir los candidatos en cada transacción, sin embargo el tiempo requerido para su construcción en cada iteración hace que los beneficios reales brindados por esta estructura no sean tan halagüeños como los esperados.

En DHP se emplea una tabla hash para reducir el número de candidatos, cuánto se puede lograr en este sentido depende del número de falsos positivos, los que son generados cuando los valores hash son idénticos para un grupo de itemsets candidatos cuyas frecuencias individuales son menores que el valor del soporte mínimo pero el valor hash asociado no es menor que dicho umbral. El número de candidatos que tienen igual valor hash está directamente relacionado con el tamaño de la tabla. Por otro lado, el espacio de memoria empleado por la tabla compite con el necesitado por al árbol hash (mediante el que se cuenta el soporte de los itemsets). De ahí que, con tablas hash grandes (para reducir la cantidad de falsos positivos) la memoria se vuelve insuficiente, sobretodo en casos donde el soporte empleado tiene valores bajos. (Agrawal, 1993)

Por su parte, Partition, emplea un método versátil para calcular los conjuntos frecuentes, realizando para ello sólo dos pasadas a través de la BD. Desafortunadamente el óptimo rendimiento de este algoritmo depende de una buena selección del tamaño de las particiones. Si las particiones son muy grandes, entonces los tidlist serán grandes también y no podrán ser colocados en la memoria principal, pero si la partición es muy pequeña pueden encontrarse muchos conjuntos frecuentes locales a una partición que no responden a los conjuntos frecuentes globales. Ello obliga a contar el soporte de muchos conjuntos candidatos que quizás no responden a la demanda de la regla a minar y además los frecuentes locales pueden diferir mucho de los globales

y el conjunto de frecuentes globales será muy grande. Por último este algoritmo considera mas candidatos que Apriori y si hay muchos conjuntos maximales frecuentes, el algoritmo se vuelve muy poco factible.

Otro de los algoritmos analizados fueron los pertenecientes a la familia Eclat. En este caso, los que emplean clases de equivalencia se benefician en el sentido de la simplicidad de su cálculo, sin embargo, los conjuntos potencialmente maximales resultantes pueden no aproximarse a los verdaderos conjuntos frecuentes maximales. Los que emplean el cálculo de los cliques maximales tienen un considerable incremento dada la complejidad de este cálculo, y los conjuntos potencialmente maximales generados responden aproximadamente a los verdaderos frecuentes maximales. Por otro lado, la estrategia de minar sólo los conjuntos maximales empleando el método híbrido no garantiza conocer los soportes de los conjuntos frecuentes que éstos contienen, por lo que para generar las RA habría que realizar otro pase por la BD para calcular dichos soportes. Por último, la estrategia DFS empleada en todos estos algoritmos (calcular los conjuntos frecuentes por cada itemset potencialmente frecuente) no garantiza conocer todos los subconjuntos infrecuentes de un candidato C, y por tanto pueden ser eliminados hasta reconocer si se cumple esta propiedad.

El cálculo de las RA, es particularmente fácil y aunque en el algoritmo descrito anteriormente no se emplea ninguna estrategia para reducir el número de reglas investigadas es posible mejorar su rendimiento considerando las relaciones entre los soportes de los subconjuntos propios de los conjuntos frecuentes usados como antecedentes en la construcción de las reglas.

2.4.3 Limitaciones de las generalizaciones estudiadas

Las generalizaciones aquí discutidas, se sustentan sobre la base de emplear dos estrategias fundamentales: el uso de una taxonomía de conceptos y la disminución del número de reglas reduciendo la cantidad de ítems en las partes derecha e izquierda de las reglas obtenidas.

El hecho de emplear la taxonomía de conceptos incrementa notablemente el número de combinaciones a explorar, pues anteriormente sólo se verificaban los nodos hojas contenidos en la taxonomía. La bondad de los resultados obtenidos depende, por un lado, de la jerarquía utilizada (definida en la totalidad de los casos de forma manual) y por otro, de la estrategia de búsqueda dentro de la jerarquía, pues en algunos casos como en EstMerge, se podan conjuntos candidatos sin el empleo de ninguna propiedad que asegure que los candidatos podados no constituyen también conjuntos frecuentes, aunque para aliviar este problema se recorre la jerarquía desde los conceptos más generales a los más particulares.

La disminución de la cantidad de RA considerando el concepto de cubrimiento, como sucede en las RAR y RMM, puede provocar que algunas reglas de gran importancia práctica sean desestimadas, pues se sustenta su búsqueda sólo en los valores de soporte y confidencia de las reglas.

2.5 Conclusiones del capítulo.

Los temas tratados en este capítulo fueron muy importantes para el posterior diseño e implementación de los algoritmos escogidos como resultado de la investigación:

- ✓ Se trataron las generalizaciones al problema del descubrimiento de las RA y las distintas derivaciones que el tema conlleva.
- ✓ Se describieron los trabajos (algoritmos) mas importantes desarrollados en cuanto a las generalizaciones especificadas.
- ✓ Y como último, pero no menos importante, se abordaron las distintas limitaciones que presentan los trabajos (algoritmos) estudiados como parte de la investigación.

Todos los temas tratados en el capitulo ofrecieron las bases y el conocimiento necesario para establecer cual de los trabajos estudiados es el mejor candidato para la implementación y mejora como resultado de la investigación. Por lo tanto, se escoge el algoritmo que resuelve el minado de reglas de asociación representativas, por ser la solución que trata de de minimizar la cantidad de elementos en el antecedente.

CAPÍTULO 3: Diseño e implementación de la herramienta.

3.1 Introducción

En este capítulo se justifica el por qué se escogen el Apriori y las RAR para la implementación, mostrando pormenores del diseño e implementación de la herramienta para los algoritmos seleccionados. El diseño debe tener flexibilidad, para que en un futuro se puedan implementar otros algoritmos. Se comienza con una breve descripción y justificación de las herramientas computacionales utilizadas. Posteriormente se continúa con las especificaciones técnicas de cada clase.

3.2 ¿Por qué escoger A_priori para la generación de ítems frecuentes?

El algoritmo A_priori es voraz. Explora todas las transacciones cada vez que comprueba los k-itemsets candidatos. Este algoritmo encuentra las asociaciones más frecuentes e itera sobre la base de datos hasta que las asociaciones obtenidas no tienen el soporte mínimo, es un método simple pero robusto, y presenta una salida intuitiva.

Los requisitos que cumple este algoritmo son los mencionados a continuación:

- No necesita decir los atributos de los lados derecho (consecuente) e izquierdo (antecedente) de las reglas pues se generan de manera automática.
- Existen variedades para tratar todo tipo de datos.
- Especificar mínimo soporte.
- Especificar máximo número de reglas.

A_priori busca iterativamente conjuntos frecuentes con cardinalidad 1 hasta k, y después usa los conjuntos frecuentes para generar las reglas de asociación. En el paso clave del descubrimiento de conjuntos frecuentes, se basa en el principio A priori, que expresa que cualquier subconjunto de un conjunto de artículos frecuente, debe ser frecuente, esto permite definir el principio de poda en A priori:

Si existe algún conjunto infrecuente, entonces no hay necesidad de generar sus superconjuntos.

3.2.1 Parámetros: soporte y confianza.

- Soporte mínimo (s): Alto ⇒ pocos conjuntos frecuentes
  ⇒ pocas reglas válidas que ocurren con frecuencia
  Bajo ⇒ muchas reglas válidas que ocurren raramente
- Confianza mínima (c): Alta ⇒ pocas reglas, pero todas "casi ciertas lógicamente"
  Baja ⇒ muchas reglas, pero muchas de ellas muy inciertas

Valores típicos: soporte = 2-10 % confianza = 70-90 %

3.3 ¿Por qué las Reglas de Asociación Representativas?

El número de reglas de la asociación es a menudo muy grande, siendo a veces inservible para las aplicaciones de la vida real. Una posible solución a este problema es restringir la extracción de reglas a aquellas que realmente se resuelvan lo que el usuario necesita, o sea, aquellas reglas que resulten nuevas o interesantes para él.

Como se mencionaba en el capitulo anterior, una RA es RAR si no existe ninguna otra RA que involucre al mismo conjunto de ítems, y cuyo antecedente sea menor que el de dicha RAR, o sea, las RAR son el mínimo juego de reglas pertenecientes a las RA que se puedan generar durante el proceso de extracción. Normalmente, el número de RAR es mucho más pequeño que el número de todas las RA, y no se necesitan ningunas medidas adicionales para determinar las RAR.

Por esta razón, y las consideraciones tratadas en el capítulo anterior, se escoge el algoritmo FastAllGenRepresentatives para su implementación. También, se escoge por el hecho de que utiliza como entrada todos los conjuntos frecuentes generados por Apriori, algoritmo escogido para resolver la primera parte del problema.

3.4 Metodología y herramientas empleadas en la implementación.

Durante el diseño e implementación de la biblioteca de clases se han utilizado varias herramientas que serán descritas a continuación, las cuales contribuyeron a que el producto tuviera mejor calidad.

C++**:**

Como requisito este trabajo ha sido implementado en el lenguaje de programación C++. El C++ es un derivado del lenguaje C. Es un lenguaje de programación orientado a objetos, con una increíble versatilidad, con el que pueden programarse desde los programas más simples a los más complicados, incluso sistemas operativos.Además es portable, es decir, un programa con el código escrito en C++, se podrá compilar en cualquier sistema operativo o sistema informático sin necesidad de cambiar casi el código fuente. Es un lenguaje multi-nivel, es decir, se puede usar tanto para programar directamente el hardware (dependiendo del sistema operativo), como para crear aplicaciones tipo Windows definidas todas por poseer una misma interfaz.

Rational Rose**:**

Para desarrollar los diagramas de clases del diseño se emplea la herramienta CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) Rational Rose, actualmente conocida como una familia de software de IBM para el levantamiento de requerimientos, diseño, construcción, pruebas y administración de proyectos en el proceso desarrollo de software. Sus productos están centrados en la metodología del Proceso Racional Unificado o RUP (Rational Unified Process) y utiliza el lenguaje de modelado UML.

UML**:**

UML (Unified Modeling Language, Lenguaje Unificado de Modelado) se ha convertido en el estándar para definir, organizar y visualizar los elementos que configuran la arquitectura de una aplicación orientada a objetos. Constituye el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. El estar apoyado por la OMG (Object Management Group) como la notación estándar para el desarrollo de proyectos informáticos constituye una de sus principales ventajas. Es útil para el desarrollo de modelaje visual de cualquier proyecto no solo informático y más aun es estándar, promueve la reutilización, puede ser utilizado con cualquier metodología a lo largo del proceso del desarrollo del software y en cualquier plataforma tecnológica donde se implementara ya sea Unix, Windows, etc.

3.4.1 Modelo de diseño.

Para lograr que en un futuro se puedan implementar nuevos algoritmos e incluso mejorar el existente se han empleado interfaces, y para mantener una estructura organizada se utilizaron paquetes a los cuales se le asignaron determinadas responsabilidades. Bajo este esquema ha sido realizado el modelo de diseño. En esta sección se efectuará una explicación detallada del diseño de la herramienta.

La biblioteca de clases tiene implementado cinco módulos o paquetes: Interfaces, Aprioriapp, Accdatos, Combinaciones y GenerarReglas. La siguiente figura muestra el modelo de diseño de la herramienta.

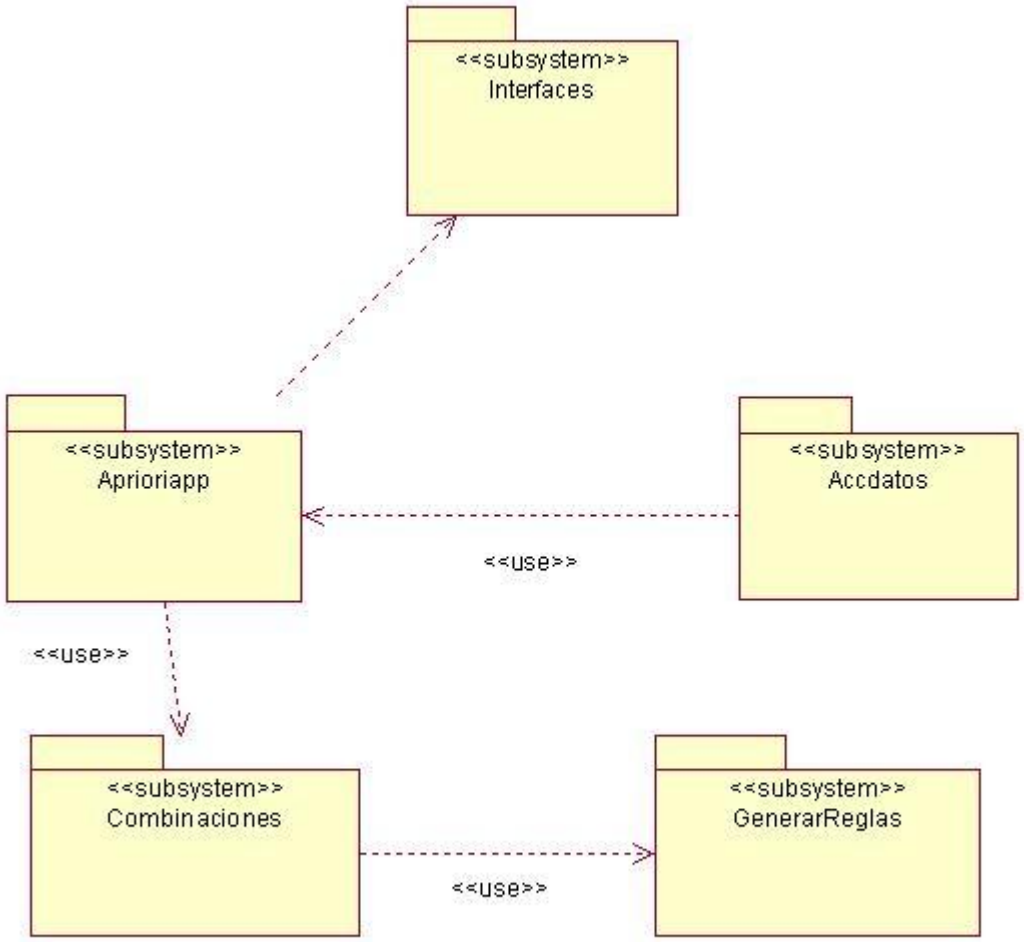


Fig.3.1. Modelo de diseño de los módulos de la biblioteca.

3.4.1.1 Módulo Aprioriapp.

En este módulo se encarga de explorar en la tabla de la BD que se está tratando en busca de los elementos que según el usuario sean frecuentes. Para ello analiza las transacciones y en cada una se ellas cuenta la cantidad de apariciones de los elementos consultados y guardando los que cumplan con el mínimo soporte especificado por el usuario. Luego, a partir de esos elementos forma combinaciones de ellos con un grado superior, realizando la misma operación formando ahora set de elementos que juntos cumplan con la condición de mínimo soporte, hasta que se encuentren todas las posibles combinaciones.

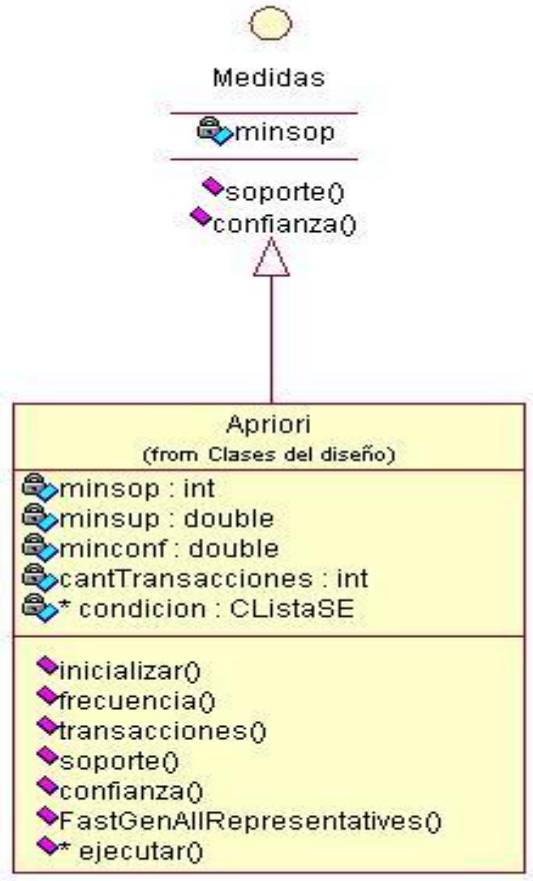


Fig.3.2.Diagrama de clases del módulo Aprioriapp.

3.4.1.2 Módulo Accdatos.

En este módulo está implementada la conexión a la BD, donde el usuario debe especificar la tabla en la de la que desea obtener resultados. Luego ce cierra la conexión mediante otro método que tiene implementado este módulo.

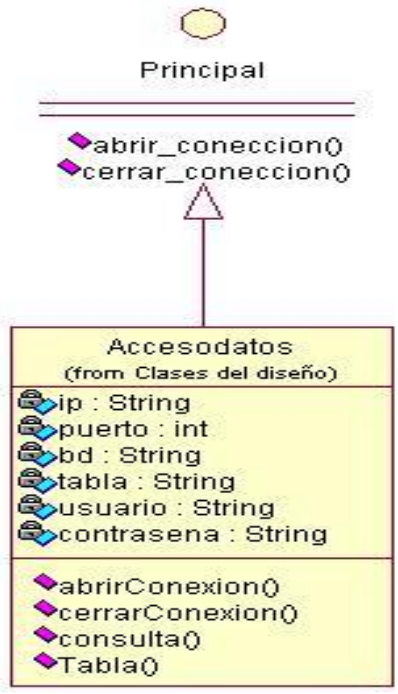


Fig.3.3. Diagrama de clases del módulo Accdatos.

3.4.1.3 Módulo Combinaciones.

Este módulo va a formar combinaciones con las variables analizadas que cumplan con el mínimo soporte especificado por el usuario, las va a ir agregando a una lista donde la longitud será igual al grado calculado en el módulo Aprioriapp. Esto sucederá hasta que no haya posibilidad de formar más combinaciones.

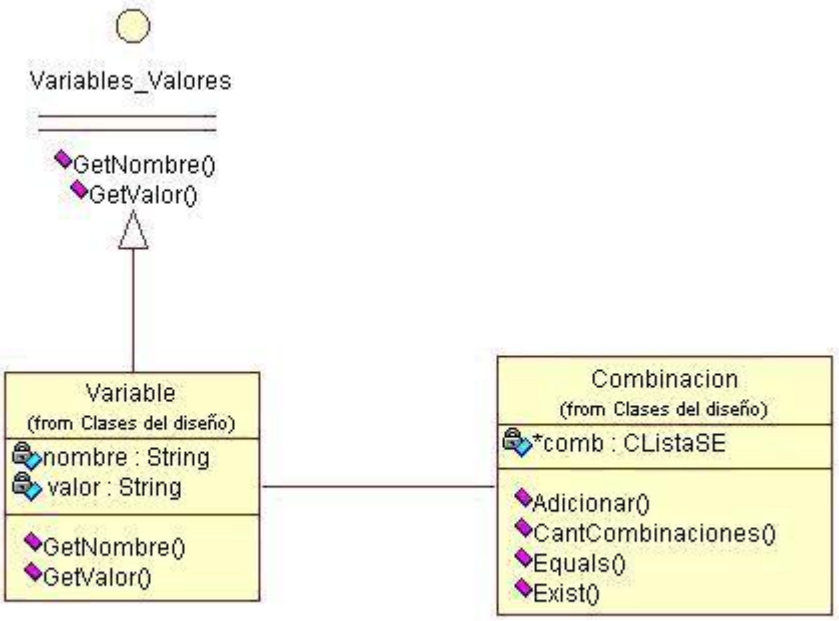


Fig.3.4. Diagrama de clases del módulo Combinaciones.

3.4.1.4 Módulo GenerarReglas.

Este módulo es el encargado de la implementación de la división de las combinaciones en parte implicativa (izquierda) y parte implicada (derecha), para así formar las diferentes reglas de asociación que cumplan con la restricción para ser representativas.

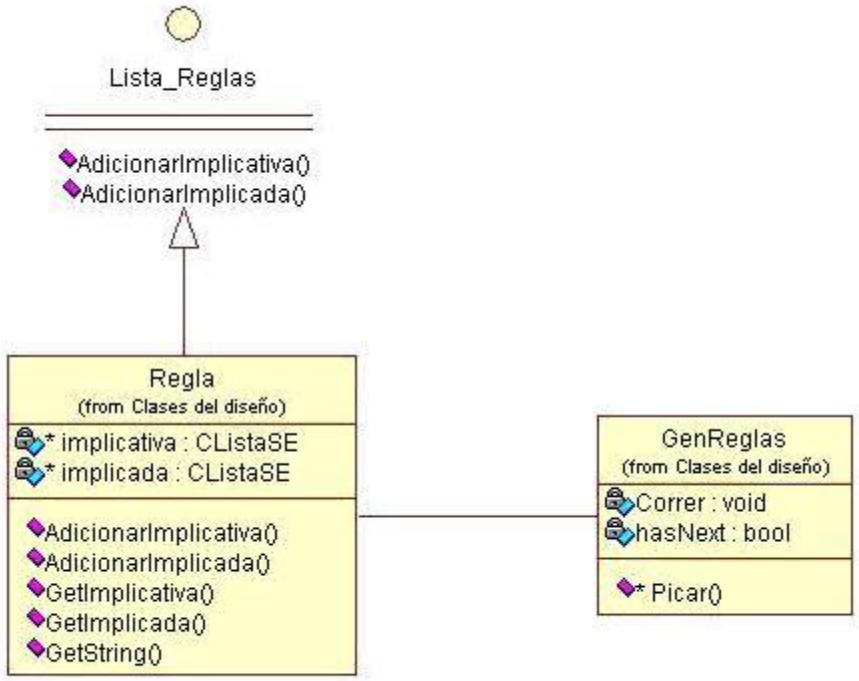


Fig.3.5. Diagrama de clases del módulo GenerarReglas.

3.4.1.5 Módulo Interfaces.

El módulo es responsable de la implementación de las interfaces, garantizando de esta forma, que sea amigable para cualquier usuario. Se han definido cuatro interfaces: Lista_Reglas, Medidas, Variables_Valores y la Principal, dándole solución visual al problema de generación de las RAR.

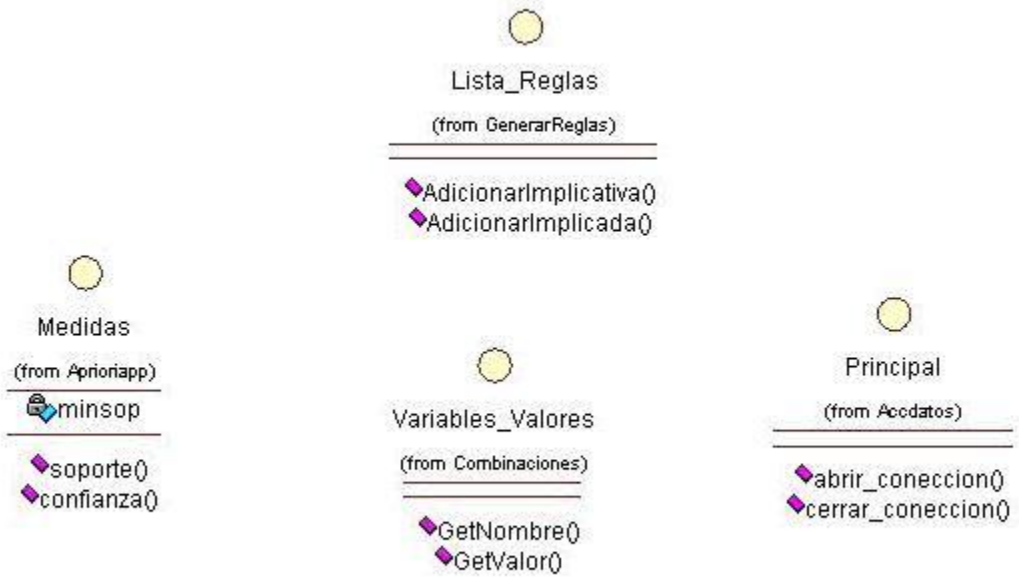


Fig.3.6. Diagrama de clases del módulo Interfaces.

3.4.2 Implementación.

En este epígrafe se describen brevemente las partes más importantes del código de la biblioteca de clases. En sus inicios se ha concebido la implementación de dos algoritmos, el primero, encuentra los elementos frecuentes y un segundo forma reglas de asociación con la combinación de dicho elementos. La implementación está concebida inicialmente para la conexión a BD MySQL y Oracle, manteniendo una estructura viable y flexible para que en trabajos futuro se le pueda realizar mejoras en cuanto lo permita.

Frecuencia**:**

Método que cuenta las incidencias de un elemento en la tabla analizada: Este método analiza las incidencias en la BD del elemento o combinación de elementos y toma como frecuentes los iguales o mayores que el mínimo soporte especificado por el usuario.

```
int frecuencia(Combinacion* comb)
{
        String restriccionVariables="";
        restriccionVariables+=comb->Valor(1)->GetNombre()+" = '"+comb->Valor(1)->GetValor()+"'";

        for(int i=2;i<=comb->CantCombinaciones();i++)
                restriccionVariables+=" and "+comb->Valor(i)->GetNombre()+" = '"
                +comb->Valor(i)->GetValor()+"'";

        String query = "Select Count(*) from "+conexion->Tabla()+" where "+restriccionVariables;

        return conexion->consulta(query);
}
```

Fig.3.7. Método que busca la frecuencia de los elementos.

Combinar**:**

Método que combina los elementos frecuentes y forma combinaciones con ellos: Forma listas de combinaciones con los elementos encontrados como frecuentes utilizando el método del reloj contador.

```
CListaSE<Combinacion*>* combinar(Combinacion* lista, int* a, int n)
{
      int current = n-1;
      int v = a[current];
      int l = lista->CantCombinaciones();
      CListaSE<Combinacion*> *ret = new CListaSE<Combinacion*>();

      while(hasNext(n, current+1, l, v))
      {
            Combinacion* nueva=new Combinacion();

            for(int i = 0; i < n; i++)
            nueva->Adicionar(lista->Valor(a[i]));

            ret->Adicionar(nueva);

            Correr(a, n, current, l);

            v = a[current];
      }

      while(current >= 0)
      {
        current--;
        Correr(a, n, current, l);
        v = a[current];
          if(hasNext(n, n - 1, l, v))
          {
                CListaSE<Combinacion*>* temp = combinar(lista,a,n);

            for(int i = 1; i <= temp->Longitud(); i++)
                ret->Adicionar(temp->Obtener(i));

            //delete temp;

            Correr(a, n, n-1, l);
      }
    }

    return ret;
}
```

Fig.3.8. Método para formar las combinaciones con los elementos frecuentes.

Picar**:**

Este método toma las combinaciones y las divide formando parte izquierda y derecha de la regla, o sea, parte implicativa e implicada de la regla.

```
CListaSE<Regla*>* Picar(Combinacion* lista, int* a, int n)
{
      int current = n-1;
      int v = a[current];
      int l = lista->CantCombinaciones();
      CListaSE<Regla*> *ret = new CListaSE<Regla*>();

      while(hasNext(n, current+1, l, v))
      {
            //String nueva;
            Regla* nueva=new Regla();

            for(int i = 0; i < n; i++)
            nueva->AdicionarImplicativa(lista->Valor(a[i]));

            for(int i = 1; i <= l; i++)
            {
               bool est = true;
               for(int j = 0; j < n; j++)
               {
               if(i == a[j])
               est = false;
               }

               if(est)
               nueva->AdicionarImplicada(lista->Valor(i));
            }
            ret->Adicionar(nueva);
```

```
            Correr(a, n, current, l);

            v = a[current];
        }

        while(current >= 0)
        {
          current--;
          Correr(a, n, current, l);
          v = a[current];
            if(hasNext(n, n - 1, l, v))
            {
                  CListaSE<Regla*>* temp = Picar(lista,a,n);

                  for(int i = 1; i <= temp->Longitud(); i++)
                      ret->Adicionar(temp->Obtener(i));

                  //delete temp;

                  Correr(a, n, n-1, l);
            }
        }

        return ret;
}
```

Fig.3.9. Método que divide las combinaciones para formar reglas a partir de ellas.

FastGenAllRepresentatives**:**

Método que elimina las reglas que no cumple con las restricciones representativas: Este método tomas las reglas de asociación formadas a partir de las combinaciones generadas y las filtra por las restricciones de soporte y confianza eliminando las que no cumplan con dichas restricciones.

```
void FastGenAllRepresentatives(CListaSE<Regla*>* reglas){

for(int i = 1; i <= reglas->Longitud(); i++)
{

double sop = soporte(reglas->Obtener(i),false);
double conf = confianza(reglas->Obtener(i));
  if(sop > minsup || conf > minconf)
  {
  reglas->Eliminar(i);
  i--;
  }
}
}
```

Fig.3.10. Método que filtra las reglas dejando solo las que cumplan con las restricciones.

3.4.3 Resultados obtenidos con el algoritmo.

En este epígrafe se muestran algunos resultados obtenidos con la herramienta implementada. La siguiente figura muestra los resultados obtenidos a partir de una consulta entre dos tablas en una BD. En este caso, el soporte especificado por el usuario es 3, con un soporte y confianza para la regla del 20% cada uno.

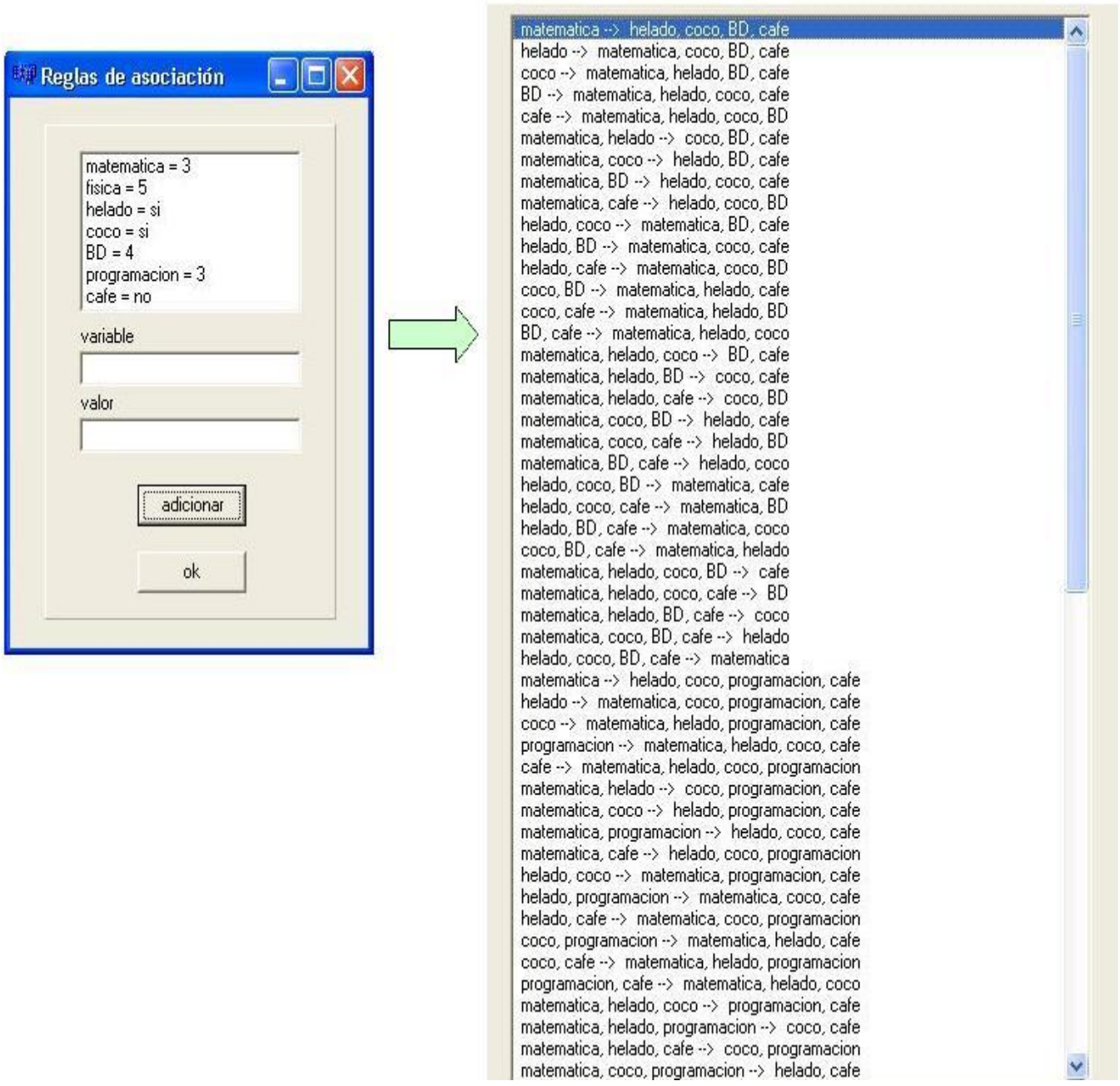


Fig.3.11. Reglas extraídas a partir de una consulta con los juegos de datos mostrados con sus valores.

La figura 3.12 muestra los resultados obtenidos a partir de otra consulta, a partir de un soporte especificado por el usuario de 2, con un soporte y confianza para la regla de un 20% y un 10% respectivamente.

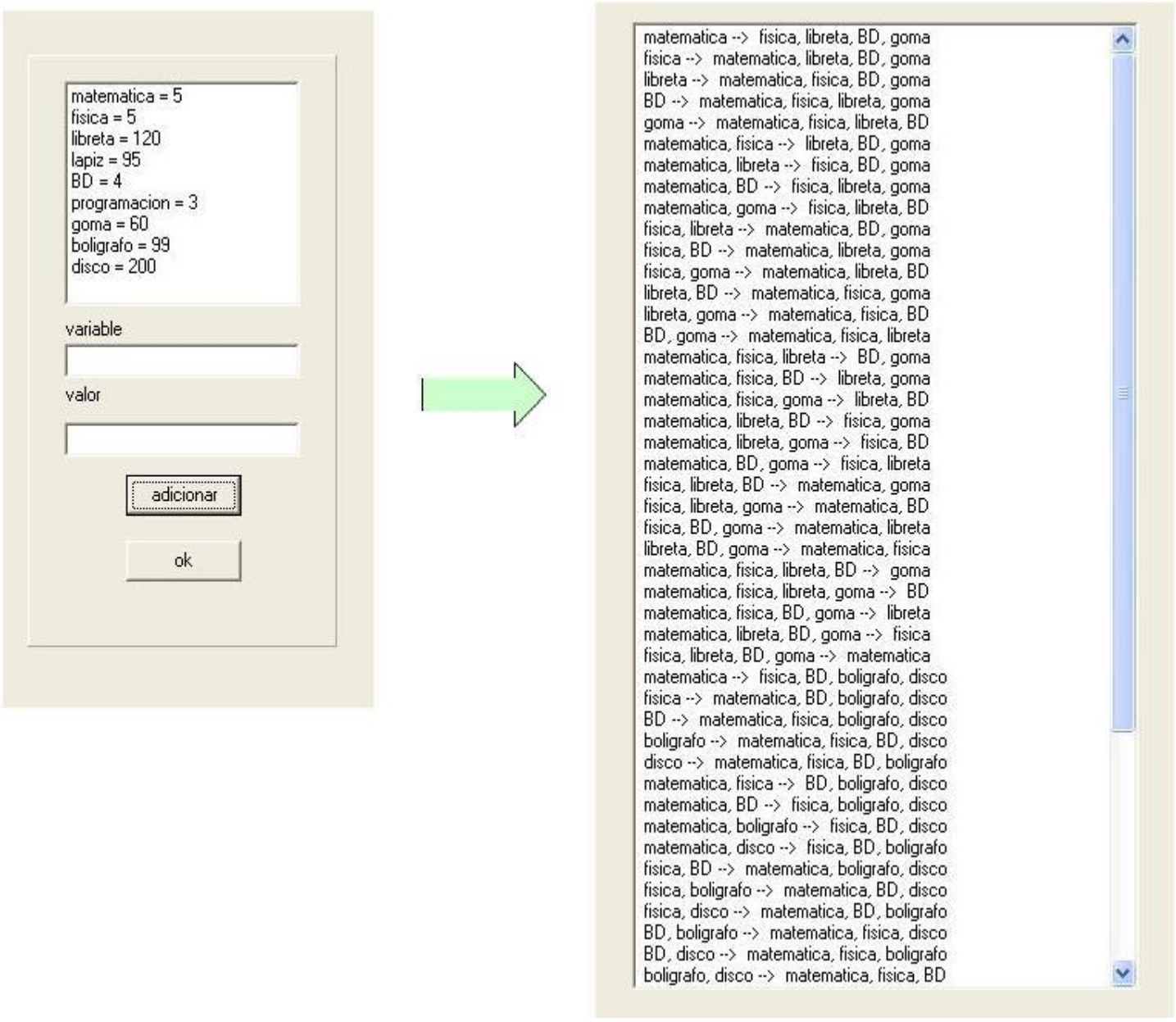


Fig.3.12. Reglas extraídas a partir de una consulta con los juegos de datos mostrados con sus valores.

La figura 3.13 muestra los resultados obtenidos a partir de una tabla, a partir de un soporte especificado por el usuario de 2, con un soporte y confianza para la regla de un 20% y un 30% respectivamente.

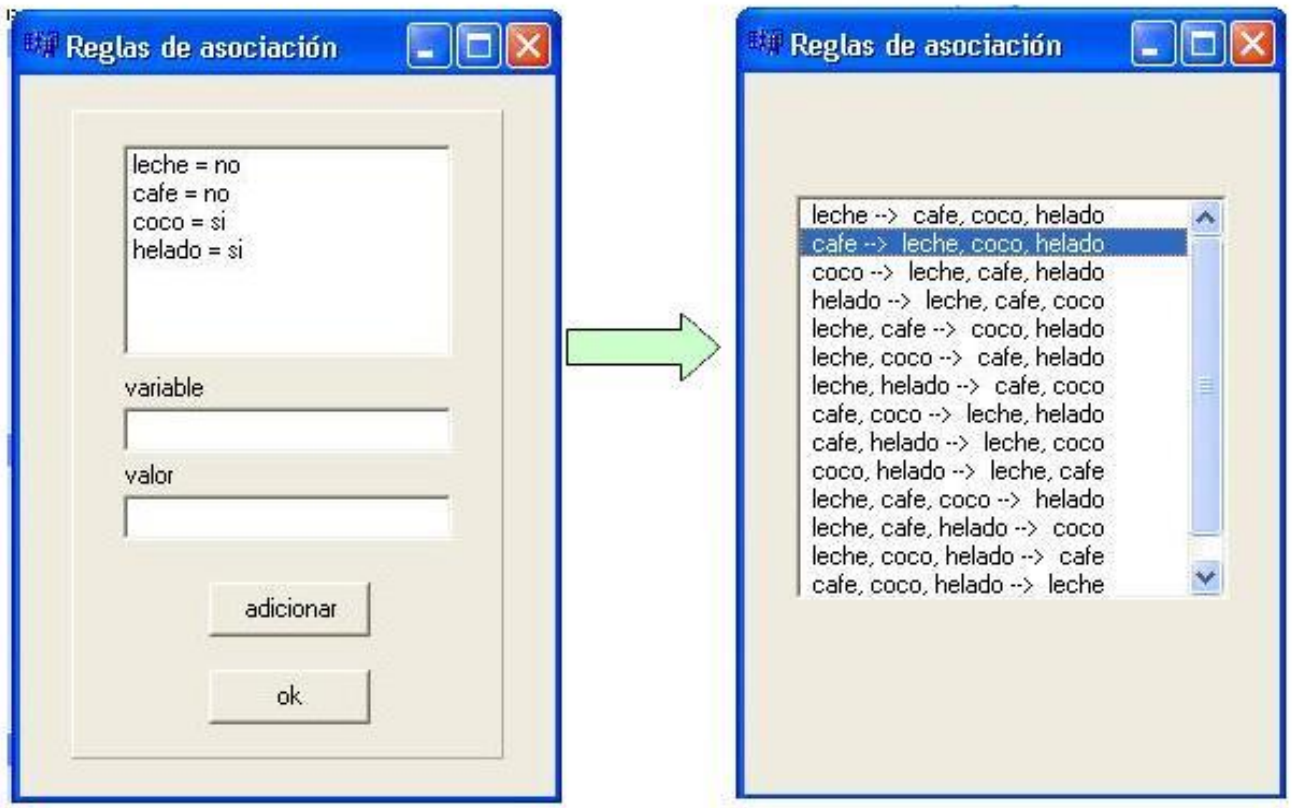


Fig.3.13. Reglas extraídas a partir de una tabla con los juegos de datos mostrados con sus valores.

3.4.3.1 Validación de los resultados.

Para validar los resultados obtenidos con la herramienta, es necesario comparar con otros resultados obtenidos a través de herramientas similares, utilizando como entrada el mismo juego de datos, para ambas herramientas, y comparar el por ciento de similitud a partir de valores de confianza y soporte iguales.

Actualmente, se han realizado varias implementaciones en el tema del minado de datos, donde uno de los exponentes del tema asociado con nuestro país es Frank Coenen, donde en conjunto con el Cenatav realizan investigaciones de este tipo. Por este motivo, se escoge

la herramienta como punto de comparación para validar los resultados. El algoritmo empleado por Coenen es una mejora del A priori, el A prioriT, que mejora considerablemente algunos aspectos, como por ejemplo, el tiempo de ejecución.

Coenen verifica su herramienta con un juego de datos de 768 transacciones, de 9 columnas. La figura a continuación presenta una muestra de las transacciones, aunque para la herramienta se comprueba con todas.

1	8	13	17	21	27	31	36	41
2	9	13	17	21	28	32	38	42
1	8	13	17	21	28	31	36	41
2	8	14	16	21	27	31	36	41
3	10	13	16	21	27	32	36	42
3	8	11	16	21	28	31	36	41
2	8	14	16	21	28	31	36	41
1	9	12	17	21	29	35	37	42
3	9	14	16	21	28	33	39	41
1	8	12	17	21	29	31	37	41
1	7	13	17	21	28	31	36	42
1	9	14	18	22	28	32	36	41
3	8	14	16	21	28	31	38	41
1	10	13	18	24	28	31	38	42
1	8	13	16	21	27	31	36	41
4	9	14	16	21	27	31	39	41
3	9	14	16	21	26	31	38	42
2	8	14	16	21	28	31	37	41
3	10	14	16	21	28	31	37	42
2	8	14	17	21	28	31	39	41
1	8	13	16	21	27	31	36	41
1	10	13	17	25	28	31	39	42
2	8	14	16	21	27	31	36	41
3	9	14	17	21	28	31	38	41
2	10	13	16	22	27	32	38	42

Fig.3.14. Muestra de elementos para la validación.

A partir de estos datos, y con un 20% y 80% de soporte y confianza respectivamente, Coenen obtiene 132 reglas, de las cuales se muestra una parte en la siguiente figura:

Fig.3.15. Reglas generadas por Coenen a partir del juego de datos anterior.

```
{1 13 41}  ->  {36}  94.47%
{1 31 41}  ->  {36}  91.36%
{1 21 31 41}  ->  {36}  90.76%
{1 13 36 41}  ->  {21}  90.05%
{1 8 21}  ->  {41}  89.9%
{31 41}  ->  {21}  89.89%
{1 41}  ->  {36}  89.06%
{31 36 41}  ->  {21}  88.75%
{1 21 41}  ->  {36}  88.51%
{1 31 36 41}  ->  {21}  88.05%
{36 41}  ->  {21}  87.24%
{1 13 36}  ->  {21}  86.95%
{1 36 41}  ->  {21}  86.28%
{1 13 21 36}  ->  {41}  85.55%
{1 13 41}  ->  {21 36}  85.08%
{36}  ->  {21}  84.24%
{1 21 31 36}  ->  {41}  83.88%
{1 31 36}  ->  {41}  82.71%
{13 21 36 41}  ->  {1}  82.35%
{1 21 36}  ->  {41}  81.29%
{21 31 36}  ->  {41}  80.91%
{1 31 41}  ->  {21 36}  80.45%
{8 31}  ->  {21 41}  80.0%
```

A partir de los mismos datos, y con los mismos valores de soporte y confianza, la herramienta proporcionada a partir de este trabajo arrojó 17 reglas, las cuales están contenidas en los resultados de Coenen, teniendo en cuenta los valores de las variables:

a = 1, b = 8, c = 13, d = 17, e = 21, f = 29, g = 31, h = 36, i = 41

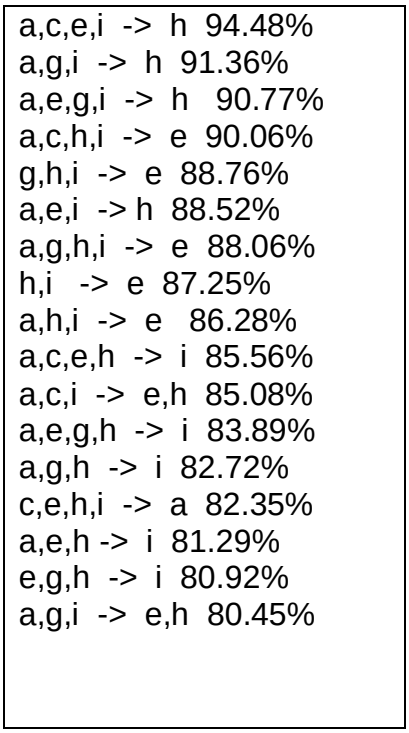


Fig.3.16. Reglas generadas por la herramienta a partir del juego de datos anteriores.

La siguiente figura muestra gráficamente los resultados obtenidos mediante los dos procedimientos, obteniendo un 12.88% de similitud en los resultados.

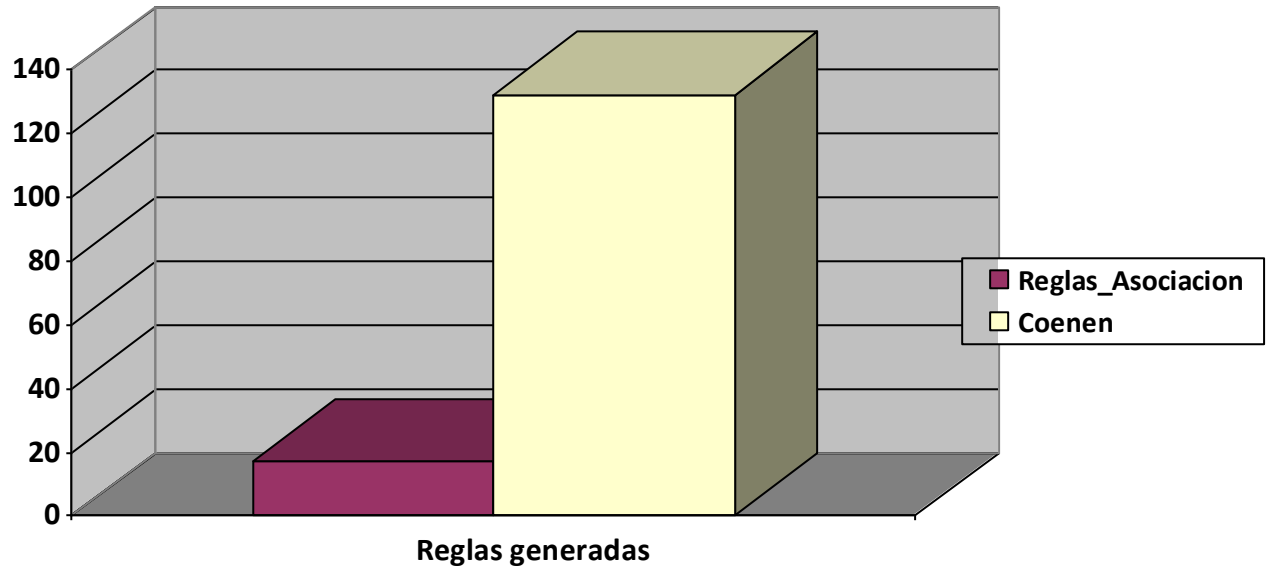


Fig.3.17. Representación gráfica de la cantidad de reglas obtenidas mediante los dos procedimientos de los resultados obtenidos.

La gran diferencia entre la cantidad de reglas obtenidas a través de los dos procedimientos, es que los algoritmos no son iguales, mientras A priori va desechando en cada pasada a la BD los elementos no frecuentes, A prioriT accede a la BD solo una vez, formando un árbol con todos aquellos elementos encontrados como frecuentes, o sea, A priori desecha elementos que A prioriT utiliza para la generación del árbol y consecuentemente, de las reglas de asociación.

3.5 Conclusiones del capítulo.

En la realización de este capítulo se estableció una arquitectura básica de la biblioteca para su posterior implementación:

- ✓ Se estudiaron las distintas herramientas y metodologías implicadas en el diseño e implementación de la biblioteca.
- ✓ Se expusieron los diferentes módulos diseñados para la creación de la biblioteca.
- ✓ Se mostraron y explicaron los métodos más importantes para el buen funcionamiento de la herramienta propuesta a partir de la biblioteca implementada.

- ✓ Se mostraron los diferentes resultados obtenido con la nueva herramienta propuesta.

Los resultados obtenidos a partir de dicha herramienta muestran las reglas de asociación que cumplen con las condiciones para ser representativas, aunque en ocasiones la calidad de las reglas obtenidas dé la medida en que las tablas a evaluar estén relacionadas, obteniéndose un 12.88% de similitud en la validación de los resultados.

CONCLUSIONES

Como resultado de la investigación, se puede concluir que el trabajo permitió recopilar los algoritmos más importantes que aparecen en la literatura sobre el tema de las reglas de asociación en bases de datos, incluyendo la metodología general de trabajo para el descubrimiento de tales conocimientos, los algoritmos para el cálculo de conjuntos frecuentes y los conceptos relacionados con las generalidades del minado de reglas, así como el empleo de taxonomías para reducir y mejorar la interpretación de las reglas encontradas. Paralelo a esto, también se concluyó que:

- ✓ Las reglas de asociación han demostrado ser un mecanismo viable para el descubrimiento de conocimientos implícitos encontrados en grandes volúmenes de información.
- ✓ Los algoritmos para obtener los elementos frecuentes en una base de datos, son una parte importante en el minado de las reglas, ofreciéndoles los elementos más importantes en la base de datos.
- ✓ Los algoritmos implementados cumplen con el propósito para el cual fueron implementados, dado que las pruebas realizadas a la herramienta lo confirman.

Ofreciendo el trabajo en general, la posibilidad a la Universidad, contar con una herramienta que le permita generar reglas de asociación, dándole cumplimiento al objetivo inicialmente planteado como resultado de la investigación.

RECOMENDACIONES

Posibles trabajos futuros estarían orientados a:

- ✓ Considerar las desventajas mencionadas en el desarrollo del trabajo con el objetivo de diseñar nuevos algoritmos para la generación de reglas de asociación en bases de datos, brindando especial atención al cálculo de reglas de asociación con soportes bajos y altas confidencias.
- ✓ Implementar los restantes algoritmos estudiados para comparar resultados.
- ✓ Realizar más pruebas a la herramienta en BD mucho más grandes.
- ✓ Realizar la migración a software libre por la tendencia que tiene la Universidad.

REFERENCIAS BIBLIOGRÁFICAS

Agrawal, R. 1993. *Mining Association Rules between Sets of Ítems in Large DataBases.* Washington, DC : s.n., 1993.

Agrawal, R, y otros. 1996. *Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining.* Menlo Park : s.n., 1996.

Amir, A. 1997. *A new and versatile Method for associations generation.Principles of Data Mining and knowledge discovery.* 1997.

Boulicaut, J y Bykowski, A. 2000. *Frequent closures as a concise representation for binary data mining.* Kyoto : s.n., 2000.

Brin, S, y otros. 1997. *Dynamic Itemset Counting and Implication Rules for Market Basket Data.* 1997.

Fayyad, U. 1995. *Knowledge discovery and Data Mining: Towards and Unifying Framework.* Oregon : s.n., 1995.

Han, J. 2001. *Mining frequent itemsets with convertible constraints.* Heidelberg, Germany : s.n., 2001.

Hangerup, T. 1990. *A guided tour of Chernoff bounds. Information processing letters.* 1990.

Hipp, J. 2000. *Algorithms for association rule mining - A general survey and comparison. .* 2000.

Holt, J. 1999. *Efficient Mining of Association Rules in Text Databases.* 1999.

Krys, M. 1998. *Representative Association Rules and Minimum Condition Maximum Consequence Association Rules. In proceedings of Principles of Data Mining and knowledge Discovery.* France : s.n., 1998.

Kryszkiewicz, M. 2001. *Closed Set based Discovery of Representative Association Rules.* Springer : s.n., 2001.

Mannila, H. 1995. *Discovering frecuent episodes in sequences. Technical Report.* Finland : s.n., 1995.

Park, J. 1997. *Using a hash based method with transaction for Mining Association Rules.* 1997.

Pasquier, N. 2000. *Algorithmes d'extraction et de r´eduction des regles d'association.* 2000.

Piatetsky, G. 1991. *Discovery, analysis and presentation of strong rules. Knowledge Discovery in Databases.* 1991.

Rougel, A. 1998. *Preprocessing of Missing Values using Robust Association Rules. In proceedings of Principles of Data Mining and knowledge Discovery.* France : s.n., 1998.

Saquer, J y Deogun, S. 2000. *Using closed itemsets for discovering.* Charlotte : s.n., 2000.

Savasare, A y Navathe, S. 1999. *An efficient Algorith for Mining Associatio Rules in Large Databases.* Atlanta : s.n., 1999.

Srikant, R. 1995. *Mining Generalized Association Rules.* Zürich : s.n., 1995.

Tirso, A. 2002. *Homo Cybersapiens. La Inteligencia artificial y la humana. Madrid: s.n., 2002*

Walczak, Z. 1998. *Selected problems and algorithms of data Mining.* 1998.

Zaki, M. 2000. *Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering.* California : s.n., 2000.

ANEXOS

Anexo 1: Algoritmos para la generación de conjuntos de ítems frecuentes

Algoritmo Apriori

Entrada: D: transacciones de la base de datos
MinSop: mínimo soporte deseado
Salida: F: Conjunto de los conjuntos de itemsets con soporte mayor que MinSop
Método:

F1 = {itemsets unitarios frecuentes}
k = 2
Mientras Fk-1 $\neq \varnothing$ hacer
Ck = AprioriGen(Fk-1)
Para todas las transacciones T $\in$ D hacer
Para todos los candidatos Z $\in$ Ck hacer
Si Z $\subseteq$ T entonces Z.count = Z.count +1
Fk = {Z $\in$ Ck | Z.count > MinSop }
k++
Devolver F = $\cup$k Fk

Utiliza las siguientes notaciones: Ck, para el conjunto de candidatos de k-itemsets, Fk, para el conjunto frecuente de k-itemsets y asociado a cada itemset se encuentra el campo count para almacenar el soporte de dicho itemset.

Algoritmo AprioriGen

Entrada: Fk-1: Conjunto frecuente (k-1)-itemsets
Salida: Ck: Conjunto candidato k-itemsets
Método:

Ck = $\varnothing$
Insertar en Ck, todos los conjuntos ordenados de ítems, estos son:
{ Z[1], ..., Z[k-1],Y[k-1] | Y, Z $\in$ Fk-1, Z[1] = Y[1], ..., Z[k-2] = Y[k-2] $\wedge$ Z[k-1]<Y[k-1] }
Borrar todos los Z $\in$ Ck, tal que algún (k-1)-subconjunto de Z no esté en Fk-1
Return Ck

Algoritmo DHP

Entrada: D: copia de la base de datos original
MinSop: Soporte mínimo especificado por el usuario
Salida: F: Conjunto de itemsets con soporte mayor que MinSop
Método:

```
F1 = ∅
Para cada transacción t ∈ D hacer
    Para cada ítem x ∈ t hacer
        x.count ++
    Para cada 2-itemset y ⊆ t hacer
        H2.Inc(y)   // Incrementa el valor de la ocurrencia del itemset en la
            // tabla hash H2
Para cada ítem x hacer
    Si x.count /nt ≥ MinSop entonces
        F1 = F1 ∪ { {x} }

H2.Eliminar(MinSop) // Eliminar todos los conjuntos contenidos en las entradas
            // hash de la tabla que no cumplan con el valor del soporte
            // mínimo
k = 2
Mientras Fk-1 ≠ ∅ hacer
    Ck = ∅
    Sea JFk la unión natural entre Fk-1 y Fk-1 siendo iguales los primeros k-2 ítems          //
Realizar la unión natural entre conjuntos frecuentes de k-1 elementos
        // que tienen en común k-2 elementos de éstos
    Para cada x ∈ JFk hacer
        Si Hk.TieneSoporte(x) entonces
            Ck = Ck ∪ {x}   // Coloco como candidatos los que no
                    //fueron eliminados de la tabla hash
    Para cada transacción t ∈ D hacer
        Para cada k-itemset x ⊆ t hacer
            Si x ∈ Ck entonces
                x.count++
        Para cada (k+1)-itemset y ⊆ t hacer
            Si (¬∃z / z = k-subconjunto de y) ∧
               (¬Hk.TieneSoporte(z)) entonces
                Hk+1.Inc(y)
    Fk = ∅
    Para cada x ∈ Ck hacer
        Si x.count / nt ≥ MinSop entonces Fk = Fk ∪ {x}
Hk.Eliminar(MinSop)
    k++
Devolver F = ∪k Fk
```

En este algoritmo se emplean las siguientes notaciones: nt es el cardinal de la base de datos inicial; Fk el conjunto que aglutina los conjuntos de ítems frecuentes de k elementos; Ck es el conjunto de candidatos de k elementos; Hk es la tabla hash para contar las ocurrencias de los conjuntos de (k+1) ítems en las transacciones.

Algoritmo Partition

Entrada: P ={p1, p2, ..., pm}, conjunto de particiones de la base de datos D,
m=$|P|$ es el número de particiones
MinSop: Soporte mínimo especificado por el usuario
Salida: F: Conjunto de los conjuntos de itemsets con soporte mayor que MinSop
Método:

```
// Fase 1
i = 1
Mientras i ≤ m hacer
    Leer una partición pi ∈ P
    Fi = GenItemsets( pi, MinSop)
                // Fi está formado por F_i^j, conjuntos frecuentes de la
                // partición i y tamaño j
// Mezcla
C = ∪ F_i^j  (i=1..m)
                // Mantener en C todos los conjuntos que resultaron
                // frecuentes para al menos una de las particiones
// Fase 2
i = 1
Mientras i ≤ m hacer
    Leer una partición pi ∈ P
    AcumularSoporte(C, pi)
Insertar en F todos los conjuntos c tales que c.count ≥ MinSop
Devolver F
```

Como se explicaba anteriormente, este algoritmo en la primera fase lee cada partición y calcula sus conjuntos frecuentes, mediante llamadas al algoritmo GetItemsets que toma una partición y el soporte mínimo requerido y devuelve todos los conjuntos frecuentes de la partición, mezcla estos conjuntos, formando el conjunto de candidatos C. En la segunda fase, se vuelve a leer cada partición, y se va acumulando para cada conjunto candidato el soporte en cada partición, mediante un llamado al procedimiento Acumular_Soporte.

Algoritmo AcumularSoporte

Entrada: C: Conjunto de candidatos globales
pi: partición de la base de datos
Salida: Se ha adicionado a cada candidato global el soporte correspondiente a la partición pi
Método:

```
Para cada 1-itemset hacer
    Generar su tidlist en la partición pi
Para cada c ∈ C hacer
    TempList = ∩ c[j].tidlist  (j=1..|c|)
    c.count = c.count + |TempList|
```

Algoritmo GenItemsets

Entrada: pi : partición de la base de datos
MinSop: Soporte mínimo especificado por el usuario
Salida: Fi : Conjunto de itemsets frecuentes de la partición pi
Método:

F_i^1 = {1- itemsets frecuentes en la partición pi junto con sus tidlist}
k = 2
Mientras $F_i^{k-1} \neq \varnothing$ hacer
Para todos los itemsets f1 $\in$ F_i^{k-1} hacer
Para todos los itemsets f2 $\in$ F_i^{k-1} hacer
// Para cada par de (k-1) itemsets en F_i^{k-1} hacer
Si f1[1] = f2[1] $\wedge$...$\wedge$ f1[k-2] = f2[k-2] $\wedge$ f1[k-1] $<$ f2[k-1] hacer
f = { f1[1] , ..., f1[k-2], f1[k-1], f2[k-1] }
// f es un k- itemset formado a partir de la pareja f1 y f2
Si todos los (k-1)-subconjunto de f están en F_i^{k-1} hacer
// f cumple la condición de mínima de frecuencia:
// todos sus subconjuntos son frecuentes
f.tidlist = f1.tidlist $\cap$ f2.tidlist
Si $|$f.tidlist$|$ $\geq$ MinSop hacer
$F_i^k = F_i^k \cup \{f\}$
Devolver Fi = $\cup$k F_i^k

Anexos 2: Algoritmos de generación para RA en BD.

Algoritmo FastGenAllRepresentatives

Entrada: F: Todos los conjuntos de itemsets frecuentes, formado por todos los F_k, determinados utilizando el algoritmo Apriori
MinConf: Mínima confidencia especificada por el usuario
Salida: RAR: Conjunto de reglas de asociación representativas
Método:

```
RAR = ∅
Para todo conjunto itemsets frecuente Z ∈ Fk, k ≥ 2 do
    Maxsup = max( { Z'.soporte /  Z⊂ Z' ∈ Fk+1} ∪ {0} )
    Si Z.soporte ≠ MaxSup entonces
        A1 = { {Z[1]}, {Z[2]}, ...,{Z[k]} } // creando los 1-antecedentes
                i = 1
        Mientras (Ai ≠ ∅) y (i<k) hacer // ciclo 1
            // Buscando los subconjuntos de ítems anteriores, para formar todas
            // las reglas posibles con menos de k antecedentes
            Para todo X∈ Ai hacer
                // Buscando consecuente para cada antecedente
                Encontrar Y∈ Fi tal que Y = X
                X.soporte = Y.soporte
                Si  Z.soporte / X.soporte ≥ MinConf entonces
                                        // Si X ⇒ Z\X es una regla de asociación
                    Si MaxSup/X.soporte < MinConf entonces
                        // Si no hay ninguna RA X ⇒ Z'\X que cubra a X ⇒ Z\X
                        RAR = RAR ∪{X⇒Z\X}
                        Ai = Ai \ {X}   // Los ítems que son antecedentes de
                                        // alguna regla ya no deben volver a emplearse
            Ai+1 = AprioriGen(Ai) // Extender los antecedentes de RA
            i = i +1
```

Algoritmo Básico

Entrada: D: base de datos de transacciones
T: Taxonomía en forma de un grafo dirigido acíclico
Salida: F: Conjunto de itemsets frecuentes
Método:
F1 = {conjuntos frecuentes de 1 elemento} // Recordemos aquí están
// elementos de cualquier nivel de la taxonomía
k = 2
Mientras (Fk-1 $\neq \varnothing$) hacer
Ck = Nuevos candidatos de tamaño k generados a partir de Fk-1
Para todas las transacciones t $\in$ D hacer
Hacer t' = t
Añadir todos los ancestros de cada ítem en t' a t', eliminando cualquier duplicación
Incrementar el contador de todos lo candidatos en Ck que estén contenidos en t'
Fk = Todos los candidatos en Ck con soporte mínimo
k = k + 1
Devolver F = $\cup$k Fk

Algortitmo Cumulate

Entrada: D: base de datos de transacciones
T: Taxonomía en forma de un grafo dirigido acíclico
Salida: F: Conjunto de itemsets frecuentes
Método:
//optimización 2
T* = el conjunto de ancestros de cada ítem calculados a partir T
F1 = {conjuntos frecuentes de 1 elemento}
k = 2
Mientras (Fk-1 $\neq \varnothing$) hacer
Ck = Nuevos candidatos de tamaño k generados a partir de Fk-1
Si (k = 2) // optimización 3
Borrar todos los candidatos en C2 que contengan un ítem y su ancestro
Borrar todos los ancestros en T* que no estén presentes en ningún candidato en Ck // optimización 1
Para todas las transacciones t $\in$ D hacer
Para cada ítem x $\in$ t hacer
Hacer t' = t
Añadir todos los ancestros de x en T* a t', eliminando cualquier duplicación
Incrementar el contador de todos los candidatos en Ck que estén contenidos en t'
Fk = Todos los candidatos en Ck con soporte mínimo
k = k + 1
Devolver F = $\cup$k Fk

Algoritmo EstMerge

Entrada: D: base de datos de transacciones
T: Taxonomía en forma de un grafo dirigido acíclico
Salida: F: Conjunto de itemsets frecuentes
Método:

// optimización 2
T* = el conjunto de ancestros de cada ítem calculados a partir T
F1 = {conjuntos frecuentes de 1 elemento}
Generar Dm, una muestra de la base de datos
k = 2
$C''_1 = \varnothing$
Mientras (Fk-1 $\neq \varnothing$ ó $C''_1 \neq \varnothing$) hacer
Ck = Nuevos candidatos de tamaño k generados a partir de Fk-1 y de C''_{k-1}
Si (k = 2) // optimización 3
Borrar todos los candidatos en C2 que contengan un ítem y su ancestro
Borrar todos los ancestros en T* que no estén presentes en ningún candidato en Ck // optimización 1
Estimar el soporte de los candidatos en Ck haciendo una pasada a través de Dm
C'_k = Candidatos en Ck que se espera que ellos o sus padres tengan mínimo soporte
Encontrar el soporte de los candidatos en $C'_k \cup C''_{k-1}$ haciendo una pasada a través de D
Borrar todos los candidatos en Ck cuyos ancestros en C'_k no tengan mínimo soporte
C''_{k-1} = Ck \ C'_k
Fk = Todos los candidatos en C'_k con soporte mínimo
Adicionar a Fk-1 todos los candidatos en C''_{k-1} con soporte mínimo
k = k + 1
Devolver F = $\cup$k Lk

Algoritmo Prutax

Entrada: F1, ..., Fn: conjuntos de ítems frecuentes
Salida: F: conjuntos candidatos frecuentes generados a partir de F1, ..., Fn,
Método:

```
Para i = n –1 hasta 1 hacer
    E.Limpiar()
    Para j = n hasta i + 1 hacer
        C = Hi ∪ Hj
        Si no ocurre que ( |C| = 2 ∧ C.ítem[2]∈ancestros(C.ítem[1]) ) entonces
            Si no ocurre que ( |C| ≠ 2 ∧ ∃S, S⊆C, ( |S| = |C| -1 : S∉ F ) entonces
                Si no ocurre que (∃ Ĉ : Ĉ ⊆O, Ĉ es ancestro de C: Ĉ ∉ F) entonces
                    C.tids = Hi.tids ∩ Hi.tids
                    Si |C.tids| ≥ min_soporte entonces
                        Añadir C a E
                        F = F ∪ {C}
Prutax(E); // Para garantizar la búsqueda en profundidad
```

Algoritmo FastGenAllRepresentatives

Entrada: F: Todos los conjuntos de itemsets frecuentes, formado por todos los Fk, determinados utilizando el algoritmo Apriori
MinConf: Mínima confidencia especificada por el usuario
Salida: RAR: Conjunto de reglas de asociación representativas
Método:

```
RAR = ∅
Para todo conjunto itemsets frecuente Z ∈ Fk, k ≥ 2 do
    Maxsup = max( { Z'.soporte / Z⊂ Z' ∈ Fk+1} ∪ {0} )
    Si Z.soporte ≠ MaxSup entonces
        A1 = { {Z[1]}, {Z[2]}, ...,{Z[k]} } // creando los 1-antecedentes
        i = 1
        Mientras (Ai ≠ ∅) y (i<k) hacer // ciclo 1
            // Buscando los subconjuntos de ítems anteriores, para formar todas
            // las reglas posibles con menos de k antecedentes
            Para todo X∈ Ai hacer
                // Buscando consecuente para cada antecedente
                Encontrar Y∈ Fi tal que Y = X
                X.soporte = Y.soporte
                Si Z.soporte / X.soporte ≥ MinConf entonces
                        // Si X ⇒ Z\X es una regla de asociación
                    Si MaxSup/X.soporte < MinConf entonces
                    // Si no hay ninguna RA X ⇒ Z'\X que cubra a X ⇒ Z\X
                        RAR = RAR ∪{X⇒Z\X}
                    Ai = Ai \ {X}  // Los ítems que son antecedentes de
                        // alguna regla ya no deben volver a emplearse
                    Ai+1 = AprioriGen(Ai) // Extender los antecedentes de RA
                    i = i +1
```

Anexo 3: Algoritmos de RA en colecciones de textos

Algoritmo FACT

Entrada: O: Conjunto de ítems
Ds: Colección de documentos
Restizq: Restricciones de la parte izquierda de las reglas
Restder: Restricciones de la parte derecha de las reglas
Rest: otras restricciones expresadas en la demanda
MinSop: Soporte mínimo especificado por el usuario
MinConf: Confidencia mínima especificada por el usuario
Salida: R: Conjunto de reglas de asociación minadas
Método:
Cand = BusqCandidatos(O, Resizq)
Para todo D $\in$ Ds hacer
Para todo X $\in$ Cand hacer
Si X $\subseteq$ K(D) entonces
B = Las palabras claves en K(D)\X que satisfacen los requerimientos en Restder y Rest y aparecen con el soporte requerido
// cada vez que aparece un conjunto B para X, se incrementa el
// contador de soporte para X y cada uno de los subconjuntos de B
Actualizar los contadores para X y todos los subconjuntos de B
Formar las asociaciones basadas en los contadores de co-ocurrencias acumuladas
Hacer R igual al conjunto anterior eliminando aquellas asociaciones que no cumplan con el soporte y la confidencia requeridas.
Devolver R

Algoritmo BusqCandidatos

Entrada: O: Conjunto de ítems a emplear en la demanda
MinSop: soporte
Restizq: Restricciones de la parte izquierda de las reglas
Salida: Cand: Conjunto de todos los ítemssets candidatos (conjunto de ítemssets a los que con soporte mayor o igual a MinSop)
Método:
Cand1 = { {A} / A$\in$ O, A.soporte $\geq$ MinSop, A satisface las restricciones Restizq}
k= 1
Mientras Candk $\neq \varnothing$ hacer
Candk+1 = { S1 $\cup$ S2 / S1, S2 $\in$ Candi, |S1 $\cup$ S2| = k + 1, $\forall$ S$\subseteq$S1$\cup$S2, |S|=k, S$\in$ Candk $\wedge$ S1 $\cup$ S2 satisface las restricciones Restizq}
k =k+1
Cand = $\bigcup_k Cand_k$
Devolver Cand

Algoritmo DescubrimientoFrasesFrecuentesMaximales

```
Entrada:    Ds: Conjunto de documentos
            MinSop1: Mínima cantidad de documentos en los que debe aparecer una
                     frase
Salida:     Fmax: Conjunto de frases frecuentes maximales
Método:
    // Fase Inicial
    Para todos los documentos d ∈ Ds hacer
        Coleccionar todos los pares ordenados dentro de d, donde la distancia
            mínima entre palabras del par es 2; de cada par se conoce también
            todas sus ocurrencias en cada documento
    C2 = Conjunto de pares ordenados frecuentes en Ds
    // Fase de descubrimiento: construcción de frases más largas por expansión y unión
    k = 2
    Fmax = ∅
    Mientras Ck ≠ ∅ hacer
        Para todas las frases candidatas f ∈ Ck  hacer
            Si f no es subfrase de alguna frase f' ∈ FMax entonces
                Si f es frecuente entonces
                    max = Expand(f) // Expandir f tanto como sea
                        // posible insertando cada vez una palabra
                        // en cualquier lugar (por delante, al final
                        // o en el medio) y verificando sus ocurrencias en
                        // Ds, eliminando aquellas subfrases de alguna
                        // frase maximal ya obtenida
                    Fmax = Fmax ∪ {max}
                    Si max = f
                        Borrar f de Ck // ya es maximal
                Sino
                    Borrar f de Ck     // por propiedad de los conjuntos
                                       // frecuentes
        Podar(Ck)    // Borrar de Ck todos los candidatos que no fueron
                     // subconjunto de alguna frase frecuente descubierta en esta
                     // iteración.
        Ck+1 = Join(Ck)   // Formar los conjuntos candidatos frecuentes de k+1
                          // elementos a partir de la unión de pares de k elementos
        k = k +1
    Devolver FMax
```

Algoritmo TEXTRISE

Entrada: E: Conjunto de entrenamiento, ejemplos o documentos
Salida: R: Conjunto de reglas generalizadas
Método:
R = E
Repetir
Para cada regla r ∈ R hacer
e´ = e tal que e∈E y $\max_{e \in E'} Similitud(e,r) = Similitud(e',r)$, con
E´= {e´/ e´∈ E, e´ no está cubierta por R}
r´ = GeneralizMásEspecifica(r, e´)
R´ = R reemplazando r por r´
Si AproxReglasTextos(R´,T) ≥ AproxReglasTextos(R,T) entonces
R = R´
Si r´ es igual a otra regla en R entonces Borrar r´ de R
Hasta que no mejore AproxReglasTextos(R,T)
Devolver R

Algoritmo GeneralizMásEspecifica

Entrada: r: regla a generalizar
e: ejemplo con el que se generaliza
Salida: r´: regla generalizada
Método:
Para cada i = 1 hasta n hacer
$A_i' = A_i \cap e_i$
r´ = ($A_1', ..., A_n', C_r \cap C_e$)
Devolver r´

GLOSARIO DE TÉRMINOS

Clasificación: permite la identificación de una función que asocia un dato en una de varias clases predefinidas.

Clustering: identifica un conjunto de categorías o clases para describir los datos.

Confidencia: % de ejemplos que una regla cubre completamente. % de aciertos de la regla.

Conjunto frecuente: representan conjuntos de elementos que están correlacionados positivamente

Datos: no es más que un número, un nombre, una dirección, una cualidad, pero también se debe entender una imagen, un símbolo, una señal de radio, un electrocardiograma, un documento, un libro, una función, una matriz, un tensor, etc., en fin, una información sobre algo.

Detección de cambios y desviaciones: descubren la mayoría de los cambios significativos en los datos desde medidas previas o valores usuales.

Item: atributo=valor, usualmente los atributos son convertidos a binario por cada valor, elemento individual.

Itemset: conjunto de posibles atributos o valores (items).

Modelo de dependencias: permite encontrar un modelo para describir las dependencias significativas entre las variables.

Reglas de asociación (RA): se basan en medidas de confianza y soporte, consideran cualquier conjunto de atributos con cualquier otro conjunto de atributos, funcionan con atributos discretos.

Regresión: intenta el aprendizaje de una función que asocia un dato para conocer el valor real de las variables de predicción y descubre las relaciones funcionales entre las variables.

Resumen: encuentra una descripción compacta para un subconjunto de datos.

Soporte de un conjunto I: % de ejemplos que la regla cubre con sus restricciones atributo – valor, número de transacciones conteniendo I.

Soporte mínimo: umbral de soporte.

Transacción: contiene un identificador TID y el conjunto de itemset.