Informe Técnico - Technical Report
DPTOIA-IT-BORRADOR
Junio, 2010

Implementación de TariyKDD Una Herramienta de Descubrimiento de Conocimiento y Evaluación de Rendimiento de Nuevos Algoritmos de Minería de Datos.

Creada por Juan Carlos Alvarado Pérez
Revisada por María Moreno García

Departamento de Informática y Automática
Universidad de Salamanca

Revisado por:

Dra. – María Moreno García
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática y Automática
Facultad de Ciencias - Universidad de Salamanca

Dr. - Pendiente de Asignar

Aprobación pendiente

Información de Autor:

Ing. Juan Carlos Alvarado Pérez
Área de Minería de Datos
Máster Oficial en Sistemas Inteligentes
Departamento de Informática y Automática
Facultad de Ciencias - Universidad de Salamanca
Plaza de la Merced S/N – 37008 - Salamanca
endimeon777@gmail.com – jcalvarado@usal.es

Resumen

En este trabajo se presenta el análisis, diseño e implementación de TariyKDD, una herramienta genérica para el Descubrimiento de Conocimiento en Bases de Datos, débilmente acoplada con el SGBD PostgreSQL.

TariyKDD comprende el desarrollo de cuatro módulos que cubren la conexión, a archivos planos y bases de datos relacionales, la selección, transformación y preparación de los datos, procesos de minería que incluyen tareas de asociación y clasificación, implementando 5 algoritmos, *Apriori, FPGrowth y EquipAsso* para asociación y *C4.5 y Mate* para clasificación, así como diferentes vistas e interpretación de los resultados.

En el presente trabajo, se evalúa el rendimiento de los algoritmos *EquipAsso*, un algoritmo para el cálculo de conjuntos de ítems frecuentes, y *Mate*, un algoritmo para la construcción de arboles de clasificación, basados en nuevos operadores del álgebra relacional, con respecto a los algoritmos *Apriori* y *FP-Growth* y *C4.5* respectivamente.

Abstract

In this work the analysis, design and implementation of TariyKDD are presented. TariyKDD is a generic tool for Knowledge Discovery in Data bases, weakly coupled with the DBGS PostgreSQL.

TariyKDD includes the development of four modules that cover the connection, to text files and relational databases, the selection, transformation and preparation of the data, mining processes that include association and classification tasks, implementing 5 algorithms, Apriori, FPGrowth and EquipAsso for association and C4.5 and Mate for classification, as well as different views and interpretation from the results.

In the present work, the performance of both, EquipAsso, an algorithm for the calculation of sets of frequent items, and Mate, an algorithm for the construction of trees of classification, based on new operators of the relational algebra, are evaluated with respect to the algorithms Apriori and FP-Growth on the one hand and C4.5 on the other, respectively.

Tabla de Contenidos

pág

Lista de Figuras

Lista de Tablas

1. INTRODUCCION

El proceso de extraer conocimiento a partir de grandes volúmenes de datos ha sido reconocido por muchos investigadores como un tópico de investigación clave en los sistemas de bases de datos, y por muchas compañías industriales como una importante área y una oportunidad para obtener mayores ganancias [42].

El Descubrimiento de Conocimiento en Bases de Datos (DCBD) es básicamente un proceso automático en el que se combinan descubrimiento y análisis. El proceso consiste en extraer patrones en forma de reglas o funciones, a partir de los datos, para que el usuario los analice. Esta tarea implica generalmente preprocesar los datos, hacer minería de datos (data mining) y presentar resultados [2, 3, 7, 23, 31]. El DCBD se puede aplicar en diferentes dominios, por ejemplo, para determinar perfiles de clientes fraudulentos (evasión de impuestos), para descubrir relaciones implícitas existentes entre síntomas y enfermedades, entre características técnicas y diagnóstico del estado de equipos y máquinas, para determinar perfiles de estudiantes ”académicamente exitosos” en términos de sus características socioeconómicas, para determinar patrones de compra de los clientes en sus canastas de mercado, entre otros muchos.

Las investigaciones en DCBD, se centraron inicialmente en definir nuevas operaciones de descubrimiento de patrones y desarrollar algoritmos para éstas. Investigaciones posteriores se han focalizado en el problema de integrar DCBD con Sistemas Gestores de Bases de Datos (SGBD) ofreciendo como resultado el desarrollo de herramientas DCBD cuyas arquitecturas se pueden clasificar en una de tres categorías: débilmente acopladas, medianamente acopladas y fuertemente acopladas con el SGBD [41].

Una herramienta DCBD debe integrar una variedad de componentes (técnicas de minería de datos, consultas, métodos de visualización, interfaces, etc.), que juntos puedan identificar y extraer eficientemente patrones interesantes y útiles de los datos almacenados en las bases de datos. De acuerdo a las tareas que desarrollen, las herramientas DCBD se clasifican en tres grupos: herramientas genéricas de tareas sencillas, herramientas genéricas de tareas múltiples y herramientas de dominio específico [31].

En este documento se presenta el trabajo de fin de Máster, fruto de la presente investigación, cuyo resultado es el desarrollo de ”TariyKDD: Una herramienta genérica de Descubrimiento de Conocimiento en Bases de Datos débilmente acoplada a un SGBD”, en la cual se implementaron los algoritmos Apriori[2], FPGrowth[19,23] y el algoritmo propuesto por Timarán EquipAsso [43, 42, 45] como tareas de asociación y por clasificación C4.5[30,31] y MateTree [46] también propuesto por Timarán, sobre los cuales se realizaron ciertas pruebas para medir su rendimiento.

El resto de este documento está organizado de la siguiente manera. En la primera sección se especifica el tema de la propuesta, enmarcándolo dentro de una línea de investigación y delimitándolo. A continuación se describe el problema objeto de estudio. En la sección 3 se especifican los objetivos generales y específicos. En la sección 4 se presenta la justificación de la propuesta de trabajo de fin de Máster. En la sección 5 se presenta el estado del arte en el área de integración de DCBD y SGBD. En la sección 6 se desarrolla todo lo concerniente al análisis orientado a objetos modelado con UML que se realizó para construir la herramienta, en la sección 7 se presenta la implementación del proyecto, en la sección 8 se presenta las pruebas realizadas y los resultados obtenidos, y finalmente en las secciones 9, 10 y 11 se presentan conclusiones, anexos y referencias bibliográficas respectivamente.

1.1 Línea De Investigación

El presente trabajo de fin de Máster, se encuentra inscrito bajo la línea de investigación de descubrimiento de conocimiento, enmarcado dentro del área de las Bases de Datos y específicamente en la subárea de arquitecturas de integración del Proceso de descubrimiento en Bases de datos con Sistemas Gestores de bases de Datos.

1.2 Alcance y Delimitación

TARIYKDD es una herramienta que contempla todas las etapas del proceso DCBD, es decir: Selección, preprocesamiento, transformación, minería de datos y visualización [2, 3, 21]. Para la etapa de minería de datos se implementaron las tareas de Asociación y Clasificación. En estas dos tareas, se utilizaron los operadores algebraicos relacionales y primitivas SQL para DCBD, desarrollados por Timarán [44, 45], con los algoritmos EquipAsso [42, 43, 45] y Mate [46], además de otros algoritmos ampliamente utilizados como lo son Apriori [2], FPGrowth [19,23] y C4.5 [30,31]. Para la etapa de Visualización se desarrolló una interfaz gráfica que le permite al usuario interactuar de manera fácil con la herramienta, además de incluir formas de visualización multidimensional, que le permiten al analista observar los resultados de forma pertinente.

Para los algoritmos implementados, se hicieron pruebas de rendimiento, utilizando conjuntos de datos reales y se los comparó con los algoritmos Apriori Híbrido [2], FP-Growth [19, 23] y C.4.5 [30, 31].

2. PROBLEMA OBJETO DE ESTUDIO

2.1 Descripción del Problema

Muchos investigadores [6, 7, 19, 37] han reconocido la necesidad de integrar los sistemas de descubrimiento de conocimiento y bases de datos, haciendo de ésta un área activa de investigación. La gran mayoría de herramientas de DCBD tienen una arquitectura débilmente acoplada con un Sistema Gestor de Bases de Datos [41].

Algunas herramientas como Alice [25], C5.0 RuleQuest [37], Qyield [8], CoverStory [28] ofrecen soporte únicamente en la etapa de minería de datos y requieren un pre y un post procesamiento de los datos. Hay una gran cantidad de este tipo de herramientas [26], especialmente para clasificación apoyadas en árboles de decisión, redes neuronales y aprendizaje basado en ejemplos. El usuario de este tipo de herramientas puede integrarlas con otros módulos como parte de una aplicación completa [31].

Otras ofrecen soporte en más de una etapa del proceso de DCBD y una variedad de tareas de descubrimiento, típicamente, combinando clasificación, visualización, consulta y clustering, entre otras [31]. En este grupo están Clementine [39], DBMiner [16, 17, 20], DBLearn [18], Data Mine [23], IMACS [5], Intelligent Miner [9], Quest [4] entre otras. Una evaluación de un gran número de herramientas de este tipo se puede encontrar en [15].

Todas estas herramientas necesitan de la adquisición de costosas licencias para su utilización. Este hecho limita a las pequeñas y medianas empresas u organizaciones al acceso a herramientas DCBD para la toma de decisiones, que inciden directamente en la obtención de mayores ganancias

y en el aumento de su competitividad.

Por esta razón, se plantea el desarrollo de una herramienta genérica de DCBD, débilmente acoplada, bajo software libre, que permita el acceso a este tipo de herramientas sin ningún tipo de restricciones, a las pequeñas y medianas empresas u organizaciones de nuestro país o de cualquier parte del mundo.

2.2 Formulación del Problema

Los grandes volúmenes de información, las técnicas tradicionales de exploración de datos, las costosas licencias de software, la posibilidad de conectarse a distintos gestores de bases de datos y archivos planos, la escaza integración de la visualización inteligente de la información para el análisis de resultados y las complicadas interfaces de usuario, hacen que sea necesaria la implentación de una herramienta genérica de descubrimiento de conocimiento bajo software libre, que integre las características que brinden solución a lo anterior

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Desarrollar una herramienta genérica para el Descubrimiento de Conocimiento en bases de datos débilmente acoplada con un sistema gestor de bases de datos, bajo los lineamientos del Software Libre, en la que se comparará el rendimiento de nuevos algoritmos de asociación y clasificación en minería de datos.

3.2 OBJETIVOS ESPECIFICOS

1. Estudiar y Analizar diferentes herramientas DCBD débilmente acopladas con un SGBD.

2. Analizar, diseñar y desarrollar programas que permitan la selección, preprocesamiento y transformación de datos.

3. Analizar, diseñar y desarrollar programas que implementen los operadores algebraicos y primitivas SQL para las tareas de Asociación y Clasificación de datos.

4. Analizar, diseñar y desarrollar programas que implementen los algoritmos EquipAsso, MateTree, Apriori Híbrido, FP-Growth y C4.5.

5. Analizar, diseñar y desarrollar programas que permitan visualizar de manera gráfica las reglas de asociación y clasificación.

6. Integrar todos los programas desarrollados en una sola herramienta para DCBD.

7. Implementar la conexión de la herramienta DCBD con múltiples SGBD.

8. Obtener conjuntos de datos reales para la realización de las pruebas con la herramienta DCBD.

9. Realizar las pruebas y analizar los resultados con la herramienta DCBD débilmente acoplada con

sistema gestor de bases de datos (SGBD).

10. Realizar las pruebas de rendimiento con los diferentes algoritmos implementados.

11. Dar a conocer los resultados de la investigación realizada a través un workshop.

4. JUSTIFICACIÓN

Todas las herramientas de DCBD o comúnmente conocidas como herramientas de minería de datos sirven en las organizaciones para apoyar la toma de decisiones semiestructuradas o no estructuradas. Este tipo de decisiones son aquellas que no se producen con regularidad o que cambian rápidamente y en las cuales no se conoce de antemano la información necesaria para tomarlas, por tanto no es fácil disponer por adelantado de procedimientos preestablecidos de apoyo. Es evidente que por su diseño, estas herramientas tienen mayor capacidad analítica que otras. Están construidas explícitamente con diversos modelos para obtener patrones insospechados a partir de los datos. Apoyan la toma de decisiones al permitir a los usuarios extraer información útil que antes estaba enterrada en montañas de datos. Diversas herramientas de minería de datos disponibles en el mercado ofrecen diferentes tipos de arquitecturas que determinan, en alguna medida, su versatilidad y su costo. Por lo general todas estas son demasiado costosas, necesitan licencias comerciales para su uso y software específico.

El desarrollo de TARIYKDD, como una herramienta DCBD bajo licencia pública GNU, permitirá que empresas u organizaciones que por su tamaño no puedan acceder a herramientas DCBD propietarias, utilicen esta tecnología para mejorar la toma de decisiones, maximicen sus ganancias con decisiones acertadas y eleven su poder competitivo, ya que el ritmo actual del mundo y la globalización así lo requieren.

Por otra parte, TARIYKDD se convierte en otro aporte más en el área del Descubrimiento de Conocimiento en bases de datos, contribuyendo a la investigación científica y al desarrollo académico, ya que en ella se pueden someter a análisis distintas técnicas y algoritmos de descubrimiento de conocimiento, pues el diseño modular de la herramienta permite integrar de manera sencilla, distintas tareas en el núcleo de la herramienta.

Por ser TARIYKDD una herramienta genérica de DCBD, puede utilizarse en diferentes campos como la industria, la banca, la salud y la educación, entre otros.

5. MARCO TEORICO

5.1 El Proceso de Descubrimiento de Conocimiento en Bases De Datos - DCBD

El proceso de descubrimiento de conocimiento en bases de datos DCBD, es el proceso que utiliza algoritmos de minería de datos, para extraer e identificar patrones, sobre datos que reposan en una BD, a los cuales previamente se ha aplicado selección, preprocesamiento, muestreo y transformación, para que finalmente sean evaluados e interpretados, con el propósito de obtener conocimiento [11].

El proceso de DCBD es interactivo e iterativo, involucra numerosos pasos con la intervención del usuario en la toma de muchas decisiones y se resumen en las siguientes etapas:

- Selección.
- Preprocesamiento / *Data cleaning*.
- Transformación / *Reducción.*
- Minería de Datos (*Data Mining*).
- Interpretación y Visualización / *evaluación.*

5.2 Arquitecturas de Integración de las Herramientas DCBD con un SGBD

Las arquitecturas de integración de las herramientas DCBD con un SGBD se pueden ubicar en una de tres tipos: herramientas débilmente acopladas, medianamente acopladas y fuertemente acopladas con un SGBD [41].

Una arquitectura es débilmente acoplada cuando los algoritmos de Minería de Datos y demás componentes se encuentran en una capa externa al SGBD, por fuera del núcleo y su integración con éste se hace a partir de una interfaz [41].

Una arquitectura es medianamente acoplada cuando ciertas tareas y algoritmos de descubrimiento de patrones se encuentran formando parte del SGBD mediante procedimientos almacenados o funciones definidas por el usuario [41].

Una arquitectura es fuertemente acoplada cuando la totalidad de las tareas y algoritmos de descubrimiento de patrones forman parte del SGBD como una operación primitiva, dotándolo de las capacidades de descubrimiento de conocimiento y posibilitándolo para desarrollar aplicaciones de este tipo [41].

Por otra parte, de acuerdo a las tareas que desarrollen, las herramientas DCBD se clasifican en tres grupos: herramientas genéricas de tareas sencillas, herramientas genéricas de tareas múltiples y herramientas de dominio específico.

Las Herramientas genéricas de tareas sencillas principalmente soportan solamente la etapa de minería de datos en el proceso de DCBD y requieren un pre- y un post-procesamiento de los datos. El usuario final de estas herramientas es típicamente un consultor o un desarrollador quien podría integrarlas con otros módulos como parte de una completa aplicación.

Las Herramientas genéricas de tareas múltiples realizan una variedad de tareas de descubrimiento, típicamente combinando clasificación, asociación, visualización, clustering, entre otros. Soportan diferentes etapas del proceso de DCBD. El usuario final de estas herramientas es un analista quien entiende la manipulación de los datos, que conoce la situación global del caso de estudio y además tiene tanto los conocimientos técnicos como científicos de los procesos de descubrimiento de conocimiento, para hacer una adecuada combinación de ellos y extraer la información oculta de las bases de datos.

Finalmente, las Herramientas de dominio específico, soportan descubrimiento solamente en un dominio específico y hablan el lenguaje del usuario final, quien necesita conocer muy poco sobre el proceso de análisis.

5.3 Implementación de Herramientas DCBD Débilmente Acopladas con un SGBD

La implementación de herramientas DCBD débilmente acopladas con un SGBD se hace a través de SQL embebido en el lenguaje anfitrión del motor de minería de datos [1]. Los datos residen en el SGBD y son leídos registro por registro a través de ODBC, JDBC o de una interfaz de cursores SQL. La ventaja de esta arquitectura es su portabilidad. Sus principales desventajas son la escalabilidad y el rendimiento. El problema de escalabilidad consiste en que las herramientas y aplicaciones bajo este tipo de arquitectura, cargan todo el conjunto de datos en memoria, lo que las limita para el manejo de grandes cantidades de datos. El bajo rendimiento se debe a que los registros son copiados uno por uno del espacio de direccionamiento de la base de datos al espacio de direccionamiento de la aplicación de minería de datos [6, 22] y estas operaciones de entrada/salida, cuando se manejan grandes volúmenes de datos, son bastante costosas, a pesar de la optimización de lectura por bloques presente en muchos SGBD (Oracle, DB2, Informix, PostgreSQL.) donde un bloque de tuplas puede ser leído a la vez (figura 5.1).

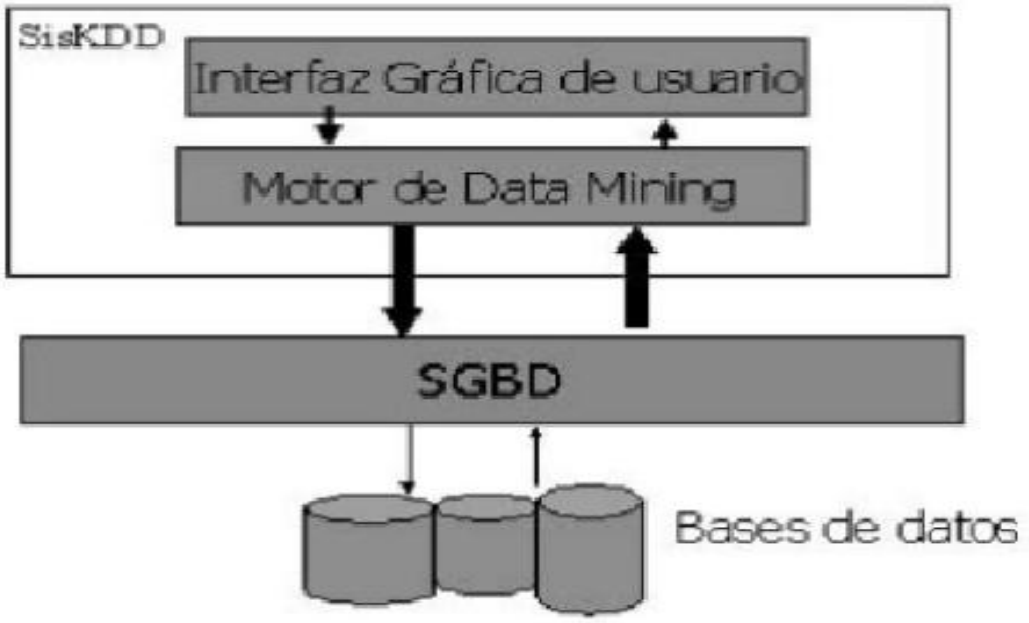

Figura 5.1. Arquitectura DCBD débilmente acoplada

5.4 Algoritmos Implementados en Tariykdd

Dentro de la herramienta de Minería de Datos TariyKDD fueron implementados los siguientes algoritmos de Asociación: Apriori, FPGrowth y EquipAsso, así como también algoritmos de clasificación: C.4.5 y Mate los cuales son explicados a continuación:

5.4.1 Apriori

La notación del algoritmo Apriori [2] es la siguiente:

Tabla 5.1. Notación algoritmo Apriori

k-itemset	Un itemset con k items
L_k	Conjunto de itemsets frecuentes k (Aquellos con soporte mínimo).
C_k	Conjunto de itemsets candidatos k (Itemstes potencialmente frecuentes)

A continuación se muestra el algoritmo Apriori:

L1 = { Conjunto de itemsets frecuentes 1 }
for (k=2; L_{k-1}≠0; k++) do begin
Ck = apriori-gen(L_{k-1}) // Nuevos candidatos
forall transacciones t ∈ D do begin
C_t = subconjunto (C_k, t); // Candidatos en t
forall candidatos c ∈ C_t do
c.cont++;
end
L_k = { c ∈ C_k|c.count ≥ minsup }
end Answer $U_k L_k$

La primera pasada del algoritmo cuenta las ocurrencias de los ítems en todo el conjunto de datos para determinar los itemsets frecuentes *1*. Los subsecuentes pasos del algoritmo son básicamente dos, primero, los itemsets frecuentes L_{k-1} encontrados en la pasada $(k - 1)$ son usados para generar los itemsets candidatos C_k, usando la función apriori-gen descrita en la siguiente subsección. Y segundo se cuenta el soporte de los itemsets candidatos C_k a través de un nuevo recorrido a la base de datos. Se realiza el mismo proceso hasta que no se encuentren más itemsets frecuentes.

Generación de candidatos en A priori. La función apriori-gen toma como argumento L_{k-1}, o sea todos los itemsets frecuentes $(k - 1)$. La función trabaja de la siguiente manera:

insert into Ck
select p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$
from L_{k-1} p, L_{k-1} q
where p.item$_1$ = q.item$_1$, ..., p.item$_{k-2}$ = q.item$_{k-2}$,
p.item$_{k-1}$ q.item$_{k-1}$;

A continuación, en el paso de poda, se borran todos los itemsets $c \in Ck$ tal que algún subconjunto $(k - 1)$ de *c* no esté en $Lk-1$:

forall itemsets c ∈ C_k do

forall subconjuntos (k − 1) s de c do
if (s ∉ L_{k-1}) then
delete c from C_k;

5.4.2 FPGrowth

Como se puede ver en [22] la heurística utilizada por Apriori logra buenos resultados, ganados por (posiblemente) la reducción del tamaño del conjunto de candidatos. Sin embargo, en situaciones con grandes patrones, soportes demasiado pequeños, un algoritmo tipo Apriori podría sufrir de dos costos no triviales:

- Es costoso administrar un gran número de conjuntos candidatos. Por ejemplo, si hay 104 Itemsets Frecuentes, el algoritmo Apriori necesitará generar más de 107 Itemsets Candidatos, así como acumular y probar su ocurrencia. Además, para descubrir un patrón frecuente de tamaño 100, como a1,…, a100, se debe generar más de 2100 candidatos en total.

- Es una tarea demasiado tediosa el tener que leer repetidamente la base de datos para revisar un gran conjunto de candidatos.

El cuello de botella de Apriori es la generación de candidatos [22]. Este problema es atacado por los siguientes tres métodos:

Primero, una innovadora y compacta estructura de datos llamada Árbol de Patrones Frecuentes o FP-tree por sus siglas en inglés (Frequent Pattern Tree), la cual es una estructura que almacena información crucial y cuantitativa acerca de los patrones frecuentes. Únicamente los itemsets frecuentes *1* tendrán nodos en el árbol, el cual está organizado de tal forma que los ítems más frecuentes de una transacción tendrán mayores oportunidades de compartir nodos en la estructura.

Segundo, un método de Minería de patrones crecientes basado en un FP-tree. Este comienza con un patrón frecuente tipo 1 (como patrón sufijo inicial), examina sus Patrones Condicionales Base (una "sub-base de datos" que consiste en el conjunto de ítems frecuentes, que se encuentran en patrón sufijo), construye su FP-tree (condicional) y dentro de éste lleva a cabo recursivamente Minería. El patrón creciente se consigue a través de la concatenación del patrón sufijo con los nuevos generados del FP-tree condicional. Un itemset frecuente en cualquier transacción siempre se encuentra en una ruta de los árboles de Patrones Frecuentes.

Tercero, la técnica de búsqueda empleada en la Minería está basada en particionamiento, con el método "divide y vencerás". Esto reduce drásticamente el tamaño del Patrón Condicional Base generado en el siguiente nivel de búsqueda, así como el tamaño de su correspondiente FP-tree. Es más, en vez de buscar grandes patrones frecuentes, busca otros más pequeños y los concatena al sufijo. Todas estas técnicas reducen los costos de búsqueda.

Diseño y construcción del árbol de Patrones Frecuentes (FP-tree). Sea $I = a_1, a_2,..., a_m$ un conjunto de ítems, y $DB = T_1, T_2,..., T_m$ una base de datos de transacciones, donde $T_i (i \in [1...n])$ es una transacción que contiene un conjunto de ítems en *I*. El soporte (u ocurrencia) de un patrón *A* o conjunto de ítems, es el número de veces que *A* esta contenida en *DB*. *A* es un patrón frecuente, si el soporte de *A* es mayor que el umbral o soporte mínimo, ξ.

Árbol de Patrones Frecuentes. A través del siguiente ejemplo se examina cómo funciona el diseño de la estructura de datos para minar con eficiencia patrones frecuentes.

Ejemplo 1: Sea la base de datos de transacciones, tabla 5.2 y $\xi=3$. Una estructura de datos puede ser diseñada de acuerdo a las siguientes observaciones:

1. Aunque sólo los ítems frecuentes jugarán un rol en la Minería de Patrones Frecuentes, es necesario leer la BD para identificar este conjunto de ítems.

2. Si se almacenan ítems frecuentes de cada transacción en una estructura compacta, se podría evitar el tener que leer repetidamente la BD.

3. Si múltiples transacciones comparten un conjunto idéntico de ítems frecuentes, estas pueden ser fusionadas en una sola, con el número de ocurrencias como contador.

4. Si dos transacciones comparten un prefijo común, las partes compartidas pueden unirse usando una estructura prefijada. Si los ítems frecuentes son ordenados de forma descendente, habrá mayor probabilidad de que los prefijos de las cadenas estén compartidos.

Tabla 5.2: Base de Datos de transacciones

TID	Items	Items frecuentes (ordenados)
100	f,a,c,d,g,i,m,p	f,c,a,m,p
200	a,b,c,f,l,m,o	f,c,a,b,m
300	b,f,h,j,o	f,b
400	b,c,k,s,p	c,b,p
500	a,f,c,e,l,p,m,n	f,c,a,m,p

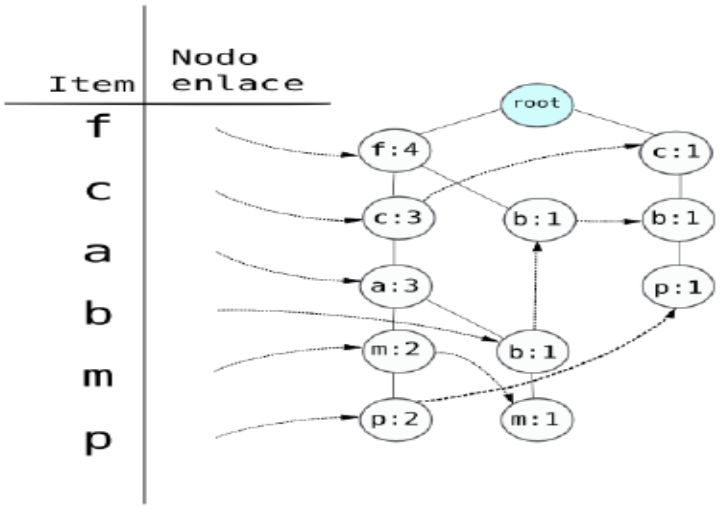

Figura 5.2: Tabla de cabeceras y Árbol FP-tree del ejemplo 1.

Con estas observaciones se puede construir un árbol de Patrones Frecuentes de la siguiente forma:

Primero, el leer DB genera una lista de ítems *frecuentes { (f:4), (c:4), (a:3), (b:3), (m:3), (p:3)* }, (el número indica el soporte) ordenados de forma descendente.

Segundo, crear la raíz del árbol con un null. Al leer la primera transacción se construye la primera rama del árbol *{(f:1), (c:1), (a:1), (m:1), (p:1)}*. Para la segunda transacción, ya que su lista de ítems frecuentes *(f, c, a, b,m)* comparte el prefijo común *(f, c, a)* con la rama existente *(f, c, a,m, p)*, el conteo de cada nodo en el árbol prefijo es incrementado en 1, dos nuevos nodos son creados, *(b:1)* enlazado como hijo de *(a:2) y (m:1)* enlazado como hijo de *(b:1).* Para la tercera transacción, como su lista de frecuentes solamente es *(f, b)* y el único prefijo compartido es *(f)*, el soporte de *(f)* se incrementa en 1, y un nuevo nodo *(b: 1)* es creado y enlazado como hijo de *(f : 3).* La lectura de la cuarta transacción lleva a la construcción de la segunda rama del árbol, *{(c:1), (b:1), (p:1)}.* Para la última transacción ya que su lista de ítems frecuentes *(f, c, a,m, p)* es idéntica a la primera, la ruta es compartida, incrementado el conteo de cada nodo en 1.

Para hacer más fácil el funcionamiento del árbol se construye una Tabla de Cabeceras en la cual cada ítem apunta a su ocurrencia en el árbol. Los nodos con el mismo nombre son enlazados en secuencia y después de leer todas las transacciones, se puede ver el árbol resultante en la figura 5.2. Por tanto un árbol de Patrones Frecuentes (FP-tree) es una estructura como se define a continuación:

- Consiste en una raíz etiquetada como null, un conjunto de árboles hijos de la raíz y una Tabla de Cabeceras de ítems frecuentes.

- Los nodos del árbol de Patrones Frecuentes o FP-tree tienen tres campos: nombre del ítem, contador y enlaces a los demás nodos. El nombre del ítem registra qué ítem representa este nodo, el contador registra el número de transacciones representadas por la porción de la ruta que alcanzan a este nodo y los enlaces llevan al siguiente nodo en el FP-tree, que tiene el mismo nombre o null si no hay nada.

- Cada entrada en la Tabla de Cabeceras de ítems frecuentes tiene dos campos, nombre del ítem y el primer nodo enlazado, al cual apunta la cabecera con el mismo nombre.

Minando Patrones Frecuentes con FP-tree. Existen ciertas propiedades del árbol de Patrones Frecuentes que facilitarán la tarea de Minería de Patrones Frecuentes:

Propiedad de nodos enlazados. Para cualquier nodo frecuente a_i, todos los posibles Patrones Frecuentes que contenga a_i pueden ser obtenidos siguiendo los nodos enlazados de a_i, comenzando desde a_i en la Tabla de Cabeceras de ítems frecuentes.

Ejemplo 2: El siguiente es el proceso de Minería basado en el FP-tree de la figura 5.2. De acuerdo a la propiedad de nodos enlazados para obtener todos los Patrones Frecuentes de un nodo a_i, se comienza desde la cabeza de a_i (en la Tabla de Cabeceras).

Comenzando por los nodos enlazados de p su Patrón Frecuente resultante es *(p:3)* y sus dos rutas en el FP-tree son *(f:4, c:3, a:3, m:2, p:2) y (c:1, b:1, p:1)*. La primera ruta indica que la cadena *"(f, c, a,m, p)"* aparece dos veces en la base de datos. Se puede observar que *"(f, c, a)"* aparece tres

veces y "*(f)"* se encuentra cuatro veces, pero con p solo aparecen dos veces. Además para estudiar qué cadena aparece con p, únicamente cuenta el prefijo de *p, (f:4, c:3, a:3, m:2).* De forma similar, la segunda ruta indica que la cadena "*(c, b, p)"* aparece sólo una vez en el conjunto de transacciones de DB, o que el prefijo de *p* es *(c:1, b:1).* Estos dos prefijos de *p*, *(f:2, c:2, a:2, m:2) y (c:1, b:1)*, forman los Sub-Patrones Base de p o llamados Patrones Condicionales Base. La construcción de un FP-tree sobre este Patrón Condicional Base lleva a únicamente una rama *(c:3).* Así que solo existe un Patrón Frecuente *(cp:3).*

Para el nodo m, se obtiene el Patrón Frecuente *(m:3)* y las rutas *(f:4, c:3, a:3, m:2)* y *(f:4, c:3, a:3, b:1, m:1).* Al igual que en el análisis anterior se obtiene los Patrones Condicionales Base de m, que son *(f:2, c:2, a:2) y (f:1, c:1, a:1, b:1).* Al construir un FP-tree sobre estos, se obtiene que el FP-tree condicional de mes *(f:3, c:3, a:3).* Después se podría llamar recursivamente a una función de Minería basada en un FP-tree (mine((*f:3, c:3, a:3)*|*m*)).

La figura 5.3 muestra cómo funciona (mine((*f:3, c:3, a:3)*|*m*)) y que incluye minar tres ítems a, c, *f*. La primera deriva en un Patrón Frecuente (am:3), y una llamada a (mine((*f:3, c:3)* | *am*)); la segunda deriva en un Patrón Frecuente *(cm:3)*, y una llamada a (mine((*f:3)* | *cm*)); y la tercera deriva en únicamente el Patrón Frecuente *(fm:3).* Con adicionales llamadas recursivas a (mine*((f:3, c:3)* | *am*)) se obtiene *(cam:3), (fam:3)* y con una llamada a (*mine((f:3)* | *cam)*), se obtendrá el patrón más largo (*fcam:3*). Similarmente con la llamada de (*mine((f:3)* | *cm))*, se obtiene el patrón (*fcm:3*). Además todo el conjunto de Patrones Frecuentes que tienen a m es *{(m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3)}* o que explicado de otra forma, por ejemplo para m los Patrones Frecuentes son obtenidos combinando a m con todos sus Patrones Condicionales Base *(f:3, c:3, a:3)*, entonces sus Patrones Frecuentes serían los ya mencionados *{(m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3)}.*

Así mismo, con el nodo b se obtiene (b:3) y tres rutas: *(f:4, c:3, a:3, b:1), (f:4, b:1) y (c:1, b:1).* Dado que los Patrones Condicionales Base de *b: (f:1, c:1, a:1), (f:1)* y *(c:1)* no generan ítems frecuentes, el proceso de minería termina. Con el nodo *a* solo se obtiene un Patrón Frecuente *{(a:3)}* y un Patrón Condicional Base *{(f:3, c:3)}.* Además su conjunto de Patrones Frecuentes puede ser generado a partir de sus combinaciones. Concatenándolas con *(a:3),* obtenemos *{(fa:3), (ca:3), (fca:3), }*. Del nodo *c* se deriva (*c:4*) y un Patrón Condicional Base *{(f:3)},* y el conjunto de Patrones Frecuentes asociados con *(c:3)* es *{(fc:3)}.* Con el nodo f solo se obtiene *(f:4)* sin Patrones Condicionales Base.

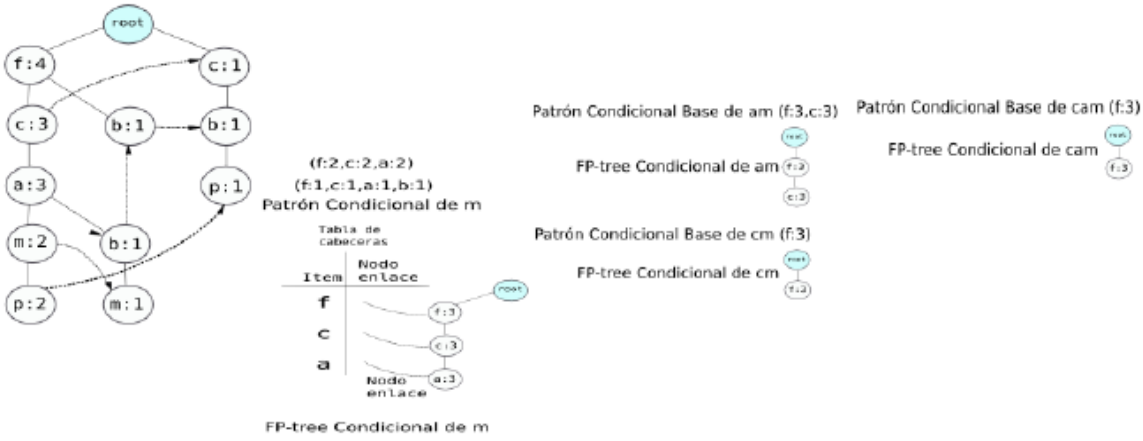

Figura 5.3: FP-tree condicional para *m*

En la siguiente tabla se muestran los Patrones Condicionales Base y los FP-trees generados.

Tabla 5.3: Patrones Condicionales Base y FP-trees Condicionales

Item	Patrón Condicional Base	FP-tree condicional
p	$\{(f:2, c:2, a:2, m:2), (c:1, b:1)\}$	$\{(c:3)\}\|p$
m	$\{(f:2, c:2, a:2), (f:1, c:1, a:1, b:1)\}$	$\{(f:3, c:3, a:3)\}\|m$
b	$\{(f:1, c:1, a:1), (f:1), (c:1)\}$	ϕ
a	$\{(f:3, c:3)\}$	$\{(f:3, c:3)\}\|a$
c	$\{(f:3)\}$	$\{(f:3)\}\|c$
f	ϕ	ϕ

Como se dijo anteriormente los Patrones Frecuentes se obtienen a partir de las combinaciones de cada uno de los ítems con sus Patrones Condicionales Base. Por ejemplo para m, sus Patrones Frecuentes *{(m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3)},* son obtenidos de combinar a m con cada uno de sus Patrones Condicionales Base *{(f:2, c:2, a:2), (f:1, c:1, a:1, b:1)}.*

5.4.3 EquipAsso. Nuevos Operadores Del Algebra Relacional Para Asociación.

- Operador Associator (α). Associator(α) es un operador algebraico unario que al contrario del operador Selección o Restricción (σ), aumenta la cardinalidad o el tamaño de una relación ya que genera a partir de cada tupla de una relación, todas las posibles combinaciones de los valores de sus atributos, como tuplas de una nueva relación conservando el mismo esquema. Por esta razón esta operación, debe ser posterior a la mayor de operaciones en el proceso de optimización de una consulta.

Su sintaxis es la siguiente: $\alpha_{\text{tam inicial, tam final}}$ (R).

El operador Associator genera, por cada tupla de la relación *R,* todos sus posibles subconjuntos (itemsets) de diferente tamaño. Associator toma cada tupla *t* de *R* y dos parámetros: tam inicial y tam final como entrada, y retorna, por cada tupla *t*, las diferentes combinaciones de atributos X_i , de tamaño tam inicial hasta tamaño tam final , como tuplas en una nueva relación. El orden de los atributos en el esquema de *R* determina los atributos en los subconjuntos con valores, el resto se hacen nulos. El tamaño máximo de un itemset y por consiguiente el tamaño anal máximo (tam final) que se puede tomar como entrada es el correspondiente al valor del grado de la relación.

Formalmente, sea $A = \{A_1, ..., A_n\}$ el conjunto de atributos de la relación *R* de grado *n* y cardinalidad m, IS y ES el tamaño inicial y final respectivamente de los subconjuntos a obtener. El operador α aplicado a *R*.

$$\alpha_{IS,ES}(R) = \{ \cup_{all} X_i \mid X_i \subseteq t_i , t_i \in R, \forall_i \forall_k (X_i = v_i(A_1), v_i(A_2), null, ..., v_i(A_k), null , v_i(A_k) null), (i = (2^n - 1) * m), (k = IS, ..., ES), A_1 A_2 ... A_k , IS = 1, ES = n\}$$

Produce una nueva relación cuyo esquema *R(A)* es el mismo de *R* de grado *n* y cardinalidad $m'=(2^n-1)*m$ y cuya extensión *r(A)* está formada por todos los subconjuntos X_i generados a partir de todas las combinaciones posibles de los valores no nulos $v_i(A_k)$ de

los atributos de cada tupla ti de *R*. En cada tupla X_i únicamente un grupo de atributos mayor o igual que IS y menor o igual que ES tienen valores, los demás atributos se hacen nulos.

Ejemplo 1: Sea la relación R(A,B,C,D):

A	**B**	**C**	**D**
a1	b1	c1	d1
a1	b2	c1	d2

El resultado de $R1 = \alpha_{2,4}(R)$, se puede observar en la figura 5.4.

- Operador Equikeep (χ). Equikeep (χ) es un operador unario que restringe los valores de los atributos de cada una de las tuplas de la relación R a únicamente los valores de los atributos que satisfacen una expresión lógica.
  Su sintaxis es la siguiente: χ expresion lógica *(R)*

- El operador EquiKeep restringe los valores de los atributos de cada una de las tuplas de la relación *R* a únicamente los valores de los atributos que satisfacen una expresión lógica expr_log, la cual está formada por un conjunto de cláusulas de la forma Atributo=Valor, y operaciones lógicas AND, OR y NOT. En cada tupla, los valores de los atributos que no cumplen la condición expr_log se hacen nulos. EquiKeep elimina las tuplas vacías i.e. las tuplas con todos los valores de sus atributos nulos.
  Formalmente, sea $A=A_1,...,A_n$ el conjunto de atributos de la relación *R* de esquema *R(A),* de grado *n* y cardinalidad *m*. Sea p una o expresión lógica integrada por cláusulas de la forma $A_i=const$ unidas por los operadores booleanos AND (^), OR (v), NOT (¬). El operador χ aplicado a la relación *R* con la expresión lógica *p*:

$$\chi_p(R) = \{t_i(A) | \forall_i \forall_j (p(v_i(A_j)) = v_i(A_j) \text{ si } p = true \text{ y } p(v_i(A_j)) = null \text{ si } p = false),\ i = 1...m',\ j = 1...n,\ m' \leq m\}$$

Produce una relación de igual esquema *R(A)* de grado n y cardinalidad *m* , donde $m \leq m$. En su extensión, cada *n-tupla* t_i, está formada por los valores de los atributos de *R,* $v_i(A_j)$, que cumplan la expresión lógica *p*, es decir $p(v_i(Y_j))$ es verdadero, y por valores nulos si $p(v_i(Y_j))$ es falso.

Ejemplo 2: Sea la relación R(A,B,C,D) y la operación $\chi_{A=a1 \vee B=b1 \vee C=c2 \vee D=d1}$ (R):

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b1	c1	d1
a2	b2	c1	d2
a1	b2	c2	d1

El resultado de esta operación es la siguiente:

A	B	C	D
a1	b1	null	d1
a1	null	null	null
null	null	c2	null
null	b1	null	d1
null	null	null	null
a1	null	c2	d1

- Algoritmo EquipAsso. El primer paso del algoritmo simplemente cuenta el número de ocurrencias de cada ítem para determinar los 1-itemsets frecuentes. En el subsiguiente paso, con el operador EquiKeep se extraen de todas las transacciones los itemsets frecuentes tamaño 1 haciendo nulos el resto de valores. Luego se aplica el operador Associator desde $I_s=2$ hasta el grado n. A continuación se muestra el algoritmo Equipasso:

L1 = {1−itemsets f recuentes};
forall transacciones t ∈ D do begin
    R = χL1 (D)
    K=2
    g = grado(R)
    R = $\alpha_{k,g}$ (R)
end
Lk = {count(R') | c.count ≥ minsup}
Respuesta = ∪Lk ;

5.4.4 Mate

El operador Mate genera, por cada una de las tuplas de una relación, todas los posibles combinaciones formadas por los valores no nulos de los atributos pertenecientes a una lista de atributos denominados Atributos Condición, y el valor no nulo del atributo denominado Atributo Clase.

Tiene la siguiente sintaxis: μ lista atributos condición; atributo clase*(R)* donde lista atributos condición es el conjunto de atributos de la relación *R* a combinar con el atributo clase y atributo clase es el atributo de *R* definido como clase.

Mate toma como entrada cada tupla de *R* y produce una nueva relación cuyo esquema está formado por los atributos condición, lista atributos condición y el atributo clase, atributo clase con tuplas formadas por todas las posibles combinaciones de cada uno de los atributos condición con el atributo clase, los demás valores de los atributos se hacen nulos.

Tabla 5.4: Resultado de R1 = $\alpha_{2,4}$ (R)

A	B	C	D
a1	b1	null	null
a1	null	c1	null
a1	null	null	d1
null	b1	c1	null
null	b1	null	d1
null	null	c1	d1
a1	b1	c1	null
a1	b1	null	d1
a1	null	c1	d1
null	b1	c1	d1
a1	b1	c1	d1
a1	b2	null	null
a1	null	c1	null
a1	null	null	d2
null	b2	c1	null
null	b2	null	d2
null	null	c1	d2
a1	b2	c1	null
a1	b2	null	d2
a1	null	c1	d2
null	b2	c1	d2
a1	b2	c1	d2

Formalmente, sea $A = \{A1, ..., An\}$el conjunto de atributos de la relación R de grado n y cardinalidad m, $LC \subset A$, $LC \neq \phi$ la lista de atributos condición a emparejar y n el número de atributos de LC, $|LC| = n'$, $n' < n$, $Ac \in A$, $Ac \cap L = \phi$ el atributo clase con el que se emparejarán los atributos de LC. El operador μ aplicado a la lista de atributos LC, al atributo clase Ac de la relación R es definido de este modo:

$$\mu_{LC;\,Ac}(R) = \{\, ti(M) \mid M = LC \cup Ac,\ LC \subset A,\ |LC| = n',\ n' < n,\ Ac \in A,\ Ac \cap LC = \phi,\ ti = Xi,\ 1 \_\ i \_\ m',\ m' = (2_{n'} - 1) * m,\ \forall_i \forall_k (\, X_i = <null, ..., v_i(A_k) ... , null, ..., v_i(Ac)>,\ v_i(A_k)\ v_i(Ac)\ \_\ null),\ 1 \_\ k \_\ n' \,\}$$

Produce una relación cuyo esquema es $R(M)$,$M = LC \subset Ac$, de grado g, $g = n' + 1$ y cuya extensión $r(M)$ de cardinalidad m', $m' = (2n' - 1) * m$ es el conjunto de g-tuplas t_i, tal que en cada

g-tupla únicamente los atributos que forman la combinación $Xi \subset LC, (k = 1..n')$ y el atributo *Ac* tienen valor, el resto de atributos se hacen nulos.

Mate empareja en cada partición todos los atributos condición con el atributo clase, lo que facilita el conteo y el posterior cálculo de las medidas de entropía y la ganancia de información. El operador Mate genera estas combinaciones, en una sola pasada sobre la tabla de entrenamiento (lo que redunda en la eficiencia del proceso de construcción del árbol de decisión).

Ejemplo 1: Sea la relación R(A,B,C,D) de la tabla 5.5. A partir de ella se obtienen las diferentes combinaciones de los atributos A,B con el atributo D, es decir, R1 = $\mu_{A,B;D}$ (R).

Tabla 5.5: Relación R

A	B	C	D
a1	b1	c1	d1
b1	b2	c2	d2

El resultado de la operación $R1 = \mu_{A,B;D}\ (R)$, se muestra en la tabla 5.6:

Función Algebraica Agregada Entro. La función *Entro()* permite calcular la entropía de una relación *R* con respecto a un atributo denominado atributo condición y un atributo clase.

Tabla 5.6: Resultado de la Operación R1 = $\mu_{A,B;D}$ (R)

A	B	C
a1	null	d1
null	b1	d1
a1	b1	d1
a1	null	d2
null	b2	d2
a1	b2	d2

Tiene la siguiente sintaxis: *Entro (Atributo; Atributo clase; R)* donde atributo es el atributo condición de la relación *R* y atributo clase es el atributo con el que se combina el atributo atributo.

Formalmente, sea $A = \{A1, ..., An, Ac\}$ el conjunto de atributos de la relación *R* o con esquema *R(A)*, extensión *r(A)*, grado *n* y cardinalidad *m*. Sea *t* el número de distintos valores del atributo clase *Ac*, $Ac \subset R(A)$ que divide a *r(A)* en *t* diferentes clases, $Ci(i = 1..t)$. Sea r_i el número de tuplas de *r(A)* que pertenecen a la clase C_i. Sea *q* el número de distintos valores $\{v_1 (A_k), v_2 (A_k), .., v_q (A_k)\}$ del atributo A_k , $A_k \subset R(A)$, el cual particiona a *r(A)* en q subconjuntos $\{S_1 , S_2 , ...S_q \}$, donde S_j contiene todos las tuplas de *r(A)* que tienen el valor $v_j (A_k)$ del atributo A_k. Sea s_{ij} el número de tuplas de la clase C_i en el subconjunto $\underline{S_j}$. La función $Entro(A_k ; Ac; R)$, retorna la entropía de *R* con respecto al atributo A_k , que se obtiene de la siguiente manera :

$$Entro(A_k; Ac; R) = \{y | y = - p_{ij} \log2 (p_{ij}), i = 1..t, j = 1..q, p_{ij} = s_{ij} / |S_j|\}$$

donde $p_{ij} = s_{ij} / |S_j|$es la probabilidad que una tupla en S_j pertenezca a la clase C_i.

La entropía de *R* con respecto al atributo clase *Ac* es:

$$Entro(Ac; Ac; R) = \{y|y = -\ p_i \log2\ (p_i\ ),\ i = 1..t,\ p_i = r_i\ /m\}$$

Donde p_i es la probabilidad que un tupla cualquier pertenezca a la clase C_i *y* r_i el número de tuplas de *r(A)* que pertenecen a la clase C_i.

Función Algebraica Agregada Gain. La función *Gain()* permite calcular la reducción de la entropía causada por el conocimiento del valor de un atributo de una relación.

Su sintaxis es: *Gain(atributo; atrib clase;R).* Donde atributo es el atributo condición de la relación *R* y atributo clase es el atributo con el que se combina el atributo atributo.

La función *Gain() permite* calcular la ganancia de información obtenida por el particionamiento de la relación *R* de acuerdo con el atributo atributo y se define:

$$Gain(A_k; Ac; R) = \{y|y = Entro(Ac; Ac; R) - Entro(A_k; Ac; R)\}$$

Operador Describe Classifier (βμ). *Describe Classifier (βμ)* es un operador unario que toma como entrada la relación resultante de los operadores Mate By, Entro() y Gain() y produce una nueva relación donde se almacenan los valores de los atributos que formaron los diferentes nodos del árbol de decisión.

La sintaxis del operador Describe Classifier es la siguiente *βμ(R)*

Formalmente, sea $A = \{A_1, .., A_n, E, G\}$ el conjunto de atributos de la relación *R* de grado $n + 2$ y cardinalidad *m*. El operador *(βμ) aplicado a R*:

$$(\beta\mu)(R) = \{t_i\ (Y\ )\ |\ Y = \{N.P, A.V, C\},\ si\ t_i = val(N),null,val(A),null,null > raiz,\ si\ t_i = val(N),val(P),val(A),val(V),val(C) > hoja,\ si\ t_i = val(N),val(P),val(A),val(V),null > nodo\ interno\}$$

Produce una nueva relación con esquema *R (Y), Y=N,P,A,V,C* donde *N* es el atributo que identifica el número de nodo, *P* identifica el nodo padre, A identifica el nombre del atributo asociado a ese nodo, *V* es el valor del atributo *A y C* es el atributo clase. Su extensión *r(Y),* está formada por un conjunto de tuplas en las cuales si los valores de los atributos son:

N null, P = null, Anull, V = null y C = null corresponde a un nodo raíz; si N null, P null, Anull, V null y Cnull corresponde a una hoja o nodo terminal y si N null, P null, A null, V null y C = null corresponde a un nodo interno.

Describe Classifier facilita la construcción del árbol de decisión y por consiguiente la generación de reglas de clasificación.

Ejemplo 2: Sea la relación *ÁRBOL (NODO,PADRE,ATRIBUTO,VALOR,CLASE)* de la tabla 5.7 resultado del operador Describe Classifier. A continuación se construye el árbol de decisión.

Tabla 5.7: Relación Árbol

NODO	PADRE	ATRIBUTO	VALOR	CLASE
N0	Null	Temperatura	Null	Null
N1	N0	Temperatura	Alta	Si
N2	N0	Temperatura	Media	No
N3	N0	Temperatura	Normal	Null
N4	N3	D_muscular	Si	Si
N5	N3	D_muscular	No	No

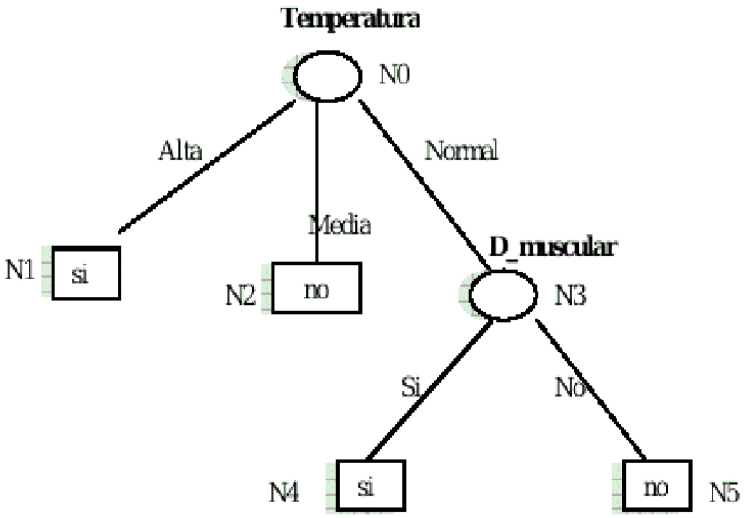

Figura 5.4: Árbol de decisión:

5.5 Poda de los Arboles Generados

La poda de los árboles generados por los algoritmos de clasificación en minería de datos, se hace necesaria por diversas razones, entre las cuales podemos mencionar la sobregeneralización, la evaluación de atributos no relevantes o insignificantes y el gran tamaño que árbol obtenido pueda alcanzar.

El caso de la sobregeneralización se presenta cuando un árbol puede haber sido construido a partir de de un conjunto de datos con ruido (noise data), por lo cual algunas ramas del árbol pueden ser engañosas o inciertas.

En cuanto a la evaluación de atributos intrascendentes, éstos deben eliminarse mediante poda ya que sólo agregan niveles en el árbol y no contribuyen a la ganancia de información.

Por último, si el árbol obtenido es demasiado profundo o demasiado frondoso, se dificultará la interpretación de las reglas obtenidas por parte del analista,

Existen dos tipos de metodología para la poda de los arboles: la pre-poda (preprunning) y la post-poda (postprunning).

En pre-poda (preprunning), la construcción del árbol se efectúa al mismo tiempo que se aplica el método de poda, de esta forma el crecimiento del árbol se detiene cuando la ganancia de información producida se inclina hacia un conjunto, al superar un umbral determinado.

En post-poda (postprunning), se efectúa una vez que se ha terminado de construir el árbol completamente, podando algunas ramas.

La pre-poda (preprunning), tiene como ventaja que no se pierde tiempo en construir una estructura que más tarde será simplificada en el árbol definitivo. El objetivo específico del analista en este caso, será buscar el mejor límite (*threshold*) que dividirá el subconjunto, para evaluar la partición con diferentes técnicas como son: con enfoque estadístico, mediante la teoría de la ganancia de información, reducción de errores, etc. Si esta evaluación es menor que el límite predeterminado, dicha división se descartará y el árbol determinará el subconjunto con la hoja más apropiada. Sin embargo, este tipo de método tiene el inconveniente de que no es sencillo detener el particionamiento en el momento idóneo, ya que un límite muy alto no permitirá la extracción de conocimiento idóneo, por otro lado, un límite demasiado bajo resultará en una simplificación insignificante.

La post-poda (postprunning) es un proceso regresivo a través del árbol de decisión, partiendo de las hojas y llegando hasta el nodo raíz, podando ramas a partir de criterios propios de cada uno de los algoritmos, como la ganancia de información.

5.6 Estado del Arte

En el desarrollo de este trabajo se ha realizado un estudio de herramientas de minería de datos elaboradas por otras universidades y centros de investigación que nos permitieron ver el estado actual de este tipo de aplicaciones. A continuación se describen las herramientas estudiadas.

5.6.1 WEKA - Waikato Environment for Knowledge Analysis

WEKA es una herramienta libre de minería de Datos realizada en el departamento de Ciencias de la Computación de la Universidad de Waikato en Hamilton, Nueva Zelanda. Los principales gestores de este proyecto son Ian H. Waitten y Eibe Frank autores del libro Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations [54] cuyo octavo capítulo sirve como tutorial de Weka y es libremente distribuido junto con el ejecutable y el código fuente. Las últimas versiones estables de Weka pueden ser descargadas de la página oficial del Proyecto [53].

La implementación de la herramienta fue hecha bajo la plataforma Java utilizando la versión 1.4.2 de su máquina virtual. Al ser desarrollada en Java, Weka posee todas las características de una herramienta orientada a objetos con la ventaja de ser multiplataforma, la última versión ha sido evaluada en entornos GNU/Linux, Macintosh y Windows siguiendo una arquitectura Cliente/Servidor, lo que permite ejecutar ciertas tareas de manera distribuida.

La conexión a la fuente de datos puede hacerse directamente hacia un archivo plano, considerando varios formatos como C4.5 y CVS aunque el formato oficial es el ARFF que se explica más adelante, o a través de un driver JDBC hacia diferentes Sistemas Gestores de Bases de Datos (SGBD).

Entre las principales características de Weka está su modularidad la cual se fundamenta en un estricto y estandarizado formato de entrada de datos que denominan ARFF (Atributte - Relation File Format) [52], a partir de este tipo de archivo de entrada se pueden aplicar los algoritmos de minería implementados en Weka. Las nuevas implementaciones y adiciones deben ajustarse a este formato.

El formato ARFF consiste en un archivo sencillo de texto que funciona a partir de etiquetas, que debe cumplir con dos secciones: una cabecera y un conjunto de datos. La cabecera contiene las etiquetas del nombre de la relación(@relation) y los atributos (@atributte), que describen los tipos y el orden de los datos. La sección datos (@data) en un archivo ARFF contiene todos los datos del conjunto que se quiere evaluar separados por comas y en el mismo orden en el que aparecen en la sección de atributos [51].

La conexión al SGBD se hace en primera instancia a través de una interfaz gráfica de usuario construyendo una sentencia SQL siguiendo un modelo relacional pero a partir de construida la tabla a minar se trabaja en adelante fuera de línea a través de flujos (streams) que se comunican con archivos en disco duro.

WEKA cubre gran parte del proceso de Descubrimiento de Conocimiento, las características específicas de minería de datos que se pueden ver son pre-procesamiento y análisis de datos, clasificación, clustering, asociación y visualización de resultados.

WEKA posee una rica colección de algoritmos que soportan el proceso de minería implementando diversos tipos de métodos como arboles de decisión, conjuntos difusos, reglas de inducción, métodos estadísticos y redes bayesianas.

Por poseer diferentes modos de interfaces gráficas, WEKA se puede considerar una herramienta orientada al usuario aunque cabe aclarar que exige un buen dominio de conceptos de minería de datos y del objeto de análisis. Muchas tareas se encuentran soportadas aunque no del todo automatizadas por lo que el proceso de descubrimiento debe ser guiado aún por un analista.

Un análisis completo de las características de WEKA con respecto a otras aplicaciones presentes en el mercado se puede encontrar en [15].

5.6.2 ADaM - Algorithm Development and Mining System

El proyecto ADaM es desarrollado por el Centro de Sistemas y Tecnología de la Información de la Universidad de Alabama en Huntsville, Estados Unidos. Este sistema es usado principalmente para aplicar técnicas de minería a datos científicos obtenidos de sensores remotos [50].

ADaM es un conjunto de herramientas de minería y procesamiento de imágenes que consta de varios componentes interoperables que pueden usarse en conjunto para realizar aplicaciones en la solución de diversos problemas. En la actualidad, ADaM (en su versión 4.0.2) cubre cerca de 120 componentes [49] que pueden ser configurados para crear procesos de minería personalizados. Se pueden integrar fácilmente nuevos componentes a un sistema o proceso existente.

ADaM 4.0.2 proporciona soporte a través del uso de componentes autónomos dentro de una arquitectura distribuida. Cada componente está desarrollado en C, C++ u otra interfaz de programación de aplicaciones y entrega un ejecutable a modo de *script* donde cada componente recibe sus parámetros a través de línea de comandos y arroja los resultados en ficheros que pueden

ser utilizados a su vez por otros componentes ADaM. Eventualmente se ofrece Servicios Web de algunos componentes por lo que se puede acceder a ellos a través de la Web.

Desafortunadamente el acceso a fuentes de datos es limitado no ofreciendo conexión directa hacia un sistema gestor de bases de datos. ADaM trabaja generalmente con ficheros ARFF [51] que son generados por sus componentes.

Dentro de las herramientas ofrecidas por ADaM encontramos soporte al preprocesamiento de datos y de imágenes, clasificación, clustering y asociación. La visualización y análisis de resultados no forman parte de la herramienta, sin embargo existe la posibilidad de ser adaptados como un componente externo. ADaM 4.0.2 ofrece un amplio conjunto de herramienta que implementa diversas metodologías dentro del área del descubrimiento de conocimiento. Existen módulos que implementan árboles de decisión, reglas de asociación, métodos estadísticos, algoritmos genéticos y redes bayesianas.

ADaM 4.0.2 no soporta una interfaz gráfica de usuario, se limita a ofrecer un conjunto de herramientas para ser utilizadas en la construcción de sistemas que cubran diferentes ámbitos. Por tal motivo, es necesario un buen conocimiento de los conceptos de minería de datos, aparte de fundamentos en el análisis y procesamiento de imágenes. No obstante, ADaM ofrece un muy buen soporte a sistemas donde se busque descubrimiento de conocimiento, un ejemplo de la implementación de ADaM puede verse en [30] donde se utilizan componentes ADaM en el análisis e interpretación de imágenes de satélite de ciclones para estimar la máxima velocidad de los vientos.

5.6.3 Orange - Data Mining Fruitful and Fun

Construido en el Laboratorio de Inteligencia Artificial de la Facultad de Computación y Ciencias de la Información de la Universidad de Liubliana, en Eslovenia y ya que fue liberado bajo la Licencia Pública General (GPL) puede ser descargado desde su sitio oficial [12].

Orange [10] en su núcleo es una librería de objetos C++ y rutinas que incluyen entre otros, algoritmos estándar y no estándar de minería de Datos y Aprendizaje automático, además de rutinas para la entrada y manipulación de datos. Orange proporciona un entorno para que el usuario final pueda acceder a la herramienta a través de *scripts* hechos en Python y que se encuentran un nivel por encima del núcleo en C++. Entonces el usuario puede incorporar nuevos algoritmos o utilizar código ya elaborado que le permiten cargar, limpiar y minar datos, así como imprimir árboles de decisión y reglas de asociación.

Otra característica de Orange es la inclusión de un conjunto de Widgets gráficos que usan métodos y módulos del núcleo central (C++) brindando una interfaz agradable e intuitiva al usuario.

Los Widgets de la interfaz gráfica y los módulos en Python incluyen tareas de minería de Datos desde preprocesamiento hasta la inducción de modelos y evaluación. Entre otras funciones estos componentes poseen técnicas para:

Entrada de datos. Orange proporciona soporte para varios formatos populares de datos. Como por ejemplo:

> **.tab.** Formato nativo de Orange. La primera línea tiene los nombres de los atributos, la segunda línea indica cual es el tipo de datos de cada columna (discreto o continuo) y en

adelante se encuentran los datos separados a través de tabuladores.

.c45. Estos archivos están compuestos por dos archivos uno con extensión **.names.** que tiene los nombres de las columnas separados por comas y otro .data con los datos separados también con comas.

Manipulación de datos y preprocesamiento. Entre las tareas que Orange incluye en este apartado tenemos visualización gráfica y estadística de datos, procesamiento de filtros, discretización y construcción de nuevos atributos entre otros métodos.

Minería de Datos y Aprendizaje automático Dentro de esta rama Orange incluye variedad de algoritmos de asociación, clasificación, regresión logística, regresión lineal, árboles de regresión y acercamientos basados en instancias (instance-based approaches).

Contenedores. Para la calibración de la predicción de probabilidades de modelos de clasificación.

Métodos de evaluación. Que ayudan a medir la exactitud de un clasificador.

Podría catalogarse como una equivocación el hecho de que Orange no incluya un módulo para la conexión a un Sistema Gestor de Bases de Datos, pero si se revisa bien la documentación de los módulos se encuentra que ha sido desarrollado uno para la conexión a MySQL (orngMySQL [13]). El módulo proporciona una entrada a MySQL a través de este modulo y de sencillos comandos en Python los datos de las tablas pueden ser transferidos desde y hacia MySQL. asimismo programadores independientes han desarrollado varios módulos entre los que se destaca uno para algoritmos de Inteligencia Artificial.

Dentro de las herramientas de minería de Datos, Orange Podría catalogarse como una Herramienta Genérica Multitarea. Estas herramientas realizan una variedad de tareas de descubrimiento, típicamente combinando clasificación, asociación, visualización, clustering, entre otros. Soportan diferentes etapas del proceso de DCBD. El usuario final de estas herramientas es un analista quien entiende la manipulación de los datos.

Orange funciona en varias plataformas como Linux, Mac y Windows y desde su sitio web [12] se puede descargar toda la documentación disponible para su instalación. Al instalar Orange el usuario puede acceder a una completa información de la herramienta, así como a prácticos tutoriales y manuales que permiten familiarizarse con la misma. Su sitio web [12] incluye tutoriales básicos sobre Python y Orange, manuales para desarrolladores más avanzados y además tener acceso a los foros de Orange en donde se despejan todos los interrogantes sobre esta herramienta.

En sí Orange está hecha para usuarios experimentados e investigadores en aprendizaje automático con la ventaja de que el software fue liberado con licencia GPL, por tanto cualquier persona es libre de desarrollar y probar sus propios algoritmos reusando tanto código como sea posible.

5.6.4 TANAGRA - A Free Software for Research and Academic Purposes

TANAGRA [35] es software de minería de Datos con propósitos académicos e investigativos, desarrollado por Ricco Rakotomalala, miembro del Equipo de Investigación en Ingeniería del Conocimiento (ERIC - *Equipe de Recherche en Ingènierie des Connaissances* [47]) de la Universidad de Lyon, Francia. Conjuga varios métodos de minería de Datos como análisis exploratorio de datos, aprendizaje estadístico y aprendizaje automático.

Este proyecto es el sucesor de SIPINA, el cuál implementa varios algoritmos de aprendizaje supervisado, especialmente la construcción visual de árboles de decisión. TANAGRA es más potente, contiene además de algunos paradigmas supervisados, clustering, análisis factorial, estadística paramétrica y no-paramétrica, reglas de asociación, selección de características y construcción de algoritmos.

TANAGRA es un proyecto open source, así que cualquier investigador puede acceder al código fuente y añadir sus propios algoritmos, en cuanto esté de acuerdo con la licencia de distribución.

El principal propósito de TANAGRA es proponer a los investigadores y estudiantes una arquitectura de software para minería de Datos que permita el análisis de datos tanto reales como sintéticos.

El segundo propósito de TANAGRA es proponer a los investigadores una arquitectura que les permita añadir sus propios algoritmos y métodos, para así comparar sus rendimientos. TANAGRA es más una plataforma experimental, que permite ir a lo esencial obviando la programación del manejo de los datos.

El tercero y último propósito va dirigido a desarrolladores novatos y consiste en difundir una metodología para construir este tipo de software con la ventaja de tener acceso al código fuente, de esta forma un desarrollador puede ver como ha sido construido el software, cuales son los problemas a eludir, cuales son los principales pasos del proyecto y que herramientas y librería usar.

Lo que no incluye TANAGRA es lo que constituye la fuerza del software comercial en el campo de la minería de Datos: Grandes fuentes de datos, acceso directo a bodegas y bases de datos (*Data Warehouses* y *Data Bases*) así como la aplicación de *data cleaning*.

Para realizar trabajos de minería de Datos con Tanagra, se deben crear esquemas y diagramas de flujo y utilizar sus diferentes componentes que van desde la visualización de datos, estadísticas, construcción y selección de instancias, regresión y asociación entre otros.

El formato del conjunto de datos de Tanagra es un archivo plano (.txt) separado por tabulaciones, en la primera línea tiene un encabezado con el nombre de los atributos y de la clase, a continuación aparecen los datos con el mismo formato de separación con tabuladores. Tanagra también acepta conjuntos de datos generados en Excel y uno de los estándares en minería de Datos, el formato de WEKA (archivos .arff). Tanagra permite cargar un sólo conjunto de datos.

A medida que se trabaja con TANAGRA y se van generando informes de resultados, existe la posibilidad de guardarlos en formato HTML.

La paleta de componentes de Tanagra tiene implementaciones de tareas de minería de Datos que van desde preprocesamiento hasta la inducción de modelos y evaluación. Entre otras TANAGRA permite usar técnicas para:

Entrada de datos con soporte para varios formatos populares de datos.

Manipulación de datos y preprocesamiento como muestreo, filtrado, discretización y construcción de nuevos atributos entre otros métodos.

Construcción de modelos mediante métodos de clasificación, que incluyen árboles de clasificación, clasificadores Naive-Bayes y métodos de regresión como regresión múltiple lineal.

Métodos de evaluación utilizados para medir la exactitud de un clasificador.

Al ser TANAGRA un proyecto Open Source es fácilmente descargable desde su sitio web [34], el cual contiene tutoriales, referencias y conjuntos de datos.

5.6.5 AlphaMiner

AlphaMiner ha sido desarrollado por el *E-Business Technology Institute* (ETI) de la Universidad de Hong Kong [24] bajo el apoyo del Fondo para la Innovación y la Tecnología (ITF) [14] del Gobierno de la Región Especial Administrativa de Hong Kong (HKSAR).

AlphaMiner es un sistema de minería de Datos, *Open Source*, desarrollado en java. Es de propósito general pero más enfocado al entorno empresarial. Implementa algoritmos de asociación, clasificación, *clustering* y regresión logística. Posee una gama amplia de funcionalidad para el usuario, al realizar cualquier proceso de minería dando la posibilidad al usuario de escoger los pasos del proceso KDD que más se ajusten a sus necesidades, por medio de nodos que el analista integra a un árbol KDD siguiendo la metodología Drag & Drop. Es por eso que el proceso KDD en AlphaMiner no sigue un orden estructurado, sino que sigue la secuencia brindada por el propósito u objetivo del analista, además brinda la visualización estadística de distintas maneras, permitiendo un análisis más preciso por parte del analista.

El principal objetivo de AlphaMiner es la inteligencia Comercial Económica o Inteligencia de Negocios (BI - *Business Inteligence*) es uno de los medios más importantes para que las compañías tomen las decisiones comerciales más acertadas. Las soluciones de BI son costosas y sólo empresas grandes pueden permitirse el lujo de tenerlas. Por tanto las compañías pequeñas tienen una gran desventaja. Alpha-Miner proporciona las Tecnología de BI de forma Económica para dichas empresas para que den soporte a sus decisiones en el ambiente de negocio cambiante rápido.

AlphaMiner tiene dos componentes principales:

1. Una base de conocimiento.

2. Un árbol KDD, que permite integrar nodos artefacto que proporcionan varias funciones para crear, revisar, anular, interpretar y argumentar los distintos análisis de datos.

AlphaMiner implementa los siguientes pasos del proceso KDD:

1. Acceso de diferentes formas a las fuentes datos.
2. Exploración de datos de diferentes maneras.
3. Preparación de datos.
4. Vinculación de los distintos modelos de minería.
5. Análisis a partir de los modelos.
6. Despliegue de modelos al entorno empresarial.

La característica más importante de AlphaMiner, es su capacidad para almacenar los datos, después de aplicar las técnicas de minería de datos, en una base de conocimiento, que puede ser reutilizada. Esta función aumenta su utilidad significativamente, y brinda un gran apoyo logístico al nivel estratégico de una empresa. Aventajando cualquier sistema tradicional de manipulación de datos. AlphaMiner proporciona una funcionalidad adicional para construir los modelos de minería de Datos, conformando una sinergia entre los distintos algoritmos de minería.

AlphaMiner se asemeja a Tariy en varias características, por un lado el lenguaje de programación en el que fue implementado es JAVA, por otro lado es *OpenSource*, tiene conectividad JDBC con los distintos gestores de bases de datos, además de conectividad con archivos de Excel. Otra semejanza es el tipo de formato que presenta para el manejo de tablas univaluadas en algoritmos de asociación, específicamente en bases de datos de supermercados, ya que después de escoger los campos de dicha base de datos se la lleva a un formato de dos columnas, una para la correspondiente transacción y otra para el ítem.

5.6.6 YALE - *Yet Another Learning Environment*

YALE [29, 47] es un entorno para la realización de experimentos de aprendizaje automático y minería de Datos. A través de un gran número de operadores se pueden crear los experimentos, los cuales pueden ser anidados arbitrariamente y cuya configuración es descrita a través de archivos XML que pueden ser fácilmente creados por medio de una interfaz gráfica de usuario. Las aplicaciones de YALE cubren tanto investigación como tareas de minería de Datos del mundo real.

Desde el año 2001 YALE ha sido desarrollado en la Unidad de Inteligencia Artificíal de la Universidad de Dortmund [48] en conjunto con el Centro de Investigación Colaborativa en Inteligencia Computacional (Sonderforschungsbereich 531).

El concepto de operador modular permite el diseño de cadenas complejas de operadores anidados para el desarrollo de un gran número de problemas de aprendizaje. El manejo de los datos es transparente para los operadores. Ellos no tienen nada que ver con el formato de datos o con las diferentes presentaciones de los mismos. El *kernel* de Yale se hace a cargo de las transformaciones necesarias. Yale es ampliamente usada por investigadores y compañías de minería de Datos.

Modelando Procesos De Descubrimiento De Conocimiento Como Arboles De Operadores con YALE. Los procesos de descubrimiento de conocimiento son vistos frecuentemente como invocaciones secuenciales de métodos simples. Por ejemplo, después de cargar datos, seo podría aplicar un paso de preprocesamiento seguido de una invocación a un método de clasificación. El resultado en este caso es el modelo aprendido, el cual puede ser aplicado a datos nuevos o no revisados todavía. Una posible abstracción de estos métodos simples es el concepto de operadores.

Cada operador recibe su entrada, desempeña una determinada función y entrega una salida. Desde ahí, el método secuencial de invocaciones corresponde a una cadena de operadores. Aunque este modelo de cadena es suficiente para la realización de muchas tareas básicas de descubrimiento de conocimiento, estas cadenas planas son a menudo insuficientes para el modelado de procesos de descubrimiento de conocimiento más complejas

Una aproximación común para la realización del diseño de experimentos más complejos es diseñar las combinaciones de operadores como un gráfico direccionado. Cada vértice del gráfico corresponde a un operador simple. Si dos operadores se conectan, la salida del primer operador se usaría como entrada en el segundo operador. Por un lado, diseñar procesos de descubrimiento de conocimiento con la ayuda de gráficos direccionados es muy poderoso. Por el otro lado, existe una desventaja principal: debido a la pérdida de restricciones y la necesidad de ordenar topológicamente el diseño de experimentos es a menudo poco intuitivo y las validaciones automáticas son difíciles de hacer.

Yale ofrece un compromiso entre la simplicidad de cadenas de operadores y la potencia de los gráficos direccionados a través del modelado de procesos de descubrimiento de conocimiento por medio de árboles de operadores. Al igual que los lenguajes de programación, el uso de árboles de operadores permite el uso de conceptos como ciclos, condiciones, u otros esquemas de aplicación. Las hojas en el árbol de operadores corresponden a pasos sencillos en el proceso modelado como el aprendizaje de un modelo de predicción o la aplicación de un filtro de preprocesamiento. Los nodos interiores del árbol corresponden a pasos más complejos o abstractos en el proceso. Esto es a menudo necesario si los hijos deben ser aplicados varias veces como, por ejemplo, los ciclos. En general, los nodos de los operadores internos definen el flujo de datos a través de sus hijos. La raíz del árbol corresponde al experimento completo.

¿Qué Puede Hacer YALE? YALE provee más de 200 operadores incluyendo algoritmos de aprendizaje automático Un gran número de esquemas de aprendizaje para tareas de regresión y clasificación incluyendo máquinas de Vectores de Soporte (SVM), árboles de decisión e inductores de reglas, operadores *Lazy*, operadores Bayesianos y operadores Logísticos. Varios operadores para la minería de reglas de asociación y *clustering* son también parte de YALE. Además, se han adicionado varios esquemas de Meta Aprendizaje.

Operadores WEKA. Todos los esquemas de aprendizaje y evaluadores de atributos del entorno de aprendizaje WEKA también están disponibles y pueden ser usados como cualquier otro operador de YALE.

5.7 Tecnología utilizada en la Implementación de Tariykdd

Durante la implementación del proyecto TariyKDD fueron utilizadas una serie de herramientas para su desarrollo. Entre las principales podemos nombrar el lenguaje de programación Java, el entorno de desarrollo NetBeans y las bibliotecas gráfica *Swing* y *Core Prefuse* de *Procesing* para Java. Además de software externo, que no se utilizó para el desarrollo, pero que fue fundamental para la implementación de la herramienta, por ejemplo, motores de bases de datos con los cuales la herramienta se conecta, com lo son: Postgres, MySQL y Oracle, también se utilizo gran variedad de herramientas de software libre que hicieron su aporte pero no están implícitas en el desarrollo por lo cual no serán abordadas como objeto de estudio.

5.7.1 Lenguaje de Programación Java

Java es un lenguaje de programación orientado a objetos y, a diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es ejecutado (usando normalmente un compilador JIT - *Just In Time*), por una máquina virtual Java, lo cual hace de él un lenguaje multiplataforma.

5.7.2 Entorno de Desarrollo NetBeans

NetBeans comprende una plataforma para el desarrollo de aplicaciones de escritorio usando Java y un Entorno integrado de desarrollo (IDE) desarrollado usando la Plataforma NetBeans.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

5.7.3 Controlador JDBC

JDBC es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

5.7.4 Controlador de Protocolo Nativo (tipo 4)

El controlador traduce directamente las llamadas al API JDBC al protocolo nativo de la base de datos figura 5.5. Es el controlador que tiene mejor rendimiento, pero está más ligado a la base de datos que empleemos que el controlador tipo JDBC-Net, donde el uso del servidor intermedio nos da una gran flexibilidad a la hora de cambiar de base de datos. Este tipo de controladores también emplea Tecnología 100 % Java.

5.7.5 Bibliotecas gráficas Swing y Core Prefuse de Processing para Java

Swing es una biblioteca gráfica para Java que forma parte de las *Java Foundation Classes* (JFC). Incluye componentes para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas. Mientras que *Processing* [40] es un lenguaje y entorno de programación de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.

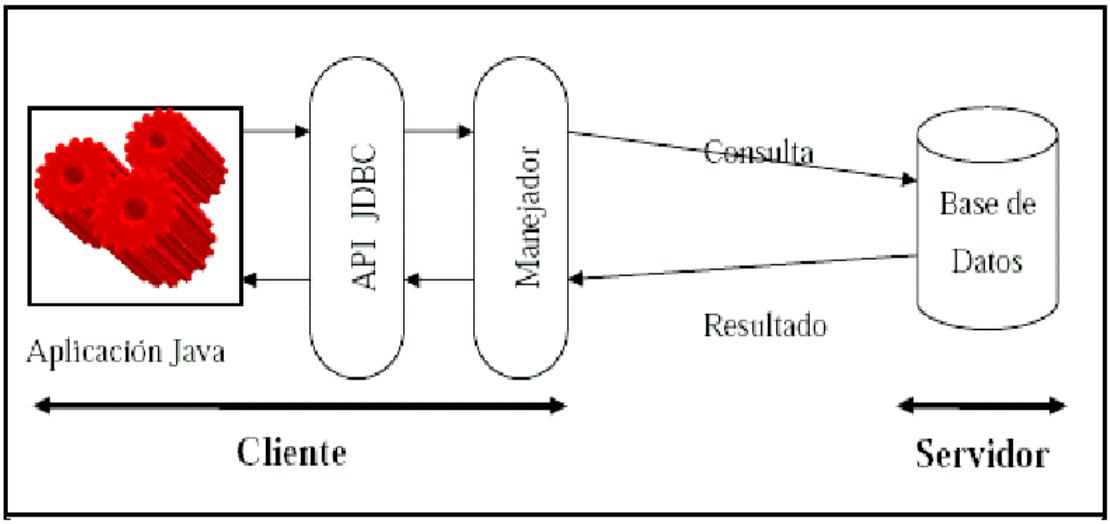

Figura 5.5: Controlador JDBC tipo 4

6. DESARROLLO DEL PROYECTO

6.1 Diseño de TariyKDD

6.1.1 Diagramas de Casos de Uso

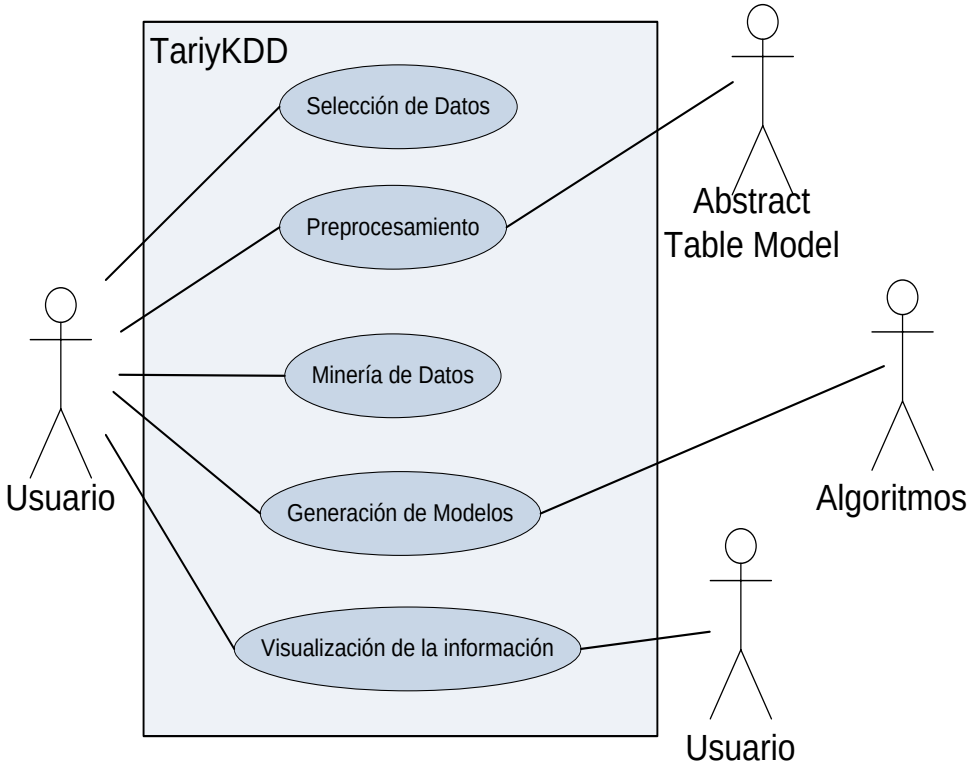

Figura 6.1: Diagrama General de TariyKDD

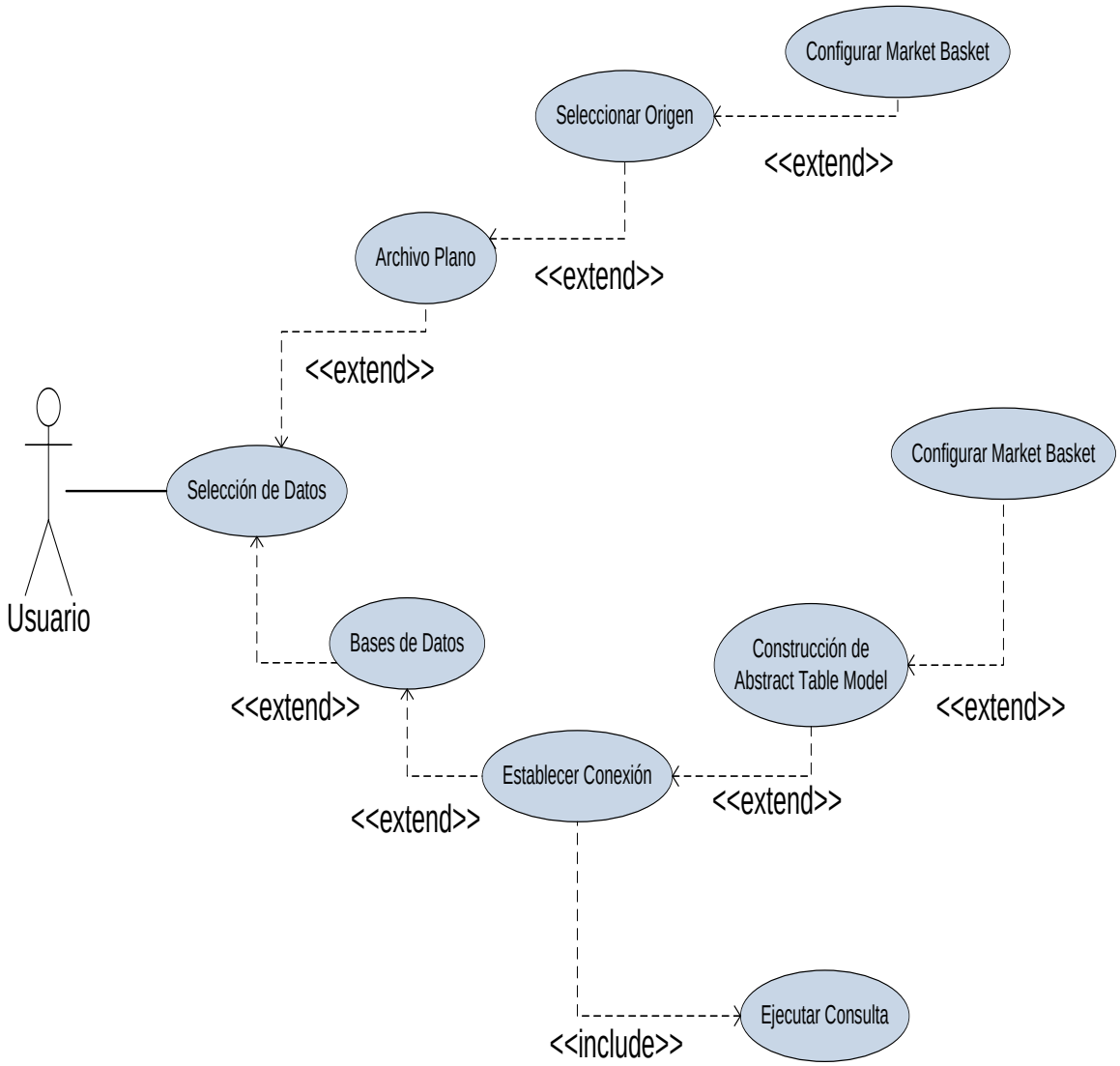

Figura 6.2: Selección de datos

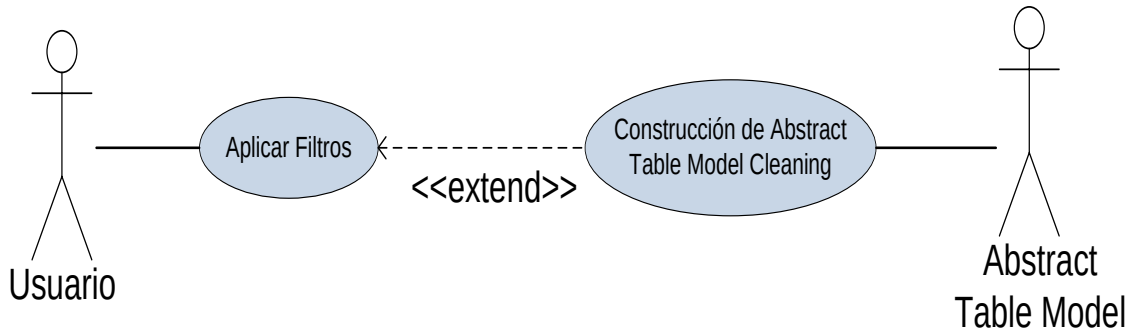

Figura 6.3: Preprocesamiento

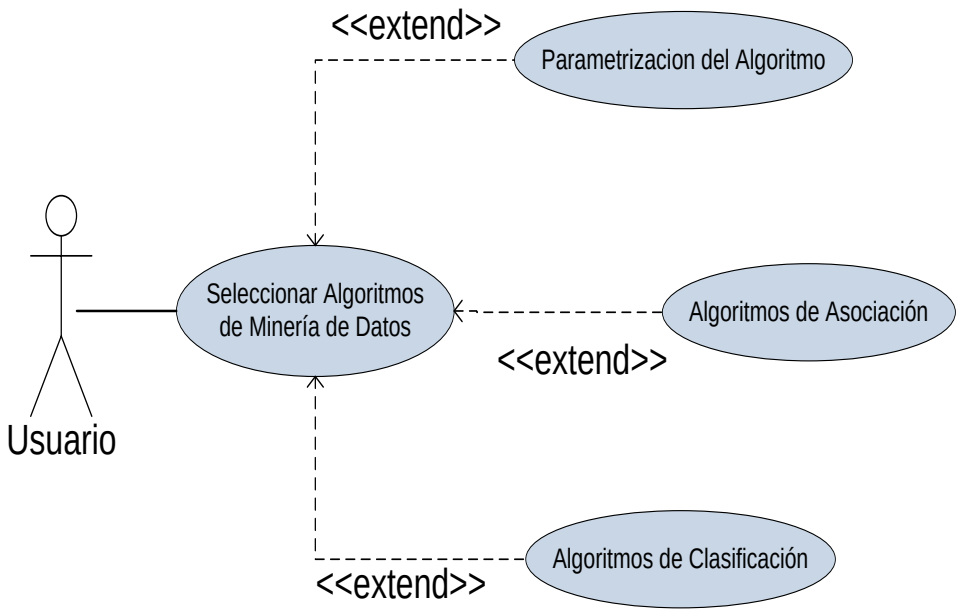

Figura 6.4: Minería de Datos

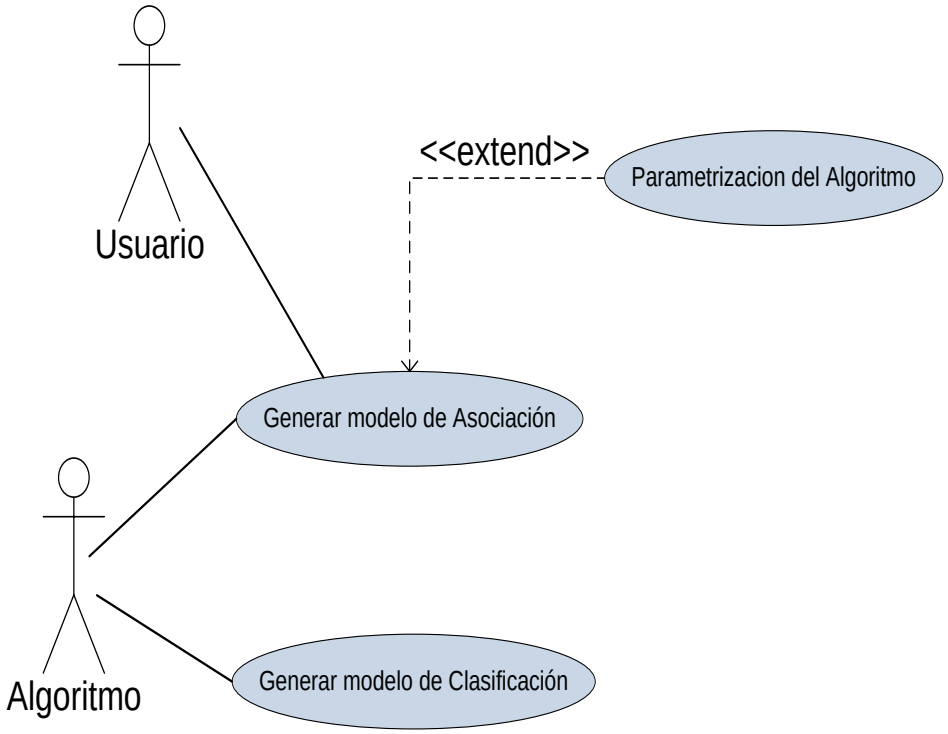

Figura 6.5: Generación de Modelos de Minería de Datos

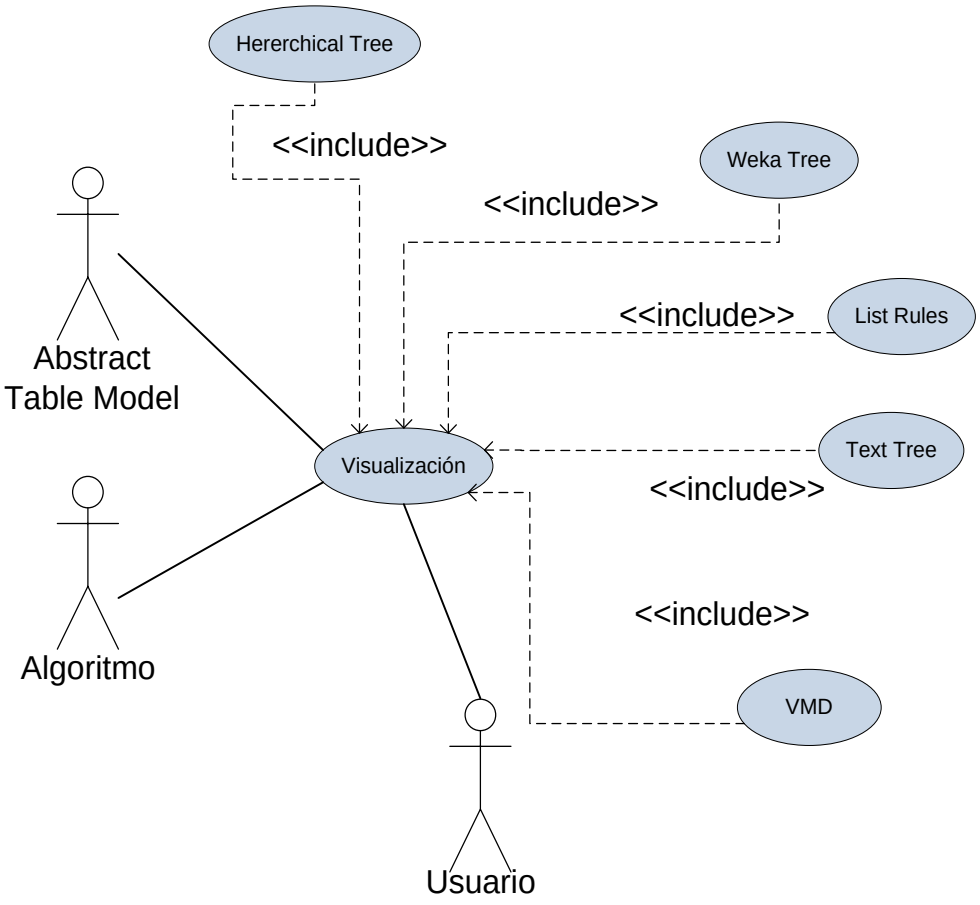

Figura 6.6: Visualización de la información

6.1.2 Diagramas de Clase

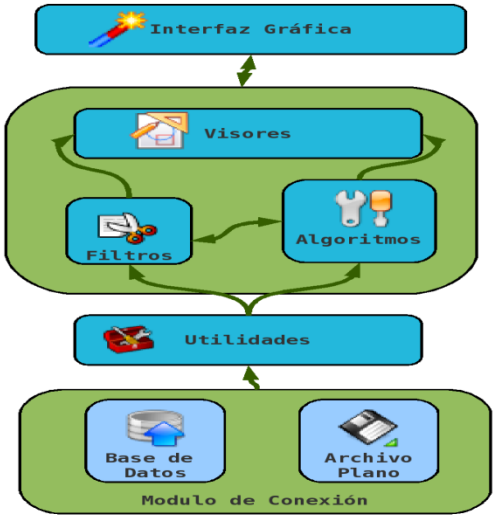

Figura 6.7: Arquitectura de capas de *TariyKDD*

Figura 6.8: Paquete KnowledgeFlow

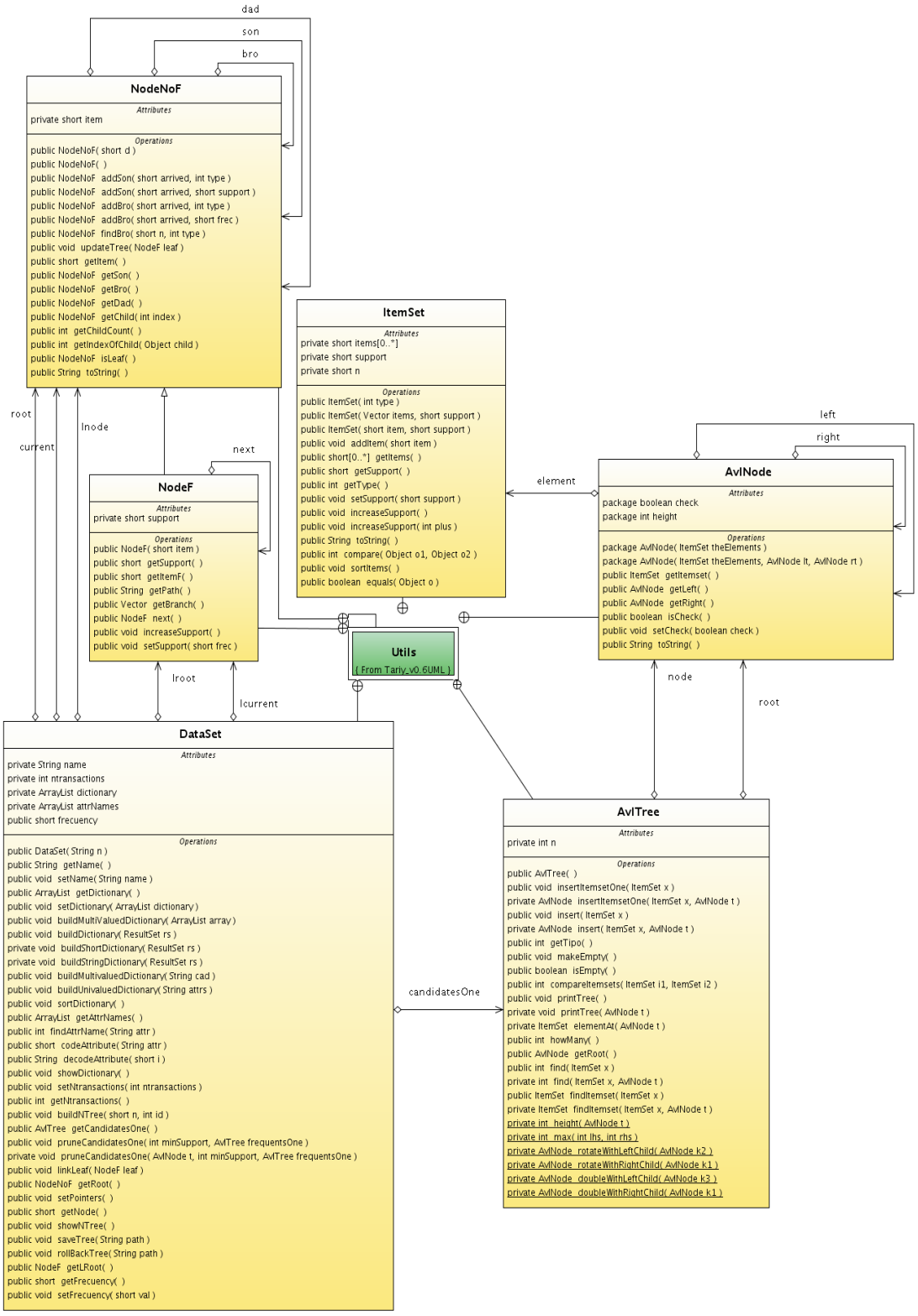

Figura 6.9: Paquete Utils

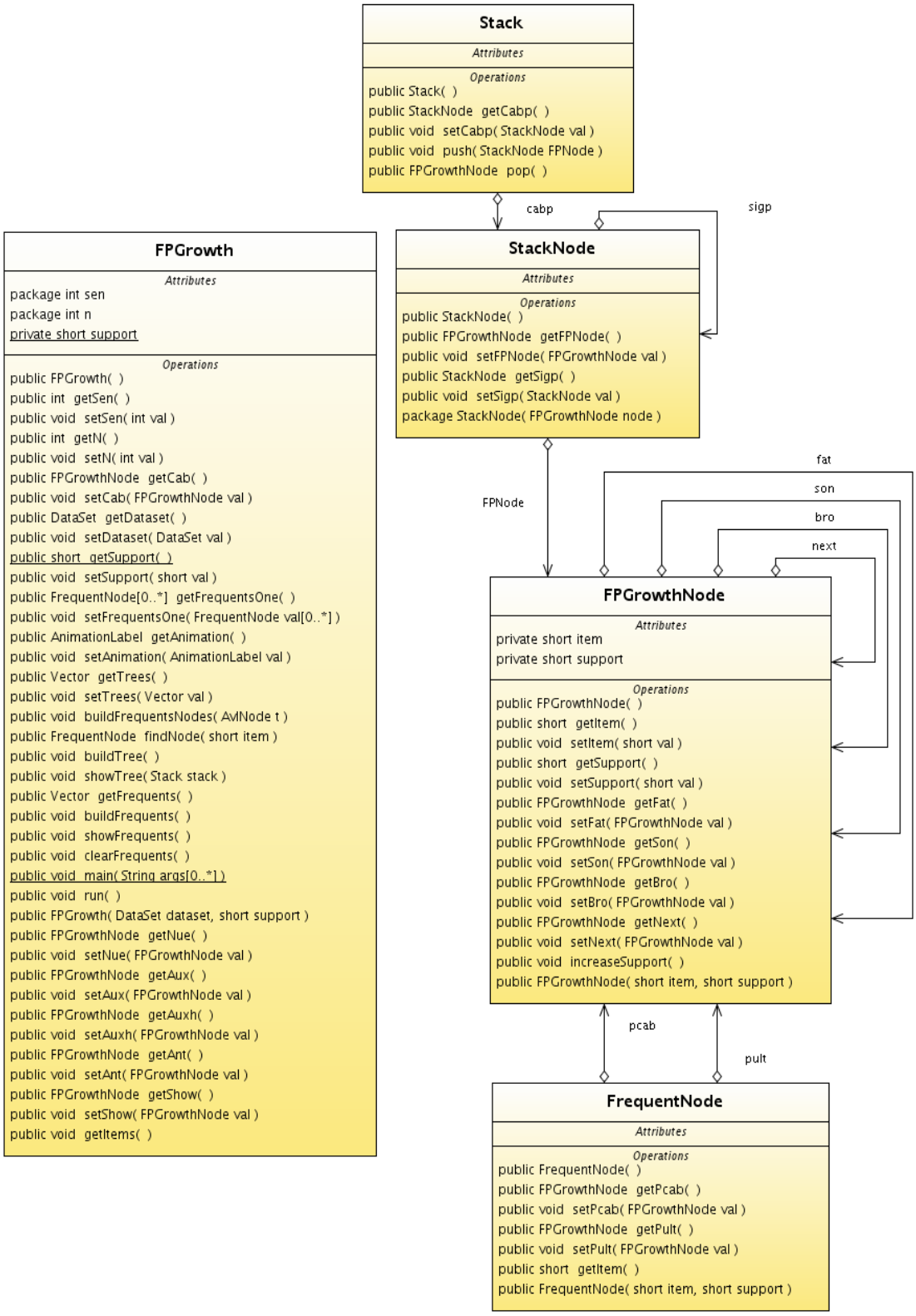

Figura 6.10: Paquete FPGrowth

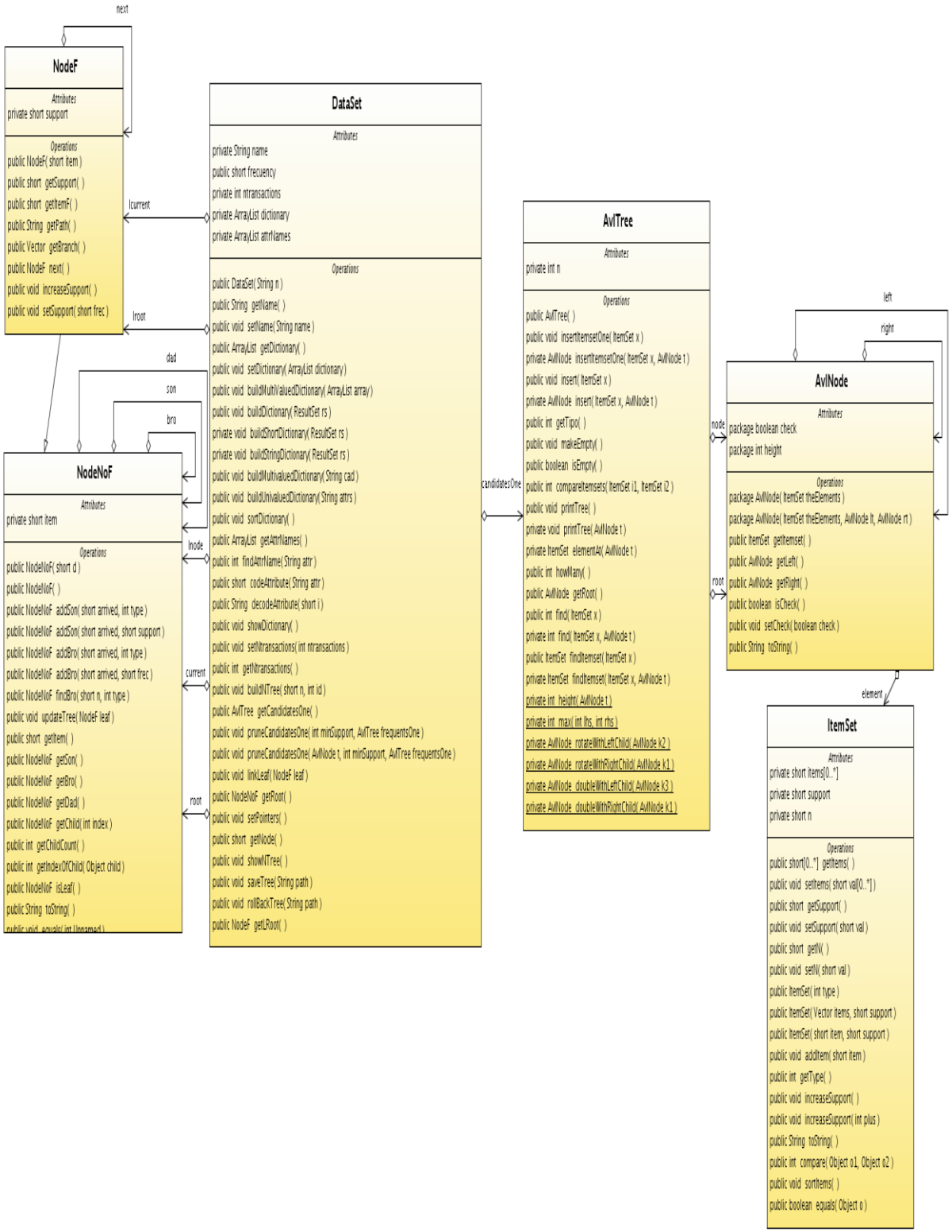

Figura 6.11: Paquete EquipAsso

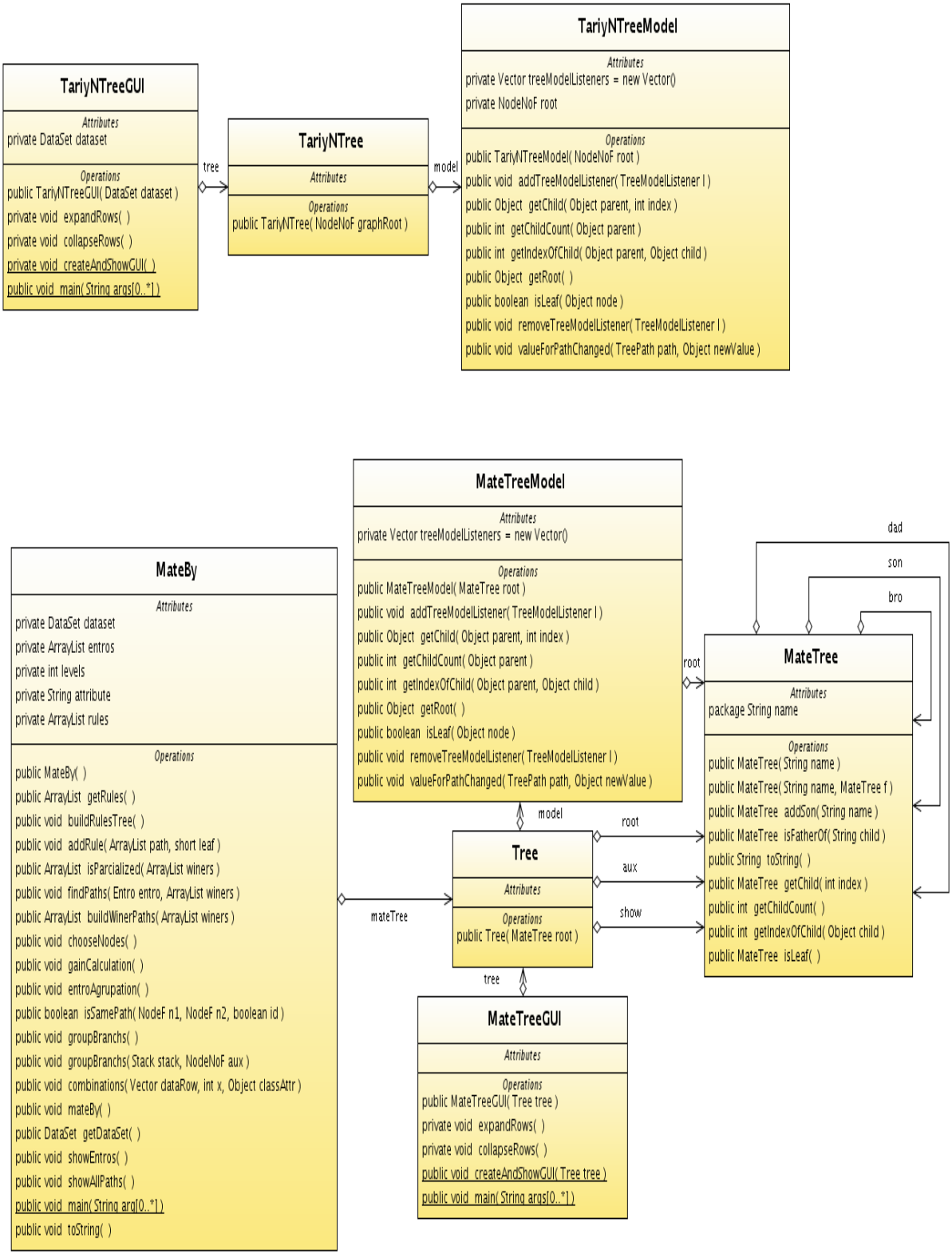

Figura 6.12: Paquete Mate

6.1.3 Diagramas de Paquetes

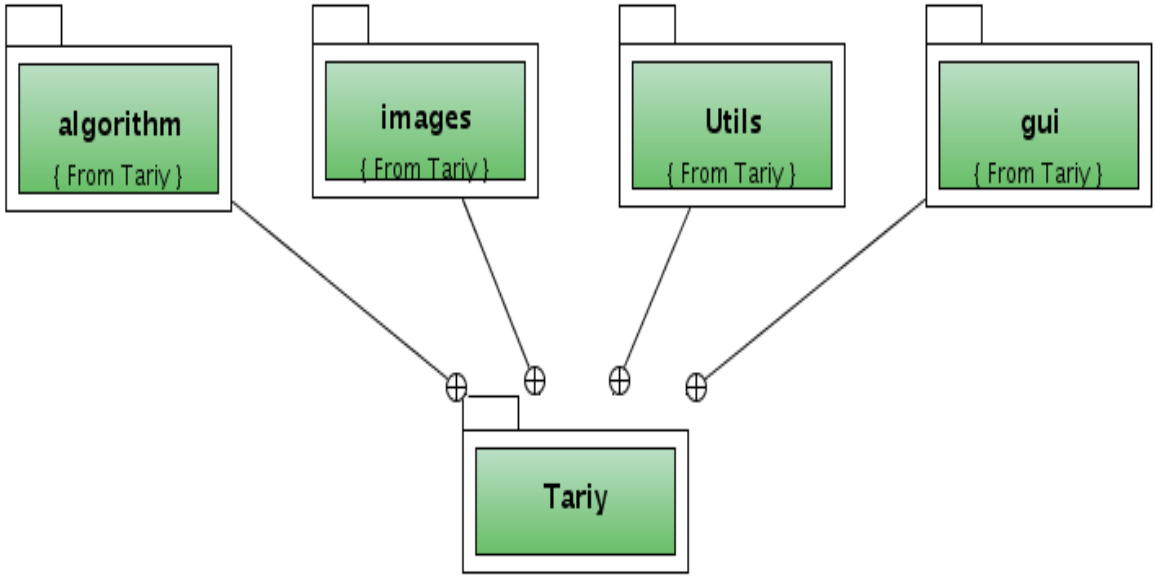

Figura 6.13: Paquete Principal

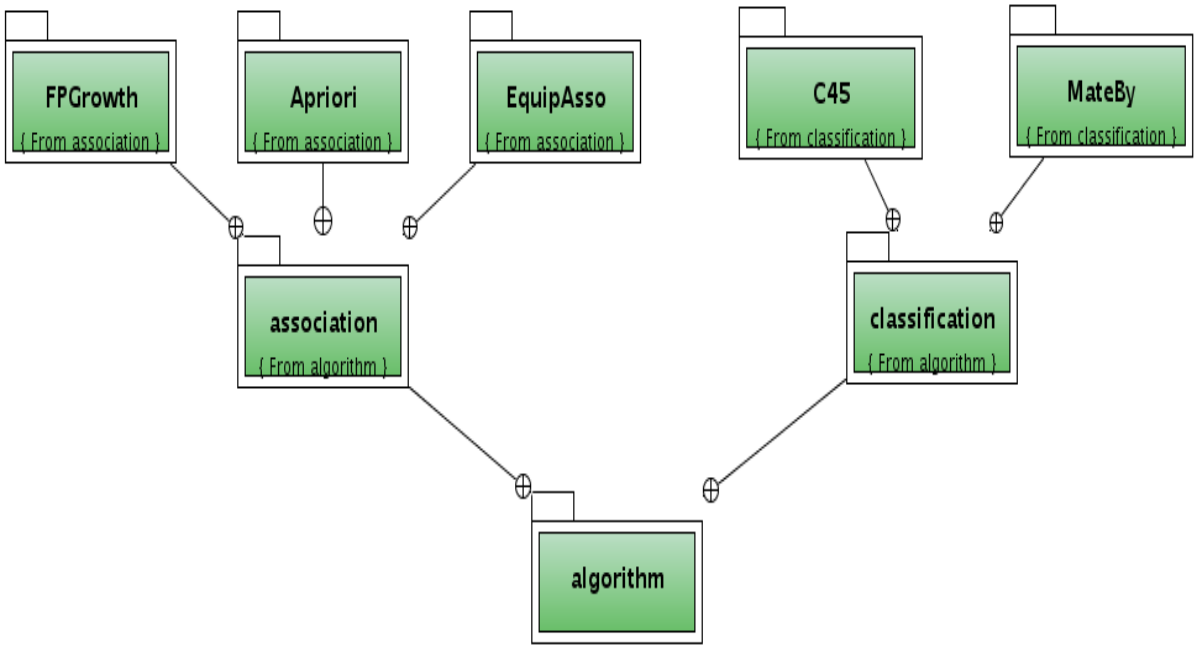

Figura 6.14: Paquete Algoritmos

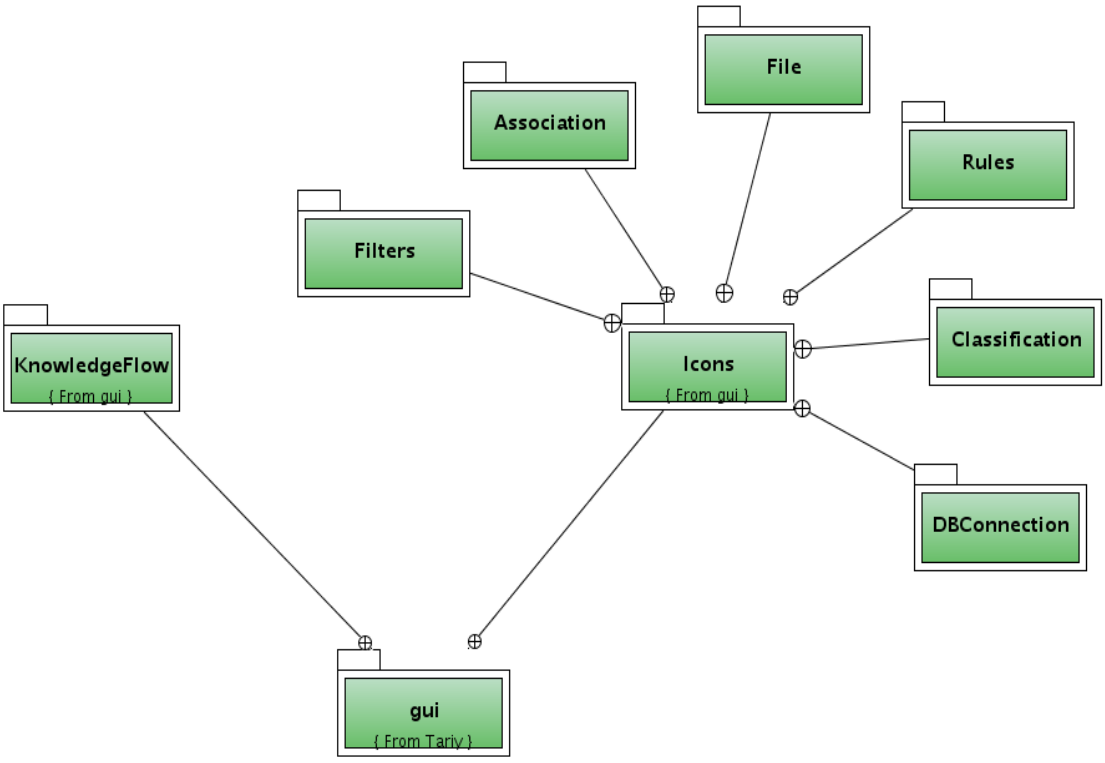

Figura 6.15: Paquete Interfaz Gráfica

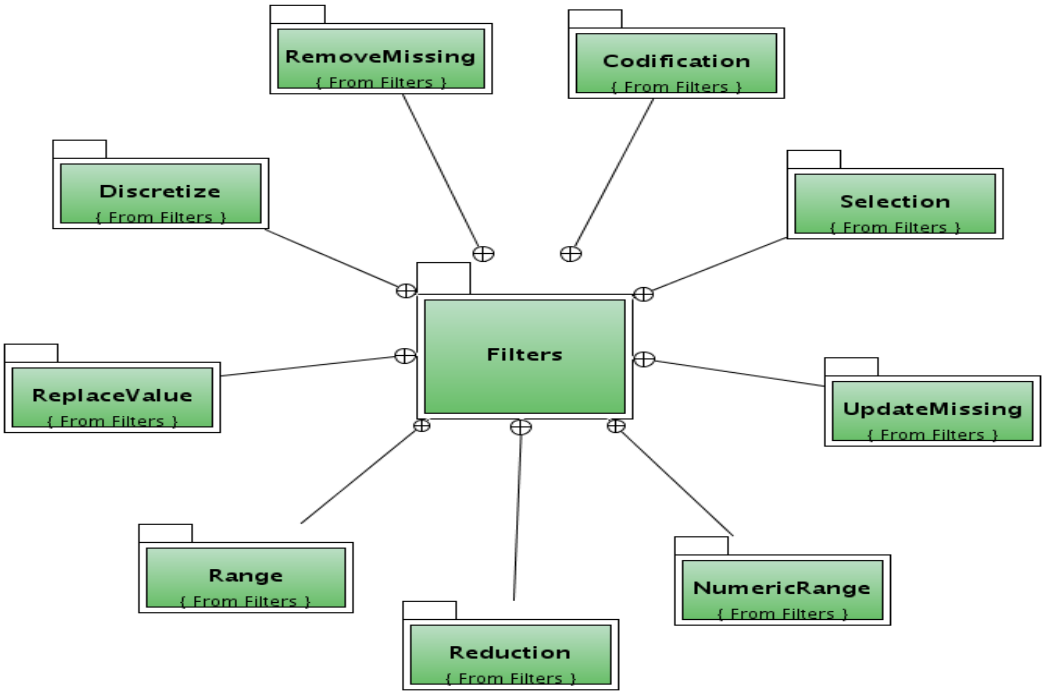

Figura 6.16: Paquete GUI Filtros

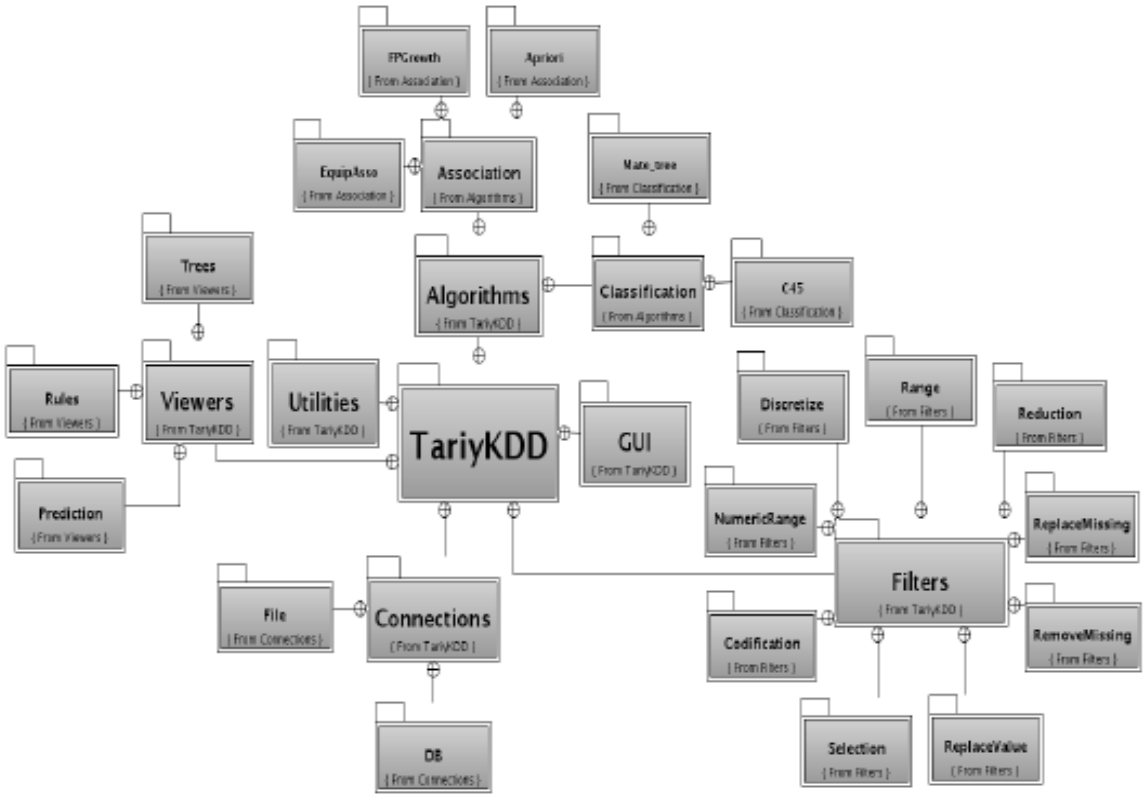

Figura 6.17: Diagrama general de paquetes

7. IMPLEMENTACIÓN

7.1 Plataforma de desarrollo de TariyKDD

Para el desarrollo de *TariyKDD* se utilizaron computadores con procesador Intel Core 2 Duo de 64 bits, 4Gb de RAM, ya que la Minería de Datos requiere grandes cantidades de memoria por el tamaño de los conjuntos de datos, disco duro Serial ATA, útil al tomar los datos desde un repositorio y al momento de realizar pruebas de rendimiento de los algoritmos, por su velocidad de transferencia, Los sistema operativos sobre los cuales se trab0ajó durante la implementación de *TariyKDD* son *Fedora Core* y Windows Vista y 7 de 64 bits. El lenguaje de programación en el que está elaborado *TariyKDD* es Java 5.0, actualización 20, con entorno de desarrollo NetBeans 6.7.1.

Dentro del proceso de Descubrimiento de Conocimiento, *TariyKDD* comprende las etapas de Selección, *Preprocesamiento*, Minería de Datos y Visualización de Resultados. De esta forma la implementación de la herramienta se hizo a través de los siguientes módulos de software cuya estructura se muestra en la figura 7.1.

7.2 Descripción de Desarrollo de TariyKDD

En primera instancia se describe de manera general la estructura de *TariyKDD* para dar una idea global de cómo fue implementada esta herramienta. A continuación, se amplía y se detalla la forma en que *TariyKDD* fue desarrollada.

- ***utils*:** Utilidades de *TariyKDD*. Dentro de este paquete encontramos clases como *DataSet*.

- ***algorithm*:** Esta compuesta de los paquetes *association* y *classification*, los cuales implementan mediante sus respectivas clases los algoritmos de asociación (Apriori, *FPgrowth y EquipAsso*) y clasificación (*C4.5 y MateBy*).
- ***gui*:** Comprende los paquetes *Icons y KnowledgeFlow*, los cuales a través de sus clases implementan la interfaz gráfica de la herramienta.

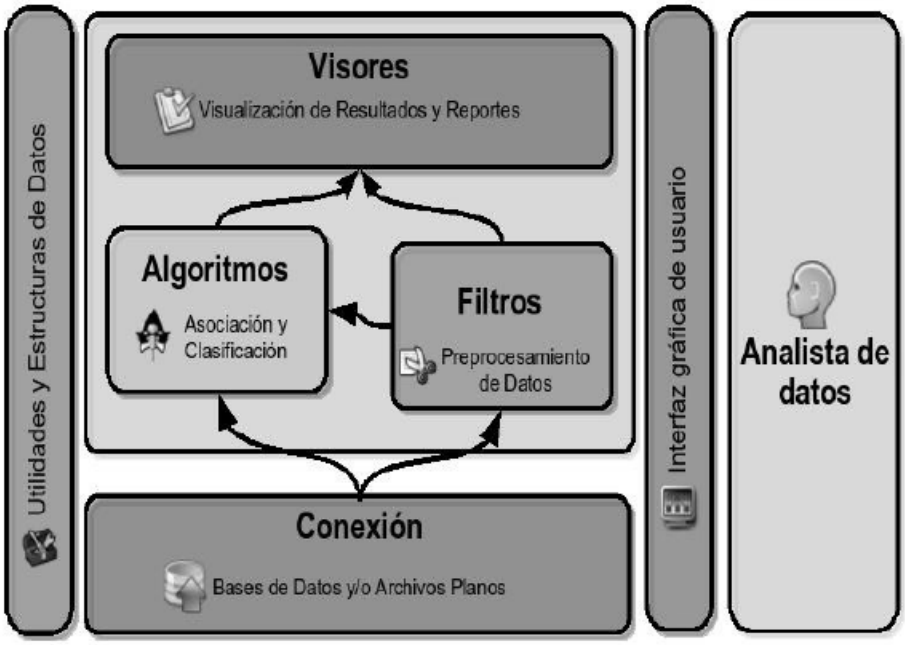

Figura 7.1: Módulos de software en *TariyKDD*

7.2.1 Paquete Utils

Esta clase está compuesta por las siguientes clases**:**

Clase AssocRules. Esta clase es la encargada de generar reglas a partir de los árboles de *itemsets* frecuentes que generan los algoritmos de asociación. Las reglas se generan teniendo en cuenta el parámetro confianza.

Clase AvlNode. En esta clase se construye la estructura interna de datos de un nodo perteneciente a un árbol *Avl*. Los atributos que maneja esta clase son los siguientes:

> ***element* de tipo itemset**: aquí se guardan los datos con los que se carga el nodo.
> ***left, right* de tipo Avlnode**: punteros hacia los hijos izquierdo y derecho respectivamente.
> ***height*** de tipo entero: altura del árbol Avl.
> ***check* de tipo booleano**: variable de control del balance del árbol.

Clase AvlTree. Esta clase es la encargada tanto de crear o armar un árbol *Avl* como de proveer los

métodos necesarios para su manejo. Un árbol *Avl* es un tipo de árbol binario que es balanceado de tal manera que la profundidad de dos hojas cualquiera en el árbol difiera como máximo en uno.

Clase BaseDatos. Esta clase es utilizada siempre que se necesite efectuar conexiones a bases de datos por medio del objeto de conexión java para *un SGBD*. Cabe aclarar que el sistema gestor de bases de datos (SGBD) inicial es *Postgres* pero el nombre del sistema gestor es totalmente parametrizable, permitiendo de esta manera la conexión a casi cualquier SGBD.

Clase fileManager. Esta clase se encarga de todo lo que tiene que ver con el manejo de archivos de acceso aleatorio y el flujo de información entre el archivo leído, el *DataSet* y el diccionario de datos.

Clase itemset. Esta clase se encarga del manejo de los conjuntos de ítems. Un *itemset* puede ser entendido como una transacción. Los atributos de esta clase son:

Clase NodeNoF. Esta clase se encarga de crear los nodos intermedios de un árbol de datos. También se implementan los métodos para enlazar un nodo a otro. Entre ellos se encuentran: *addSon, addBro y findBro* que se encargan de adicionar un hijo, adicionar un hermano y buscar si un nodo tiene hermanos respectivamente.

Clase NodeF. Esta clase extiende a la clase *NodeNoF*. Este tipo de nodos son las hojas de las ramas. Lo que las hace diferentes, son dos atributos adicionales, uno es el soporte, que se encarga de llevar a cabo el conteo de veces que una transacción repetida ha sido tenida en cuenta, y el otro es un puntero a otra hoja. En el momento de recorrer un árbol, el tipo de nodo nos permite saber que hemos llegado al final de una rama.

Clase Transaction. Esta clase es la encargada de manejar los *itemsets* o conjuntos de ítems por cada transacción. Cada uno de los *itemsets* son almacenados en vectores. Esta clase se encarga de alimentar esos vectores, permite hacer consulta sobre ellos u organizarlos.

7.2.2 Paquete algorithm

Este paquete está compuesto a su vez por los paquetes *association y classification*, los cuales implementan los algoritmos de Asociación (*Apriori, FPgrowth y EquipAsso*) y Clasificación (*C4.5 y MateBy*).

7.2.3 Paquete association

El paquete *association* está conformado por los paquetes que implementan los algoritmos de asociación y que se explican a continuación:

Paquete A priori. La implementación del algoritmo *apriori* se realizó utilizando una clase denominada *apriori.java* que interactúa directamente con otras clases definidas en el paquete *Utils* como *DataSet, Transaction, AvlTree e itemset*. Al igual que las otras clases que implementan algoritmos de asociación, en el constructor de la clase *Apriori* se usan 2 parámetros: una instancia de la clase *DataSet*, donde vienen comprimidos el conjunto de datos desde el módulo de conexión o el módulo de filtros, y un entero corto, que es suministrado por el usuario, que será usado como el soporte del sistema durante la ejecución de este algoritmo.

De igual manera, en la construcción de la clase se instancia e inicializa un objeto de la clase *AvlTree* que almacenará los *itemsets* frecuentes tipo 1, el cual es alimentado haciendo uso del método *pruneCandidatesOne* de la clase *DataSet*, el cual recibe el soporte del sistema como parámetro. En este punto también se instancia e inicializa un arreglo que guardará los diferentes árboles balanceados (*AvlTree*), que contendrán los *itemsets* Frecuentes de cada tipo y que serán calculados por la herramienta. Al disponer ya de los *itemsets* frecuentes tipo 1, estos son almacenados en la posición 0 de este arreglo.

Para disparar la ejecución del algoritmo usamos el método *run* el cual ejecuta un ciclo dentro del cual el método *makeCandidates* se encarga de generar todos los *itemset* candidatos para cada iteración. Para esto el método *makeCandidates* llama al objeto *auxTree*, instancia de la clase *AvlTree* que conserva el ultimo árbol de *itemset* frecuentes y pregunta cuantos *itemsets* tiene en ese instante utilizando el método *howMany* de esta clase.

Para cada elemento encontrado dentro del árbol de *itemsets* frecuentes se recorre un ciclo con los demás miembros del árbol formando combinaciones que generan un nuevo *itemset* candidato haciendo uso del método *combinations* que recibe como *parametros* el árbol *auxTree* de *itemset* Frecuentes y un árbol *AvlTree* donde se guardaran los *itemsets* Frecuentes tipo $k + 1$ llamando *frequents*. Se aprovecha la ventaja de que los *itemsets* están ordenados en el árbol y se verifica que el *itemset* evaluado coincida en sus $n - 1$ primeros ítems, siendo n el tamaño de ese *itemset*, con el próximo *itemset* del árbol. De ser así se instancia un nuevo *itemset* con los $n - 1$ ítems coincidentes y los 2 últimos ítems de cada *itemset* involucrado. De esta manera se generan *itemsets* candidatos de tamaño $n + 1$.

Cada *itemset* candidato es pasado como parámetro al método *increaseSupport* donde es contado el número de ocurrencias de ese *itemset* candidato dentro del conjunto de datos. En este punto se hace una consideración importante y si los *itemsets* candidatos que se están construyendo son de tipo 3 o superior se evalúa el paso de poda, esto es, se descompone el *itemset* candidato construido en sus $n - 2$ posibles combinaciones del tipo anterior, siendo n el tamaño del *itemset* candidato que se está generando. No se evalúan las dos últimas combinaciones pues fueron éstas las que generaron el candidato que se está analizando y se tiene certeza de que existen en el árbol de *itemsets* Frecuentes. Cada una de las combinaciones generadas son buscadas en *auxTree*, hay que recordar que este es el árbol que contiene todos los *itemsets* frecuentes del orden anterior al que se está generando. Si una de las combinaciones del *itemset* candidato actual no es encontrada en el árbol *auxTree*, este *itemset* ya no es considerado y se procede a evaluar el próximo.

Si el *itemset* candidato actual es de tipo 2 o ha superado el paso de poda se procederá a hacer su conteo. Para ello, se hace una nueva instancia de la clase *Transaction* donde se carga una a una las transacciones del conjunto de datos y se compara con el contenido del *itemset*. Si coinciden los elementos del registro y el *itemset*, este último incrementa su soporte interno en uno.

Al final de este proceso se cuenta con un *itemset* candidato con su soporte establecido, este se compara con el soporte del sistema y de superarlo es incluido en el árbol *frequents*. Cuando ya se han evaluado todos los candidatos para un árbol *auxTree*, se pregunta si se han generado nuevos *itemsets* frecuentes en *frequents*. Si el método *howMany* de *frequents* arroja existencias se almacena en un array de una dimensión de árboles frecuentes y se asigna a *auxTree* el árbol *frequents* para iniciar de nuevo el proceso de generación de candidatos. Esto se hará hasta que el árbol *frequents* resulte vacío (no hayan nuevos *itemsets* frecuentes). En el arreglo de árboles frecuentes quedan almacenados todos los *itemsets* frecuentes organizados por tipo que serán pasados al Módulo de

visores para ser visualizados como reglas.

Paquete FPgrowth. Las clases que implementan el algoritmo *FPgrowth* se encuentran agrupadas en el paquete con el mismo nombre y las más importantes para su funcionamiento son *FPGrowth, FPGrowthNode, FrequentNode, BaseConditional, BaseConditionals, Conditionals y AvlTree*.

El algoritmo *FPGrowth* se basa en la generación de *itemsets* candidatos a partir de un árbol N-Ario. Para comprender mejor la implementación de *FPGrowth*, se deben revisar primero las clases que forman la estructura del árbol *N-Ario*, las cuales han sido llamadas *FrequentNode* y *FPGrowthNode*.

Clase FrequentNode. Para la posterior creación de *itemsets* frecuentes se almacenan en un arreglo de objetos de tipo *FrequentNode* los *itemsets* frecuentes-1 o de tamaño 1. Cada uno de los ítems frecuentes-1 de esta lista se encuentra enlazados a un nodo del árbol N-ario, de ahora en adelante *FPtree*, que tenga el mismo nombre que el *itemset* frecuente-1.

Clase FPGrowthNode. Esta clase proporciona la estructura del *FPTree*, en donde cada nodo tiene un valor de *ítem*, un soporte y los punteros respectivos para realizar los enlaces entre nodos (Punteros al nodo hijo, al nodo padre, al nodo hermano y al siguiente nodo con el mismo nombre).

Clase FPGrowth. FPGrowth es la clase principal del paquete, tiene los métodos más importantes del algoritmo y a través de los cuales se obtienen los *itemsets* frecuentes.

Los parámetros que el algoritmo *FPGrowth* recibe en su constructor son, *DataSet* y el soporte suministrado por el usuario. El primer paso de la implementación de *FPGrowth* es crear la lista con los *itemsets* frecuentes-1, la cual se almacena en el arreglo *frequentsOne*. El siguiente paso es construir el *FPtree*, para esto se lee cada una de las transacciones de *DataSet* y para aquellos ítems cuyo soporte sea mayor o igual al mínimo se los almacena en la clase del paquete *Utils Transaction*. A partir de estos ítems filtrados se construye el *FPtree*.

Para una mejor compresión sobre la construcción del *FPTree* se puede observar el conjunto de datos de la tabla 7.1 y su representación gráfica en la figura 7.2

Como se puede observar en la figura 7.2, cada uno de los ítems frecuentes-1 se encuentra enlazado a un nodo del *FPtree*, por ejemplo el ítem frecuente-1, 5 se encuentra enlazado al nodo de *FPtree* 5, cuyo soporte es 1 y a la vez cada nodo de *FPtree* se enlaza al siguiente nodo con el mismo nombre, en el caso del nodo 5, este se enlaza a otro nodo con valor 5 y cuyo soporte es 1.

El siguiente paso de la implementación cosiste en recorrer la estructura de los ítems frecuentes-1 y por cada uno de estos construir sus Patrones Condicionales Base, los cuales se obtienen recorriendo *FPtree* desde cada nodo enlazado por los ítems frecuentes-1 a través de su puntero al nodo padre hasta la raíz. Por ejemplo, como se puede observar en la figura 7.2, para el ítem frecuente-1, 5 sus Patrones Condicionales Base son dos: (1,2:1) y (3,1,2:1), donde el número después de ":" es el soporte del Patrón, el cual corresponde al mismo soporte que tenga el ítem frecuente-1 en cuestión. Cada uno de los Patrones Condicionales Base se almacenan en la clase *BaseConditional* y todo el conjunto de Patrones Condicionales Base de un ítem frecuente-1 se almacenan en la clase *BaseConditionals*. En la tabla 7.2 se pueden observar los Patrones Condicionales Base de cada uno de los ítems frecuentes-1.

Tabla 7.1: Conjunto de datos transaccional

Transacción	Lista de items
T01	{(1,2,5)}
T02	{(2,4)}
T03	{(2,3)}
T04	{(1,2,4)}
T05	{(1,3)}
T06	{(2,3)}
T07	{(1,3)}
T08	{(1,2,3,5)}
T09	{(1,2,3)}

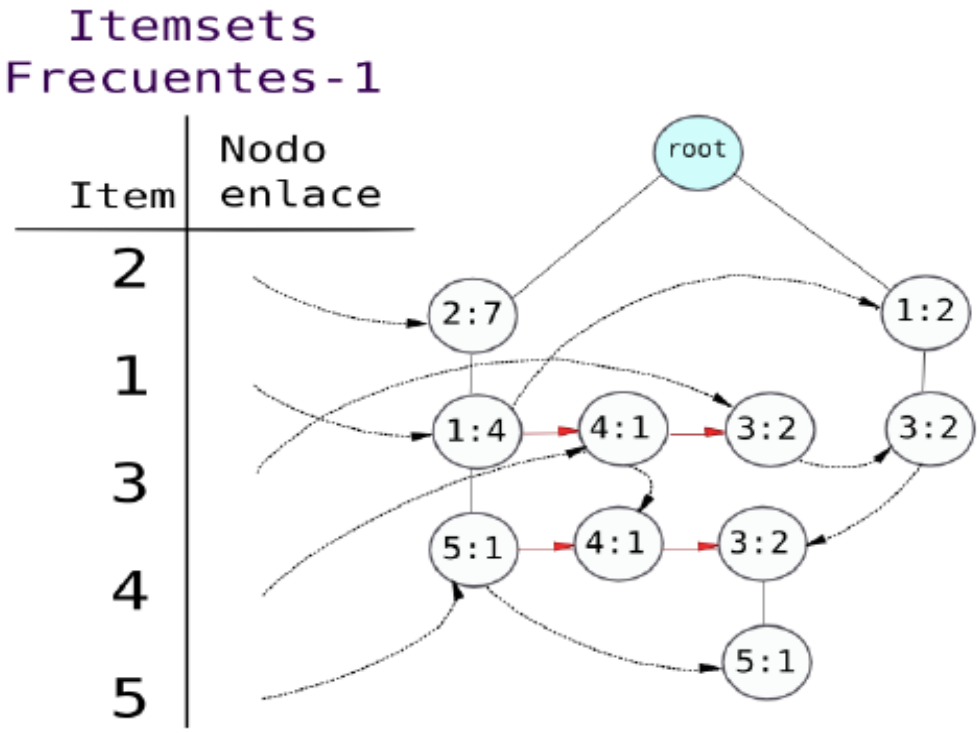

Figura 7.2: Árbol *Fptree*

Cuando se han construido todos los Patrones Condicionales Base de un ítem frecuente-1, el siguiente paso de la implementación es determinar cuáles de sus ítems tienen soporte mayor o igual al mínimo. Para esto se debe sumar cada uno de los soportes que un ítem tenga en cada Patrón Condicional Base, los ítems que cumplan con el soporte van a conformar los Patrones Condicionales, los cuales se almacenan en la clase *Conditionals*, por ejemplo, para un soporte mínimo igual a 2, los Patrones Condicionales del ítem frecuente-1 son (1:2) y (2:2), se puede observar que el ítem 3 no cumple con el soporte, por tanto no es Patrón Condicional. Los Patrones Condicionales de los ítems frecuentes-1 se pueden observar en el árbol Fptree de la Figura 7.2.

Tabla 7.2: Patrones Condicionales e *itemsets* frecuentes

Item	Patrones Condicionales Base	Patrones Condicionales	Patrones Frecuentes
5	{(1,2:1),(3,1,2:1)}	(2:2, 1:2)	(2,5:2),(1,5:2)(2,1,5:2)
4	{(2:1),(1,2:1)}	(2:2)	(2,4:2)
3	{(2:2),(1:2),(1,2:2)}	(2:4, 1:2),(1:2)	(2,3:4),(1,3:2),(2,1,3:2)
1	{(2:4)}	(2:4)	(2,1:4)

El último paso de la implementación es determinar el conjunto de *itemsets* Frecuentes. Para lo cual se hace uso de la clase *Combinations*, la cual toma los Patrones Condicionales y los combina con los ítems frecuentes-1. Por ejemplo, tenemos que los Patrones Condicionales del *ítem* frecuente-1, 5 son (2:2, 1:2), entonces, 5 se combina con sus Patrones Condicionales y se obtienen los *itemsets* Frecuentes (2,5), (1,5) y (2,1,5). Para los demás ítems, podemos ver sus *itemsets* Frecuentes en la tabla 7.2.

Así como para *Apriori y EquipAsso* los *itemsets* Frecuentes se almacenan en un Vector de árboles AVL balanceados, en donde en cada posición del Vector, se encuentran almacenados un tipo de *itemsets* Frecuentes. Es decir en la posición 0 del Vector se encuentran los *itemsets* Frecuentes tipo 1, en la posición 1 los *itemsets* Frecuentes tipo 2 y así mismo el resto de *itemsets* frecuentes. Algoritmo *EquipAsso* El paquete *EquipAsso* dentro del módulo de algoritmos contiene dos clases encargadas de la implementación y aplicación del algoritmo *EquipAsso* orientadas a dar soporte al descubrimiento de reglas de asociación dentro del proceso KDD. La primera *EquipAsso*, encargada de la carga de datos y un primer filtrado de ellos, donde se carga solo aquellos ítems que pasan el umbral soporte dado (proceso *EquiKeep* dentro del algoritmo *EquipAsso*) y la otra clase *Combinations* que se encarga de generar todas las combinaciones de un determinado tamaño para cada uno de los registros cargados del conjunto de datos y sus respectivo conteo (proceso *Associator* dentro del algoritmo *EquipAsso)*.

La clase principal es *EquipAsso*, al hacer una instancia de esta clase se debe pasar como parámetros una instancia de la clase *DataSet*, llamada *dataset* y que contiene una versión comprimida del conjunto de datos, y un entero corto, que se llama *support* y que cumple la tarea de soporte del sistema. Estas instancias son asignadas a atributos internos de esta clase.

La clase *EquipAsso* cuenta con un atributo de tipo arreglo llamado *Trees* donde se almacenan los diferentes árboles de *itemsets* frecuentes organizados por tamaño. El primer conjunto de *itemsets* son los de tamaño 1 y podemos obtenerlo al llamar al método *pruneCandidatesOne* de la clase *DataSet* que fue pasada como parámetro al constructor de la clase. Este método devuelve un árbol *AvlTree* que es insertado en la posición 0 del arreglo *Trees* y que contiene el conjunto de *itemsets* Frecuentes tamaño 1. Los demás conjuntos de *itemsets* Frecuentes (tamaño 2, tamaño 3, ...) serán almacenados en las siguientes posiciones de ese arreglo organizados en árboles *AvlTree*. En el siguiente listado de código podemos observar el constructor de la clase *EquipAsso*.

Una vez instanciado un objeto *EquipAsso* se inicia el proceso del algoritmo llamando al método *run* de esta clase. Dentro de este método se inicia un ciclo controlado por el método *findInDataset* donde se recorre el conjunto de datos. Este método declara un objeto del tipo *Transaction* el cual, a través de su método *loadTransaction*, carga los registros del conjunto de datos. En este punto se

realiza un primer filtrado cargando únicamente aquellos ítems que sobrepasen el soporte del sistema y están por ende contenidos en el conjunto de *itemsets* frecuentes tamaño 1 y que ha sido almacenado en la primera posición del arreglo *Trees.*

Es por eso que son enviados como parámetros instancias del *dataset* actual y del *AvlTree* que contiene los *itemset* Frecuentes tamaño 1 para solo cargar del *dataset* los ítems que están en este conjunto. Posterior a este filtrado se pasa esta transacción como parámetro a una instancia de la clase *Combinations* para encontrar sus posibles combinaciones. Puede darse el caso de que ninguno de los ítems provenientes del *dataset* supere el soporte, ni sea encontrado entre los *itemsets* frecuentes uno, por lo que se cargará una transacción vacía esta es descartada y se continúa con la siguiente transacción.

La clase *Combinations* recibe como parámetro un arreglo que contiene los ítems validados de cada transacción provenientes del conjunto de datos. Este arreglo es cargado en el constructor de esta clase y posteriormente, con la llamada del método *letsCombine* de la clase *Combinations*, se genera un determinado grupo de combinaciones dependiendo del tamaño que se quiera generar, combinaciones tamaño 2, tamaño 3, etc según sea el caso, con cada una de estas combinaciones crea objetos de la clase *ítemset*. El método *letsCombine* recibe como parámetro un árbol *AvlTree* llamado *treeCombinations* en el cual se van insertando los objetos *itemset* que han sido generados.

Se hace uso de un árbol balanceado *AvlTree*, para almacenar este tipo de conjuntos de *itemsets* para agilizar las búsquedas de un determinado *itemset* generado. Si un *itemset* generado a partir de una combinación ya existe en el árbol *treeCombinations* el soporte interno de este se incrementa en uno. De esta manera, la primera vez que se ejecute el método *letsCombine* se generarán las combinaciones tamaño 2 de cada transacción válida del conjunto de datos y quedarán almacenadas en *treeCombinations* el total de *itemsets* generados por el *dataset* y su correspondiente soporte. Con posterioridad a este paso se procede a barrer el árbol *treeCombinations* para seleccionar aquellos *itemsets* Frecuentes que hayan superado el soporte del sistema. Esta función se realiza haciendo uso del método *pruneCombinations*, de la clase *EquipAsso*, que recibe como parámetros el árbol *treeCombination* y un árbol *AvlTree* llamado *treeFrequents*, donde se guardaran únicamente los *Itemsets* Frecuentes.

Al final de este método se liberan los recursos ocupados por *treeCombinations* y contaremos con un objeto *treeFrequents* donde están almacenados todos los *Item-sets* frecuentes de un determinado tamaño. Para concluir el proceso se comprueba si el árbol *treeFrequents* contiene elementos, se usa el método *howMany* de la clase *AvlTree* para este propósito. De ser así este árbol es almacenado en el arreglo *Trees* de la clase *EquipAsso* en su respectiva posición de acuerdo al tamaño de los *itemsets* que contiene y el método *findInDataset* retorna una señal de true al ciclo principal del método *run* que controla la continuación del algoritmo. El proceso se repetir esta vez calculando las combinaciones e *Itemsets* Frecuentes tamaño 3. El algoritmo se detiene cuando el método *howMany* del árbol *treeFrequents* devuelva 0 o 1, que quiere decir que ya no se han generado *Itemsets* Frecuentes en esta iteración y el proceso de *EquipAsso* ha terminado. En este caso el método *findInDataset* retorna una señal de *false* y el ciclo principal de la clase *EquipAsso* concluye.

7.2.4 Paquete classification

El paquete *classification* está conformado por los paquetes que implementan los algoritmos de clasificación, los cuales se explican a continuación:

Paquete c45. Este paquete implementa el algoritmo de clasificación C.4.5 [33], el cual es una técnica de minería de datos que permite descubrir conocimiento por medio de la clasificación de atributos a través de la ganancia de información de los mismos, aplicando fórmulas para obtener la entropía de cada atributo con respecto a otros.

En primera instancia se hace un conteo de los distintos valores en el atributo objetivo, con el propósito de encontrar una entropía inicial, con la cual calcularemos las entropías de cada atributo, para saber cuál es la que brinda la mayor ganancia de información, el atributo con mayor ganancia es el próximo nodo de árbol de decisión, el cual es una estructura que se inicia en el nodo raíz o atributo objetivo, además está compuesta por nodos internos, ramas y hojas. Los nodos representan atributos, las ramas son reglas de clasificación y las hojas representan a una clase determinada.

El proceso es iterativo hasta que no haya atributos que clasificar. En TARYI C 45 esta implementado de la siguiente forma: El flujo de entrada es un conjunto de datos presentados en un *TabelModel*, el cual permite la conexión del algoritmo con distintos filtros de la etapa *datacleaning*. También hacemos uso de una estructura de árbol N-Ario, en una clase la cual llamamos *TreeCounter*, para hacer el conteo eficiente de las múltiples combinaciones de los valores de un atributo con respecto a otros.

Al resultado de este conteo, se le aplican las formulas de entropía para encontrar el atributo con mayor ganancia de información, la cual es obtenida a partir del siguiente criterio.

La ganancia de información de un atributo A, con respecto a un conjunto de ejemplos S es:

$$Gain(S, A) = entropía(S) - \sum v \in V\,alores(A)|Sv|/|S|entropía(Sv)$$

Donde: *Valores(A)* es el conjunto de todos los valores del atributo A. *Sv* es el subconjunto de *S* para el atributo *A* que toma valores *v*.

$entropía(S) = -\sum pi\ Log_2\ pi$ donde *pi* es la probabilidad que un ejemplo arbitrario pertenezca a la clase *Ci*.

Después de encontrar el atributo que presenta mayor ganancia de información, es vinculado a la ruta de una rama, para repetir el proceso recursiva e iterativa mente hasta conformar el árbol de decisión, que es implementado en un árbol eneario gráfico denominado *finalTree*, el cual extiende la clase de Java *Jtree*.

Ejemplo de funcionamiento del algoritmo C 45 en TARYI: Primero en la tabla 7.3 se pueden observar datos de entrada, para el algoritmo C45:

En la tabla 7.3 el Atributo seleccionado como target o clase es "JUGAR TENNIS", lo cual significa que el objetivo de la Minería de Datos gira en torno a este atributo. Aplicamos el conteo, utilizando la estructura *TreeCounter*, sobre el atributo *Target*, con lo cual obtenemos 9 valores "SI" y 5 valores "NO", como lo observamos en la figura 7.3.

A estos resultados aplicamos la fórmula para obtener la entropía inicial.

$$E([9+, 5-]) = -(9/14)\ Log_2\ (9/14) - (5/14)\ Log_2 (5/14) = 0{,}940$$

Aplicamos nuevamente el proceso de conteo, relacionando cada atributo, con el atributo objetivo, con el propósito de encontrar aquel que brinde mayor ganancia de información, se presenta el ejemplo de conteo con el atributo "VIENTO" en el figura 7.4.

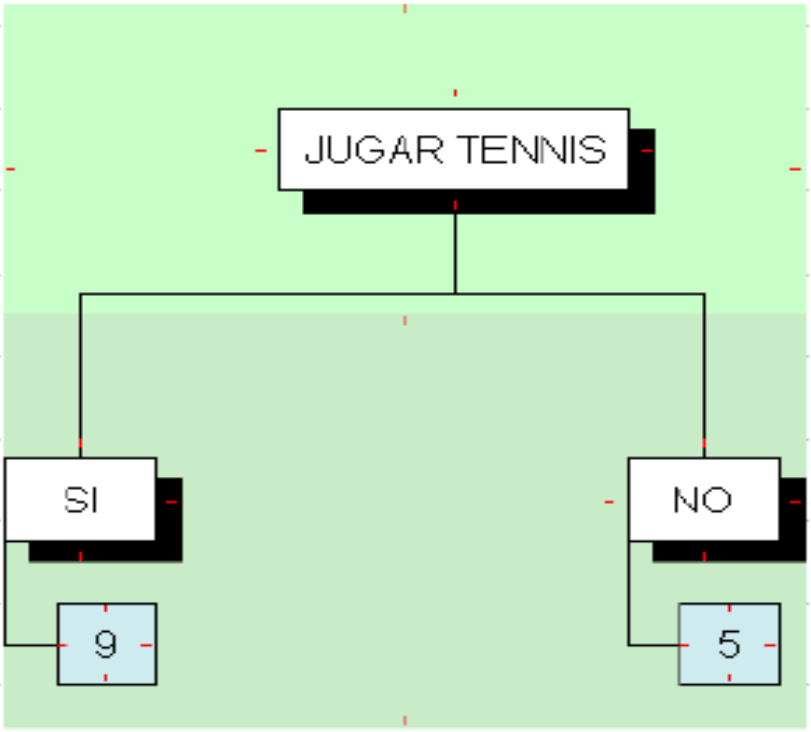

Figura 7.3: Conteo *target*

Tabla 7.3: Conjunto de datos

DIA	ESTADO	TEMPER.	HUMEDAD	VIENTO	Jugar Tennis
D1	Soleado	Caliente	Alta	Debil	No
D2	Soleado	Caliente	Alta	Fuerte	No
D3	Nublado	Caliente	Alta	Debil	Si
D4	Lluvioso	Templado	Alta	Debil	Si
D5	Lluvioso	Fresco	Normal	Debil	Si
D6	Lluvioso	Fresco	Normal	Fuerte	No
D7	Nublado	Fresco	Normal	Fuerte	Si
D8	Soleado	Templado	Alta	Debil	No
D9	Soleado	Fresco	Normal	Debil	Si
D10	Lluvioso	Templado	Normal	Debil	Si
D11	Soleado	Templado	Normal	Fuerte	Si
D12	Nublado	Templado	Alta	Fuerte	Si
D13	Nublado	Caliente	Normal	Debil	Si
D14	Lluvioso	Templado	Alta	Fuerte	No

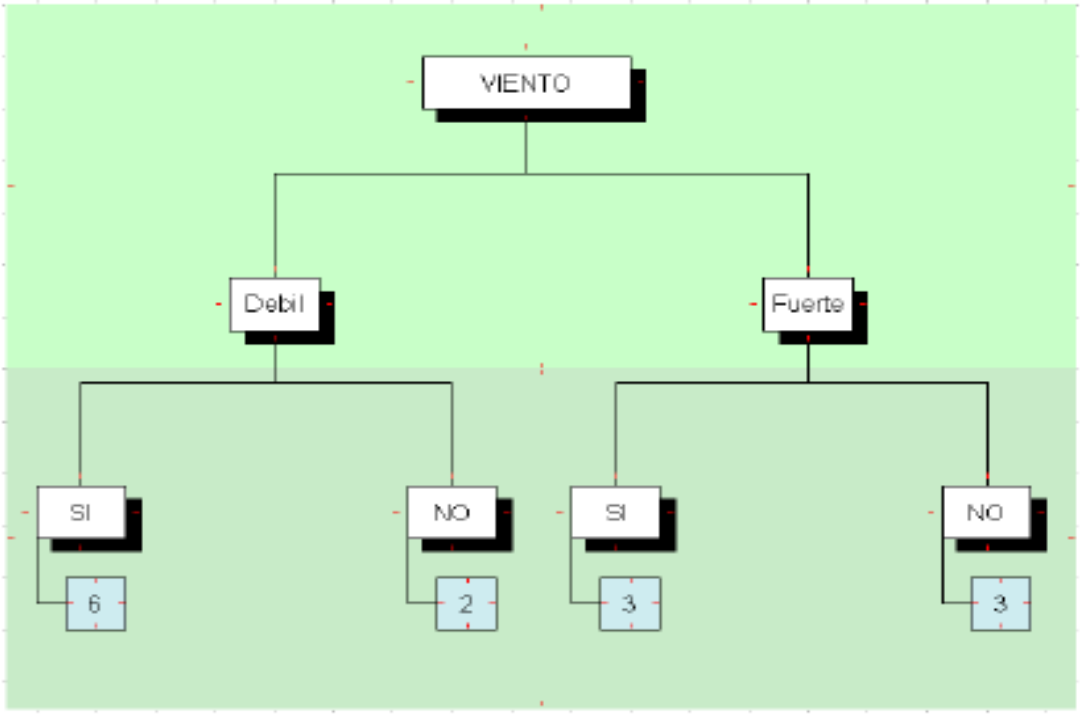

Figura 7.4: Conteo viento

Con estos resultados y la entropía inicial aplicamos la fórmula para obtener la entropía del atributo ”VIENTO” relacionado con el atributo objetivo, de la siguiente forma:

El atributo ”VIENTO” posee dos valores los cuales son débil y fuerte:
$|Sdebil| = [6+, 2-] Sf\ uerte = [3+, 3-]$

$Gain(S,Viento) = entropía(S) - (Sdebil * entropía(Sdebil) + Sf\ uerte * entropía(Sf\ uerte)$ = 0,940 − 8/14entropía(Sdebil) − 6/14entropía(Sf uerte)

Calculamos $E(Sdebil) = E([6+, 2-]) = -(6/8)\ Log_2(6/8) - (2/8)\ Log_2(2/8) = 0{,}811$
$E(Sf\ uerte) = E([3+, 3-]) = -(3/6) log_2(3/6) - (3/6) log_2(3/6) = 1$

Reemplazando : $Gain(S, viento) = 0{,}940 - 0{,}463 - 0{,}428 = 0{,}048$

De una forma similar se aplican los anteriores procedimientos con cada atributo, para encontrar el atributo que brinde mayor ganancia de información para este nivel. Los resultados obtenidos son:

$Gain(S,Estado) = 0.246\ Gain(S,Humedad) = 0.151\ Gain(S,Temperatura) = 0.029$

Podemos observar que la mayor ganancia de información la brinda el atributo ESTADO igual a 0.246, lo cual significa que el nodo inicial en el árbol de decisión es ESTADO, a partir de este atributo encontraremos los siguientes nodos que suministren mayor ganancia en los tres valores del atributo, los cuales son ”Soleado”, ”Nublado” y ”Lluvioso”, también encontramos que valor ”Nublado” se parcializa hacia una decisión la cual es Jugar Tenis. El árbol de este nivel es el siguiente:

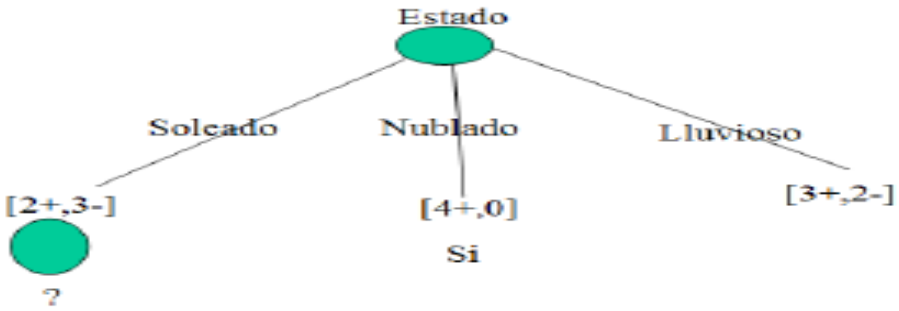

Figura 7.5: árbol parcial

Ahora se encuentra el atributo que brinde la mayor ganancia para cada valor del atributo "ESTADO", haciendo el conteo de forma eficiente al encontrar los resultados para los tres valores simultáneamente. Como ejemplo lo haremos para Humedad, combinado con el atributo Estado, tal y como se puede observar en la figura 7.6.

A continuación se aplica la formula de entropía para el valor Soleado, del atributo Estado, de la siguiente forma:

Humedad(Alta[0+, 3−], Normal[2+, 0−])
Gain(Soleado, Humedad) = 0,970 − (3/5) * 0 − (2/5) * 0 = 0,970

Temperatura(Caliente[0+, 0−], Templado[1+, 1−], fresco[1+, 0−]
Gain(Soleado, Temperatura) = 0,970 − (2/5) * 0 − (2/5) * 1 − (1/5) * 0 = 0,570

Viento(Débil[1+, 2−], fuerte[1+, 1−])
Gain(Soleado, Viento) = 0,970 − (3/5) * 0,918 − (2/5) * 1 = 0,019

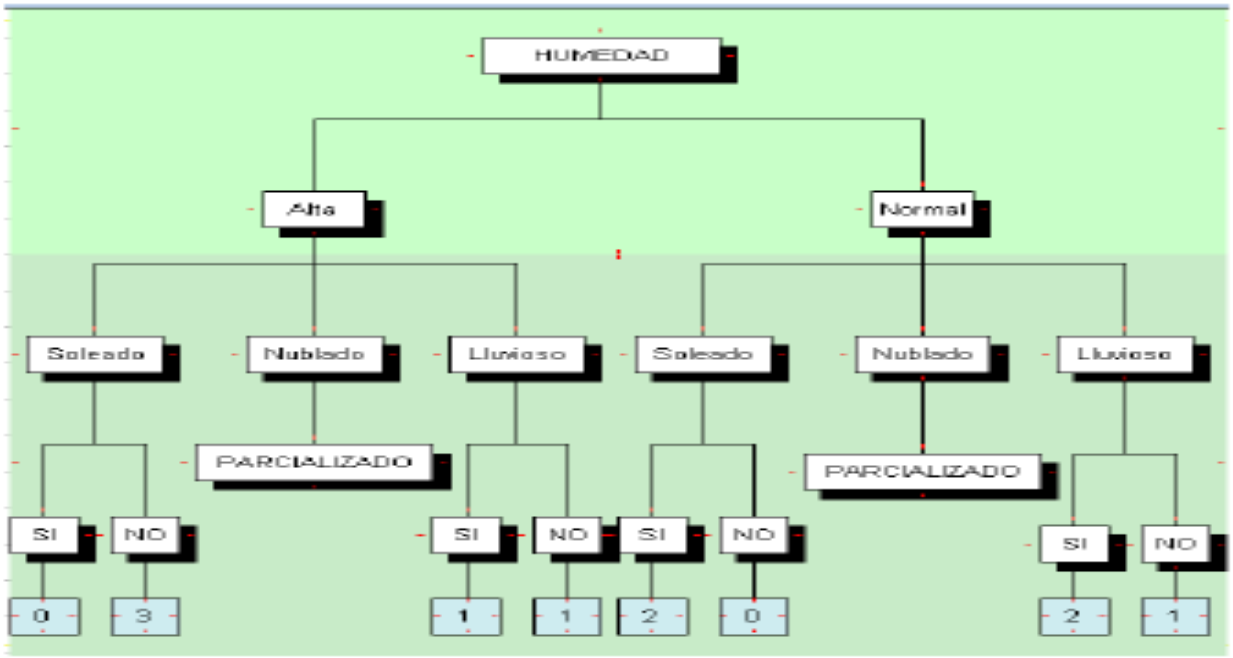

Figura 7.6: Conteo humedad estado

Podemos observar que para el valor Soleado, el atributo ganador es Humedad cuyos valores son Alta y Normal los cuales se parcializan en este nivel. Para el atributo Lluvioso, después de realizar el conteo se aplica la formula de entropía

Humedad(Alta[1+, 1−], Normal[2+, 1−])
Gain(Lluvioso, Humedad) = 0,97 − (2/5) * 1 − (3/5) * 0,917 = 0,0198

Temperatura(Caliente[0+, 0−], Templado[2+, 1−], fresco[1+, 1−]
Gain(Lluvioso, Temperatura) = 0,97 − 0 − (3/5) * 0,917 − (2/5) * 1 = 0,0198

Viento(Débil[3+, 0−], fuerte[0+, 2−])
Gain(Lluvioso, Viento) = 0,970 − (3/5) * 0 − (2/5) * 0 = 0,970

Podemos observar que el atributo ganador es Viento cuyos valores Fuerte y débil se parcializan en este nivel. A continuación se presenta el árbol de decisión definitivo:

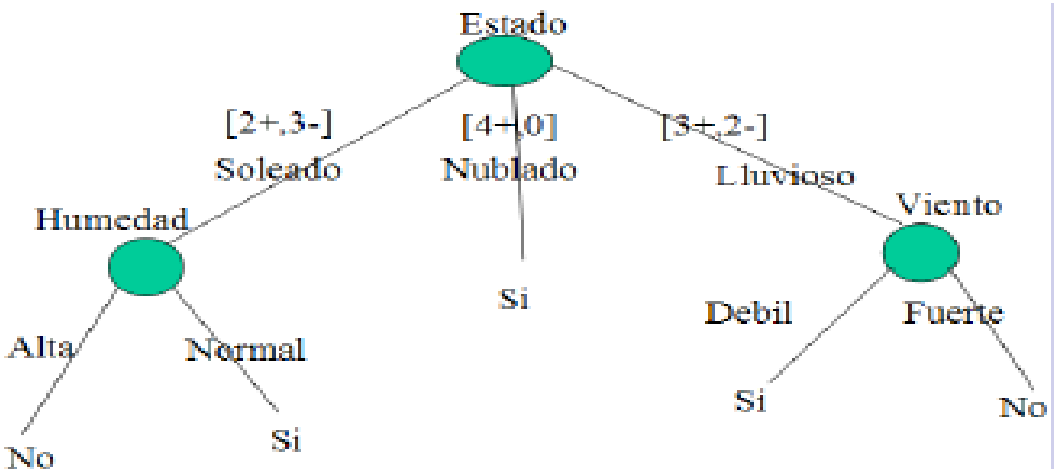

Figura 7.7: árbol definitivo

Paquete mate. A continuación se describen las clases implementadas en la programación del algoritmo *MateBy*. Se habla más en detalle acerca de las estructuras de datos y los métodos más utilizados.

Clase MateBy. Inicialmente se crea un objeto de la clase *MateBy*. Este objeto tiene la siguiente estructura:

dataSet de tipo *DataSet*: es un árbol *N-ario* en el que se almacenan las combinaciones que generadas y en el cual es posible llevar un conteo de cada una de las ocurrencias de una combinación dada.

entros de tipo A*rrayList*: es un arreglo de datos en el que se va a almacenar las agrupaciones de combinaciones que tienen que hacerse para el cálculo de la entropía.

levels de tipo entero: se usa para almacenar el número del nivel de una hoja en el árbol *dataSet*.

attribute de tipo *String*: aquí se almacena el nombre del atributo que se encuentra inmediatamente después de la raíz y por el cual se inició el recorrido del árbol en una búsqueda.

rules de tipo *ArrayList*: aquí se almacenan las reglas generadas. Este arreglo se alimenta del árbol *dataset* después del cálculo de la ganancia.

mateTree de tipo *Tree*: este es un árbol N-ario se crea para ser mostrado gráficamente.
Ya se ha descrito la estructura principal sobre la cual se va a desarrollar el algoritmo. A continuación se describen los métodos principales con los que se lleva a cabo la tarea de clasificación. Se omiten métodos triviales tales como recorridos del árbol, consultas e inserciones al mismo.

Método MateBy. Inicialmente se cargan los datos para alimentar al algoritmo. Cada transacción se almacena en un vector que es pasado como parámetro al Método *combinations*. El atributo clase no se almacena en este vector y es pasado como otro parámetro al Método *combinations*. En este Método se aplica el operador *MateBy* generando todas las posibles combinaciones de las transacciones con el atributo clase. Cada una de las combinaciones se almacena en el árbol N-ario *dataset*, cada rama del árbol representa una combinación diferente. Si se presenta el caso en el que una combinación se repite, esta no es almacenada nuevamente. Las hojas del árbol tienen una estructura diferente al resto de nodos en el árbol que les permite guardar el número de ocurrencias de una combinación. Este campo en las hojas es llamado *soporte* y serán de gran utilidad posteriormente en el cálculo de la ganancia. La figura 7.8 muestra las estructuras mencionadas anteriormente. Al final se retorna *dataset*.

Método groupBranchs. Este método es el encargado de agrupar ramas del árbol o combinaciones que se están asociadas y que deben unirse para luego calcular la ganancia. El criterio de agrupación se basa en la coincidencia del número de niveles y de la ruta de cada hoja en el árbol. Ya que cada rama tiene un soporte asociado, al momento de agruparlas es necesario conservar ese soporte y además calcular el soporte acumulado al agrupar distintas ramas. Tanto estos datos como las hojas agrupadas son almacenadas en una estructura especial llamada *entro*. Cada uno de los elementos entro es almacenado en un arreglo llamado *entros* para mejorar la manipulación de estos objetos. El proceso continua hasta recorrer todas las ramas del árbol y hasta haber agrupado todas las hojas relacionadas entre sí.

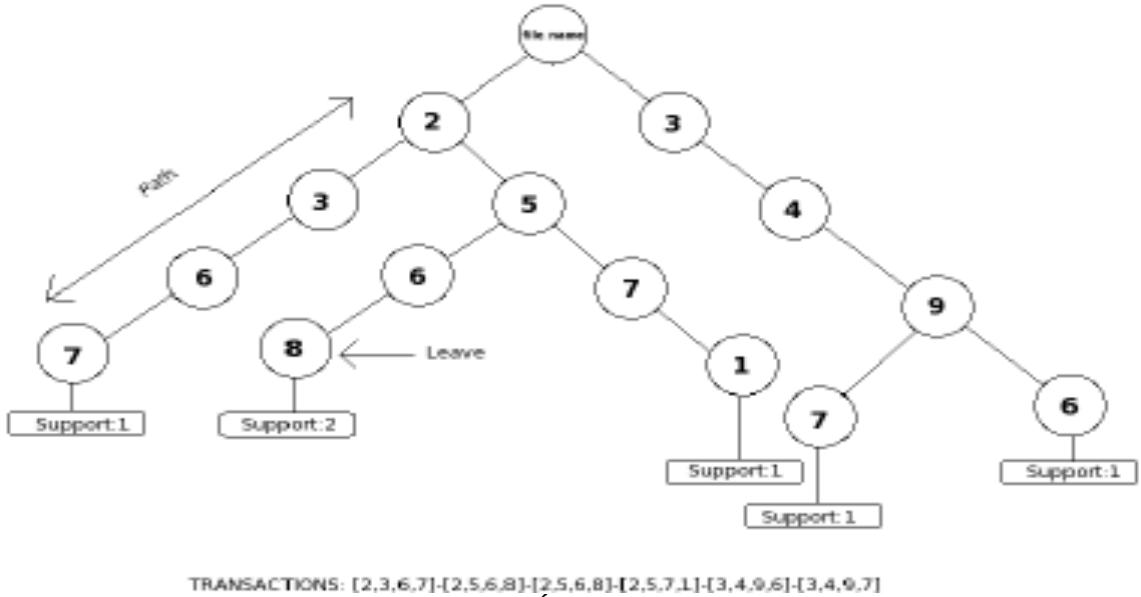

Figura 7.8: Árbol de decisión

Método entroAgrupation. Dentro de éste método se recorre el arreglo *entros* y se recuperan las hojas agrupadas para calcular la entropía por medio del Método *calculateEntropy*. Este nuevo dato es almacenado en entro por cada grupo de hojas.

Método gainCalculation. Dentro de método se recurre nuevamente al arreglo *entros*. Hasta este momento se ha calculado la entropía de cada grupo de hojas y ahora pasamos a agrupar nuevamente para calcular la ganancia. La agrupación se hace de tal manera que las rutas de cada una de las hojas almacenadas en entro tenga el mismo número de niveles y que los nombres de los atributos que están inmediatamente después de la raíz coincidan, de esta manera y debido a la organización del árbol al que hacen referencia las hojas se logra organizar las combinaciones de tal forma que se puede calcular la ganancia. Una de las partes más complejas en este paso es el control de los soportes. En el arreglo *entros* se intercala, por cada grupo de hojas, el valor de la entropía y el soporte acumulado. Los ciclos implementados se ocupan de recorrer adecuadamente las estructuras para poder obtener todos los datos necesarios de para calcular la ganancia. Otro punto importante es que a medida que se recorre *entros* se van comparando las ganancias obtenidas y solo se trabaja por las ramas que obtengan el mayor valor en cada iteración.

Método chooseNodes. Este método se ocupa de seleccionar los nodos que van a ir en el árbol de reglas. De las agrupaciones de hojas que obtuvieron la mayor ganancia se debe establecer la hoja que forma parte de la regla que va a pasar al árbol de reglas. A partir de la hoja se recorre el árbol de abajo hacia arriba para sacar la ruta. Esta ruta representa la regla generada.

Método buildRulesTree. Este Método se encarga de construir el árbol de reglas que es mostrado gráficamente. Este árbol es de tipo *Tree* y será mostrado a través de un *Jtree*. Cada regla se decodifica al pasar del árbol *dataset* al árbol *rulestree*. Para esto se utiliza el diccionario construido al momento de cargar los datos a la aplicación por medio de la llamada al Método *getRules*. Aquí l que se hace es hacer el proceso inverso al de la codificación para obtener los nombres reales de los atributos. Estos nombres son los que son mostrados finalmente en el árbol gráfico.

Método getRules. Se instancia un arreglo de cadenas en el que se van a almacenar los nombres de los atributos que se obtengan de cruzar los punteros de la rama en el árbol *dataSet* y el diccionario.

7.2.5 Paquete GUI

Dentro de la implementación de una interfaz gráfica amigable para el proyecto *TariyKDD*, se trabajó utilizando las funcionalidades del proyecto *Matisse*, el constructor de interfaces gráficas de usuario (GUI) propia de *NetBeans* 5.0 que permite un diseño visual de las formas y su posterior programación.

Dentro del paquete *gui* de *TariyKDD* reposan dos paquetes: *KnowledgeFlow* e *Icons*. El primero contiene las formas utilizadas en la Interfaz principal de la aplicación. En el paquete *Icons* se aborda la programación para cada uno de los iconos involucrados en el proceso de descubrimiento de conocimiento y a los cuales tiene acceso el usuario para desempeñar una determinada tarea por ejemplo realizar una conexión a una base de datos, ejecutar un determinado algoritmo o utilizar un visor para desplegar la información obtenida.

Paquete KnowledgeFlow. Se hablará primero de la implementación del paquete *KnowledgeFlow*. La interfaz principal de la aplicación está contenida dentro de la clase *Chooser*, cuya

implementación se muestra en la figura 7.9 y a su vez implementa el uso de diversas clases propias de la biblioteca gráfica *Swing* de Java como *JTabbedPanne*, *JsplitPane*, *JscrollPane*, *JLabel* así como extensiones de la clase *JPanel* donde se implementan funcionalidades propias a la aplicación como un área de trabajo donde tendrá lugar la construcción de un experimento KDD y donde, a través de la metodología de Arrastrar y Soltar (*Drag'n Drop*), se dispondrán los iconos que representan una determinada tarea dentro del proceso. Se extiende también *JComponent* donde se implementan funcionalidades de los iconos que representan cada acción dentro de la aplicación.

Dentro de la interfaz principal se pueden distinguir algunas secciones como son un Selector, en el cual se permite escoger una etapa dentro del proceso KDD, un panel, donde se despliegan los iconos asociados a una determinada etapa y una Área de Trabajo, donde se organizan los iconos seleccionados y se construye un experimento de descubrimiento de conocimiento.

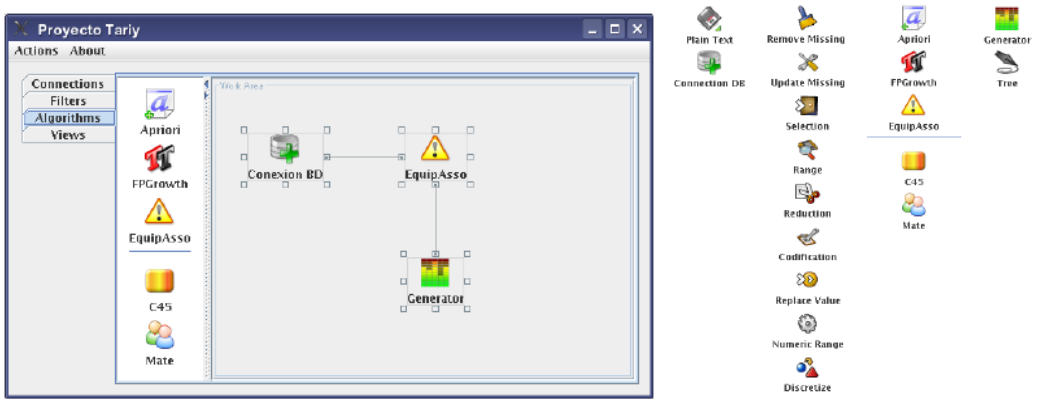

Figura 7.9: Clase *Chooser*. Interfaz principal de la aplicación

Cada una de estas partes se constituyen como una nueva clase en el proyecto, que se gestionan con instancias propias de Java las cuales se mencionaron anteriormente. Con un *JTabbedPanne* se monta la selección de un determinado proceso desplegando etiquetas que identifican a cada uno de ellos, "*Connections*", para escoger una conexión hacia un conjunto de datos específica, puede ser a una base de datos o cargando un archivo plano, "*filters*", donde se da la opción de escoger un determinado filtro que modificará el conjunto de datos que se haya cargado, "*Algorithms*", sección en la cual se escoge el algoritmo oportuno para realizar las tareas de minería de datos como tal y "*Views*", donde se despliegan opciones de visualización para organizar y mostrar al usuario los resultados obtenidos.

Al interactuar con las etiquetas del *JTabbedPanne* se verá modificado el contenido de la clase *Container*, la cual es una extensión de *JSplitPanne* y divide este componente en dos, izquierda y derecha. En el izquierdo se cargará un panel correspondiente a cada etapa seleccionada con el *JTabbedPanne*, los posibles paneles que se cargan se visualizan en la figura 7.9, y en el derecho se carga una instancia de la clase *MyCanvas*, que provee la funcionalidades de *Drag'n Drop* (Arrastrar y Soltar) entre los iconos involucrados en un experimento.

Cada panel extiende a *JPanel* y está conformado por instancias de la clase *JLabel*, cada una de las cuales representará un icono perteneciente a esa etapa. En la instanciación de cada *JLabel* se carga una imagen alusiva a cada icono y un texto que lo identifica dentro del panel. Las imágenes correspondientes a cada icono, y en general todas las imágenes utilizadas en la herramienta, se

cargan directamente del *Classpath* de la aplicación dentro del paquete *images*. Es importante aclarar que en este momento también se asigna el nombre de cada *JLabel*, en su propiedad *name* a través del método *setName*, ese mismo nombre será relacionado posteriormente para identificar el icono seleccionado por el usuario desde el panel, luego, cada *JLabel* se adiciona a su respectivo panel.

Se asigna una imagen al *JLabel* utilizando el método *getClass().getResource*(Ruta de la imagen). La clase *MyCanvas* es la encargada de implementar las funcionalidades de *Drag'n Drop* para los iconos involucrados en un determinado experimento. Al seleccionar con el ratón un determinado icono desde un panel, el *JLabel* asociado a ese icono es capturado escaneando los eventos del ratón cuando un icono es presionado y posteriormente liberado.

Al adicionar un *MouseListener* al objeto *container*, el *JSplitPanne* que contiene los paneles y el área de trabajo, este captura el evento *pressed* del ratón cuando este ha sido presionado. En ese momento, el evento es capturado y a través del método *getPoint* tenemos la posición dentro del contenedor donde fue generado el click. El método *findComponent*() devolverá el componente encontrado en el punto donde fue presionado el ratón.

Cuando el ratón se libera, este evento es igualmente capturado y se procede a identificar cual fue el icono seleccionado desde el panel para desplegarlo correctamente sobre el objeto *MyCanvas*. Esto se efectúa al consultar la propiedad *name* del objeto *pressed* que fue capturado durante el evento anterior. Dependiendo del nombre del componente se procede a instanciar un objeto de la clase *Icon*.

La clase *Icon* fue construida extendiendo a un *JPanel* y contiene una instancia de la clase *JLabel*, que contiene la imagen y el texto asociados a ese icono, y un total de 8 instancias de la clase Conector, esta clase fue diseñada para identificar secciones dentro del icono donde el usuario pueda hacer un click y relacionar un icono con otro.

Se implementa dentro de esta clase un menú desplegable donde se pueda incluir funciones propias para cada icono y la inclusión de una animación que se activa cuando el icono está realizando un proceso determinado. Estas funcionalidades serán heredadas a todas las clases que extiendan de la clase *Icon*.

El menú desplegable se logra al usar las clases *JPopupmenú* y *JmenúItem*. Por defecto, todas las instancias de *Icon* y las clases que hereden de él tendrán implementada la opción *Delete*, que borrará el icono del área de trabajo y descargará la clase *Icon* correspondiente de la clase *MyCanvas* que la contiene. Para ello, se instancia un objeto de la clase *JmenúItem* y se inicializa su propiedad de name con el valor "*Delete*", posteriormente se procede a adicionar esta instancia de *JmenúItem* al *JPopupmenú* asociado a la clase *Icon*. Cuando un usuario seleccione la opción *Delete* se dispara el método *mnuDeleteActionPerformed* que se encarga de descargar las conexiones que tenga asociado este icono con otros iconos y borrar el componente del área de trabajo. Nuevas adiciones al menú desplegable pueden ser hechas al instanciar objetos *JmenúItem* e incluirlos dentro del *JPopupmenú* de la clase *Icon*.

La clase Conector fue construida extendiendo *JComponent* ya que debía proveer facilidades de captura de los eventos del ratón y ser dibujado de manera especial cuando esté disponible y diferente cuando ya haya sido seleccionado.

Al método le llega una instancia *g* del la clase *Graphics* que será el objeto donde podremos

dibujar. A través de los métodos de esta clase podemos escoger colores para el conector y dibujar un rectángulo de 7x7 pixels que representará el área del conector que puede ser pulsada por el usuario para su selección y dibujaría un rectángulo relleno en el centro del conector si este ha sido ya seleccionado.

La clase *Icon* como tal no es incluida directamente sobre la clase *MyCanvas*, o área de trabajo. Existen una serie de clases que extiende a la clase *Icon* y se corresponden con cada tarea desempeñada dentro del proceso KDD. Existen un total de 7 extensiones a la clase *Icon* y estas son *DBConnectioIcon, fileIcon, filterIcon, AssociationIcon, ClassificationIcon, RulesIcon y TreeIcon* asociadas a la conexión con una base de datos, conexiones con archivos planos, filtros para la selección y preprocesamiento de un conjunto de datos, algoritmos de asociación, algoritmos de clasificación, visualización de resultados de reglas de asociación y visualización de árboles de decisión respectivamente. Estas 7 clases, junto con otras que apoyan la tarea que cumplen, están contenidas en 7 paquetes dentro del paquete *Icons* que será explicado posteriormente y que forma parte a su vez del paquete *gui*.

Para cada una de las clases heredadas se maneja de manera independiente el contenido de su *JPopupmenú* así como la imagen y el texto asociados a cada tipo de icono.

La clase *MyCanvas.* Será la encargada de gestionar las conexiones entre los iconos y su movimiento sobre el área de trabajo. Esta clase extiende un *JPanel* y monitoréa los eventos del ratón referentes a los *clicks* del usuario. Los eventos que tiene contemplados son *MousePressed* (ratón presionado), *MouseDragged* (click sostenido), *MouseReleased* (ratón liberado), *MouseClicked* (un click sencillo).

Para el evento *MousePressed*, se identifica si el click fue sobre un icono o sobre uno de sus conectores. Si fue sobre un icono, éste es almacenado en la variable *selectedIcon*, pero si fue sobre un conector se almacena en la variable *selectedConector*. Para esta tarea se utiliza el método *findComponentAt(MouseEvent*) que se explicó anteriormente. Al final de este método, y en general todos los métodos que involucran cambios sobre el área de trabajo, se llama al método *repaint*()que se encarga de redibujar la clase *MyCanvas* invocando directamente el método *paint(Graphics g)* de esta clase y que se explicará posteriormente.

Para el evento *MouseDragged* se identifica cual fue el componente seleccionado en el evento *MousePressed*, si se trata de un icono este método se encarga de redibujarlo a medida que se mueve el ratón pero si se trata de un conector, se traza una línea entre el conector seleccionado y el puntero del ratón a medida que este se mueve. Estas dos acciones se realizan en el método *paint(Graphics g)* de la clase *MyCanvas*.

Para el evento *MouseRelease* se debe identificar igualmente qué componente está seleccionado si se trata de un icono, este evento se limita a liberar la variable *selectedIcon* para no seguir cambiando su posición, si se trata de un Conector se debe identificar en qué punto es liberado, si coincide con un conector de otro icono que esté disponible se establecerá una relación entre ambos iconos trazando una línea entre sus conectores involucrados. Esta relación será almacenada en un arreglo donde se guardarán instancias de la Clase *Connections*. Esta clase se construye a partir de los dos conectores involucrados en la relación y servirá posteriormente para trazar todas las relaciones existente sobre el área de trabajo durante la invocación al método *paint(Graphics g)*.

Durante el evento *MouseClicked* evalúa si el click fue sobre un conector que está seleccionado,

eliminando esa relación o si el click fue sobre un icono, desplegando el menú emergente.

La clase *MyCanvas* tiene sobrecargado el método *paint(Graphics g)* lo que permite aprovechar las propiedades de dibujo de la clase que extiende *(JPanel)* así como adicionar nuevas figuras al interactuar sobre la clase *Graphics* asociada a esta clase y que nos provee de diferentes métodos para trazar figuras, escoger colores y repintar los componentes gráficos que están contenidos dentro de la clase y que hayan cambiado de posición.

Durante la implementación de este método primero se hace una llamada al método *paint(Graphics g)* de la clase superior para que haga un redibujado de los componentes que contiene y de esta manera redibujar todos los iconos que posee en sus nuevas posiciones así como los bordes y colores propios de la clase heredada. Posteriormente se recorre el arreglo *connections* que posee todas la conexiones entre lo conectores y los iconos a los que pertenecen trazando líneas que relacionan de manera visual un icono con uno o más de ellos. Al recorrer el arreglo *connections* se extraen de él objetos de la Clase *Connection*. Esta clase consiste de dos instancias de la Clase Conector, identificadas con los nombres *from* y *to*, haciendo alusión a él conector desde donde viene la relación y hacia dónde va. Recordemos que la Clase Conector hereda a la clase *JComponent*, es decir que hereda los métodos *getX()* y *getY()* para calcular su posición dentro del la clase que los contiene, que en este caso es *MyCanvas*. De esta manera podemos calcular las coordenadas de los conectores y trazar las líneas que representan la relación.

Paquete Icons. Dentro del paquete GUI también se encuentra contemplado un paquete llamado *Icons* que se encarga de organizar cada una de las extensiones hechas a la clase *Icon* y aquellas clases que sirven de apoyo para las acciones propias de cada una de estas clases, por ejemplo las formas de la interfaz gráfica encargadas de capturar información del usuario y que se disparan al seleccionarlas del menú contextual de cada icono.

Cada clase se encuentra contenida dentro de un nuevo paquete lo que quiere decir que dentro del paquete *Icons* existe un total de 7 nuevos paquetes que son *DBConnection*, *file, filters, Association, Classification, Rules y Tree*. Durante esta explicación se abordarán los paquetes *Association, Classification, Rules y Tree*. Los tres primeros paquetes se explicarán con más detalle en otras secciones de este capítulo dado el nivel de complejidad que estos revisten.

Paquete Association. En este paquete se encuentra la clase *AssociationIcon* y la clase *configureSupport*. Esta extiende la clase *Icon* adicionando dos nuevas entradas al menú contextual que se corresponde a Configure, para configurar el soporte pedido al usuario, y *Run*, que ejecuta el algoritmo escogido por el usuario y que puede contener tres estados al hacer alusión al algoritmo Apriori, FPGrowth o EquipAsso. Estos estados se muestran en la figura 7.10.

Figura 7.10: Estados de la Clase *AssociationIcon*

En la gráfica se puede apreciar el contenido del menú desplegable de cada icono: *Delete*, por defecto presente en todas las clases que hereden de *Icon*, *Configure*, desplegará una instancia de la Clase *configureSupport* donde el usuario introduce el soporte del sistema para este experimento. Figura 7.11 ilustra el contenido de esta ventana.

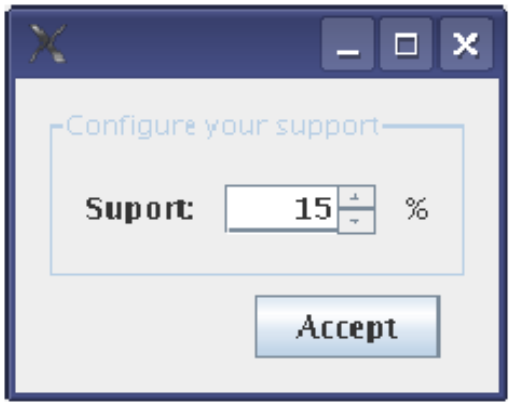

Figura 7.11: Captura del soporte del sistema

La última opción del menú es *Run*, aquí esta clase hará instancias de las clases Apriori, FPGrowth o EquipAsso según sea el caso pasando como parámetros un objeto de la clase *DataSet*, que puede provenir de una instancia de la clase *DBConnectionIcon* o de una instancia de *fileIcon* con la cual se haya establecido una relación, y el soporte del sistema capturado con la ventana anteriormente descrita.

El resultado de ejecutar cualquiera de las clases de asociación que se han implementado devolverá a la Clase *AssociationIcon* un Vector el cual contiene un arreglo de los arboles Avl (instancias de la Clase *AvlTree*) que organizan el conjunto de itemset frecuentes obtenidos al ejecutar un determinado algoritmo. Este vector es guardado en la variable *trees*, variable global a esta clase que posteriormente será pasada a un objeto de la Clase *RulesIcon* para que despliegue las reglas obtenidas al analizar el conjunto itemsets frecuentes.

Paquete Classification. En este paquete se encuentra la clase *ClassificationIcon*. Esta extiende la clase *Icon* adicionando una nueva entrada al menú contextual que se corresponde a *Run*, que ejecuta el algoritmo escogido por el usuario y que puede contener dos estados al hacer alusión al algoritmo C45 o Mate. Estos estados y las entradas al menú contextual se muestran en la figura 7.12.

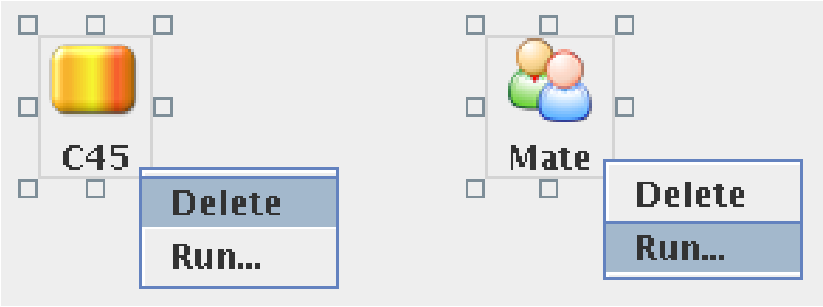

Figura 7.12: Estados de la Clase *ClassificationIcon*

La única opción implementada en el menú es *Run* ya que este tipo de algoritmos no requiere información directa del usuario aparte del conjunto de datos. Al seleccionar la opción *Run* esta clase hará instancias de las clases C45 o Mate según sea el caso pasando como parámetros un objeto que implemente la interfaz TableModel dentro de la variable *dataIn*, que puede provenir de una instancia de la clase *filterIcon*, donde se ha seleccionado el atributo clase del conjunto de datos, con la cual se ha establecido una relación.

El resultado de ejecutar cualquiera de los algoritmos de clasificación será dos objetos de las Clases *JPanel y ArrayList*. El primero contendrá una extensión de *JPanel* donde se despliega un objeto de tipo *JTree* que contiene gráficamente el árbol de decisiones que resulta del análisis y el *ArrayList* tendrá el conjunto de reglas que están contenidos dentro del árbol de decisiones de una manera textual. Estas dos clases serán pasadas como parámetros a la Clase *TreeIcon* encargada de desplegar de manera visual su contenido.

Paquete Rules. Después de aplicar a un conjunto de datos, cualquiera de los tres algoritmos, Apriori, EquipAsso o FPGrowth, a través de este paquete los itemsets frecuentes obtenidos pueden ser observados en forma de reglas de asociación.

El paquete Rules forma parte del paquete *Icons*, que a su vez forma parte del paquete GUI. Las clases que conforman este paquete son: *RulesIcon, configureConfidence, RulesTableModel y showRules*.

RulesIcon extiende a la clase *Icon* que forma parte del paquete *KnowledgeFlow* y a su vez del paquete GUI, por tanto automáticamente hereda su menú con la opción por defecto *delete*, la cual permite al usuario eliminar el icono visualización de reglas de asociación o en la interfaz llamada *Generator*. Por otra parte, la clase *RulesIcon* añade al menú de *Icon* las opciones Configure y *Run.*

La opción Configure abre una pequeña ventana, controlada por la clase *configureConfidence*, la cual hace uso de un *JSpinner*, clase descendiente de *javax.swing*, que permite introducir en un campo de entrada un valor numérico a través del cual el usuario determina la confianza con la cual se van a mirar las reglas de asociación. La ventaja de usar un *JSpinner* es la facilidad con la que se pueden validar las entradas, ya que mediante la clase *SpinnerNumberModel* se establece el límite inferior y superior que el usuario está en condición de escribir y si éste introduce una entrada inválida, en cuanto se pierda el foco del *JSpinner* éste tomará el último valor correcto.

Por otra parte, la opción *Run* muestra las reglas de asociación al usuario en una JTable. Las reglas de asociación se deben almacenar en un array de cadenas, en la clase *AssocRules*. Antes la clase *AssocRules* guarda las reglas sin codificar, o sea, guarda los códigos del diccionario de datos.

A continuación, las reglas se decodifican, o sea que la regla *1 & 2 => 3* se almacenará en el array de la clase *AssocRules*, pero de la siguiente manera: *Jabón & Arroz => Champú*. De la misma manera todas las reglas de asociación encontradas se almacenan en la clase *AssocRules*.

A partir de las reglas almacenadas en la clase *AssocRules*, se construye un modelo propio para JTable y de esta forma se muestran las reglas en la tabla. Para construir el modelo, entonces las reglas almacenadas en un array de cadenas se pasan al constructor de la clase *RulesTableModel*, la cual implementa los respectivos métodos para construir el modelo.

La tabla que muestra los datos se encuentra en la clase *showRules*, e implementa un evento del

ratón para que cuando el usuario haga click en el encabezado Rules de la tabla, las reglas se ordenen por un criterio que ha sido determinado ”Pepita de oro”, o sea que en las primeras posiciones se indicaran las reglas cuyo antecedente sea menor que su consecuente, por ejemplo la regla: Arroz ¿Jabón, Desodorante, es una ”pepita de oro”, ya que un producto lleva a comprar dos más. Y cuando el usuario haga click en el encabezado *Confidence* de la tabla, las reglas se ordenaran de mayor a menor confianza. Para implementar el click sobre el encabezado de la tabla, se hace lo siguiente:

Se añade un “escucha” a la tabla a través del método: *addJTableHeaderListener*. El “escucha” que se encuentra en la función *addJTableHeaderListener*, va a estar pendiente del evento click del ratón mediante la clase *MouseAdapter*. Mediante el método de JTable, *convertColumnIndexToModel* se obtiene la columna en la cual el usuario hizo click. La tabla que muestra las reglas tiene 3 columnas, por tanto si el código retornado es 1 quiere decir que el usuario hizo click en la segunda columna, por tanto las reglas serán ordenadas de acuerdo al criterio “pepita de oro”. Pero si el código retornado es 2, entonces el criterio de ordenamiento es descendiente de acuerdo a la confianza. Para mayor detalle se puede observar el código fuente del método que supervisa los eventos del ratón, a continuación:

7.2.6 Paquete Conexión

El paquete *DBConnection* se encuentra contenido dentro del paquete *Icons* que a su vez se encuentra dentro del paquete GUI. En este paquete se encuentran las clases necesarias que soportan una conexión a bases de datos a través de un driver JDBC. Hacen parte de este paquete las clases *DBConnectionIcon, connectionWizard, Table, MyCanvasTable, SelectorTable y ScrollableTableModel*.

DBConnectionIcon. El paquete *DBConnectionIcon* es una clase que extiende a *Icon* del paquete GUI *KnowledgeFlow* y se encarga de proveer un menú desplegable que guíe por las etapas de conexión, selección y carga de datos desde una base de datos. La figura 7.13 muestra la implementación de la clase *DBConnectionIcon* con sus opciones de menú.

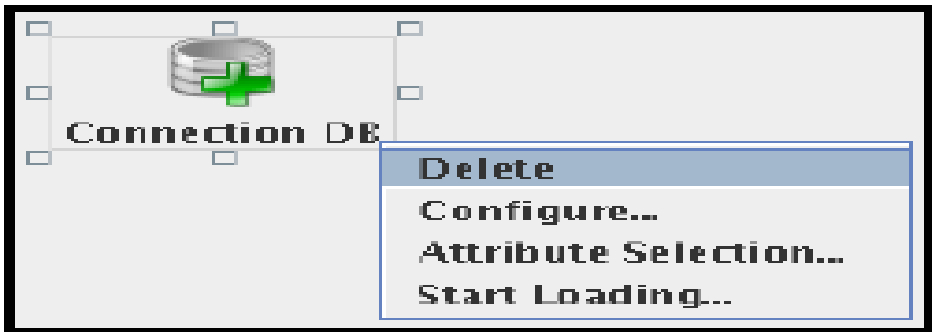

Figura 7.13: Implementación de la Clase *DBConnectionIcon Menú 1*

Al heredar de la clase *Icon*, *DBConnectionIcon* posee por defecto la opción *Delete* en su menú para eliminar este icono del área de trabajo. Al escoger la siguiente opción, Configure, se visualiza una instancia de la clase *connectionWizard*, la implementación de esta clase se puede ver en la figura 7.14.

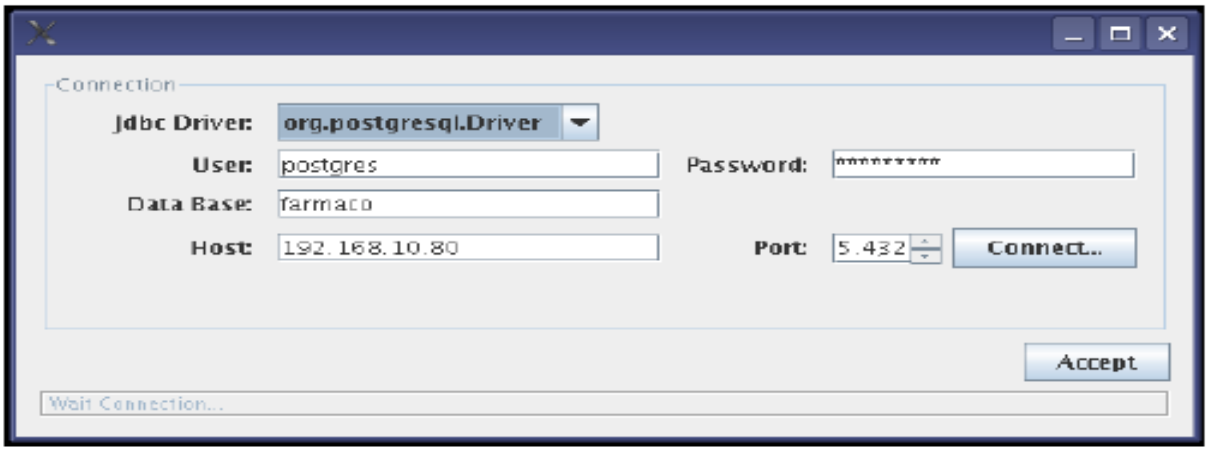

Figura 7.14: Implementación de la Clase *DBConnectionIcon Menú 2*

En ésta se recoge la información necesaria para establecer una conexión a través del controlador JDBC escogido en el campo JDBC Driver. En este punto hay que aclarar que se hace uso del API SQL de Java, que es un conjunto de clases dispuestas a la interacción con controladores JDBC y al cual se accede importando el paquete *java.sql*. Las clases e interfaces involucradas en la conexión a bases de datos son:

***java.sql.DriverManager*.** Provee los servicios básicos para el manejo de un conjunto de controladores JDBC
java.sql.DatabaseMetaData. Información comprensiva sobre la base de datos en su totalidad.
java.sql.Connection. Una conexión (sesión) con una base de datos específica. Se ejecutan las sentencias SQL y los resultados se devuelven dentro del contexto la conexión.
java.sql.Statement. El objeto usado para ejecutar una sentencia estética SQL y devolver los resultados que esta produce.
java.sql.ResultSet. Una tabla de los datos que representan un sistema del resultado de la base de datos, que es generado generalmente ejecutando una sentencia SQL que consulta a la base de datos.
java.sql.SQLException. Una excepción que proporciona la información de un error durante el acceso a bases de datos u otros errores durante la conexión.

A partir del nombre del Driver capturado en la forma se instancia la clase del controlador a través de la instrucción:

Class.forName(JDBCDriver name);

Para el caso de PostgreSQL la instrucción será la siguiente:

Class.forName("org.postgresql.Driver");

Con la información referente se construye la url hacia la base de datos y usando la clase *DriverManager* se obtiene una instancia de la *Interface Connection*. El siguiente fragmento de código ilustra esta acción:

```
url = CabeceraDelControlador + "//" + Servidor + ":" + Puerto+ "/" +NombreDeLaBaseDeDatos;
connection = DriverManager.getConnection(url, usuario, password);
```

Un ejemplo de la construcción de una conexión a una base de datos PostgreSQL, llamada minería, a través del usuario ”user” con los valores por defecto para el Servidor y el Puerto será la siguiente:

```
url = "jdbc:postgresql://localhost:5432/mineria";
connection = DriverManager.getConnection(url,"user","t3o4g2o");
```

La interfaz *Connection* obtenida al pulsar el botón Accept de la Clase *connectionWizard* devuelve una instancia de esta conexión al *DBConnectionIcon* que se almacena en la variable connection.

La siguiente opción en el menú desplegable de *DBConnectionIcon* es *Attribute Selection*. Esta alternativa genera un objeto de la clase *SelectorTable*. Esta clase utiliza diferentes objetos del *API Swing* de Java para desplegar y solicitar al usuario información referente al conjunto de datos que quiere minar. Usa un *JComboBox* para listar las tablas que pertenecen a la bases de datos con la cual se estableció conexión, *JRadioButtons* para solicitar al usuario cómo debe ser considerado el conjunto de datos, como un conjunto multivaluado o bajo la metodología de canasta de mercado, un *JTextArea* para mostrar la sentencia SQL que se está construyendo, un *JTable* que visualizará el contenido de la consulta que se logre construir y una instancia de la clase *MyCanvasTable*, que es una extensión de la clase *JPanel* y es utilizada para desplegar las tablas y relaciones que se establezcan para generar una consulta de manera visual usando la metodología *Drag'n Drop*.

Durante el constructor de la clase se carga el *JComboBox* con los nombres de las tablas que contiene la base de datos conectada. Para obtener esta información se utiliza el método *getTables* de la interfaz *DatabaseMetaData*. Este método devuelve un *ResultSet* que se recorre para alimentar un Vector que se pasa como parámetro en la construcción del *JComboBox*.

A partir de las tablas que despliegue el *JComboBox* podemos escoger de esa lista la(s) tabla(s) que queremos relacionar en el análisis. La figura 7.15 muestra esta acción. Éstas aparecerán dentro del área del *Attribute* Selector, una instancia de la Clase *MyCanvasTable*, donde podemos interactuar con las tablas de forma gráfica a y establecer relaciones al tiempo que construimos la sentencia SQL para consultar la base de datos. Esta sentencia se construye al interior de la Clase *JTextArea*

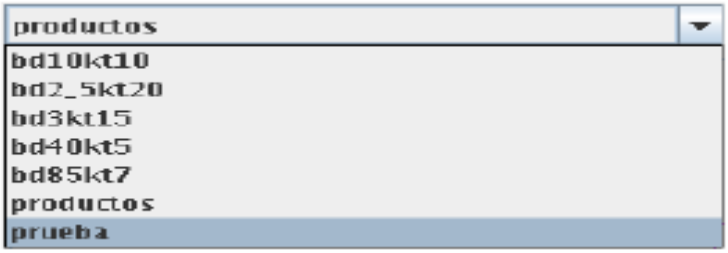

Figura 7.15: Tablas de la conexión organizadas en un *JComboBox*

Para la interacción de las tablas y las relaciones dentro de *MyCanvasTable* se crearon las Clases *Edge, ConectorTable, Attribute, Table. ConectorTable* es similar a la Clase Conector del paquete GUI *KnowledgeFlow* y es usado para establecer y marcar relaciones entre los atributos de las tablas involucradas. La Clase *Attribute* está conformado por una instancia de *ConectorTable* más un *JLabel* con el nombre del atributo y la posibilidad de una imagen que indique su selección. La particularidad de la Clase *ConectorTable* es que dependiendo del tipo de atributo desplegará una

imagen diferente como conector si se trata de un atributo de tipo clave primaria, clave foránea o clave mixta. Un ejemplo de esta situación se puede apreciar en la figura 7.16.

El conjunto de objetos *Attribute* conforman un objeto *Table* que puede estar conformado por un número *n* de atributos más un *JLabel* que sirva de título a la tabla. La implementación final de la Clase *Table* se puede apreciar en la figura 7.16.

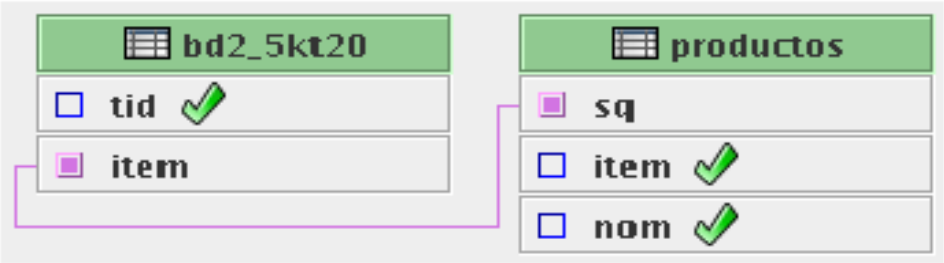

Figura 7.16: Implementación de la Clase *Table*

La Clase *Edge* cumple la función de guardar los objetos *ConectorTable* involucrados en una relación. De esta manera, la Clase *MyCanvasTable* guarda en un arreglo los objetos *Edges* que representan las relaciones establecidas hasta el momento de manera que al recorrerlo, pueda identificar los conectores involucrados, ubicarlos dentro de la forma y trazar líneas para representar las relaciones entre tablas. Este procedimiento es similar al efectuado en la interfaz principal *(Clase MyCanvas)* con los diferentes tipos de iconos y sus relaciones.

La Clase *MyCanvasTable* se encarga también de monitorear los eventos del ratón para identificar clicks dentro de las tablas seleccionadas y marcarlos, seleccionar una tabla para su desplazamiento y marcar relaciones entre los atributos de las tablas.

Al medida que se marca un atributo de una determinada tabla, éste se adicionará al *JTextArea* que construye la sentencia SQL dentro de su campo SELECT y se incluirá el nombre de la tabla de ese atributo dentro del campo FROM, de la misma forma, al establecer una relación se identifican las tablas relacionadas y se incluyen en el campo WHERE de la consulta. Cuando la selección de atributos ha finalizado se puede ver una preliminar de la consulta al pulsar el botón *Execute*. Aquí se instancia un objeto de la Clase *ScrollableTableModel* al cual se pasa como parámetros el atributo *connection* de la clase *MyCanvasTable*, que representa la conexión a la base de datos, junto con una versión textual del contenido del *JTextArea*, que se constituye como la sentencia SQL que queremos consultar.

La Clase *ScrollableTableModel* se encarga de realizar las consultas a la base de datos usando las interfaces *Statament y ResulSet*. La interfaz *Statement* se crea a partir de la interfaz *Connection* que llega como parámetro en el constructor de la clase. Esta interfaz, a través de su método *executeQuery (String query*) dispara una consulta a la base de datos y retorna el resultado en una variable de tipo *ResulSet*. El *query* que se pasa como parámetro a este método es el que se construyó como parámetro al constructor de esta clase.

La Clase *ScrollableTableModel* es la encargada de recorrer el *ResulSet* que contiene el resultado de la consulta y construir un modelo de tabla que es pasado al *Jtable* de la Clase *MyCanvasTable* para visualizar los resultados de la consulta como se puede apreciar en la figura 7.17.

nommat	nombre	nfinal	estado
Compiladores	Alvaro Molina	4.2	A
Telematica I	Alvaro Molina	3.3	A
Bases de Datos	Alvaro Molina	2.4	R
S.O. II	Antonio Gonzalez	4.4	A
Admon. C. Computo	Antonio Gonzalez	3.7	A

Figura 7.17: Construcción de la previsualización de datos en un JTable

Una vez validados los datos que se quiere minar se confirma la aceptación al pulsar el botón *Accept*. En ese momento devolverá la interfaz principal y el último paso para construir el conjunto de datos se realiza a través de la opción *Start Loading* del menú contextual de la clase *DBConnectionIcon* donde, según sea el caso, se invocarían los métodos *loadMarketBasketDataSet*(), para cargar una instancia del *DataSet* que cubre el modelo de canasta de marcado (Tablas Univaluadas), o *loadMultivaluedDataSet*(), para cargar instancias de *DataSet* de conjuntos multivaluados. Esta instancia de la Clase *DataSet* reposará como atributo de la Clase *DBConnectionIcon* para ser pasada en el momento de una conexión a instancias de las Clases *filterIcon* o *AssociationIcon*.

Paquete file. A través de este paquete el usuario puede establecer una conexión con un Archivo Plano, para de esta forma obtener el conjunto de datos. El paquete file forma parte del paquete *Icons*, que a su vez forma parte del paquete GUI. Las clases que conforman este paquete son: *fileIcon, Openfile y fileTableModel.*

fileIcon extiende a la clase *Icon* que forma parte del paquete *KnowledgeFlow* y a su vez del paquete GUI, por tanto automáticamente hereda su menú con la opción por defecto *delete*, la cual permite al usuario eliminar el icono de conexión al Archivo Plano. Por otra parte, la clase *fileIcon* añade al menú de *Icon* las opciones *Open* y *Load*.

La opción *Open* es la encargada de abrir una ventana en la cual el usuario elige el conjunto de datos que va a minar, esta opción se maneja a través de la clase *Openfile.* La clase *Openfile* muestra el conjunto de datos a través de una *Jtable*. La clase de *Swing, JfileChooser* mediante su método *getAbsolutePath*, retorna como una cadena de texto la ruta completa en donde se encuentra el archivo.

La cadena de texto obtenida pasa como parámetro al constructor de la clase *fileTableModel*, la cual se encarga de construir una JTable a través de la cual se muestran los datos del archivo plano.

A través de la clase *fileTableModel* se construye un modelo propio para la *Jtable* que indicará los datos. Construir un modelo para una JTable es suministrarle la información concerniente al nombre y número de las columnas y las filas, en el caso de las filas se deben suministrar los datos.

Para construir un modelo propio para JTable, la clase *fileTableModel*, debe extender a *AbstractTableModel*, quien implementa los métodos necesarios para la construcción de una JTable, se debe también implementar los métodos *getColumnName*(int col), *getRowCount*(), *getColumnCount*() y *getValueAt(int rowIndex, int colIndex*) de acuerdo al tipo de estructura que almacena los datos, entonces, a través de la cadena de texto que indica la ruta completa en donde se encuentra el archivo plano, se abre un flujo con tal archivo y mediante el método *dataAndAtributes*

de la clase *fileManager*, encargada de administrar todo lo referente a Archivos Planos, se obtiene la siguiente información del archivo seleccionado por el usuario: en un array de objetos retorna los nombres de las columnas y en una matriz de objetos los datos en sí. De esta forma y teniendo los métodos antes mencionados debidamente implementados, los datos se mostrarán en una *Jtable* de acuerdo al modelo proporcionado.

La opción Load se encarga de, a partir del flujo de comunicación abierto con un archivo plano, construir un *DataSet* o estructura en forma de árbol N-Ario para almacenar los datos de manera comprimida.

7.2.7 Paquete filtros

El módulo filtros o *data cleaning*, se encarga de hacer un refinamiento de los datos en dos etapas, por un lado hace un proceso de limpieza sobre datos corruptos, vacíos, ruidosos, inconsistentes, duplicados, alterados etc., por otro lado hace una selección de estos datos para escoger aquellos que brinden información de calidad, aplicando distintas técnicas como son muestreos, discretizaciones y otras. De esta forma obtenemos datos depurados según el objetivo del analista, para que posteriormente se pueda aplicar el núcleo KDD o de minería de Datos sobre datos coherentes, limpios y consistentes.

A este módulo, pertenecen los filtros: *Remove Missing, Update Missing, Selection, Range, Reduction, Codification, Replace Value, Numeric Range y Discretize*, los cuales extienden la clase *AbstractTableModel* de Java, con el objetivo de alimentarse y presentar sus resultados a través de la clase denominada TableModel, que es el medio por el cual comunican sus flujos de datos, es así como los filtros pueden recibir los datos de entrada de otros filtros o de la conexión directa de a las bases de datos, pero siempre utilizando la clase TableModel. De la misma forma los datos de salida en los filtros, se enviarán utilizando el mismo filtro *RemoveMissing* formato.

Las clases que se encargan de Mostrar resultados y de hacer la configuración de los filtros, son todos los módulos *Show y Open*, respectivamente, estos extienden la clase de Java denominada *javax.swing.JFrame*, ya que presentan una interfaz a modo de ventana que permite mantener un dialogo constante con el analista.

Filtro RemoveMissing. El objetivo especifico de este filtro, es eliminar todas las transacciones que contengan campos vacíos.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
2	Alta	Alto		Si	No	
3	Baja	Alto	Bajo	No	Si	No
4		Alto		No		
5	Media	Bajo	Alto	Si	Si	Si
6	Baja	Bajo	Alto	Si	Si	
7	Alta	Bajo	Alto		No	
8	Alta	Bajo	Bajo		Si	No
9	Alta	Alto	Bajo	Si		Si
10	Baja	Bajo	Alto	Si		
11	Media	Bajo	Bajo		Si	No
12	Alta		Alto		Si	Si
13	Baja	Alto	Alto	Si	Si	No
14	Baja	Alto		No	No	Si

Figura 7.18: Datos de entrada antes de aplicar *RemoveMissing*

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
3	Baja	Alto	Bajo	No	Si	No
5	Media	Bajo	Alto	Si	Si	Si
13	Baja	Alto	Alto	Si	Si	No

Figura 7.19: Datos de Salida después de aplicar *RemoveMissing*

Filtro UpdateMissing. El objetivo especifico de este filtro, es remplazar los campos vacíos de un atributo específico, por un valor otorgado por el analista. El filtro *UpdateMissing* consta de 3 clases las cuales son *OpenUpdateMissing, ShowUpdate Missing* y el núcleo principal la clase *UpdateMissing.*

Ejemplo: Al aplicar el filtro *UpdateMissing*, en la tabla de la figura 7.20 se remplaza en el atributo "ALERGIA ANTIBIOTICO", todos los campos vacíos, por el valor "JC". La tabla resultante se muestra en la figura 7.21.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
2	Alta	Alto		Si	No	
3	Baja	Alto	Bajo	No	Si	No
4		Alto		No		
5	Media	Bajo	Alto	Si	Si	Si
6	Baja	Bajo	Alto	Si	Si	
7	Alta	Bajo	Alto		No	
8	Alta	Bajo	Bajo		Si	No
9	Alta	Alto	Bajo	Si		Si
10	Baja	Bajo	Alto	Si		
11	Media	Bajo	Bajo		Si	No
12	Alta		Alto		Si	Si
13	Baja	Alto	Alto	Si	Si	No
14	Baja	Alto		No	No	Si

Figura 7.20: Datos de Entrada antes de aplicar UpdateMissing

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
2	Alta	Alto		Si	No	
3	Baja	Alto	Bajo	No	Si	No
4		Alto		No		
5	Media	Bajo	Alto	Si	Si	Si
6	Baja	Bajo	Alto	Si	Si	
7	Alta	Bajo	Alto	JC	No	
8	Alta	Bajo	Bajo	JC	Si	No
9	Alta	Alto	Bajo	Si		Si
10	Baja	Bajo	Alto	Si		
11	Media	Bajo	Bajo	JC	Si	No
12	Alta		Alto	JC	Si	Si
13	Baja	Alto	Alto	Si	Si	No
14	Baja	Alto		No	No	Si

Figura 7.21: Datos de Salida después de aplicar *UpdateMissing*

Filtro Selection. El objetivo especifico de este filtro, es hacer una selección de atributos y del atributo objetivo sobre un conjunto de entrada. El filtro *Selection* consta de 3 clases las cuales son *OpenSelection, ShowSelection* y el núcleo principal *Selection.*

Ejemplo: En la tabla original figura 7.22 se puede observar 6 Atributos, al aplicar el filtro *Selection*, se puede observar en la figura 7.23 la tabla que sólo contiene los atributos sobre los cuales hicimos la selección, los cuales son: AZUCAR SANGRE, ALERGIA ANTIBIOTICO y como atributo objetivo ADMINISTRAR FARMACO.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	Si
2	Alta	Alto	Alto	Si	No	Si
3	Baja	Alto	Bajo	No	No	Si
4	Media	Alto	Alto	No	Si	No
5	Media	Bajo	Alto	Si	Si	No
6	Baja	Bajo	Alto	Si	Si	Si
7	Alta	Bajo	Alto	Si	No	Si
8	Alta	Bajo	Bajo	No	Si	Si
9	Alta	Alto	Bajo	Si	Si	No
10	Baja	Bajo	Alto	Si	Si	Si
11	Media	Bajo	Bajo	Si	Si	Si
12	Alta	Bajo	Alto	Si	Si	No
13	Baja	Alto	Alto	Si	Si	Si
14	Baja	Alto	Bajo	No	No	Si

Figura 7.22: Datos de Entrada 1

AZUCAR_SANGRE	ALERGIA_ANTIBIOTICO	ADMINISTRAR_FARMACO_F
Alto	No	Si
Alto	Si	Si
Alto	No	Si
Alto	No	No
Bajo	Si	No
Bajo	Si	Si
Bajo	Si	Si
Bajo	No	Si
Alto	Si	No
Bajo	Si	Si
Bajo	Si	Si
Bajo	Si	No
Alto	Si	Si
Alto	No	Si

Figura 7.23: Datos de Salida después de aplicar *Selection*

Filtro Range. El objetivo principal de este filtro, es escoger una muestra sobre un conjunto de entrada, especialmente útil para minería con algoritmos de clasificación. El filtro *Range* consta de 3 clases las cuales son *OpenRange, ShowRange* y el núcleo principal, la clase *Range.*

Técnica de Aleatorios: Esta técnica se encarga de seleccionar una muestra del conjunto de entrada aleatoriamente, a partir de una semilla y la función *Random* de Java. Como ejemplo apliquemos este filtro a la tabla de la figura 7.22

	PRESION_ARTERIAL	AZUCAR_SANGRE	INDICE_COLESTER...	ALERGIA_ANTIBIOTI...	OTRAS_ALERGIAS	ADMINISTRAR_FAR...
2	Alta	Alto	Alto	Si	No	Si
10	Baja	Bajo	Alto	Si	Si	Si
12	Alta	Bajo	Alto	Si	Si	No
11	Media	Bajo	Bajo	Si	Si	Si
5	Media	Bajo	Alto	Si	Si	No

Figura 7.24: Datos de Salida después de aplicar la Técnica de Aleatorios

Técnica de 1 en n. Esta técnica se encarga de seleccionar una muestra, que tomara cada transacción en saltos de *n* en *n, n* es el valor de salto otorgado por el analista. Como ejemplo apliquemos este filtro a la tabla de la figura 7.22.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	Si
4	Media	Alto	Alto	No	Si	No
7	Alta	Bajo	Alto	Si	No	Si
10	Baja	Bajo	Alto	Si	Si	Si
13	Baja	Alto	Alto	Si	Si	Si

Figura 7.25: Datos de Salida después de aplicar la Técnica de 1 en *n*

Técnica de Primeros n. Esta técnica se encarga de seleccionar una muestra, a partir de las primeras **n** transacciones del conjunto de datos de entrada. Como ejemplo apliquemos este filtro a la tabla de la figura 7.22.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	Si
2	Alta	Alto	Alto	Si	No	Si
3	Baja	Alto	Bajo	No	No	Si
4	Media	Alto	Alto	No	Si	No
5	Media	Bajo	Alto	Si	Si	No
6	Baja	Bajo	Alto	Si	Si	Si
7	Alta	Bajo	Alto	Si	No	Si

Figura 7.26: Datos de Salida después de aplicar la Técnica de primeros *n*

Filtro Reduction. El objetivo específico de este filtro, es hacer una reducción en el número de transacciones, manteniéndolas o eliminándolas, con diferentes técnicas y parámetros de selección. El filtro *Reduction* consta de 3 clases las cuales son *OpenReduction, ShowReduction* y el núcleo principal *Reduction*.

Técnica de Reducción por Rango. Esta técnica se encarga de reducir el conjunto de entrada, en un rango de transacciones, a partir de una transacción inicial y otra final, de acuerdo al parámetro escogido por el analista, el cual puede ser eliminar o mantener dichas transacciones. Apliquemos este filtro a la tabla de la figura 7.22.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8

Figura 7.27: Reducción por rango, Manteniendo los datos

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
11	Media	Bajo	Bajo	Si	Si	3
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1
14	Baja	Alto	Bajo	No	No	7

Figura 7.28: Reducción por rango, Removiendo los datos

Técnica de Reducción de Transacciones por Atributo. Esta técnica se encarga de reducir el conjunto de entrada, dependiendo del tipo de datos que contenga el atributo seleccionado, los cuales pueden ser alfabéticos o numéricos, si son numéricos se debe suministrar un valor limítrofe y si son alfabéticos el analista deberá seleccionar los atributos de su interés. Además de acuerdo al parámetro escogido por el analista, el filtro eliminará o mantendrá las transacciones seleccionadas.

Ejemplo: Apliquemos ahora el filtro con valores alfabéticos, y seleccionemos como atributo de reducción a "INDICE COLESTEROL", y como valor de este atributo a "ALTO". También escogemos como parámetro de selección, Mantener el conjunto, Esto significa que el filtro buscará cualquier valor diferente a "ALTO" en este atributo, y eliminará su transacción correspondiente como se muestra en la figura 7.29

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
10	Baja	Bajo	Alto	Si	Si	8
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1

Figura 7.29: Reducción por atributo Alfabético, Manteniendo los datos

Filtro Codification. El objetivo específico de este filtro, es realizar una codificación sobre el conjunto de datos de entrada. El filtro *Codification* consta de 2 clases las cuales son *ShowCodification* y el núcleo principal la clase *Codification*. Este filtro realiza la codificación, asignando un código único a cada valor de un atributo, a lo largo de toda la tabla, además crea un diccionario de datos, para su posterior decodificación. Como ejemplo apliquemos este filtro a la tabla de la figura 7.22.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
0	3	5	7	9	11
0	3	5	8	9	12
1	3	6	7	9	13
2	3	5	7	10	14
2	4	5	8	10	15
1	4	5	8	10	16
0	4	5	8	9	17
0	4	6	7	10	18
0	3	6	8	10	19
1	4	5	8	10	16
2	4	6	8	10	15
0	4	5	8	10	19
1	3	5	8	10	11
1	3	6	7	9	19

Figura 7.30: Datos de Salida, después de aplicar la Codificación

Paralelamente se crea el diccionario de datos, para la posterior decodificación, como se muestra en la figura 7.31

INDICE	ATRIBUTO	VALOR
0	PRESION_ARTERIAL	Alta
1	PRESION_ARTERIAL	Baja
2	PRESION_ARTERIAL	Media
3	AZUCAR_SANGRE	Alto
4	AZUCAR_SANGRE	Bajo
5	INDICE_COLESTEROL	Alto
6	INDICE_COLESTEROL	Bajo
7	ALERGIA_ANTIBIOTICO	No
8	ALERGIA_ANTIBIOTICO	Si
9	OTRAS_ALERGIAS	No
10	OTRAS_ALERGIAS	Si
11	ADMINISTRAR_FARMACO_F	1
12	ADMINISTRAR_FARMACO_F	4
13	ADMINISTRAR_FARMACO_F	9
14	ADMINISTRAR_FARMACO_F	6
15	ADMINISTRAR_FARMACO_F	3
16	ADMINISTRAR_FARMACO_F	8
17	ADMINISTRAR_FARMACO_F	2
18	ADMINISTRAR_FARMACO_F	5
19	ADMINISTRAR_FARMACO_F	7

Figura 7.31: Diccionario de datos

Filtro ReplaceValue. El objetivo específico de este filtro, es remplazar uno o varios valores de un atributo seleccionado, por otro valor suministrado por el analista. El filtro *RemplaceValue* consta de 3 clases las cuales son: *OpenRemplaceValue*, *ShowRemplaceValue* y el núcleo principal la clase *RemplaceValue.*

> **Ejemplo:** apliquemos este filtro a la tabla de la figura 7.22, el atributo "PRESION ARTERIAL" tiene 3 valores, "Baja", "Media" y "Alta", seleccionamos los valores "Baja" y "Alta" del atributo, para ser remplazados por el valor "Extremos", por consiguiente la nueva tabla quedaría como se muestra en la figura 7.32.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Extremos	Alto	Alto	No	No	1
Extremos	Alto	Alto	Si	No	4
Extremos	Alto	Bajo	No	No	9
Media	Alto	Alto	No	Si	6
Media	Bajo	Alto	Si	Si	3
Extremos	Bajo	Alto	Si	Si	8
Extremos	Bajo	Alto	Si	No	2
Extremos	Bajo	Bajo	No	Si	5
Extremos	Alto	Bajo	Si	Si	7
Extremos	Bajo	Alto	Si	Si	8
Media	Bajo	Bajo	Si	Si	3
Extremos	Bajo	Alto	Si	Si	7
Extremos	Alto	Alto	Si	Si	1
Extremos	Alto	Bajo	No	No	7

Figura 7.32: Datos de Salida, después de aplicar el filtro *RemplaceValue*

Filtro NumericRange. El objetivo específico de este filtro, es eliminar los valores de un atributo numérico, que están por fuera de un rango determinado por el analista. El filtro *NumericRange* consta de 3 clases las cuales son *OpenNumericRange*, *ShowNumericRange* y el núcleo principal la clase *NumericRange.*

> **Ejemplo:** Al aplicar el filtro *NumericRange*, con un límite inferior igual a 2 y un límite superior igual a 5, sobre el atributo numérico "ADMINISTRAR FARMACO", se eliminan 9 valores, permaneciendo en la tabla los valores comprendidos en dicho rango incluyendo los valore limites 2 y 5. El nuevo conjunto de datos se muestra en la figura 7.34.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8
11	Media	Bajo	Bajo	Si	Si	3
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1
14	Baja	Alto	Bajo	No	No	7

Figura 7.33: Datos de Entrada 2

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	
Alta	Alto	Alto	Si	No	4
Baja	Alto	Bajo	No	No	
Media	Alto	Alto	No	Si	
Media	Bajo	Alto	Si	Si	3
Baja	Bajo	Alto	Si	Si	
Alta	Bajo	Alto	Si	No	2
Alta	Bajo	Bajo	No	Si	5
Alta	Alto	Bajo	Si	Si	
Baja	Bajo	Alto	Si	Si	
Media	Bajo	Bajo	Si	Si	3
Alta	Bajo	Alto	Si	Si	
Baja	Alto	Alto	Si	Si	
Baja	Alto	Bajo	No	No	

Figura 7.34: Datos de Salida, después de aplicar el filtro *NumericRange*

Filtro Discretize. El objetivo específico de este filtro, transformar un valor numérico discontinuo en un rango continuo. El filtro *Discretize* consta de 3 clases las cuales son *OpenDiscretize*, *ShowDiscretize* y el núcleo principal la clase *Discretize. El filtro suministra dos técnicas des discretización las cuales son:*

Técnica de discretizar con número de rangos. Esta técnica se encarga de tomar el mínimo y el máximo valor del atributo numérico seleccionado, con el objetivo de hacer una división clasificatoria, dependiendo del número de rangos otorgado por analista y también teniendo en cuenta los rangos extremos: desde -infinito hasta el mínimo valor y desde el máximo valor hasta +infinito.

Ejemplo: En primera instancia aplicamos la técnica de discretizar con número de rangos a la tabla de la figura 7.33, seleccionamos al atributo numérico "ADMINISTRAR FARMACO", y segmentaremos al conjunto en 3 rangos, lo cual significa que se construirán 3 rangos con valores continuos, de los cuales 2 pertenecen a los rangos extremos: desde el mínimo valor hasta infinito y desde el máximo valor hasta +infinito, ubicando de esta forma el valor del atributo en un rango determinado, la tabla con los valores discretizados con esta técnica, se muestra en la figura 7.35.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	(-Infinity : 1]
Alta	Alto	Alto	Si	No	(1.0 : 8.0]
Baja	Alto	Bajo	No	No	[9 : + Infinity)
Media	Alto	Alto	No	Si	(1.0 : 8.0]
Media	Bajo	Alto	Si	Si	(1.0 : 8.0]
Baja	Bajo	Alto	Si	Si	(1.0 : 8.0]
Alta	Bajo	Alto	Si	No	(1.0 : 8.0]
Alta	Bajo	Bajo	No	Si	(1.0 : 8.0]
Alta	Alto	Bajo	Si	Si	(1.0 : 8.0]
Baja	Bajo	Alto	Si	Si	(1.0 : 8.0]
Media	Bajo	Bajo	Si	Si	(1.0 : 8.0]
Alta	Bajo	Alto	Si	Si	(1.0 : 8.0]
Baja	Alto	Alto	Si	Si	(-Infinity : 1]
Baja	Alto	Bajo	No	No	(1.0 : 8.0]

Figura 7.35: Datos de Salida al aplicar el filtro *Discretize* con Número de Rangos

Técnica de discretizar con el tamaño del rango. Esta técnica se encarga de delimitar rangos, en incrementos, según el tamaño del rango otorgado por el analista, dentro de los valores comprendidos entre el mínimo y máximo valor del atributo numérico seleccionado, también se tiene en cuenta los rangos extremos los cuales son desde el infinito hasta el mínimo valor, y desde el máximo valor hasta el + infinito.

Ejemplo: Aplicamos la técnica de discretizar con el tamaño del rango a la tabla de la figura 7.33, seleccionamos el atributo numérico"ADMINISTRAR FARMACO", y escogemos como tamaño del rango el valor 3, lo cual significa que los rangos se construirán en incrementos de 3, de esta forma se crean 5 rangos con valores continuos, de los cuales 2 pertenecen a los rangos extremos: desde el mínimo valor hasta infinito y desde el máximo valor hasta +infinito. La tabla resultante se muestra en la figura 7.36.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	(-Infinity : 1]
Alta	Alto	Alto	Si	No	(1.0 : 4.0]
Baja	Alto	Bajo	No	No	[9 : + Infinity)
Media	Alto	Alto	No	Si	(4.0 : 7.0]
Media	Bajo	Alto	Si	Si	(1.0 : 4.0]
Baja	Bajo	Alto	Si	Si	(7.0 : 8.0]
Alta	Bajo	Alto	Si	No	(1.0 : 4.0]
Alta	Bajo	Bajo	No	Si	(4.0 : 7.0]
Alta	Alto	Bajo	Si	Si	(4.0 : 7.0]
Baja	Bajo	Alto	Si	Si	(7.0 : 8.0]
Media	Bajo	Bajo	Si	Si	(1.0 : 4.0]
Alta	Bajo	Alto	Si	Si	(4.0 : 7.0]
Baja	Alto	Alto	Si	Si	(-Infinity : 1]
Baja	Alto	Bajo	No	No	(4.0 : 7.0]

Figura 7.36: Datos de Salida, antes de aplicar el filtro Discretize con el tamaño del rango.

8. VISUALIZACIÓN

Una visualización es eficiente si cuenta una historia con los recursos precisos y oportunos, de forma adecuada para la comprensión humana. Uno de esos recursos es la usabilidad, o facilidad de uso que hace que la visualización de los gráficos sean legibles e intuibles, utilizando el espacio, los colores, las formas y la holística del contenido de la manera más optima, un segundo criterio a considerar, es la demanda computacional que exige dicha visualización, pues generalmente el volumen de los datos suele ser bastante grande.

En Tariy, las técnicas de visualización que se han implementado, son gráficos de árboles enearios y gráficos multivariantes, los cuales son muy adecuados para la extracción analítica de conocimiento en bases de datos, que acompañadas de técnicas de minería de datos hacen un equipo óptimo para la obtención de resultados.

8.1 Gráficos de Árbol

Los arboles, son gráficos de jerarquía de una sola pieza que no tienen ciclos (num), los arboles simplifican las tareas de toma de decisiones y clasificación, abordan el objeto de estudio de una forma bastante ilustrativa sin embargo en grandes volúmenes de datos, se incurre en la pérdida del contexto.

En Tariy se ha optado por representar tres tipos de árboles, para graficar las reglas resultantes de aplicar los algoritmos de Mineria de Datos C45, y MateTree, en la tarea de clasificación, los cuales generan tanto el gráfico a ser ilustrado, como las reglas consecuentes del proceso. Dichos arboles van desde una forma simple de representación como un árbol en forma textual Figura 8.1, pasando por el típico árbol de jerarquía de carpetas Figura 8.2, y por último una forma mucho más interactiva y eficaz de representar la información como lo es el árbol Weka Figura 8.3, el cual, a parte de la representación, permite la interacción por parte del usuario brindado la posibilidad de redimensionarlo para que sea manipulable (Figura 8.4), y que no pierda el contexto general. Asimismo, ofrece la posibilidad de observar la proporción relativa en términos de porcentaje de las variables de un atributo Figura 8.4, y permite el auto escalamiento, para tener una panorámica general (Figura 8.3).

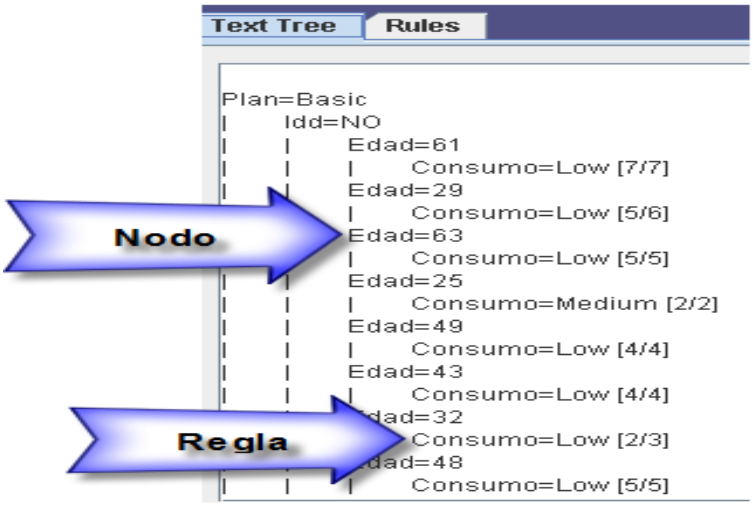

Figura 8.1: Árbol Textual

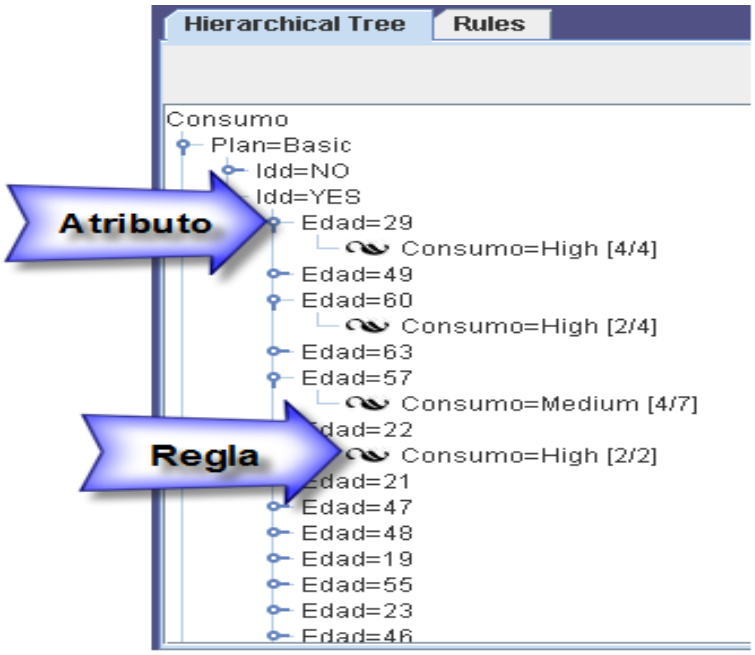

Figura 8.2: Árbol Jerárquico de Carpetas

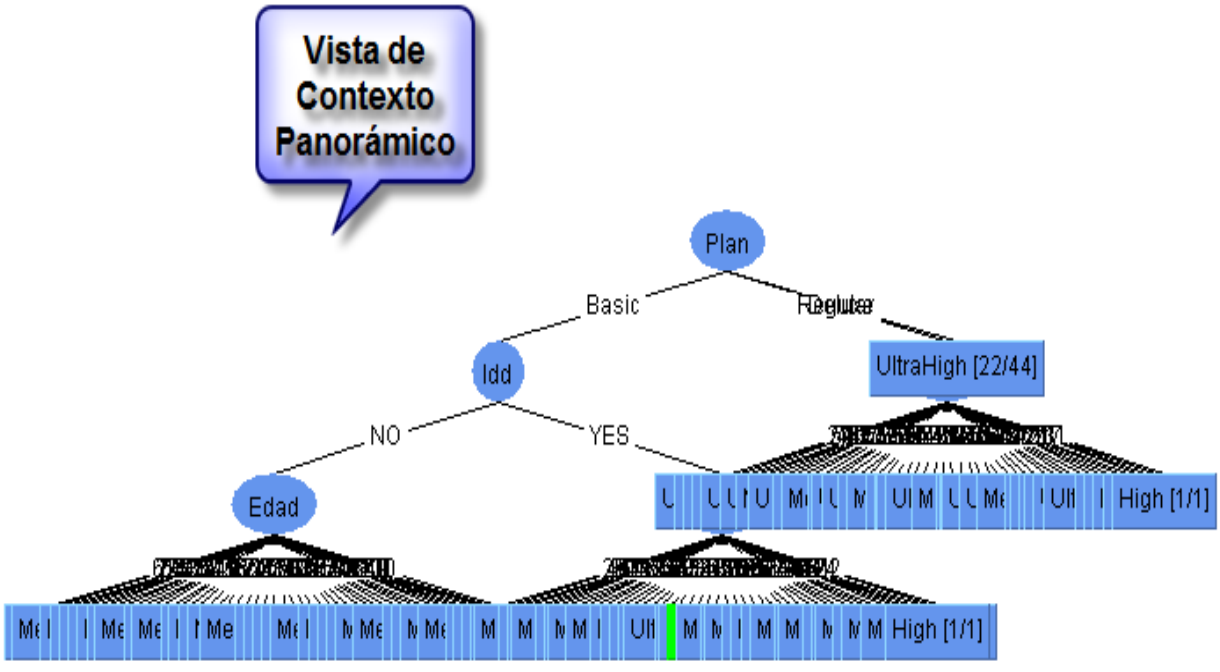

Figura 8.3: Árbol Weka Contexto Panorámico

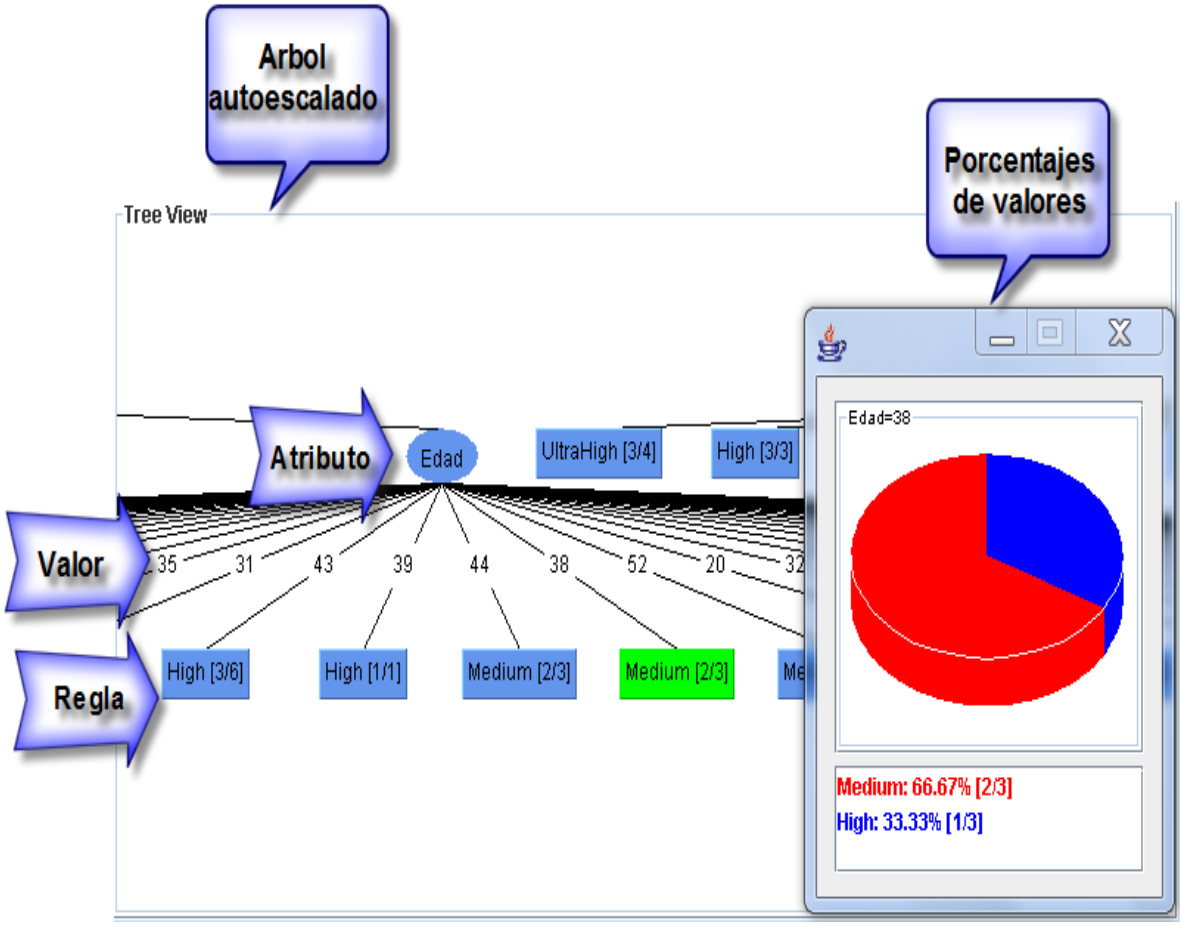

Figura 8.4: Árbol Weka Auto escalado y con porcentaje de resultados

8.2 GRÁFICO MULTIDIMENSIONAL

En este tipo de grafico no se reduce la dimensionalidad del conjunto, pues esto implicaría pérdida de información, y la visualización multivariante intenta representar la información con la mínima pérdida [32].

Una forma de visualización muy eficiente en cuanto a espacio y legibilidad de información es la representación radial, la cual es una técnica de visualización multidimensional capaz de representar conjuntos de datos de tres o más dimensiones en un espacio bidimensional.

Tomamos como base una circunferencia que es segmentada en el número de clases (columnas) del mismo modo que un gráfico de pastel, de forma equidistante de manera que se tiene una repartición del espacio en n dimensiones representadas por radios que parten del centro del círculo de forma análoga a las coordenadas paralelas pero con mejor uso del espacio. Cada eje de la clase es su vez dividido teniendo en cuanta el número de atributos de dicha clase, de manera que aproveche al máximo el espacio del segmento de radio (Figura 8.5).

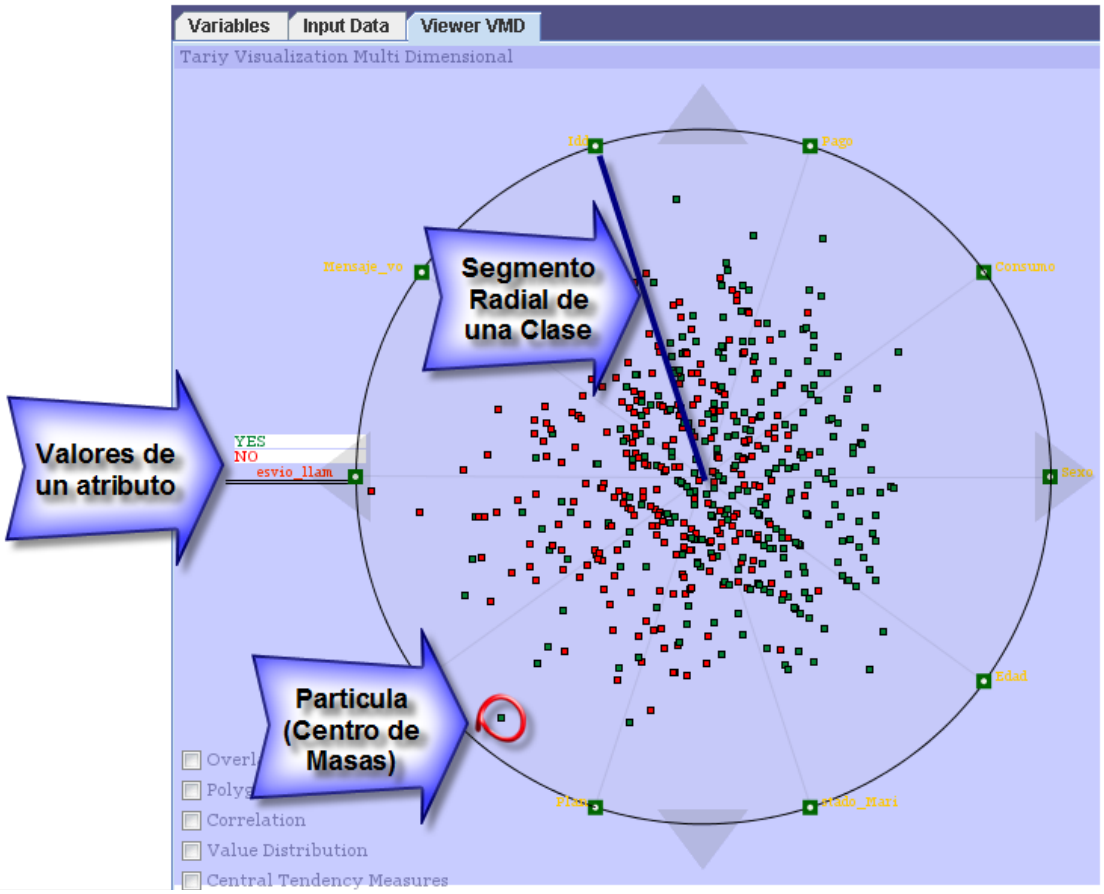

Figura 8.5: Grafico Multivariante, División y Segmentación

En primera instancia se hace una representación de los registros (filas) por medio de puntos ya que es la forma más simple de representación (Figura 10), además no se ve afectado por los cambios en el espacio de trabajo como por ejemplo el escalado o la rotación, dichos puntos son ubicados en las coordenadas donde la suma de las fuerzas de los ejes sea igual a cero [38].

Como se mencionó anteriormente, cada registro es representado por un único punto que se posiciona dentro del sistema radial por medio de un método de la física "El Centro de Masas" [36], el cual es un punto geométrico que dinámicamente se comporta como si estuviese sometido a la resultante de las fuerzas sobre él, obteniendo una posición única en el plano en función de todas sus variables.

En cada encabezado el usuario tiene la posibilidad de consultar la variable categórica, de manera que haciendo clic sobre el nombre de la clase aparece un menú que lista cada una de los distintos atributos. A cada uno de ellos se le puede asociar un color que represente a dicho atributo en el conjunto total (Figura 8.6). La manera de realizar ésta representación es coloreando los puntos en el plano. Sobre los radios de las clases se distribuyen los valores de acuerdo al porcentaje de ocurrencia, los cuales se pueden visualizar haciendo clic sobre el radiobutton de la clase.

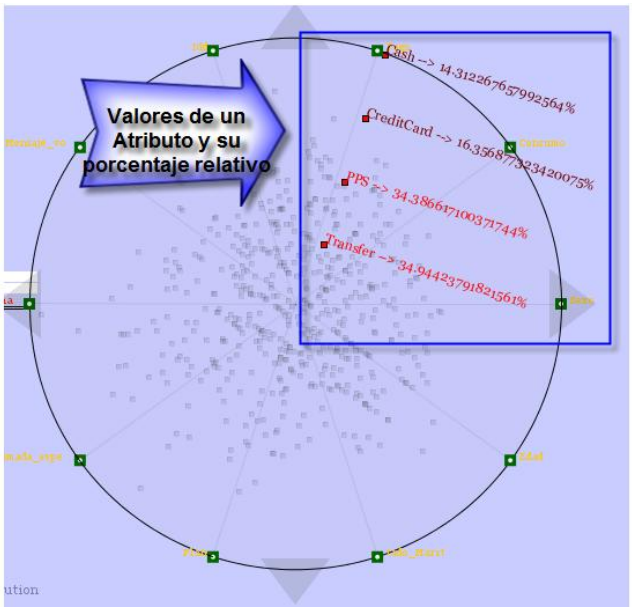

Figura 8.6: Grafico Multivariante, Valores de un Atributo

Una de las opciones a ser representadas es la vista por polígonos (Figura 8.7), que es equivalente a la representación en coordenadas paralelas circulares. Con los polígonos se visualiza todas las relaciones de las transacciones de la base de datos. Esta representación puede ser tanto individual como grupal, la vista individual se realiza haciendo clic sobre el punto que representa el registro. De esta forma por medio de un polígono relaciona todos los valores de los atributos para este registro. La representación de polígonos grupales es simplemente la presentación simultánea de los polígonos individuales de todas las partículas dibujadas sobre la circunferencia, este tipo de gráfico es muy útil ya que puede otorgar las capacidades de descubrimiento de patrones de las coordenadas paralelas, así como la posibilidad de comparar clases o grupos de clases.

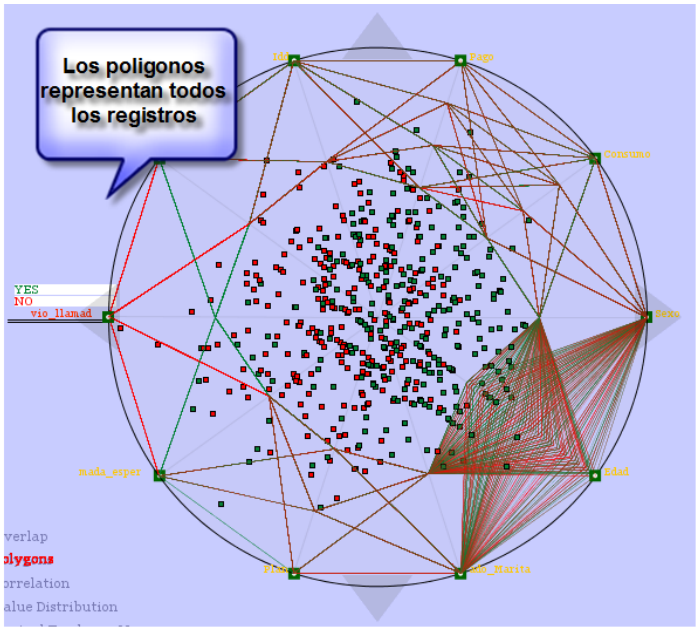

Figura 8.7: Grafico Multivariante, Vista por Polígonos

Se incluye también gráficos por distribuciones, los cuales pretenden resaltar algunas características importantes para aportar conocimiento sobre donde están concentrados y distribuidos la mayoría de los valores de cada variable desde una vista global, y destacarlo mediante recursos visuales como la preatentiva, que se logra al ensanchar ó estrechar el radio que representa la variable, con objeto de destacar las zonas de la escala de valores en donde existe una mayor concentración ó frecuencia (Figura 8.8), con el objetivo de resumir e interpretar conjuntos de datos, describiendo los contenidos y exponiendo las características principales.

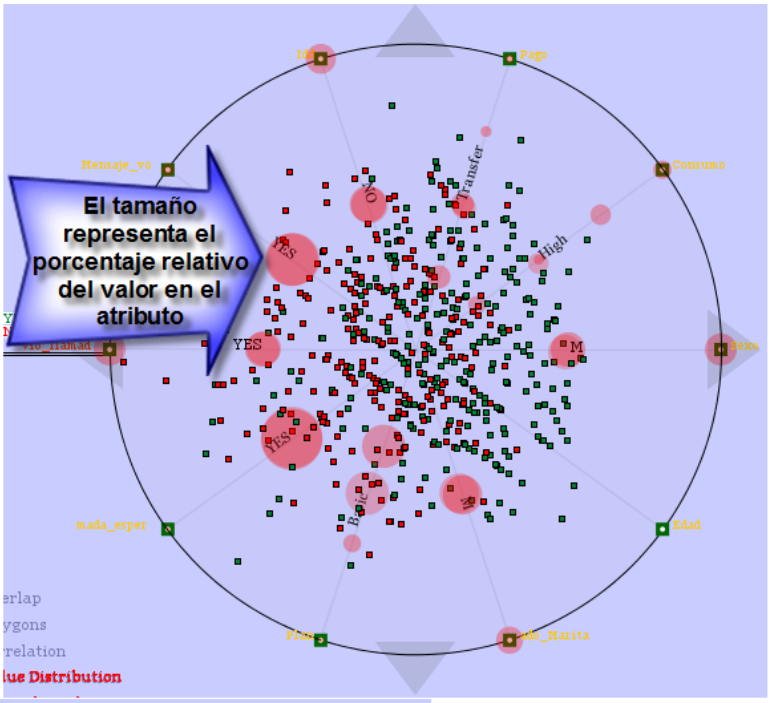

Figura 8.8: Grafico Multivariante, Vista por Distribución

9. PRUEBAS Y RESULTADOS

9.1 Rendimiento Algoritmos de Asociación

El conjunto de datos utilizados en las pruebas pertenecen a las transacciones de uno de los supermercados más importantes del departamento de Nariño (Colombia) durante un periodo determinado. El conjunto de datos contiene 10.757 diferentes productos. Los conjuntos de datos minados con la herramienta TariyKDD se muestran en la tabla 8.1.

Tabla 9.1: Conjuntos de Datos y Nomenclatura

Nomenclatura	Numero de Registros	Numero de Transaccions	Promedio items por transaccion
BD85KT7	555.123	85.692	7
BD40KT5	194.337	40.256	5
BD10KT10	97.824	10.731	10

Para cada conjunto de datos se realizó preprocesamiento y transformación de datos con el fin de eliminar los productos repetidos en cada transacción y posteriormente se cargaron las tablas objeto de un modelo simple (i.e. una tabla con esquema *Tid, Item*) a la estructura de datos *DataSet* descrito en el capítulo 7 en la sección de implementación.

Se evaluó el rendimiento de los algoritmos Apriori, FP-*Growth y Equipasso*, comparando los tiempos de respuesta para diferentes soportes mínimos. Los resultados de la evaluación del tiempo de ejecución de estos algoritmos, aplicados a los conjuntos de datos BD85KT7, BD40KT5 y BD10KT10, se pueden observar en las figuras 9.1, 9.2 y 9.3 respectivamente.

En general, observando el comportamiento de los algoritmos *FP-Growth y Equipasso* con los diferentes conjuntos de datos, se puede decir que su rendimiento es similar, contrario al tiempo de ejecución de Apriori, que se ve afectado significativamente a medida que se disminuye el soporte.

Tabla 9.2: Tiempos de ejecución tabla BD85KT7

BD85KT7			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
4.15	750	166	85
4.75	362	162	82
5.35	365	164	83
5.95	365	162	83
6.55	120	159	80

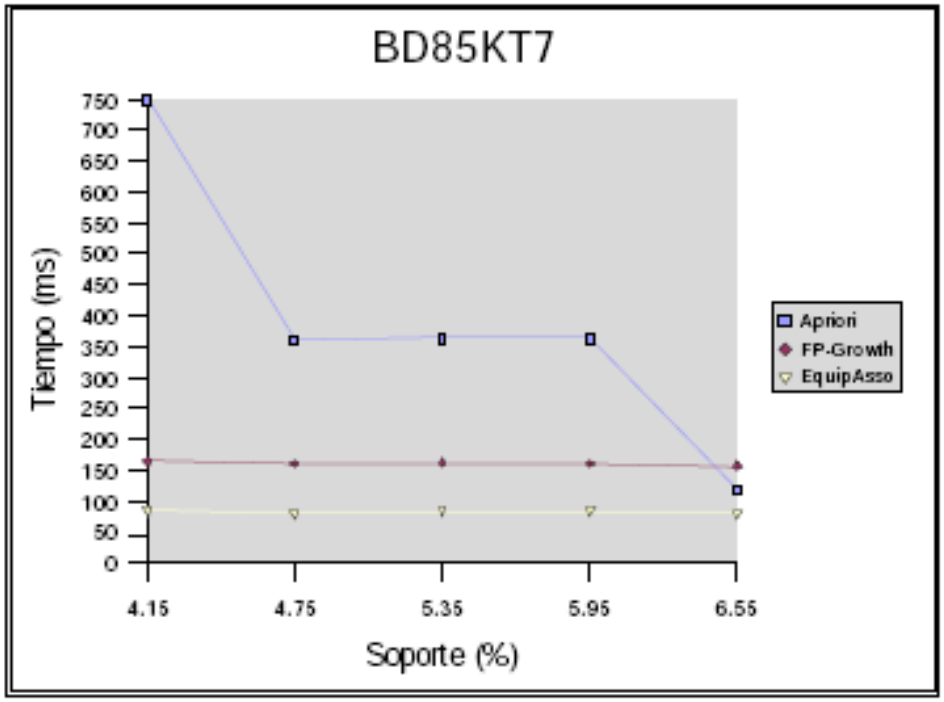

Figura 9.1: Rendimiento BD85KT7

Tabla 9.3: Tiempos de ejecución tabla BD40KT5

BD40KT5			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
1.90	268	66	29
2.00	265	64	29
2.10	132	63	27
2.20	45	61	27
2.30	44	61	27

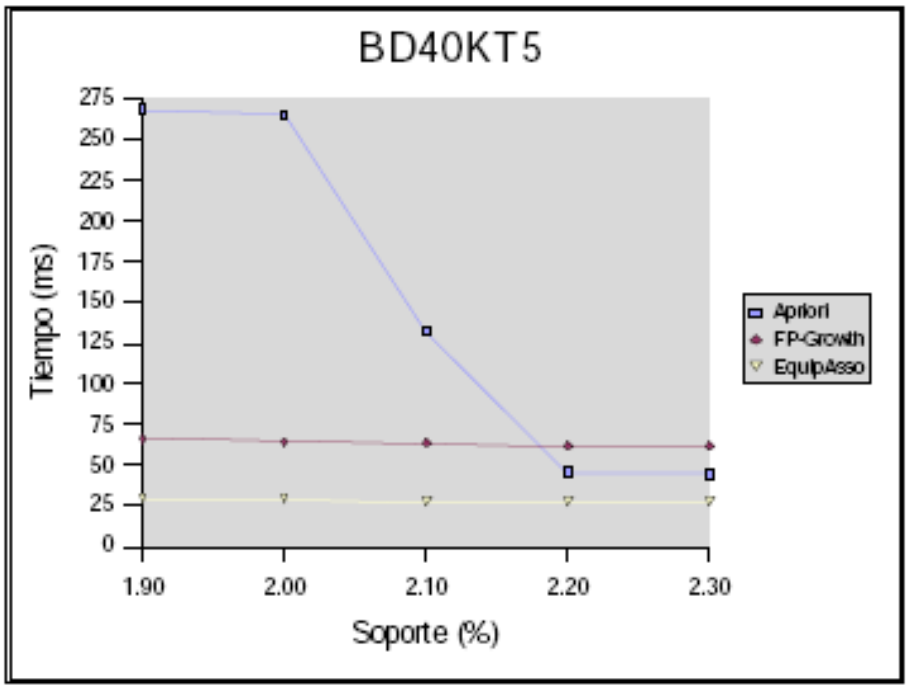

Figura 9.2: Rendimiento BD40KT5

Tabla 9.4: Tiempos de ejecución tabla BD10KT10

BD10KT10			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
3.00	525	28	15
4.00	182	26	14
5.00	105	25	13
6.00	53	24	13
7.00	52	24	13

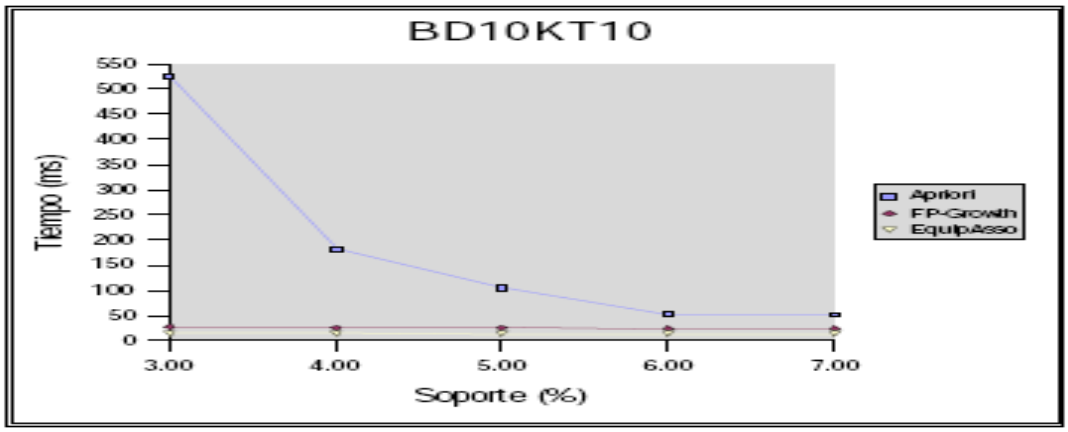

Figura 9.3: Rendimiento BD10KT10

Analizando el tiempo de ejecución de únicamente los algoritmos *FP-Growth* y *EquipAsso* (figuras 9.4, 9.5 y 9.6) para los conjuntos de datos BD85KT7, BD40KT5 y BD10KT10. Con soportes más bajos, el comportamiento de estos algoritmos sigue siendo similar.

Tabla 9.5: Tiempos de ejecución tabla BD85KT7

BD85KT7			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
1.00	-	5130	1225
1.50	-	730	270
2.00	-	212	205
2.50	-	202	202
3.00	-	185	187

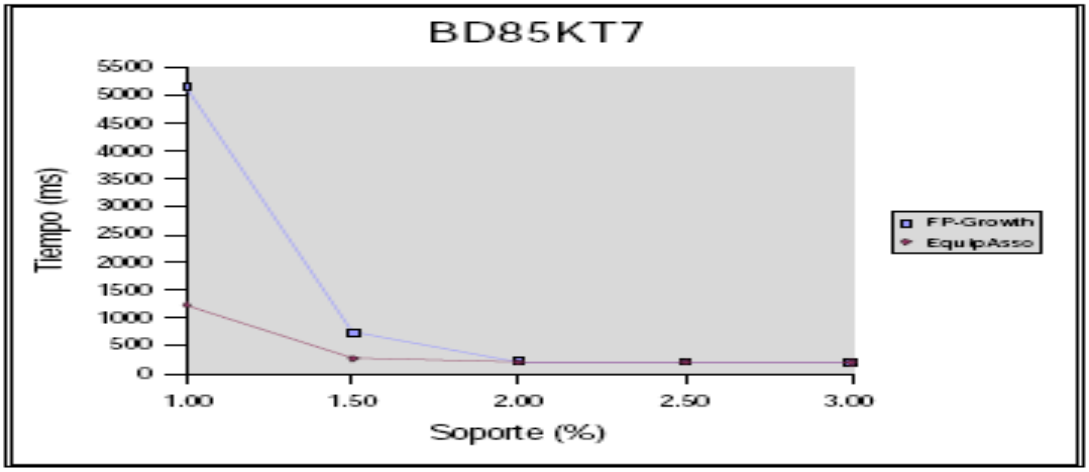

Figura 9.4: Rendimiento BD85KT7

Tabla 9.6: Tiempos de ejecución tabla BD40KT5

BD40KT5			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
0.10	-	965	741
0.20	-	425	290
0.30	-	240	168
0.40	-	156	121
0.50	-	124	105

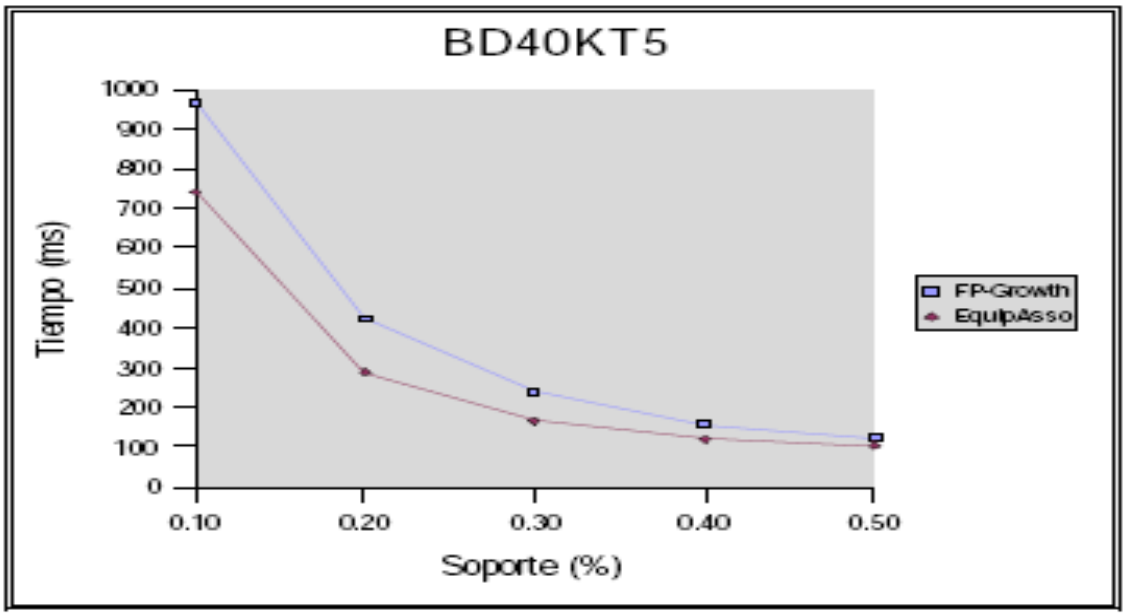

Figura 9.5: Rendimiento BD40KT5

Tabla 9.7: Tiempos de ejecución tabla BD10KT10

BD10KT10			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
0.50	-	257	181
0.75	-	133	93
1.00	-	78	60
1.25	-	61	50
1.50	-	46	42

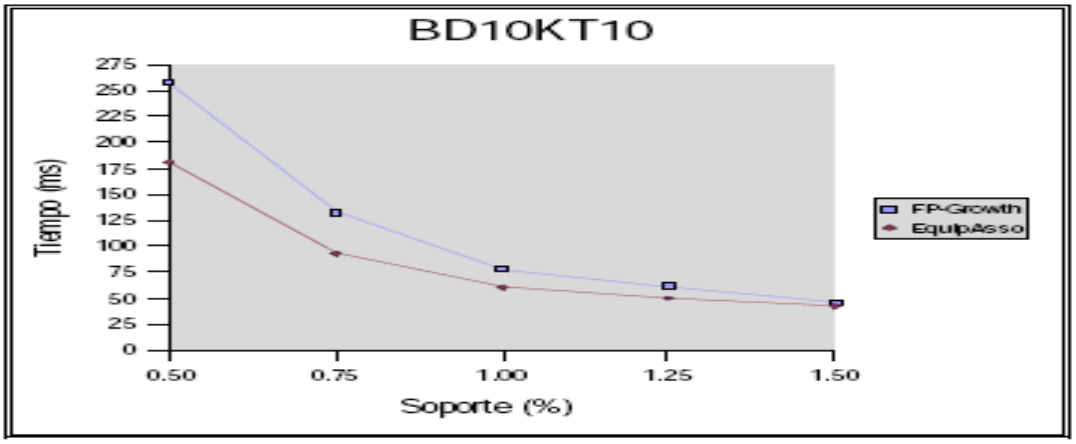

Figura 9.6: Rendimiento BD10KT10

9.2 Rendimiento Algoritmos de Clasificación

Para realizar las pruebas de clasificación se seleccionó la base de datos histórica de los estudiantes de la Universidad de Nariño, compuesta por información personal y académica de 20328 estudiantes. Este conjunto de datos fue construido por miembros del Grupo de Investigación GRiAS de la Universidad de Nariño. Mayores detalles de este conjunto pueden ser consultados en [27].

Después de los procesos de limpieza y transformación de datos se trabajó con una única tabla o vista minable que contó con un total de 26 atributos que se mencionan en la tabla 8.8. listando sus posibles valores.

Tabla 9.8. Campos en el conjunto UDENAR

Atributo	Valores
edad	A, B, C, D
edading	A, B, C, D
fecha_ing	A, B, C, D, E
ingresos	A, B, C, D, E
val_matricula	A, B, C, D, E
nestrato	0, 1, 2, 3, 4, 5, 6
naturaleza	Oficial, Privado

Atributo	Valores
jornada	Completa, Mañana, Tarde, Noche
calendario	A, B, Flexible
sexo	M, F
ponderado	A, B, C, D
ocu_madre	1, 2, 3, 4, 5
mas_una_a_cargo	S, N
vive_con_familia	S, N
tiene_hermanos_u	S, N
especial	S, N
padre_vive	S, N
estado_civil	0, 1, 2, 3, 4
madre_vive	S, N
tipo_residencia	0, 1, 2, 3
zona_nac	Norte, Norten, Occidente, Occidenten, Oriente, Orienten, Sur, Surn
semestred	1, 2, 3, 4, 5
cod_facultad2	1, 2, 3, 4, 5, 6, 7, 8, 18, 22, 32
claseal	1, 2, 3
claserend	1, 2, 3, 4, 5
clasepromedio	A, B, C, D, E

Como se ve en la tabla varios atributos fueron discretizados. Los respectivos valores y detalle de este proceso se explican en las tablas 9.9. a 9.24 respectivamente.

Tabla 9.9. Discretización atributo Edad

EDAD	DISCRETIZACIÓN	CANTIDAD
Menores e iguales a 18	A	827
Mayores de 18 y menores que 22	B	3634
Mayores e iguales que 22 y Menores de 26	C	4856
Mayores e iguales que 26	D	11012

Tabla 9.10. Discretización atributo Edad_ingreso

EDAD DE INGRESO	DISCRETIZACIÓN	CANTIDAD
Menores e iguales a 18	A	9054
Mayores de 18 y menores que 22	B	7370
Mayores e iguales que 22 y Menores de 26	C	2594
Mayores e iguales que 26	D	1311

Tabla 9.11. Discretización atributo Fecha ingreso

FECHA DE INGRESO	DISCRETIZACIÓN	CANTIDAD
Antes de 1990	A	1022
Después o igual a 1990 y Menores de 1995	B	4852
Después o igual a 1995 y Menores de 2000	C	5978
Después o igual al 2000 y Menores de 2003	D	5046
Mayores o iguales de 2003	E	7621

Tabla 9.12. Discretización atributo Ingresos

INGRESOS	DISCRETIZACIÓN	CANTIDAD
Menores que 3	A	6101
Mayores o iguales a 3 y menores que 6	B	6350
Mayores que 6 y menores que 9	C	3325
Mayores que 9 y menores que 12	D	1828
Mayores de 12	E	2725

Tabla 9.13. Discretización atributo Valor Matricula

VALOR DE MATRICULA	DISCRETIZACIÓN	CANTIDAD
Menores de 50000	A	3529
Mayores o iguales que 50000 y menores que 100000	B	5348
Mayores o iguales que 100000 y menores que 200000	C	6762
Mayores o iguales que 200000 y menores que 500000	D	3636
Mayores o iguales que 500000	E	1054

Tabla 9.14. Discretización atributo Naturaleza

NATURALEZA	DISCRETIZACIÓN	CANTIDAD
Oficial	Oficial	13571
No Oficial	Privado	6757

Tabla 9.15. Discretización atributo Ocupación Madre

OCUPACION MADRE	DISCRETIZACIÓN	CANTIDAD
Ama de Casa	1	9049
Profesional	2	572
Empleada	3	313
Independiente	4	200
Otro	5	10194

Tabla 9.16. Discretización atributo Ponderado

PONDERADO	DISCRETIZACIÓN	CANTIDAD
Menores que 30	A	43
Mayores o iguales que 30 y menores que 50	B	2689
Mayores o iguales que 50 y menores que 70	C	16601
Mayores o iguales que 70 y menores o iguales que100	D	1623

Tabla 9.17. Discretización atributo Estado Civil

ESTADO CIVIL	DISCRETIZACIÓN	CANTIDAD
Soltero	0	20651
Casado	1	5
Separado	2	251
Divorciado	3	24
Unión Libre	4	25

Tabla 9.18. Discretización atributo Tipo Residencia

TIPO_RESIDENCIA	DISCRETIZACIÓN	CANTIDAD
No propia	0	5181
Propia	1	14995
Propia pagando cuotas	2	308
Arrendando o Anticresis	3	472

Tabla 9.19. Discretización atributo Semestre

SEMESTE	DISCRETIZACIÓN	CANTIDAD
Semestre 1 hasta semestre 4	1	8859
Semestre 5 hasta semestre 8	2	3539
Semestre 9 hasta semestre 10	3	1390
Egresados	4	1383
Graduados	5	5157

Tabla 9.20. Discretización atributo Facultad

FACULTAD	DISCRETIZACIÓN	CANTIDAD
Artes	1	2097
Ciencias Agrícolas	2	1532
Derecho	3	1208
Ciencias Económicas y Administrativas	4	2424
Ingeniería	5	2549
Ciencias Pecuarias	6	1715
Ciencias Naturales y Matemáticas	7	3701
Ciencias Humanas	8	4009
Educación	18	793
Ingeniería Agroindustrial	22	537
Ciencias de la Salud	32	390

Tabla 9.21. Discretización atributo Zona_nac

ZONA_NAC	DISCRETIZACIÓN	CANTIDAD
Norte del departamento de Nariño	Norten	66
Sur del departamento de Nariño	Surn	16931
Oriente del departamento de Nariño	Orienten	1512
Occidente del departamento de Nariño	Occidentenn	595
Norte de Colombia	Norte	62
Sur de Colombia	Sur	410
Oriente de Colombia	Oriente	23
Occidente de Colombia	Occidente	729

Tabla 9.22. Discretización atributo Clase_al

CLASE_AL	DISCRETIZACIÓN	CANTIDAD
Normal	1	18641
Reingreso	2	846
Retirado	3	1469

Tabla 9.23. Discretización atributo Clase_rend

CLASE_REND	DISCRETIZACIÓN	CANTIDAD
No ha perdido materias	1	7852
Ha perdido 1 materia	2	3339
Ha perdido 2 materia	3	2576
Ha perdido 3 materia	4	2047
Ha perdido más de 3 materias	5	5142

Tabla 9.24. Discretización atributo Clase_promedio

CLASE_PROMEDIO	DISCRETIZACIÓN	CANTIDAD
Menor a 2	A	2391
Mayor o igual a 2 hasta 3	B	2934
Mayor o igual a 3 hasta 3.5	C	5166
Mayor o igual a 3.5 hasta 4.0	D	6850
Mayor o igual a 4.0 hasta 5.0	E	3615

Se hicieron dos tipos de pruebas sobre el conjunto UDENAR evaluando los tiempos de ejecución de los algoritmos de clasificación implementados en TariyKDD (C4.5 y Mate). Las primeras pruebas variaban el número de atributos con el que se trabajaba y en las segundas se varió el número de registros con los que se construyó el modelo.

En la primera prueba se construyeron modelos seleccionando en primera instancia 22 atributos y el total de registros (20328), la clase objetivo con la que se evaluó el modelo fue *clase_rendimiento*. Posteriormente se eliminó uno a uno los atributos hasta un límite de 4 columnas. Los resultados de esta prueba se pueden ver en la Figura 9.7.

Otra prueba consistió medir la importancia de un atributo en el conjunto, observando cómo variaba la ganancia de información al quitar y poner uno de los 22 atributos que conforman la muestra para evaluar la ausencia o presencia de un determinado atributo en la construcción de un modelo. Los resultados se pueden observar en la Figura 9.8.

La última prueba sobre los atributos de la tabla consistió en eliminar en cada prueba que generaba mayor ganancia de información, en cada experimento para determinar los atributos más relevantes. Los resultados de esta prueba se resumen en la Figura 9.9.

Tabla 9.25. Tiempos de ejecución en el conjunto UDENAR al reducir atributos.

Número de Atributos	Tiempo De Ejecución (ms)	
	C4.5	Mate
22	1.495,67	7.157,33
21	1.670,33	6.931,33
20	1.701,67	7.700,33
19	1.441,33	7.143,33
18	1.445,67	6.461,67
17	1.327,33	6.990,00
16	1.195,67	6.211,67
15	1.139,00	5.969,33
14	1.083,33	5.900,67
13	1.080,67	5.986,67
12	916,00	5.827,67
11	854,00	5.044,67
10	696,33	4.530,33
9	609,00	4.786,33
8	494,33	4.931,67
7	429,67	3.922,00
6	353,00	4.091,33
5	237,33	3.713,00
4	157,67	2.338,67

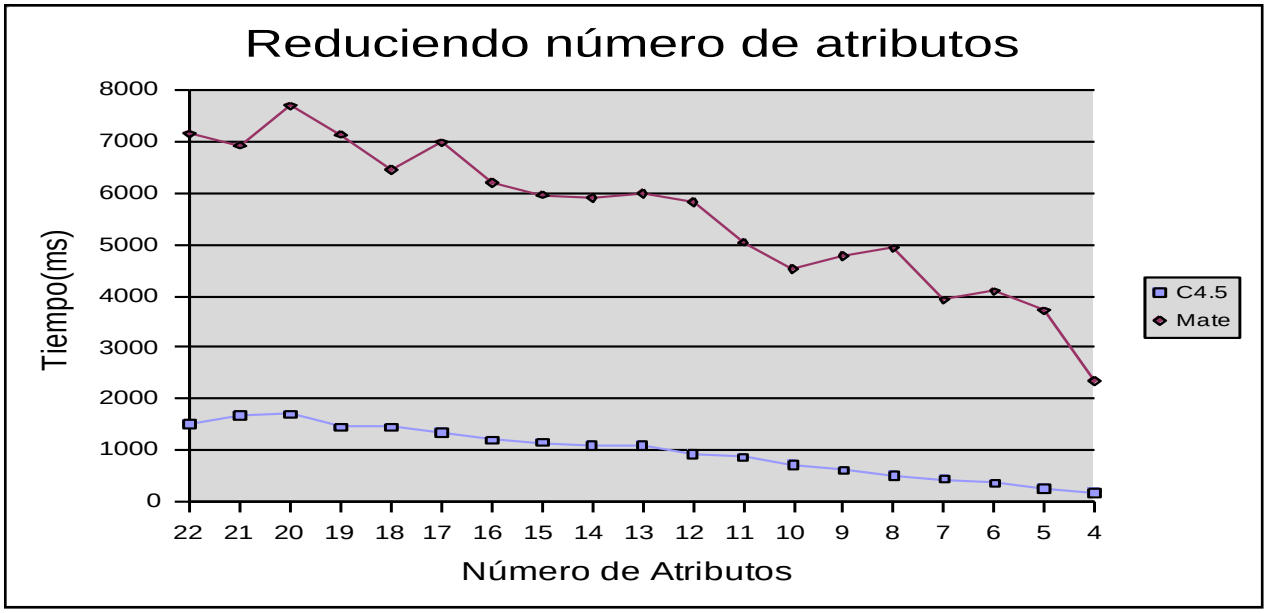

Figura 9.7. Rendimiento de algoritmos en el conjunto UDENAR al reducir atributos.

Tabla 9.26. Tiempos de ejecución en el conjunto UDENAR al quitar y poner un atributo.

Atributo	TIEMPO DE EJECUCIÓN (ms)	
	C4.5	MATE
Ninguno	1.495,00	6.276,00
cod_fac2	1.643,00	6.918,00
semd	1.248,33	5.610,67
Zona_nac	1.073,33	4.988,67
Tip_resid	1.155,33	5.671,00
Madre_vive	1.132,33	6.177,33
Est_civil	1.152,33	5.863,67
Padre_vive	1.164,00	6.159,33
Especial	1.182,33	5.832,67
Tiene_hermanos	1.192,67	6.288,67
Vive_c_famil	1.129,33	6.202,33
Mas_u_a_c	1.127,00	5.715,00
Ocu_madre	1.107,00	5.417,33
Ponderado	1.134,33	5.438,33
sexo	1.161,67	5.689,67
Calendario	1.159,33	5.644,67
jornada	1.110,00	5.921,33
Naturaleza	1.135,33	5.952,67
estrato	1.138,67	6.019,33
Vlr_matricula	1.144,33	5.960,67
Ingresos	1.158,33	5.706,33
Edad_ingreso	1.178,00	6.076,33

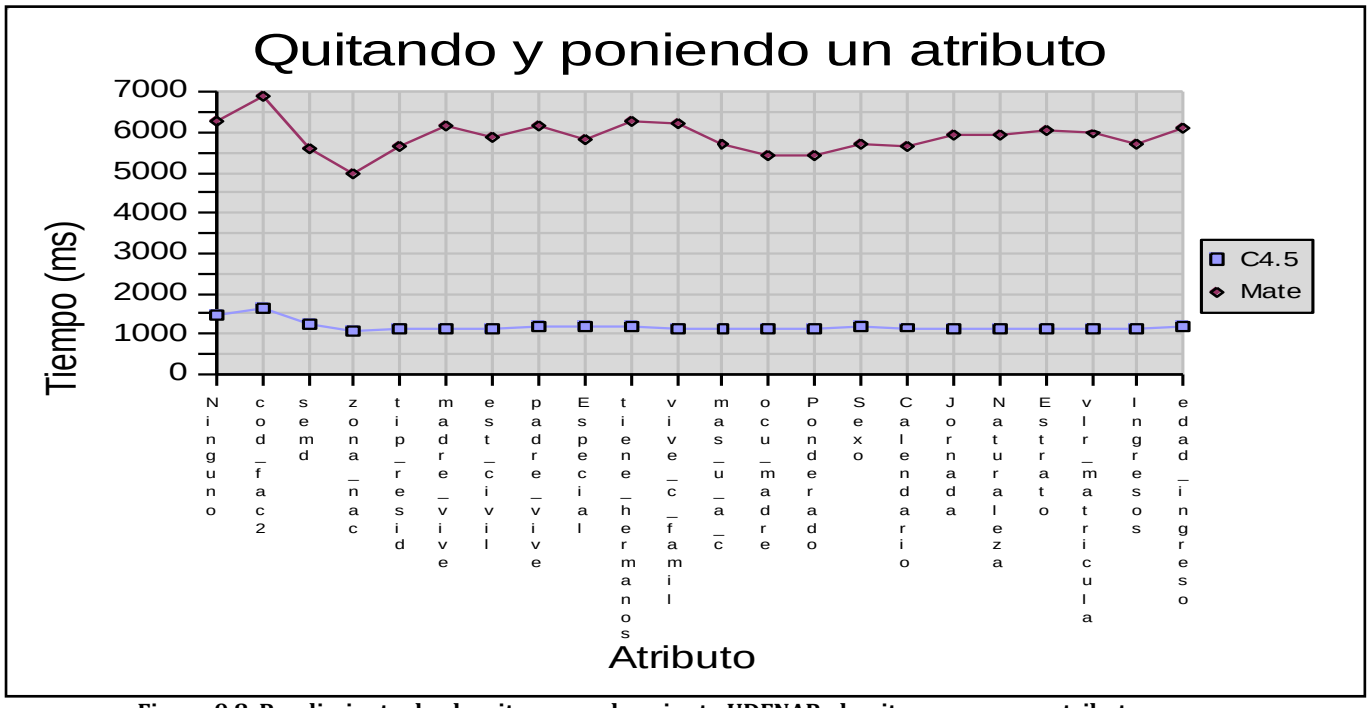

Figura 9.8. Rendimiento de algoritmos en el conjunto UDENAR al quitar y poner un atributo.

Tabla 9.27. Tiempos de ejecución en el conjunto UDENAR al eliminar el atributo ganador.

Atributos	TIEMPO DE EJECUCIÓN (ms)		Número de Reglas
	C4.5	MATE	
Ninguno	1.495,00	6.276,00	429
cod_fac2	1.697,00	9.040,00	398
semd	1.688,33	7.541,67	417
Sexo	1.515,67	7.851,67	358
Edad_ingreso	1.448,67	8.875,67	410
Ponderado	1.337,33	9.555,33	389
Vlr_matricula	1.356,67	13.634,33	381
Zona_nac	1.320,67	13.965,67	384
Ingresos	1.372,00	13.327,00	293
estrato	1.350,33	17.925,67	219
Est_civil	1.121.33	13.524,67	209
Ocu_madre	1.026,67	11.557,00	158
Vive_c_famil	867,33	9.587,67	121
jornada	722,67	6.181,00	59
Tip_resid	598,67	4.552,67	34
Tiene_hermanos	432,00	3.559,67	26
Calendario	331,00	2.186,00	16
Naturaleza	227,00	1.272,00	8
Mas_u_a_c	137,33	700,00	5
Padre_vive	79,33	366,33	3
Madre_vive	31,33	153,33	2

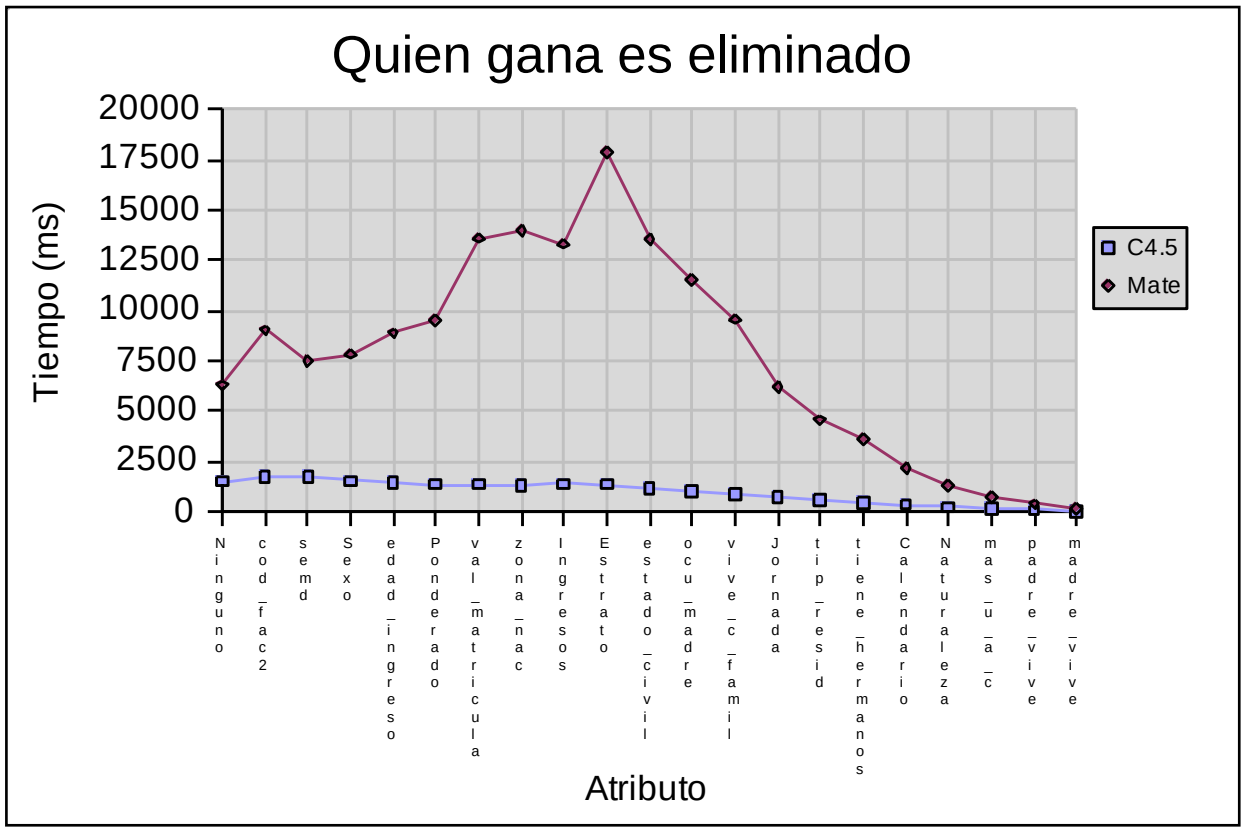

Figura 9.9. Rendimiento de algoritmos en el conjunto UDENAR al eliminar el atributo ganador.

A partir de la última prueba se construyo una vista minable con los 5 atributos más relevantes (*cod_facultad, semestred, sexo, edad_ingreso, ponderado*) más la clase (*clase_rendimiento*) y se tomaron muestras desde 20000 hasta 2000 registros disminuyendo el tamaño del conjunto en 2000 registros para cada medición. Un resumen de la prueba se puede ver a continuación en la Figura 8.10.

Tabla 9.28. Tiempos de ejecución en el conjunto UDENAR al reducir el número de registros.

Registros	TIEMPO DE EJECUCIÓN (ms)		Número de Reglas
	C4.5	MATE	
20000	258,33	3.595,67	226
18000	249,00	3.322,67	212
16000	222,33	2.880,67	216
14000	192,67	2.804,00	224
12000	160,33	2.338,33	228
10000	130,00	2.358,33	225
8000	108,33	1.730,67	213
6000	83,00	1.323,67	216
4000	60,00	957,00	200
2000	38,33	523,67	204

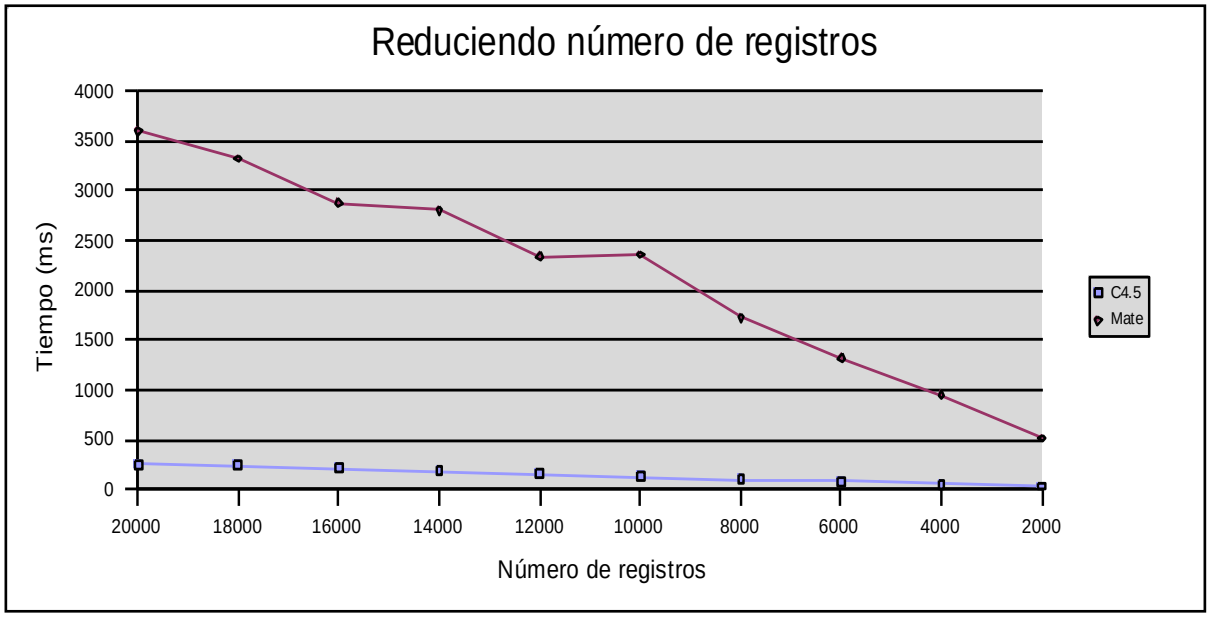

Figura 9.10. Rendimiento de algoritmos en el conjunto UDENAR al reducir el número de registros.

En la figura 9.7 podemos observar como al reducir el número de atributos bajan los tiempos de ejecución de los algoritmos siendo más notable para el algoritmo Mate.

Se puede observar también que la ausencia de ciertos atributos hace aumentar los tiempos de ejecución en ambos algoritmos determinando la relevancia de dicho atributo dentro del conjunto.

Este comportamiento es más visible en la figura 9.8 donde atributos como cod_facultad y zona_nac repercuten positiva o negativamente en la ejecución de los algoritmos.

Analizando en la figura 9.9 se encuentra un comportamiento especial que afecta en mayor medida al algoritmo Mate, a medida que se eliminan atributos relevantes esto afecta a los tiempos de ejecución y resulta más difícil para los algoritmos construir modelos que definan a la clase con los atributos con los que dispone.

En la figura 9.10 se ve claramente como el número de registros afecta en mayor medida al algoritmo Mate y conforme se disminuye el tamaño del conjunto, se mejoran los tiempos de ejecución.

De todas maneras y en forma general se ve claramente un mejor comportamiento del algoritmo C4.5 con respecto a Mate.

9.3 Rendimiento del Formato de Compresión Tariy Vs Formato Arff

Un Análisis del formato para compresión de datos descrito en el capítulo 7 del presente trabajo, en la sección Arquitectura *Tariy*, con respecto al formato ARFF de la herramienta de minería de Datos WEKA [51, 52] se muestra en el siguiente tabla, donde se registra el tamaño en disco de cada formato al almacenar conjuntos de datos con diferente número de transacciones y atributos.

Tabla 9.29: Análisis de formatos de almacenamiento

Archivo ARFF	Núm. Instancias	Núm. Atributos	Tam. ARFF (KB)	Tam. Tariy (KB)
mushroom	8124	23	726.30	133.81
titanic	2201	4	64.60	0.17
tictactoe	958	10	26.50	14.00
soybean	683	36	194.10	55.46
vote	435	17	32.20	8.87
contact-lenses	24	5	1.10	0.23
weather.nominal	14	5	0.57	0.16

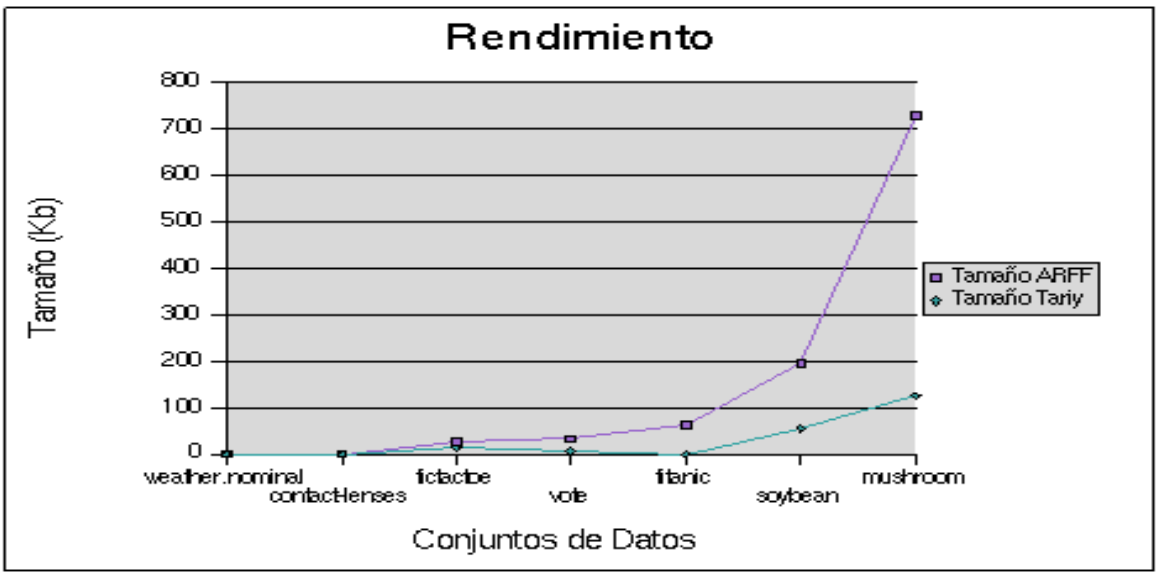

Figura 9.11: Rendimiento formatos de almacenamiento

10. CONCLUSIONES

En este proyecto se diseñó e implementó una herramienta débilmente acoplada un SGBD que da soporte a las etapas de conexión, preprocesamiento, minería y visualización del proceso KDD. Igualmente se incluyeron en el estudio nuevos algoritmos de asociación y clasificación propuestos por Timaran [42, 43, 44, 45].

Para el desarrollo del proyecto se hizo un Análisis de varias herramientas de software libre que abordan tareas similares a las que se pretendió en este trabajo. Se identificaron las limitaciones y virtudes de estas aplicaciones y se diseñó una metodología para el desarrollo de una herramienta que cubriera las falencias encontradas.

Teniendo en cuenta la intención de liberar la herramienta se establecieron patrones de diseño que hicieran posible el acoplamiento de nuevas funcionalidades a cada uno de los módulos que lo componen, facilitando así la inclusión futura de nuevas características y la mejora continua de la

aplicación, así como la implementación de un sistema de control de versiones que facilite el trabajo colaborativo y permita el acceso a la herramienta a través de Internet.

La construcción de TariyKDD comprendió el desarrollo de cuatro módulos que cubrieron, el proceso de conexión a datos, tanto a archivos planos como a bases de datos relacionales, la etapa de preprocesamiento, donde se implementaron 9 filtros para la selección, transformación y preparación de los datos, el proceso de minería que comprendió tareas de asociación y clasificación, implementando 5 algoritmos, *Apriori, FPGrowth y EquipAsso* para asociación y *C4.5 y Mate* para clasificación y el proceso de visualización de resultados, utilizando tablas y árboles para generar informes de los resultados y reglas obtenidas. Igualmente se incluyó un módulo para la predicción de nuevos registros a partir de los modelos construidos con los algoritmos de clasificación. Estos desarrollos fueron logrados usando en su totalidad herramientas de código abierto y software libre.

Se desarrolló un modelo de datos que facilitó la aplicación de algoritmos de asociación sobre bases de datos enmarcadas en el concepto de canasta de mercado donde la longitud de cada transacción es variable.

Se realizaron pruebas para evaluar la validez de los algoritmos implementados. Para el plan de pruebas de la tarea de asociación, se utilizaron conjuntos de datos reales de las ventas de un supermercado de la Caja de Compensación familiar de Nariño compuesta por más de 85000 transacciones. Para Clasificación, se trabajó con la base de datos histórica de los estudiantes de la Universidad de Nariño, compuesta por información personal y académica de cerca de 20400 estudiantes.

Analizando las pruebas obtenidas para la tarea de clasificación podemos concluir que es recomendable el uso del algoritmo *C4.5* para conjuntos de datos extensos donde el número de atributos a analizar sea grande. Solo bajo conjuntos de datos pequeños y con reducido número de atributos resulta viable aplicar el algoritmo *Mate* bajo una arquitectura débilmente acoplada.

Analizando las pruebas obtenidas para Asociación se concluye que los algoritmos *FPGrowth* y *EquipAsso* obtienen muy buenos tiempos de respuesta al ser aplicados en conjuntos grandes pero a medida que se disminuye el criterio de soporte se ve un mejor comportamiento por parte del algoritmo *EquipAsso*. Definitivamente no es viable aplicar el algoritmo *Apriori* bajo conjuntos grandes, limitando su uso a muestras pequeñas de datos.

Se cuenta con una versión estable de TariyKDD con la capacidad de extraer reglas asociación y clasificación bajo una arquitectura débilmente acoplada con un SGBD y con la capacidad de mostrar dichas reglas de forma visual e intuible para el analista, esta herramienta fue desarrollada bajo los lineamientos del software libre.

Una vez que se han descrito los resultados más relevantes que se han obtenido durante la realización de este proyecto, se sugiere una serie de recomendaciones como punto de partida para futuros trabajos:

- Realizar mayores pruebas de rendimiento de esta arquitectura e implementar otras técnicas de descubrimiento de conocimiento, como la aplicación de Redes Neuronales y la implementación de nuevas tareas de Asociación y Clasificación.

- Implementar otras tareas y algoritmos de minería de datos, como clustering y reglas de

asociación secuenciales.

- Implementar nuevos filtros e interfaces de visualización que permitan el mejoramiento continuo de TariyKDD.

- Acoplar gráficos estadísticos y a los conjuntos de datos cargados para obtener una información inicial de sus características e implementar nuevas técnicas de visualización inteligente de la información para un análisis holístico del estudio.

- Diseñar una interfaz que permita la ejecución de primitivas SQL de minería de datos fuertemente acopladas al sistema gestor de base de datos.

- Liberar, compartir y difundir una versión estable de TariyKDD con la capacidad de descubrir conocimiento en bases de datos.

Finalmente este trabajo permitió aplicar los conocimientos adquiridos en el Máster Oficial en Sistemas Inteligentes y en especial los de las materias *Mineria de Datos y Visualización Inteligente de la Información*.

11. BIBLIOGRAFIA

[1] R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In The Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, 1996.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In VLDB Conference, Santiago, Chile, 1994.

[3] R. Agrawal , T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. ACM SIGMOD, 1993.

[4] R. Agrawal , M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The quest data mining system. In 2Ao Conference KDD y Data Mining, Portland, Oregon, 1996.

[5] R. Brachman and T. Anand. The Process of Knowledge Discovery in Databases: A first Sketch, Workshop on Knowledge Discovery in Databases. 1994.

[6] S. Chaudhuri. Data mining and database systems: Where is the intersection? In Bulletin of the Technical Committee on Data Engineering, volume 21, Marzo 1998.

[7] M. Chen, J. Han, and P. Yu. Data mining: An overview from database perspective. In IEEE Transactions on Knowledgeand Data Engineering, 1996.

[8] Quadrillion Corp. Q-yield. http://www.quadrillion.com/qyield.shtm, 2001.

[9] C[11] IBM Corporation. Intelligent miner. http://www4.ibm.com/software/data/iminer, 2001.

[10] J Demsar and B Zupan. Orange: From experimental machine learning to interactive data mining. Technical report, Faculty of Computer and Information Science, University of Liubliana, Slovenia,
http://www.ailab.si/orange/wp/orange.pdf, 2004.

[11] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview, in advances in knowledge discovery and data mining. In AAAI Pres / The MIT Press, 1996.

[12] Faculty of Computer and Slovenia Information Science, University of Liubliana. Orange, fruitful and fun. http://www.ailab.si/orange, 2006.

[13] Faculty of Computer and Slovenia Information Science, University of Liubliana. Orange's interface to mysql.
http://www.ailab.si/orange/doc/modules/orngMySQL.htm, 2006.

[14] Goverment of Hong Kong. Innovation and technology fund. http://www.itf.gov.hk, 2006.

[15] M. Goebel and L. Gruenwald. A survey of data mining and knowledge discovery software tools. In SIGKDD Explorations, volume 1 of 1, June 1999.

[16] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, K. Koperski, G. Liu, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. Zaiane, S. Zhang, and H. Zhu. Dbminer: A system for data mining in relational databases and data warehouses. In CASCON: Meeting of Minds.

[17] J. Han, Y. Fu, W. Wang, J. Chiang, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. Zaiane. Dbminer: A system for mining knowledge in large relational databases. In The second International Conference on Knowledge Discovery & Data Mining.

[18] J. Han, Y. Fu, and S. Tang. Advances of the dblearn system for knowledge discovery in large databases. In International Joint Conference on Artificial Intelligence IJCAI, Montreal, Canada, 1995.

[19] J. Han and M. Kamber. Data Mining Concepts and Techniques. Morgan Kaufmann Publishers, 2001.

[20] J. Han, Y. Fu, W. Wang, J. Chiang, O. Zaiane, and K. Koperski. DBMiner: Interactive Mining of Multiple-Level Knowledge in Relational Databases. ACM SIGMOD, Montreal, Canada, 1996.

[21] J. Han and J. Pei. Mining frequent patterns by pattern-growth: Methodology and implications. In SIGKDD Explorations, volume 2:14-20, 2000.

[22] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In ACM SIGMOD, Dallas, TX, 2000.

[23] T. Imielnski and H. Mannila. A database perspective on knowledge discovery. In Communications of the ACM.

[24] E-Business Technology Institute. E-business technology institute, the university of hong kong. http://www.eti.hku.hk, 2005.

[25] Isoft S.A. Alice. http://www.alice-soft.com/html/prod alice.htm, 2001.

[26] Kdnuggets. http://www.kdnuggets.com/software, 2001.

[27] Lombana C, Narvaez R, Viteri G, Dulce E. Detección de patrones de bajo rendimiento y/o deserción de los estudiantes de la Universidad de Nariño con técnicas de minería de datos, VIII Convocatoria Alberto Quijano Guerrero, Sistema de Investigaciones Universidad de Nariño, 2006.

[28] C. Matheus, P. Chang, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. In IEEE Transactions on Knowledge and Data Engineering, volume 5, 1993.

[29] Mierswa, M Wurst, R Klinkenberg, M Scholz, and T Euler. Yale: Rapid prototyping for complex data mining tasks. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), 2006.

[30] NASA National Aeronautics and Space Administration. Tropical cyclone windspeed indicator. http://pm-esip.nsstc.nasa.gov/cyclone/.

[31] G. Piatetsky-Shapiro, R. Brachman, and T. Khabaza. An Overview of Issues in Developing Industrial Data Mining and Knowledge Discovery Applications. 1996.

[32] Inselberg A. Multidimensional Dectective.

[33] J.R. Quinlan. C4.5: Programs for Machine Leraning. Morgan Kaufmann Publishers, 1993.

[34] R. Rakotomalala. Tanagra project. http://chirouble.univ-lyon2.fr/ricco/tanagra/en/tanagra.html, 2006.

[35] R. Rakotomalala. Tanagra. In TANAGRA: a free software for research and academic purposes, volume 2, pages 697–702. EGC'2005, 2005.

[36] M. Inestroza and R. Therón. Visualización de Datos Multidimensionales: Una nueva propuesta

[37] RuleQuest Research Inc. C5.0. http://www.rulequest.com, 2001.

[38] Sándor Kromesch and Sándor Juhász , High Dimensional Data Visualization

[39] SPSS. Clementine. http://www.spss.com/clementine, 2001.

[40] Project Processing, http://processing.org/

[41] R. Timarán. Arquitecturas de integración del proceso de descubrimiento de conocimiento con sistemas de gestión de bases de datos: un estado del arte, en revista ingeniería y competitividad. Revista de Ingeniería y Competitividad, Universidad del Valle, 3(2), Diciembre 2001.

[42] R. Timarán. Descubrimiento de conocimiento en bases de datos: Una visión general. In Primer Congreso Nacional de Investigación y Tecnología en Ingeniería de Sistemas, Universidad del Quindío Armenia, Octubre 2002.

[43] R. Timarán and M. Millán. Equipasso: An algorithm based on new relational algebraic operators for association rules discovery. In Fourth IASTED International Conference on Computational Intelligence, Calgary, Alberta, Canada, July 2005.

[44] R. Timarán and M. Millán. Equipasso: un algoritmo para el descubrimiento de reglas de asociación basado en operadores algebraicos. In 4Aa Conferencia Iberoamericana en Sistemas, Cibernética e Informática CICI 2005, Orlando, Florida, EE.UU., Julio 2005.

[45] R. Timarán, M. Millán, and F. Machuca. New algebraic operators and sql primitives for mining association rules. In IASTED International Conference Neural Networks and Computational Intelligence, Cancun, Mexico, 2003.

[46] R. Timarán. Nuevas Primitivas SQL para el Descubrimiento de Conocimiento en Arquitecturas Fuertemente Acopladas con un Sistema Gestor de Bases de Datos. PhD thesis, Universidad del Valle, 2005.

[47] Université Lumiere Lyon 2. Eric equipe de recherche en ingénierie des conaissances. http://chirouble.univ-lyon2.fr, 2006.

[48] Artificial Inteligence Unit of the University of Dortmund. Artificial inteligence unit of the university of dortmund. http://www-ai.cs.uni-dortmund.de, 2006.

[49] Information Technology The University of Alabama in Huntsville and Systems Center. Adam 4.0.2 components.
http://datamining.itsc.uah.edu/adam/documentation.html.

[50] Information Technology The University of Alabama in Huntsville and Systems Center. Algorithm development and mining system.
http://datamining.itsc.uah.edu/adam/index.html.

[51] Waikato ML Group. Attribute-relation file format (arff).
http://www.cs.waikato.ac.nz/ ml/weka/arff.html.

[52] Waikato ML Group. Collections of datasets.
http://www.cs.waikato.ac.nz/ml/weka/indexd atasets.html.

[53] Waikato ML Group. The waikato environment for knowledge analisys.
http://www.cs.waikato.ac.nz/ml/weka.

[54] I. Waitten and F. Eibe. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. 2001.