INTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MÉXICO

Algoritmo ID3 en un Ambiente Distribuido

TESIS PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A

IVAN OLMOS PINEDA

Asesor:	Dra. Maricela Quintana López	
Comité de Tesis:	Dr. Jesús Antonio Sánchez Velásquez	
	Dr. Edgar E. Vallejo Clemente	
Jurado:	Dr. Edgar E. Vallejo Clemente	Presidente
	Dr. Jesús Antonio Sánchez Velásquez	Secretario
	Dra. Maricela Quintana López	Vocal

Atizapán de Zaragoza, Edo. De México, Agosto 31 del 2001

DEDICATORIA

A mis padres y hermana, por el gran apoyo que me han brindado para seguir superándome.

A todos mis familiares, gracias por su confianza.

RECONOCIMIENTO

Agradezco al Fondo de Apoyo a los Programas de Posgrado del I. T. E. S. M. (F. A. P. P. I.) por brindarme la oportunidad de estudiar la Maestría en MCC en el Tec de Monterrey Estado de México.

RESUMEN

El presente documento presenta un algoritmo basado en el método ID3 tradicional capaz de operar en una Base de Datos Distribuida con fragmentación horizontal primaria. El algoritmo propuesto pretende obtener un árbol de clasificación sin la necesidad de reunir en un solo sitio el total de registros que se encuentran almacenados en los diversos sitios de una base distribuida.

El algoritmo se ha diseñado de tal manera que toma en cuenta cada uno de los patrones presentes en cada sitio, tratando de que el árbol final se pueda aplicar a cualquier sitio para clasificar sus registros. La ventaja que presenta el árbol global es el hecho de considerar cualquier posible patrón que se encuentre en cualquier sitio, abarcando una gama más amplia de patrones.

El algoritmo se basa en el esquema cliente – servidor, en el cuál, un sitio denominado servidor, llevará el control de la construcción del árbol, mientras que el resto de los sitios se encargarán de operar directamente con los datos. Cada sitio estará generando un vector de ganancias de cada atributo en cada sitio. Estos vectores son enviados al servidor en conjunto con el número de registros asociados a los sitio. Una vez recolectados estos vectores, el servidor sumará las ganancias de cada atributo correspondientes a cada sitio, multiplicando cada ganancia por la proporción de registros de la cual se ha obtenido, seleccionando aquél que presente una mayor ganancia. Además, para disminuir el flujo de datos, los sitios le informan al servidor sobre los costos de manejar el dominio global de cada atributo o el dominio local, proceso que determinará cuales serán las ramas que se considerarán en el árbol que se construye en el servidor.

Una vez determinado el atributo ganador, el servidor se encargará de difundir dicho resultado en conjunto con el valor del dominio a trabajar. Este algoritmo considera el caso cuando algunos sitios finalizan antes que otros el desarrollo de una rama. Para ello, se han implementado técnicas que permiten discernir los patrones de los sitios que finalizan de aquellos sitios que aún pueden seguir desarrollando una rama. En caso de existir patrones propios de los sitios que han finalizado, estos son asociados a una rama la cual es etiquetada con una clase y el resto de los sitios continuarán el desarrollo. Además se considera el caso cuando no existen patrones disjuntos entre los sitios. Expuestos los puntos que se han considerado, se presenta formalmente el algoritmo.

Con la finalidad de comprobar su funcionamiento, se realizaron diversas pruebas sobre un conjunto de datos de los cuales se conoce perfectamente sus patrones, con el fin de manipularlos y verificar el funcionamiento de cada consideración que se ha hecho en el algoritmo. También se realizaron algunas pruebas con una base de datos del mundo real.

Finalmente, se exponen las ventajas y desventajas que se han observado al probar el algoritmo con las pruebas realizadas, proponiendo puntos en los cuales se podría mejorar el funcionamiento del algoritmo propuesto.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

NOTACION

$A, B, ..., Z$:	Conjuntos cualesquiera
$\{...\}$	Conjunto de elementos
$\forall$	Para todo elemento
$\exists$	Existe, para algún elemento
U	Conjunto universo
$\varnothing$	Conjunto vacío
$\in$	Pertenencia, pertenece a, es un elemento de
$\notin$	No pertenece, no es un elemento de
$\cup$	Unión
$\subseteq$	Contenido, incluido (entre conjuntos)
$\Re$	Conjunto de tuplas de una tabla (relación) cualquiera
$\Re^k$	Relación (o sitio) "*k*" de una BDD cualquiera
Y	Número de sitios de una BDD cualquiera
R_j	Fragmento horizontal "*j*" de una $\Re$ cualquiera.
$\Im$	Conjunto de entrenamiento ($\Im \subseteq \Re$)
m	Número de tuplas de una $\Re$ cualquiera
m^k	Número de tuplas(o registros) pertenecientes a $\Re^k$.
n	Número de atributos de una $\Re$ cualquiera
D_x	Dominio del atributo x, donde $D_x = \{valor_1, valor_2, ...\}$
N	Número de atributos en una $\Re$ cualquiera
Ψ	Conjunto de los N dominios de una $\Re$ cualquiera
β_x	Cardinalidad de D_x en $\Re$ (número de elementos que conforman a D_x)
d_{ji}	Valor de la tupla "*i*" en el campo "*j*" en una $\Re$ cualquiera

d_i	Tupla o registro "i" de una $\Re$ cualquiera
v_i^j	Valor en la posición "i" del dominio D_j, esto es, $v_i^j \in D_j$ tal que $v_i^j = valor_i$
$\|a\|$	Cardinalidad de "a".
$=$	Igualdad
$\neq$	Diferente de
$+$	Más, adición
$-$	Menos, diferencia
xy, $x*y$	Producto cartesiano
$<$	Menor que
$\leq$	Menor o igual
$>$	Mayor que
$\geq$	Mayor o igual que
$a \leftarrow b$	"a" toma el valor de "b"
Σ	Suma
$\prod$	Producto
$Pr(x)$	Probabilidad de x
$Pr_v(x)$	Verosimilitud del evento x
$Pr[A\|B]$	Probabilidad del evento A dado B
$\log_2$	Logaritmo en base 2
$f(x)$	Imagen de x a través de la función f
π	relación constante entre la circunferencia y su diámetro con valor 3.141592...
μ	Media
σ	Desviación estándar
$\sqrt{}$	Raíz cuadrada
$\aleph \rightarrow C$	Si se cumple $\aleph$, entonces se da "C".
x^n	Producto cartesiano $\overbrace{x*x*\cdots*x}^{n \text{ veces}}$

1 ANTECEDENTES

Desde la aparición de los primeros sistemas de cómputo, surgió la necesidad de organizar la información con el fin de tener un control eficiente de los datos almacenados. El uso de archivos fue una de las primeras formas de organización, que evolucionaron en sistemas de almacenamiento más complejos como los sistemas de Bases de Datos (BD).

Los sistemas de BD establecen un conjunto de reglas y criterios para el almacenamiento de la información (en este trabajo se maneja indistintamente el concepto "datos" e "información"), permitiendo enfrentar problemas como:

- ***Redundancia***: Se presenta cuando se tienen varias copias de una información en un sistema. Esto origina entre otros, desperdicio en el espacio de almacenamiento.

- ***Inconsistencia***: Consecuencia directa de la redundancia, caracterizada por no actualizarse todas las copias de una determinada información.

- ***Integridad de los datos***: Se puede detectar con mayor facilidad cualquier dato corrupto en la base.

- ***Compartir datos***: Los usuarios por medio de las aplicaciones, pueden realizar diferentes operaciones sobre un conjunto común de información.

- ***Seguridad***: Se puede establecer restricciones de seguridad a diferentes niveles para el acceso y/o modificación de la información.

Para organizar la información en una BD, se crearon diversos esquemas: jerárquico, de red y el relacional. El esquema que se popularizó fue el relacional, debido a la simplicidad con la que se pueden agrupar los datos por medio de tablas relacionales. Estas tablas tienen un índice o atributo denominado "*campo llave*" por medio del cuál se establece el vínculo con otras tablas, permitiendo agrupar de manera eficiente un conjunto de datos. Adicionalmente, las operaciones que se implementaron para las consultas en bases relacionales están sustentadas en la teoría matemática de relaciones, por lo cuál son muy sólidas y confiables. Por está razón, el esquema adoptado en este trabajo es el relacional.

Las operaciones de consulta clásicas arrojan como resultado un subconjunto de los datos originales. Estas se forman mediante combinaciones de las siguientes operaciones básicas [[1], [3]]:

1) *Selección*: Operación que obtiene un subconjunto de las tuplas de una relación, manteniendo todos los atributos de la relación original.

2) *Proyección*: Se obtiene una relación donde el número de atributos es menor o igual a la relación original, eliminando tuplas repetidas.

3) *Unión*: Se reúnen dos relaciones en una sola, siempre y cuando estas tengan el mismo número de atributos y sus dominios de los atributos correspondientes sean idénticos.

4) *Diferencia*: Es la operación donde se obtienen las tuplas que pertenecen a una relación pero no pertenecen a la otra (bajo las mismas consideraciones que la unión).

Consultas más complejas que requirieran de analizar la información (por ejemplo, análisis estadísticos), se realizaban de manera manual. Pronto se vio la necesidad de automatizar este tipo de tareas, con el objetivo de obtener "conocimiento" basado en la información, dada por patrones, tendencias, etc., que la información como tal, no proporciona. Esta automatización originó el proceso que se conoce como "descubrimiento del conocimiento en Bases de Datos", que consiste precisamente en identificar aquellos patrones útiles, válidos y entendibles por el usuario. Estos primeros conceptos dieron paso a lo que hoy conocemos como Minería de Datos.

Desde los primeros trabajos de análisis de información hasta la actualidad, la Minería de Datos a adoptado muy diversas técnicas de análisis. Actualmente existe una gran diversidad de algoritmos para el análisis de los datos que emplean desde técnicas estadísticas hasta métodos de inteligencia artificial (IA). Estos algoritmos, independientemente de su forma de operar, fueron diseñados para funcionar con BD's centralizadas.

El desarrollo de las BD's centralizadas fue impulsado por dos factores principalmente: el poder de procesamiento de las nuevas computadoras y el decremento en su costo. Su popularidad aumentó a tal grado, que pronto diversos sectores (industrial, científico, de educación, entre otros) utilizaron las BD para almacenar y administrar su información. Con el incremento en su uso, surgió otra necesidad: compartir información entre los diversos sistemas. Este fue un punto clave en el desarrollo de las redes de computadoras.

El objetivo fundamental de las redes de computadoras es compartir los recursos de diversos sistemas (hardware, software, datos), buscando su integración. Lograr esto implicó que varios sectores se desarrollaran para cumplir con las nuevas exigencias, entre los cuales estuvieron: medios de comunicación que permitieran el intercambio de información entre los sistemas; protocolos encargados de controlar la comunicación entre estos, etc.

El nuevo entorno generado por las redes impulsó la aparición de las Bases de Datos Distribuidas (BDD) las cuales buscan entre otros puntos, aprovechar el poder de cómputo que en conjunto pueden ofrecer las computadoras, permitiendo compartir la información, ejecutar aplicaciones de manera local o global, cómputo local e independiente, entre otras.

Actualmente, con las redes tanto locales como amplias (incluyendo a Internet), las BDD tienen las condiciones necesarias para desarrollarse. Esto implica que son necesarias nuevas herramientas que permitan trabajar con la información contenida en este tipo de base de datos.

Los trabajos que se han realizado en Minería de Datos para BDD han seguido diferentes vertientes. Algunas de éstas se han enfocado en Internet, apoyando los análisis automatizados, que han beneficiado al comercio electrónico [4]. Con técnicas de Minería de Datos se puede detectar fraudes basados en el comportamiento de cierta cuenta bancaria, por ejemplo [4], [5], [7], [17]. También se han aplicado a nuevos campos de la investigación, como la espacial, permitiendo catalogar las estrellas a partir de las características obtenidas de las imágenes de los telescopios (aplicaciones adicionales se encuentran en [4]). De hecho, el sistema SKYCAT es una BD de aproximadamente 3 terabytes que contiene imágenes de aproximadamente 2 billones de estrellas, las cuales deben ser catalogadas [34].

La investigación ha llevado a la creación de diversas organizaciones, como el Centro Nacional de Minería de Datos de Estados Unidos (NCDM [5]), la cuál ha organizado reuniones de trabajo para discutir las diversas tendencias que se siguen en el terreno de la Minería de Datos. Aquí se han propuesto diversos trabajos enfocados a sistemas distribuidos o paralelos, que trabajen con los grandes volúmenes de información que se manejan en las BD's actuales.

De aquí surge la necesidad de desarrollar técnicas enfocadas al análisis de datos almacenados en bases distribuidas, es decir, estudiar e implementar algoritmos de Minería de Datos distribuidos que, como sus antecesores, nos permitan automatizar el proceso de descubrimiento de información, ahora en un ambiente distribuido.

Organización del documento

La estructura del documento es la siguiente: en los primeros 3 capítulos se hace la presentación de los antecedentes, planteamientos y objetivos.

Posteriormente, se presentan dos capítulos introductorios a las Bases de Datos y la Minería de datos. En el Capítulo 4 se hace la presentación de los conceptos bajo los cuales se rigen las bases de datos relacionales, tanto centralizadas como distribuidas.

En el Capítulo 5 se presenta el concepto de Minería de Datos, así como se presentan algunos de los métodos más populares. Al final del capítulo, se hace una breve introducción a los sistemas paralelos y distribuidos, los cuales han afectado considerablemente al enfoque centralizado que tuvo en sus inicios la Minería de Datos.

En el Capítulo 6 se hace la presentación del algoritmo que se propone en el presente trabajo de tesis. Dicho capítulo inicia exponiendo porqué se tomo el algoritmo ID3 como base para el algoritmo que se propone en el trabajo. Posteriormente, se señalan las consideraciones que se han tomando en cuenta para el diseño del algoritmo distribuido basado en ID3, finalizando con la presentación formal del algoritmo que se propone.

En el Capítulo 7 se analizan y detallan las pruebas que se realizaron para verificar el funcionamiento del algoritmo propuesto. En el Capítulo 8 se presentan las conclusiones que se han sacado del presente trabajo de investigación, señalando los aspectos positivos y negativos del algoritmo propuesto. En base a lo anterior, se hace una serie de comentarios enfocados al trabajo a futuro sobre el algoritmo, buscando mejorar su funcionamiento.

2 PLANTEAMIENTO

El presente trabajo busca adaptar un algoritmo de Minería de Datos para un proceso de clasificación sobre una BDD con fragmentación horizontal. Conseguir este objetivo plantea la necesidad de estudiar la naturaleza de las BDD así como los algoritmos de minería existentes, con el fin de tener un amplio conocimiento teórico sobre el campo que se va a trabajar.

Las BDD's requieren de algoritmos que permitan analizar automáticamente el gran volumen de información que pueden contener. Esto nos lleva a desarrollar nuevos algoritmos o adaptar los existentes para su aplicación en bases distribuidas.

En el caso de modificar un algoritmo existente, el proceso debe tomar en cuenta aspectos como:

1) Esquema bajo el cuál se encuentra la base distribuida (relacional, jerárquico, de red).

2) Fragmentación aplicada a la BDD.

3) La ubicación de los datos.

Las BDD parten de un conjunto de criterios que sirven para dividir y ubicar la información de una BD centralizada en los distintos sitios. Estos criterios deben considerar: la cantidad de sitios que existirán, los costos y frecuencias de acceso, la replicación que se manejará, etc. En base a esto, se realiza en primera instancia la fragmentación de los datos de la relación original. Esta tiene dos formas básicas de realizarse: fragmentación horizontal y fragmentación vertical. El primer tipo opera sobre las tuplas de las relaciones, obteniéndose sub-relaciones que forman un subconjunto de la relación original. La fragmentación vertical divide a la relación en base a sus atributos. Ambas se conocen como fragmentaciones primarias. Debido a que la fragmentación horizontal conserva la estructura básica de las relaciones (las tuplas mantienen el número original de atributos), la propuesta de tesis se enfocará a trabajar en BDD's con fragmentación horizontal primaria.

Partiendo de una BDD bien construida, el objetivo del trabajo será seleccionar un algoritmo de Minería de Datos que se adapte mejor a las características de la base distribuida, buscando lograr la extracción de patrones (conocimiento) de una BDD con fragmentación horizontal.

Para la selección del algoritmo, es necesario estudiar las técnicas más comunes presentes en la Minería de Datos [7], [9], [11], [13], [16], ente las que se encuentran:

- ***Árboles de decisión***: esta técnica recibe como entrada un objeto con sus atributos, y da como resultado la clase a la que pertenece el objeto, realizándose un proceso de clasificación. Cada nodo interno corresponde a un valor de un atributo y las ramas están etiquetadas con los posibles resultados de la prueba. Las hojas representan el valor de una clase. Algoritmos como ID3, CART, y C4.5 son señalados como los más utilizados [6], [7], [8], [9], [10], [13], [14], [16].

- ***Redes neuronales***: Estos modelos se basan en una red de nodos (neuronas) altamente interconectados entre si. Los modelos ajustan los pesos asociados a cada neurona de la red, dependiendo de un conjunto de valores de entrada. Los pesos son ajustados por medio de funciones lineales o no lineales, proceso que se conoce como entrenamiento. Dependiendo de los ajustes, la red presentará una amplia gama de resultados. Por ello, son muy adaptables a los patrones de los valores de entrada, pero al mismo tiempo, las vuelve muy impredecibles [23], [24], [25], [26], [27], [28].

- ***Modelos de dependencias probabilísticas***: aquí se hace uso de redes bayesianas, donde las evaluaciones se basan en probabilidades [11].

- ***Modelos relacionales***: aquí se hace uso de un análisis lógico (programación lógica inductiva) para obtener los modelos buscados.

Seleccionado un algoritmo, se buscará modificarlo con el objeto de que los resultados que arroje al trabajar sobre una BDD sean válidos y/o equivalentes de los que se obtendría al trabajar con un algoritmo del mismo tipo, pero centralizado, con la ventaja que el algoritmo distribuido aprovechará los recursos de cómputo de la BDD.

Figura 2.1. Esquema de trabajo de un algoritmo centralizado sobre una BD centralizada

Se plantea que uno de los puntos fundamentales a resolver es como integrar los resultados parciales de cada sitio para formar un resultado global. En la Figura 2.1, podemos observar que aplicar un algoritmo centralizado a una BD centralizada genera un único resultado que representa los patrones de la BD (resultado que puede estar compuesto de un conjunto de patrones). Sin embargo, esto no sucede en una BDD, donde aplicar el mismo algoritmo a cada base local, arrojara un conjunto de "N" resultados independientes entre sí. Por ello, es necesario el estudio de técnicas que permitan aprovechar el cómputo paralelo de una BDD y a su vez, generen un resultado global del análisis hecho.

Existen diversas estrategias que son utilizadas para atacar problemas de multiprocesamiento. Estas son divididas en dos áreas: el cómputo paralelo y el distribuido. Ambas comparten la

característica de realizar cálculos en el mismo instante de tiempo en diferentes procesadores. En una BDD, los datos de cada sitio son distintos, por lo que el diseño de un algoritmo para estas bases puede ser de dos formas: el mismo algoritmo trabaje sobre los diversos datos de la BDD (instrucción simple, datos múltiples, o SIMD) o diversos fragmentos de un algoritmo trabajen sobre cada sitio (instrucción múltiple, datos múltiples, o MIMD). En ambos esquemas, el resultado final se puede obtener de las siguientes formas:

- *Con intercambio de información.* Aquí, los algoritmos que se ejecutan en cada sitio requieren importar datos o resultados de otros sitios para continuar sus cálculos. Por ello, el resultados final puede generarse al mismo tiempo en todos los sitios o en alguno en particular, dependiendo del método.

- *Sin intercambio de información.* En este método, los algoritmos trabajan de manera independiente en cada sitio y en caso de ser necesario, en un proceso final, reunir los resultados generados en cada uno del ellos para obtener el resultado global.

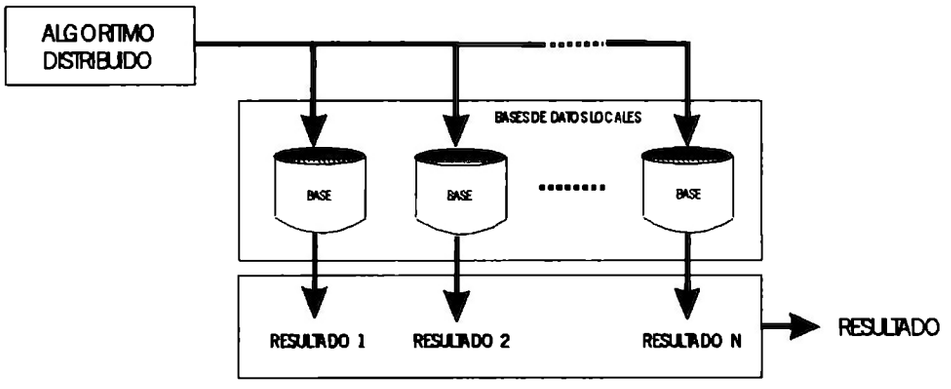


Figura 2.2 Esquema de trabajo de un algoritmo Distribuido concurrente en una BDD.

La Figura 2.2. muestra un algoritmo sin intercambio de información. Observemos que cada algoritmo se ejecuta de principio a fin sin necesidad de datos o resultados de otro sitio. Los resultados de cada sitio pueden formar parte directamente del resultado final o, pasar por un proceso de post – procesamiento en el cuál se conjugan los resultados parciales de cada sitio en uno solo.

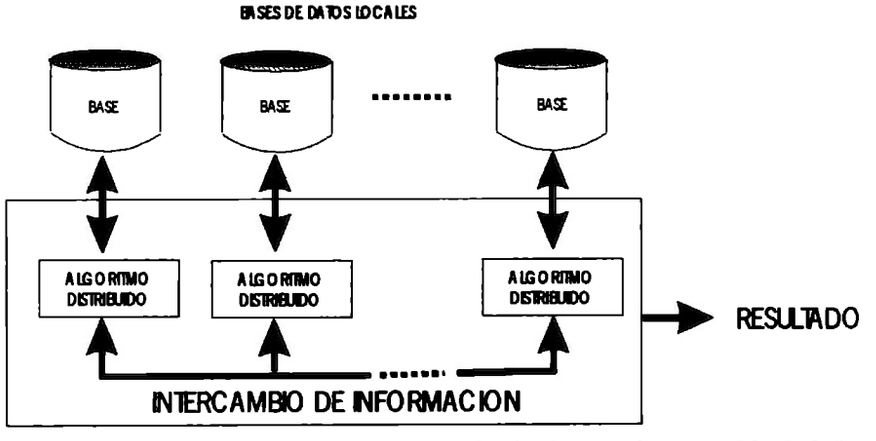


Figura 2.3. Esquema de trabajo de un algoritmo Distribuido con intercambio de información.

En contraparte, como lo muestra la Figura 2.3, la construcción con intercambio de información requiere la interacción de cada sitio para la construcción de los resultados locales. El intercambio puede ser de datos sin procesar o resultados provenientes de los cálculos locales. El resultado final puede generarse de manera simultánea en todos los sitios o uno de ellos llevar la construcción del mismo. Un aspecto muy cuidado en éste esquema es el volumen de información a intercambiar, ya que si es muy elevado, el sistema puede presentar fallos en un mayor grado.

Considerando los aspectos ya señalados, se busca obtener un algoritmo que permita realizar un proceso de clasificación aprovechando el procesamiento distribuido que una BDD ofrece.

3 OBJETIVOS

Nuestro objetivo es contribuir a la evolución de los algoritmos de Minería de Datos mediante la modificación de un algoritmo para que pueda aplicarse a una BDD con fragmentación horizontal.

Realizar está tarea, implica estudiar la naturaleza que impera en una BDD, así como los algoritmos de Minería de Datos existentes. El resultado de estos estudios permitirá escoger un algoritmo de clasificación que se adapte mejor a la naturaleza de las BDD's.

Para lograr la consecución de este objetivo, es necesario realizar los siguientes pasos:

1. Estudiar las características de una BD centralizada.
2. Analizar las características del funcionamiento de una BDD con fragmentación horizontal primaria.
3. Analizar los diferentes tipos de algoritmos existentes en Minería de Datos para seleccionar el más apropiado para las BDD's con fragmentación horizontal primaria.
4. Realizar las modificaciones necesarias al algoritmo seleccionado, con el objetivo de obtener un algoritmo de Minería de Datos distribuido que opere adecuadamente sobre nuestra BDD seleccionada.
5. Analizar el comportamiento del algoritmo con base a los resultados que arroje.
6. Exponer las conclusiones del trabajo (ventajas, desventajas, trabajo a futuro).

4 BASES DE DATOS

La aparición de las BD's y la gran cantidad de herramientas desarrolladas, tanto en software como en hardware, han permitido administrar la información con una mayor eficiencia. A mediados de la década de los 60's [1] aparecieron las primeras BD's comerciales, donde la cantidad de datos almacenada era relativamente pequeña, siendo su administración asequible por un solo sistema (a estos sistemas se les denominó *bases de datos centralizadas*). Estas bases se albergaban en enormes sistemas de cómputo sumamente costosos, por lo que pocos centros de investigación contaban con uno. Con el desarrollo de la tecnología, los sistemas de cómputo evolucionaron en sistemas con una mayor capacidad de procesamiento, más económicos y mayor facilidad en su instalación. Esto propició que escuelas, centros de investigación, comercios, etc., pudieran hacer uso de las nuevas generaciones de computadoras, aprovechando sus velocidades para procesar la información. Con éste desarrollo, las BD's encontraron una amplia aceptación entre los nuevos usuarios, debido a la facilidad ofrecida para administrar y controlar su información.

La constante evolución en el campo computacional llevó a la aparición de las primeras redes de computadoras alrededor de los años 70's. Estas redes permitieron por primera vez comunicar a distintos sistemas de cómputo, y por ende, compartir tanto sus recursos como su información. Con ésta evolución, fue prioritario el desarrollo de programas que permitieran administrar todo el conjunto de datos que ahora se podía compartir. Fue entonces cuando nace el concepto de las *Bases de Datos Distribuidas*, las cuales permiten almacenar la información en diferentes sitios o lugares, y por medio de consultas remotas, accederla.

4.1 BASES DE DATOS CENTRALIZADAS

4.1.1 ¿Por qué Bases de Datos Centralizadas?

Las bases de datos originalmente se desarrollaron bajo un concepto centralizado. La idea de "centralizado" se refiere a que un solo programa o aplicación administra el total de la información presente en la base. A este sistema se le denomina "Sistema Manejador de Base de

Datos" (SMBD, o sus siglas en inglés, DBMS), el cuál controla el acceso, la manipulación y la actualización de la base.

Actualmente, los SMBD permiten consultas o actualizaciones secuenciales o simultáneas (concurrentes). Cuando el SMBD permite operaciones simultáneas, el sistema debe considerar la consistencia de la información, debido a que diferentes aplicaciones pueden estar operando sobre un mismo dato, siendo más compleja su administración que una base de acceso secuencial.

Las BD's centralizadas se desarrollaron de manera importante basándose en conceptos concurrentes, permitiendo que varios usuarios realizarán tareas diversas sobre una misma base. La representación de este tipo de bases se ilustra en la Figura 4.1, en donde se observa que la interacción entre la base y el entorno es mediante el SMBD, ofreciendo apoyo a los usuarios para el acceso a los datos. Los usuarios acceden a los datos mediante las aplicaciones o software del sistema, que están basadas en las operaciones soportadas por el SMBD. Mediante las herramientas ofrecidas por el SMBD, el administrador de la base puede encargarse de las operaciones propias de mantenimiento de la información, utilizando el lenguaje de manipulación de datos (LMD o DML en inglés).

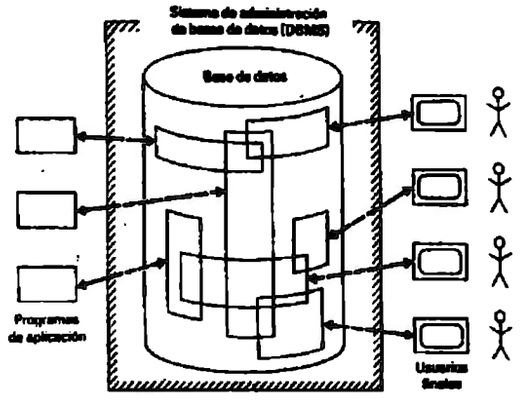


Figura 4.1. Esquema de una BD centralizada

Las BD's centralizadas ofrecen un mayor control sobre los datos, permitiendo manejar problemas como la redundancia. Por ejemplo, antes de almacenar nueva información, el SMBD verifica que no se encuentre previamente almacenada. Lo anterior apoya al control de la consistencia de datos, evitando que se actualice un conjunto de datos y sus replicas no (inconsistencia). Por otra parte, permite establecer criterios de seguridad para restringir el acceso a los datos. Esto resulta útil dentro de las organizaciones, ya que el acceso a los datos depende del nivel jerárquico del usuario. En general, la administración de la información se volvió una tarea más sencilla y más económica, requiriéndose un número muy reducido de personal calificado.

Otra característica importante de las BD's es el crecimiento: las BD's pueden incrementar su capacidad de almacenamiento de datos según las necesidades del usuario. Sin embargo, pronto se observó que el ritmo de crecimiento en la información almacenada era superior a la capacidad de procesamiento de los sistemas (por ejemplo, el FinCEN AIS es una asociación encargada de

verificar que no se efectúen fraudes a cuantas bancarias, siendo complicada la tarea debido a la cantidad de transacciones que se realizan actualmente y que continúa creciendo [17]). Para resolver está problemática y aprovechando los constantes avances tecnológicos, se emplearon sistemas multiprocesador, basados en un esquema centralizado de control. Sin embargo, estos sistemas no pudieron dar respuesta a todas las necesidades de los usuarios, ya que problemas como la eficiencia en la búsqueda y recuperación de la información continuaban, principalmente en consultas remotas, donde el "servidor" tenía que enviar por medio de una red las consultas deseadas. Pronto se observó que dividir la BD centralizada en fracciones más pequeñas podría ayudar a resolver el problema, ya que cada fragmento sería controlado por un sistema independiente con capacidad de procesamiento local, permitiendo realizar búsquedas más rápidas por tratarse de un número menor de datos. Por otra parte, cada sitio podría acceder a los recursos de otros sistemas, permitiendo compartir tanto datos como programas. Esto dio paso a las nuevos sistemas de bases de datos distribuidas que, por lo anterior, se basan en sistemas del tipo centralizado interconectados para compartir recursos. Debido a la importancia ya expuesta de las bases centralizadas, se expone a continuación los esquemas más utilizados así como definiciones básicas de los elementos que las conforman.

4.1.2 Estructura de una Base de Datos Centralizada

Los SMBD establecen estructuras predefinidas para los datos, así como las operaciones permitidas sobre estos. Las BD se han clasificado en tres grupos de acuerdo al enfoque del esquema en que se encuentra almacenada la información:

- Enfoque relacional.
- Enfoque jerárquico.
- Enfoque de red.

El *enfoque jerárquico* usa una representación de "árbol", donde los datos se representan por medio de registros y las asociaciones por medio de ligas, que pueden representar asociaciones de uno a muchos. Ese enfoque permite modelar de una manera natural jerarquías del mundo real (jefe, subjefes, empleados, etc) desde un registro denominado raíz. Sin embargo, tiene problemas propios de la estructura, ya que se debe de tener conciencia de la jerarquía de los datos para poderlos modificar [1].

El *enfoque de red* también usa la representación de ligas para las asociaciones, pero la diferencia radica en que este último usa una estructura más general, ya que se pueden representar asociaciones más complejas. Aquí, cada registro puede tener cualquier número de superiores inmediatos, permitiéndose modelar correspondencias de muchos a muchos. La representación es más compleja tanto a nivel de registros como para el LMD, ya que es necesario expresar más sentencias de control para búsquedas originadas por el número de ligas superiores de un registro [1].

Los esquemas expuestos no gozan de tanta popularidad como *el relacional*, debido a que este es más sencillo en su interpretación y manipulación al momento de ser implementado, y además,

está sustentado en una sólida teoría matemática de relaciones. Por lo anterior, el presente trabajo se enfoca a este tipo de bases.

En el *enfoque relacional*, los datos se representan mediante *tablas*, que en gran parte de la bibliografía se denominan *relaciones*. La definición formal del término relación es la siguiente *[1]*:

> **Definición 1: Relación**
> *Dado el conjunto $\{D_1, D_2, ..., D_n\}$, donde cada D_i con $1 \leq i \leq n$ es un conjunto de elementos cualesquiera, se dice que $\Re$ es una relación sobre estos n conjuntos si es un conjunto de "m" tuplas o registros distintos $d_j = < \quad , v_j^2, ..., v_j^n >$, $1 \leq j \leq m$, tal que $\in D_1$, $v_j^2 \in D_2$, ..., $v_j^n \in D_n$. Los conjuntos D_1 hasta D_n son los dominios de $\Re$ y $U = D_1 x D_2 x ... x D_n$ representa al conjunto universo de todas las relacione posibles. El valor "n" es el grado de $\Re$ y "m" es su cardinalidad.*

Una forma alternativa de definir las relaciones se basa en el producto cartesiano de "*n*" conjuntos. Esto es, sean D_1, D_2, ..., D_n "*n*" conjuntos, donde el producto cartesiano se denota como $D_1 \times D_2 \times ... \times D_n$, que representa todas las "*m*" tuplas posibles $<v_j^1, v_j^2, ..., v_j^n>$ tales que v_j^1 pertenece a D_1, v_j^2 pertenece a D_2, ..., v_j^n pertenece a D_n. En base a esto, se define una relación $\Re$ sobre los conjuntos D_1, D_2, ..., D_n si es un subconjunto del producto cartesiano $D_1 \times D_2 \times ... \times D_n$.

En la literatura de las BD's relacionales, la información del usuario es agrupada en tablas. De hecho, una BD puede contener una infinidad de tablas, las cuales son relacionadas por medio de las llaves primarias (las cuales se explican más adelante). Para el presente trabajo, denotaremos por medio de $\Re$ a una y solo una de estas tablas. Cada $\Re$ está formada de un conjunto de atributos fijos. A cada atributo lo denotaremos por medio de A_i, donde los posibles valores de este atributo, es decir, su dominio, lo expresaremos por D_i. Por medio de Ψ representaremos al conjunto de atributos que componen a una $\Re$, esto es, si $\Re$ tiene un grado "*n*", tendremos que:

$$\Psi = \{ A_1, A_2, ..., A_n \}$$

Si "*n*" tiene el valor de uno, a la relación se le conoce como *unaria*, si es dos, *binaria* y así sucesivamente. En general, una relación de grado "*n*" se denomina n-aria.

Normalmente no se define ningún orden entre las tuplas de una $\Re$. No pasa lo mismo desde el punto de vista matemático. Esto es, si se intercambian dos elementos de una relación, ya no serán equivalentes la original y la modificada. Sin embargo, para la teoría en BD's no se es tan estricto, manteniéndose la equivalencia. De manera similar, se mantiene la equivalencia entre dos relaciones si el orden de los atributos es alterado [3].

Dentro de las BD's, todo valor que integra a una $\Re$ es atómico. Dicho de otra manera, en la intersección de una fila y una columna, debe encontrarse uno y solo un valor. Para lograr esto, se han establecido una serie de normalizaciones, que son una serie de condiciones que verifican este hecho (en la teoría de BD's, comúnmente se emplea desde la primera hasta la tercera forma normal, aunque existen algunas otras [3]).

Cuando se trabaja con BD's relacionales, comúnmente existen atributos cuyos valores tienen la característica de ser únicos en una $\Re$, los cuales son utilizados para identificar a las tuplas de la relación. A estos atributos de les denomina *campos llave*. Los campos llave pueden estar formados por uno o varios atributos (incluso pueden abarcar el total de los atributos de una $\Re$). Aquel atributo que representa a una $\Re$ se denomina *llave primaria* y aquellos atributos que no son designados como llave primaria, se les conoce como *llaves alternas*. En cualquier caso, dentro del esquema relacional, cualquier $\Re$ debe tener una única llave primaria. En caso de que ningún atributo cumpla las características de ser llave primaria, generalmente se crea un atributo con estas características (por ejemplo, se puede añadir un campo numérico creciente o decreciente en el cuál ningún valor se repita).

Además de la propiedad expuesta, las llaves primarias cumplen otras características. En particular, ningún componente de una llave primaria debe tener un valor nulo. Esto es, si consideramos que A_x es un campo llave, debe cumplirse que:

$$\forall v_j^x \in \Re \mid v_j^x \in D_x, v_j^x \textit{ no es un valor nulo, para } 1 \leq j \leq m$$

Los valores de los campos llave no deben depender de los valores de otros atributos, es decir, tienen la propiedad de ser valores independientes. Sin embargo, otros atributos que no sean llaves si pueden depender de los campos llave.

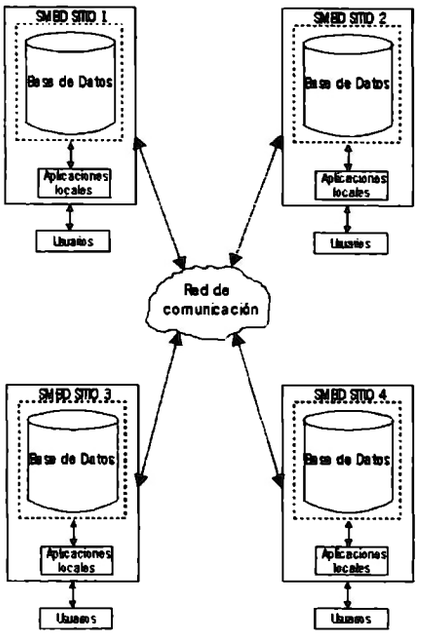
4.2 BASES DE DATOS DISTRIBUIDAS

Una Base de Datos Distribuida está formada por un conjunto de BD's independientes entre sí que trabajan en conjunto para comportarse como una sola BD. Por esto, muchos conceptos que se manejan en bases centralizadas, se generalizan a bases distribuidas. Por ejemplo, encontramos el equivalente al SMBD centralizado. El sistema encargado de administrar una base distribuida se denomina Sistema Manejador de Bases de Datos Distribuidas (SMBDD o DDBMS en inglés). Las tareas realizadas son más complejas que las de un SMBD centralizado, debido a que ahora se debe pensar en datos y consultas tanto locales como remotas.

En una BDD, la información se encuentra distribuida dentro de un conjunto de bases "locales", las cuales, en su conjunto forman la base global. Formalizando, una BDD se compone de S^1, S^2, ...,S^Y bases locales, donde cada S^i representa los datos almacenados en el sitio "i" (cada S^i contiene un conjunto de $\Re$'s). Es importante señalar que estos datos no necesariamente son excluyentes entre si, por lo que si S es el conjunto total de información de la base original, tendremos que: $S = S^1 \cup S^2 \cup ... \cup S^Y$. Una BDD se define de la siguiente manera [3]:

> **Definición 2: Base de Datos Distribuida**
> *Una BDD es una colección de datos que está distribuida sobre diferentes computadoras (sitios) de una red. Cada sitio de la red tiene capacidad de procesamiento local y puede ejecutar aplicaciones locales. Además, cada uno de ellos puede participar en la ejecución de una aplicación global, que requiera acceder a los datos de diversos sitios usando un subsistema de comunicación.*

Una BDD no necesariamente debe estar en sitios geográficos distintos. Físicamente la base puede localizarse en sitios sumamente distantes (distintas ciudades, naciones, por ejemplo) o incluso dentro de un mismo edificio o habitación. Para ilustrar este hecho observemos la Figura 4.2, donde la BDD está integrada por 4 sitios, los cuales se comunican por medio de una red (Internet por ejemplo). Observemos que cada sitio tiene su propio SMBD apoyado por un procesamiento local independiente, capaz de ejecutar aplicaciones locales.



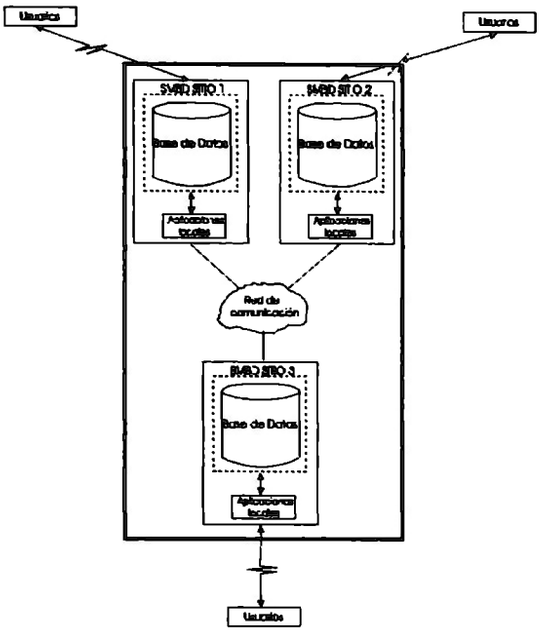


Figura 4.2. BDD ubicada en sitios geográficamente distintos comunicada por una red.

Figura 4.3. BDD ubicada en un mismo sitio comunicada por una red.

En la Figura 4.3, la BDD se encuentra ubicada en un mismo sitio (un mismo edificio o habitación, por ejemplo), dividida en 3 bases o sitios. Observemos que tiene las mismas características señaladas en el caso anterior: un SMBD para cada sitio, procesamiento independiente y una red de comunicación. Estas ilustraciones también nos ayudan a comprender lo siguiente: una BDD es un sistema distribuido integrado por un conjunto finito de sitios que ofrecen la capacidad de realizar diferentes cálculos en el mismo instante de tiempo (paralelismo). Una máquina paralela también ofrece está característica, sólo que el control de los procesadores es llevado por un solo sistema. Por ejemplo, los cálculos que requiere el SMBD de la Figura 4.1. pueden llevarse a cabo en una máquina multiprocesador, pero el control de la BD continúa estando en un solo SMBD, por ello, no se considera como BDD.

Los sitios de las BDD's se comunican por medio de una red de computadoras. En la teoría de las redes, comúnmente se denominan *host* a las computadoras que integran a una red. En BDD, normalmente se denominan *sitios* (en este trabajo se emplea el término sitio).

Físicamente, una red está conectada por algún tipo de cable por el cuál fluye la información (medio guiado) o por algún tipo de comunicación inalámbrica (medio no guiado). Dentro de los medios guiados se encuentra el par trenzado, cable coaxial, fibra óptica, etc., mientras que entre los no guiados están las microondas, señales satelitales, etc.

Independientemente del medio de transmisión, las redes se configuran de distintas maneras. En la actualidad, distinguimos dos tipos: las *redes locales* (LAN) y las *redes amplias* (WAN)[41]. Estas redes son montadas bajo diversas *topologías*. Existen varios tipos, como estrella, bus, anillo, jerárquica, completamente conectada, etc. Estas topologías generalmente se implementan de manera física, pero actualmente se configuran redes *híbridas*, las cuales físicamente tienen una topología, como bus, por ejemplo, pero lógicamente funciona como una estrella. Dependiendo de lo anterior, la BDD tendrá que ajustarse a la configuración de la red sobre la cuál se encuentra montada.

Hay muchos factores que se pueden mencionar con respecto a configuraciones de redes, y que finalmente repercuten en la eficiencia de la misma. Por ello, una BDD debe considerar la configuración de la red para buscar un mejor rendimiento.

4.3 REFERENCIA DE LA ARQUITECTURA PARA UNA BDD

No existe una referencia general para la arquitectura de una BDD. Sin embargo, existe una idea que conceptualmente es muy útil para entenderla, representada en la Figura 4.4, la cuál puede servir para entender el proceso de construcción [3].

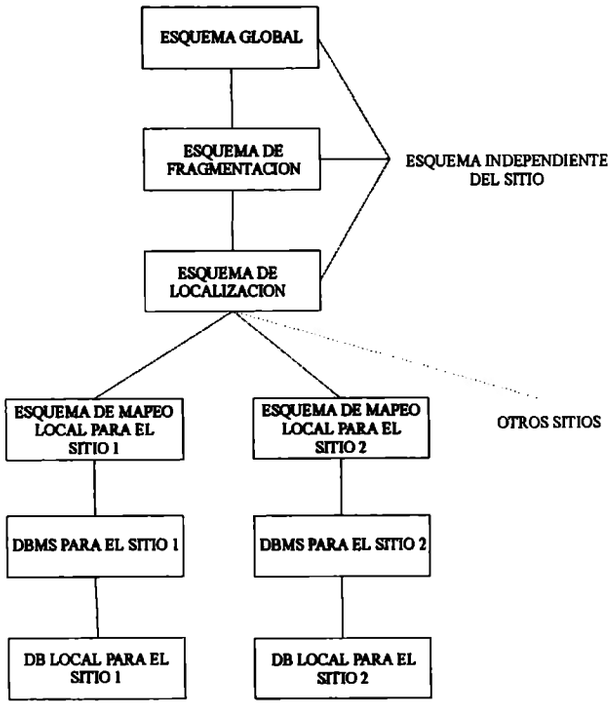


Figura 4.4 Arquitectura general para construir una BDD.

En primera instancia, se ilustra un esquema que engloba el total de la información de la base distribuida. Este se conoce como *esquema global*, y en el se definen todos los datos contenidos en la BDD (este es equivalente al esquema de las relaciones en una base centralizada), como los dominios de los atributos globales, así como las relaciones que se establecen entre estos.

El esquema global es dividido en distintas porciones o fragmentos, en lo que se conoce como *esquema de fragmentación*. Esta se puede llevar a cabo de muy diversas maneras: horizontal, vertical y/o combinaciones (sección 4.4.). Cuando una base ha sido fragmentada, el siguiente paso es la ubicación de los fragmentos. El *esquema de localización* realiza está tarea, ubicando los fragmentos de acuerdo a diversos criterios: frecuencias de acceso, costos de comunicación, costos de almacenamiento, costos de recuperación, entro otros. Es importante señalar que dependiendo del mapeo efectuado en el esquema de localización, se determina si una base distribuida va a ser redundante o no redundante.

Los tres niveles superiores de este esquema son independientes de cada sitio. En contraparte, las operaciones realizadas en los tres niveles inferiores se refieren más a las tareas locales de cada sitio. Cada uno de ellos cuenta con un mapeo de localización de imágenes físicas locales. Este esquema se conoce como *esquema de mapeo local*, y depende del SMBD de cada sitio. Es claro que los esquemas de mapeo de cada sitio pueden ser distintos entre si. En el nivel inmediato inferior encontramos al SMBD de cada sitio, el cuál trabaja directamente con la base asociada a este. Como se observa en la Figura 4.4, cada sitio es independiente uno de otro, por el hecho de tener un SMBD exclusivo, que administra un conjunto de datos que pertenecen al sitio, pero mantienen una relación unos con otros por el esquema global del que se derivan.

Un SMBDD debe llevar a cabo las operaciones de un SMBD centralizado y tener todas las funciones requeridas para el manejo de la distribución de los datos. Estas últimas operaciones deben de actuar de manera transparente para el usuario, es decir, debe de darse la impresión como si se tratara de una sola BD. Normalmente se identifican dos niveles de transparencia: *transparencia de fragmentación* y *transparencia de localización*. La transparencia de fragmentación es el más alto grado de transparencia. En éste, el usuario o aplicación trabajan en relaciones globales, mientras que, en la transparencia de localización el usuario o aplicación trabajan sobre fragmentos de una relación global. Sin embargo, en ambos casos, no se conoce donde se encuentran localizados los fragmentos.

Otro concepto importante es la redundancia (copias idénticas de información en distintos sitios). En el esquema de localización y dependiendo de las necesidades del usuario, se pueden establecer lineamientos que permitan replicar una porción de las relaciones globales en distintos sitios, pero también, por el contrario, se puede evitar cualquier posibilidad de replicación de información.

Las BDD's han demostrado gran eficiencia principalmente por la autonomía que presenta cada sitio del resto del sistema. Esto permite la ejecución de aplicaciones que trabajen de manera local, y en caso de ser necesario, se accede a la información de otros sitios. Lo anterior favorece a la distribución de las tareas que se efectúan sobre la base. Adicionalmente, en caso de presentarse problemas en un sistema local, el resto de la base distribuida puede seguir funcionando, lo cuál implica una gran tolerancia a las fallas o caídas del sistema.

4.4 FRAGMENTACIÓN DE DATOS

Una decisión fundamental en el diseño de una BDD gira en torno a la ubicación de los datos. Ello implica buscar la mejor forma de fragmentarlos, considerando las necesidades del usuario así como el desempeño del sistema.

El análisis debe incluir aspectos como:

- Los beneficios que presentará la distribución de los datos comparado con un esquema centralizado.
- La disponibilidad de los datos para consultas o actualizaciones.
- La seguridad que debe implementarse ante la intromisión de extraños.
- El desempeño global del sistema, considerando la frecuencia de acceso a los datos así como los costos de consultas remotas.
- La tolerancia que la BDD ofrecerá ante fallas o caídas de sistemas locales.
- Las limitantes relacionadas con la infraestructura que será utilizada, así como los costos de implementación y mantenimiento, entre otros.

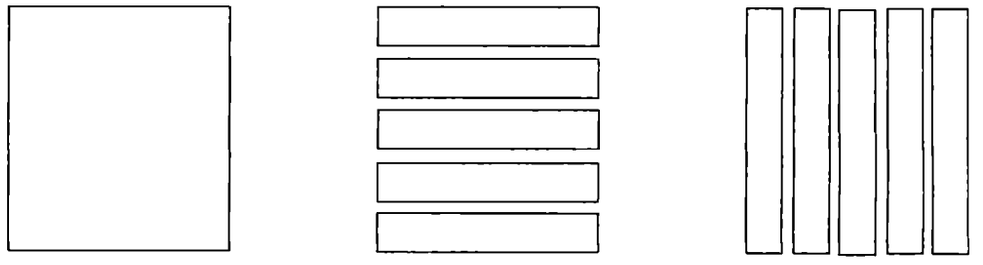
Existen dos diseños de fragmentación: si se parte de una BD centralizada, el diseño que se realiza para fragmentarla se conoce como *descendente*. Por otra parte, si se parte de un conjunto finito de BD's centralizadas para integrarlas en una BDD, el diseño se conoce como *ascendente*. La idea que se presentó en la Figura 4.4 corresponde a un diseño descendente, el cuál es adoptado en el presente documento.

En este diseño, el proceso se inicia identificando las relaciones que se establecen en el esquema global. Estas relaciones dependen de la información que deberá contener la BDD. El diseñador, de acuerdo a su criterio, establecerá las relaciones que más se ajusten a la naturaleza de los datos.

Una vez identificadas las relaciones, se continúa con el esquema de fragmentación. Este se lleva a cabo de dos formas básicas: *fragmentación horizontal* o *fragmentación vertical*. A partir de una $\Re$, Figura 4.5. a), la fragmentación horizontal divide en subconjuntos a los registros, conservándose la estructura de los atributos de la relación original, Figura 4.5 b). En cambio, la fragmentación vertical divide a la relación en base a sus atributos, Figura 4.5 c). Generalmente, en este tipo de fragmentación es necesario que el campo llave sea replicado en cada fragmento resultante.

A partir de una fragmentación horizontal de una $\Re$ cualquiera, se puede inducir una nueva fragmentación horizontal en otra $\Re'$ por medio de los campos llaves que las relacionan, conocida como *fragmentación horizontal derivada*. Análogamente se pueden obtener fragmentaciones verticales derivadas. Asimismo se pueden realizar fragmentaciones mixtas o híbridas, que se realizan aplicando primero una fragmentación horizontal y al resultado se la aplica una

fragmentación vertical (HV) o por el contrario, se aplica primero la vertical y posteriormente la horizontal (VH). Estas ideas se ilustran en la Figura 4.5 d) y e).



a) Relación original **b) Fragmentación horizontal** **c) Fragmentación vertical**

d) Fragmentación HV **e) Fragmentación VH**

Figura 4.5 Tipos de fragmentación para una BDD

Cualquier tipo de fragmentación debe cumplir con tres condiciones:

- Ser completa, es decir, que todos los datos de la relación global pertenecen a algún fragmento.
- La relación global se debe de poder reconstruir a partir de los fragmentos.
- Los fragmentos deben ser disjuntos.

La fragmentación horizontal es la que comúnmente se emplea en los diseños de las BDD's, ya que la estructura de los atributos es la misma tanto para la relación original como para los fragmentos resultantes. Con ello, cualquier consideración aplicable a la estructura de la relación original, continúa siendo válida para los fragmentos. Por ejemplo, si consideramos a la Tabla 2 como una relación global que ha sido dividida en dos fragmentos, Tabla 3 y Tabla 4, podemos observar que estas últimas tienen los mismos atributos que la Tabla 2. De esto podemos concluir que cualquier tipo de relación o dependencia entre estos atributos continúa en los fragmentos derivados de la relación original.

4.4.1 Fragmentación horizontal

Existen dos tipos de fragmentación horizontal: primaria y derivada. Para realizar una fragmentación horizontal primaria, uno puede emplear dos métodos:

- Estadístico, definido por el número de referencias de las aplicaciones a los fragmentos.
- Por predicados, el cuál hace uso de propiedades lógicas o semánticas de las tuplas.

Estadístico: la fragmentación horizontal primaria basada en el método estadístico es utilizada cuando se conoce de antemano la frecuencia de uso que se tiene sobre las distintas tuplas de una $\Re$. Sin embargo, está condición no siempre se conoce, ya que requiere un análisis histórico de los datos, por lo que el método más utilizado es el que se explica a continuación.

Predicados: este no requiere un análisis previo, ya que se basa en las propiedades lógicas o semánticas que los registros. Este método se basa en el uso de los *predicados simples,* que se define como sigue:

Definición 3: Predicado simple

Sea una relación $\Re$ sobre un conjunto de atributos Ψ cualquiera, un predicado simple a_{ij} definido sobre $\Re$ se denota como:

$$a_{ij}: A_i \; \theta \, Valor$$

donde θ es un operador relacional $=, <, \leq, >, \geq, \neq$ y $Valor \in D_i$. Se denota como $P_i=\{a_{i1}, a_{i2}, \ldots, a_{i\beta}^{i}\}$ al conjunto de todos los predicados simples derivados de A_i y al conjunto global de predicados simples con:

$$P = \bigcup_{i=1}^{n} P_i$$

De hecho, un predicado simple hace una sola prueba sobre un atributo de una $\Re$. Por ejemplo, si tomamos en cuenta la relación $\Re$ de la Tabla 1, un posible predicado simple para el atributo *age* es *age = young*.

Un predicado simple se llama *relevante* cuando permite distinguir a dos fragmentos que son referenciados de distinta manera por al menos una aplicación. A partir de los predicados simples se pueden derivar los predicados *minitérmino*, que son el resultado de combinaciones booleanas de predicados simples por medio del conectivo lógico "$\wedge$". Esto es, si nos basamos en un conjunto P de una $\Re$ cualquiera, el conjunto de predicados minitérmino $M = \{m_1, m_2, \ldots, m_k\}$ se denota como:

$$M = \{m_t \mid m_t = a_{ij}^* \wedge \ldots \wedge a_{xy}^*\} \; 1 < t < k$$

donde el número de predicados que conforman a cada m_t es variable, y $a_{ij}^* = a_{ij}$ o $a_{ij}^* = \neg a_{ij}$. Con los conceptos anteriores, la fragmentación horizontal se puede definir de la siguiente manera:

Definición 4: Fragmentación horizontal

Dada una relación $\Re$ cualquiera, los fragmentos horizontales derivados de R están dados por:

$$R_j = SL_{Fj}(\Re)$$

donde F_j es la fórmula de selección empleada. Si F_j está en forma normal conjuntiva, es un predicado minitérmino, esto es, $F_j \in M$.

Una fragmentación exige que el conjunto de fórmulas de selección empleadas sea mínimo, es decir, todos sus predicados deben ser relevantes. Lo anterior favorece a evitar la redundancia de datos. También es deseable que cualesquiera dos tuplas de cualquier fragmento tengan la misma probabilidad de ser referenciadas por una aplicación cualesquiera. En caso de no contar con un historial de las aplicaciones que hacen uso de los registros, se puede realizar una estimación de los posibles accesos de las aplicaciones a los registros, aunque esto puede no resultar muy preciso. Este equidad en las referencias favorecerá a obtener un equilibrio de cargas de trabajo para los sitios que alberguen a estos registros.

Establecer el conjunto de fórmulas para una fragmentación resulta en ocasiones sumamente complejo, debido a que el dominio de los atributos suele ser grande y aunado a esto, manejar las referencias de las aplicaciones a los registros lo complica aún más. Por ello, la experiencia en el uso y comportamiento de los datos en una BD, generalmente es la herramienta en la cuál se apoya una buena fragmentación.

4.4.2 Ubicación de fragmentos horizontales

El proceso de ubicación de fragmentos es una tarea compleja, debido a la cantidad de variables que se involucran en el proceso. Generalmente las variables que se consideran se basan en los costos de: almacenamiento, de consultas y actualizaciones, referentes a la replicación de datos, de comunicación, así como de procesamiento. En este trabajo se considerará que todos los sitios tienen la misma capacidad de procesamiento.

El objetivo es conseguir que la BDD presente un comportamiento "óptimo" ante las operaciones de consulta y actualización que se realicen sobre cada fragmento. Estas operaciones son efectuadas por las aplicaciones que dan soporte a las necesidades del usuario final.

El concepto de "optimo" requiere un exhaustivo y complejo análisis computacional, el cuál derivará en una función de ubicación que, ante cualquier enfoque, presente el mejor desempeño de la BDD. La función que se use para encontrar este modelo debe considerar:

- El costo que implica almacenar R^i en el sitio "j".
- El costo de realizar una consulta al fragmento R^i en el sitio "j".
- El costo de realizar una actualización al fragmento R^i en el sitio "j" (en caso de que se maneje replicación, considerar cada uno de los sitios donde se encuentra replicada).
- El costo de la comunicación entre el sitio "j" y cada uno de los sitios pertenecientes a la BDD.

Por ello, se considera que la ubicación intenta minimizar una función de costo combinado, de la cuál se obtiene un mayor o menor rendimiento, esto es, las respuestas a las consultas de los usuarios se realicen en un menor tiempo. Sin embargo, un costo mínimo no necesariamente implica un mayor rendimiento para una operación particular, debido a que una de las variables puede tener el peor comportamiento para una operación específica, pero las otras presentan el mejor comportamiento, provocando que en conjunto, se obtenga el menor costo.

Representemos a las aplicaciones presentes en la BDD por medio de $Q=\{q_1, q_2,..., q_c\}$ y al costo de comunicación por unidad entre el sitio "*a*" y el sitio "*b*" con t_{ab}, asumiendo que el costo es el mismo tanto para una consulta como para una actualización. De acuerdo con Pelaggati **[3]**, una primera aproximación para determinar la ubicación de R_j en el sitio "*i*" sin considerar replicación y costos de comunicación, está dada por la expresión siguiente:

$$G_j^i = \sum_{k=1}^{c} f^i(q_k) * (r_{kj} + u_{kj})$$

donde:

j: índice del fragmento.
i: índice del sitio.
k: índice de la aplicación.
$f^i(q_k)$: frecuencia de uso de la aplicación *k* en el sitio *i*.
r_{kj}: número de consultas de la aplicación *k* sobre R_j.
u_{kj}: número de actualizaciones de la aplicación *k* sobre R_j.

Esta expresión obtiene la ganancia de ubicar al fragmento R_j en el sitio "*i*". La fórmula es aplicada a los "Y" sitios, ubicándose al fragmento en el sito que presentó la mayor ganancia. Esto es, el sitio ganador S^g se obtiene de:

$$S^g = \max\{G_j^i \mid 1 \le i \le Y\}$$

Por otra parte, se puede optar por minimizar los costos de comunicación así como de almacenamiento de un fragmento en un sitio. Si denotemos con a^i al costo de almacenar un fragmento en el sitio "*i*", S^g se obtendrá de la siguiente expresión:

$$S^g = \min\{t_{ik} * G_j^i + a^i \mid 1 \le i \le Y, 1 \le k \le Y, i \ne k\}$$

En caso de manejarse una BDD con replicación, debemos localizar todos aquellos sitios S^i donde es más barato ubicar una réplica de R_j que importarla o recuperarla de otro sito de la BDD. La siguiente ecuación muestra este hecho:

$$Gr_j^i = \sum_{k=1}^{m} f^i(q_k) r_{ki} - C * \sum_{k=1}^{m} \sum_{\substack{i'=1 \\ i' \ne i}}^{Y} f^{i'}(q_k) u_{ki}$$

En está expresión, "*C*" representa el radio entre el costo de una actualización y el costo de recuperación. Normalmente, los costos de actualización son más caros debido a los mensajes de control que estos involucran, por lo que $C \ge 1$. Todos aquellos sitios "*i*" cuyos valores Gr_j^i resulten positivos, son candidatos a ubicar una réplica. En caso de que todos los valores resulten negativos, se optará por colocar una réplica en el que presentó el valor máximo.

Se podría continuar exponiendo ecuaciones y esquemas que permitan llevar a cabo la ubicación. Sin embargo, por la dependencia que existe entre las variables que intervienen en este proceso, hallar el mejor modelo es sumamente complejo. Por ello, la teoría en BDD's ha optado por señalar las ventajas y desventajas de los modelos de ubicación más estudiados. De estas conclusiones, se obtiene aquel que, comparado con el resto de los modelos existentes, presenta un mejor comportamiento ante las variables que se han establecido como entrada. Notemos que lo anterior no necesariamente implica que los modelos elegidos sean los óptimos, simplemente son los mejores comparados con el resto. En las referencias [3], [31], [32] se puede encontrar más información sobre este aspecto.

En el presente capítulo se han introducido los conceptos generales bajo los cuales se rige una BD y una BDD. Estos conceptos nos han ayudado a entender la estructura de los datos sobre los cuales un algoritmo de Minería de Datos va a tener que operar.

Ahora pasaremos a estudiar los métodos de Minería de Datos más populares que se han desarrollado, con el objetivo de entender su naturaleza y funcionamiento desde la perspectiva original bajo la cuál fueron concebidos, es decir, en un ambiente centralizado o de una sola tabla.

5 MINERIA DE DATOS

En respuesta a las necesidades de análisis de grandes volúmenes de información de forma automática o semi - automática, surgió la Minería de Datos, que da una perspectiva diferente a los métodos estadísticos tradicionales. De hecho, es un campo multidisciplinario, en el cuál los resultados son producto de aplicar diversas técnicas de procesamiento así como de visualización.

En este capítulo se expondrán los conceptos básicos usados en la Minería de Datos. Posteriormente, se dará una perspectiva general de los métodos más usados dentro de esta disciplina.

5.1 DEFINICIÓN

Las BD son capaces de almacenar grandes volúmenes de información, satisfaciendo la necesidades de almacenar datos de manera ordenada. Sin embargo, también es necesario realizar tareas de análisis con el fin de obtener un mayor conocimiento de estos datos, como serían las tendencias, patrones, etc. Dichos análisis se iniciaron principalmente desde la perspectiva estadística [5], donde se obtienen como resultado porcentajes que asocian de alguna manera a los distintos campos de los datos. La Minería de Datos es otra forma de analizar la información, buscando reglas o patrones entendibles y útiles para el usuario. La principal diferencia entre la Minería de Datos y las técnicas estadísticas tradicionales se encuentra en que está última parte de una hipótesis que es construida y posteriormente validada con los datos. En contraste, el manejo del descubrimiento en la Minería de Datos en cierto sentido es llevada a cabo por los mismos patrones y las hipótesis son automáticamente extraídas de los datos. En otras palabras, la Minería de Datos es manejada por los datos, mientras que la estadística es manejada por los humanos.

En general, un sistema que busca patrones en una BD está realizando una inferencia sobre los datos. La forma de inferir la información se realiza de dos maneras: por *deducción* o *inducción* [14].

- ***Deducción:*** En la deducción, el resultado obtenido es una consecuencia lógica de la información de la BD. Por ejemplo, los SMBD ofrecen operadores básicos de deducción de

información, como es el "join", el cuál al operar sobre dos tablas relacionales obtiene una sola tabla basada en las relaciones de ambas (mediante sus campos llave).

- ***Inducción:*** Es una técnica que infiere información basada en la generalización de los datos almacenados en la BD. La información aportada o generada se considera de alto nivel o conocimiento, y se le llama *modelo*.

La diferencia más importante entre ambas es la naturaleza del resultado. En la deducción, los resultados están basados en el mundo real de la BD (datos obtenidos directamente de la información), por lo que son correctos o precisos (por ejemplo, al aplicar el operador join a dos tablas, el resultado es un subconjunto de la información original donde no se deducen o infieren patrones), mientras que los resultados de la inducción se basan en generalidades de la BD caracterizados por los modelos, pero no necesariamente son los más precisos, ya que pueden diferir del comportamiento de la información de la BD. La Minería de Datos se considera como una forma de inferencia inductivo.

Antes de proseguir, es conveniente definir formalmente el concepto de Minería de Datos. Se dice que la Minería de Datos es el proceso de automatizar el descubrimiento de patrones [2]. Esto se refiere principalmente a identificar aquellas tendencias y patrones que la información como tal no proporciona, construyendo modelos que apoyen a su identificación. En ocasiones y dependiendo del método seleccionado, se pueden realizar ciertas predicciones. Otra perspectiva está enfocada a las bases comerciales, donde la Minería de Datos se define como el proceso de extraer información válida que previamente era desconocida, aplicable a grandes BD y que pueda ser usada para realizar decisiones de negocios[7].

El concepto de Minería de Datos, como lo observamos en las definiciones expuestas, varía incluso entre autores. Heikki, por ejemplo, hace mención que el término Minería de Datos se llega a manejar de manera similar a lo que se conoce como Knowledge Discovery in Databases (KDD) [34] o, en otros casos, se considera como parte del proceso de KDD, que consiste desde la recolección de los datos, pre procesamiento de los mismos (eliminando posibles inconsistencias o ruido de los mismos), implementación de una técnica de análisis (algoritmos de Minería de Datos), formato de los resultados, así como la visualización de estos [2].

Cada perspectiva es muy válida. Por ello y para evitar confusiones, en este trabajo se adoptará la siguiente definición:

> **Definición 5: Minería de Datos**
> *La minería de datos es un proceso de análisis y exploración sobre un conjunto de datos de una BD, deseablemente automático, que busca encontrar patrones, predicciones, etc., con un significado válido y aplicable al conjunto de datos que se han explorado y de utilidad para el usuario.*

La exploración o aprendizaje[1] realizada por la Minería de Datos puede ser un proceso semi - automático o automático. Al primero se le denomina "***Minería de Datos dirigida***" (aproximación *top-down*), debido a que la exploración es supervisada por el usuario, quien le indica al sistema donde buscar o explorar la información de la base. Por otro lado, en la "***Minería de Datos no dirigida***" (button-up), el propio sistema busca los patrones en base a propiedades comunes de los datos, es decir, es un proceso totalmente automático en el cuál el usuario no interactúa con el sistema.

La exploración realizada en la Minería de Datos tiene los siguientes elementos en común:

- ***Representación del modelo:*** Es el lenguaje utilizado para describir los patrones por descubrir. Esta representación debe ser lo suficientemente representativa para que se produzca un modelo lo más preciso posible. Un *modelo* se entiende como una simplificación de la naturaleza de los datos.

- ***Evaluación del modelo:*** estima qué tanto concuerdan los patrones encontrados con la realidad de la información. En este punto se realiza una validación de los resultados, buscando que representen lo más fielmente posible situaciones válidas.

- ***Método de búsqueda:*** Aquí se involucran dos factores: la búsqueda de parámetros y la búsqueda del modelo. En la búsqueda de parámetros, el algoritmo debe hallar los parámetros que satisfagan los criterios de evaluación en base a los datos observados o analizados. La búsqueda del modelo es similar a la búsqueda de parámetros: el algoritmo generalmente obtiene una familia de modelos, de los cuales se debe elegir aquellos que se ajusten mejor a los datos. Normalmente se lleva a cabo por medio de un análisis estadístico.

Los puntos descritos anteriormente están encaminados a obtener resultados válidos y confiables. Al respecto, se pide que los resultados cubran los siguientes requisitos:

- La información descubierta previamente debe haber sido desconocida o no haber sido hipotetizada (o considerada).

- La información descubierta debe ser válida. Esto significa que los resultados deben corresponder a una realidad y no caer en situaciones de sobre optimismo (o lo contrario). Esta tarea generalmente es realizada por el usuario o experto que estudia los datos, el cuál en base a su conocimiento y/o experiencia, decide cuales de los patrones obtenidos son válidos.

- Los resultados deben ser útiles, esto es, debe ser posible trasladar está información en algún negocio o acción ventajosa en favor del usuario final.

Existen diversos problemas que se deben enfrentar para obtener resultados que cumplan los puntos anteriores. Estos se exponen en la siguiente sección.

---

[1] El aprendizaje es el proceso en el cuál un sistema adquiere nuevo conocimiento de su entorno o cuando este organiza el actual conocimiento para hacerlo más sencillo de usar[40].

5.2 PROBLEMAS QUE ENFRENTA LA MINERIA DE DATOS

En el proceso de análisis de la información, los métodos basados en la Minería de Datos deben considerar factores que afectan la veracidad de los resultados. Estos se catalogan en dos grupos: problemas propios de la información y los problemas propios del método.

Dentro de los problemas propios de la información o de la BD, encontramos los siguientes [7], [8], [10][14]:

- ***Tipos de datos:*** Los datos pueden ser de distintos tipos, ya sea numéricos, binarios, texto, etc. Si la totalidad de los datos son de un solo tipo, en general los algoritmos no tendrán problema para trabajar con estos (ya que existen algoritmos encaminados para cada tipo de datos). No obstante, las BD's pueden albergar datos de muy diversa índole, representando un problema para la elección del algoritmo, ya que algunos de estos pueden trabajar bien con un subconjunto de los datos, pero no con otro. También se suelen realizar mapeos a datos numéricos no continuos, debido a que estos son más económicos al momento de ser manipulados por un algoritmo, aunque esto no siempre es posible.

- ***Forma de almacenamiento:*** Las BD's pueden estar en distintos formatos de almacenamiento. La forma más comúnmente usada es el esquema relacional. Sin embargo, la mayoría de los algoritmos de Minería de Datos considera que la información se encuentra en una sola tabla (un formato plano), por lo que es necesario en muchas ocasiones realizar vistas globales de la BD.

- ***Ruido:*** El ruido consiste de información falsa que por error humano o del sistema, fue almacenada en la BD y por ende, genera patrones o predicciones imprecisas. También se considera como ruido a los datos replicados.

- ***Valores faltantes:*** Generalmente, y al igual que en el ruido, son provocados por errores humanos al introducir la información en la base. Los valores faltantes conllevan a problemas similares a los expuestos en el ruido, y ambos son atacados por medio de métodos estadísticos, ya sea para verificar la veracidad de los datos o para calcular el valor de un dato faltante.

Por otra parte, los problemas asociados a los métodos se relacionan principalmente a la dimensión que guardan los espacios de búsqueda (la cantidad de datos) en los cuales tienen que operar. Generalmente, entre mayor sea este espacio de búsqueda, las operaciones que realizan los métodos se incrementan, consumiendo mayores recursos y tiempo para su ejecución. Una forma de resolver este problema es restringir al espacio de búsqueda, de tal manera que no afecte a la veracidad de los resultados. Para ello, se busca identificar que tipo de datos son fundamentales para los patrones de aquellos que no lo son tanto, eliminando estos últimos del espacio de búsqueda inicial.

En la minería dirigida, este problema tiene una menor trascendencia, debido a que el espacio de búsqueda inicial es restringido de acuerdo a las consideraciones hechas por el usuario, es decir, ya no se considera al total de la BD.

El problema de la búsqueda de patrones se presenta fuertemente en la minería no dirigida, debido a que los algoritmos por sí mismos deben de explorar todas las posibles combinaciones de los patrones para construir el modelo que buscan. Lo anterior implica que incluso en BD de pequeño tamaño, el número de operaciones a realizar sea extremadamente grande. Por ello, es fundamental contar con buenas estrategias de búsqueda que, al momento de la ejecución, vayan restringiendo el espacio de búsqueda a aquellos datos que se determine son más importantes para el modelo [2], [4][5][7][9][14].

5.3 INTERACCIÓN "MINERÍA DE DATOS" Y "BASES DE DATOS CENTRALIZADAS"

Los métodos de Minería de Datos generalmente toman como entrada a un conjunto de datos almacenados en una BD, aunque esto no es imperativo, ya que podrían existir sistemas que alimenten a los algoritmos por medio de sensores, por ejemplo. El presente trabajo se basa en el hecho que los datos de entrada se encuentran en una BD, tal y como se esquematiza en la Figura 5.1.

Figura 5.1. Proceso esquemático de la Minería de Datos

En el proceso de exploración o aprendizaje, el sistema debe recopilar suficiente información para crear patrones confiables basados en la BD de entrada. En la Minería de Datos, generalmente se realiza un proceso de aprendizaje, en el cuál se extraen los patrones y, un proceso de prueba, el cuál se encamina a verificar la veracidad de los patrones obtenidos. Por ello, la BD de entrada suele dividirse en dos conjuntos: el conjunto de "***entrenamiento***" y el conjunto de "***prueba***".

En BD's relacionales, tanto el conjunto de entrenamiento como el de prueba son un subconjunto de la BD original. Formalmente, el conjunto de entrenamiento se define como (está definición es aplicable al conjunto de prueba):

> **Definición 6. Conjunto de entrenamiento**
> *Sea U un conjunto universo y $\mathcal{R}$ una relación cualquiera sobre U. Un conjunto de entrenamiento $\mathcal{I}$ es una relación no vacía cualquiera, donde $\mathcal{I} \subseteq \mathcal{R}$.*

La definición nos sugiere que pueden existir un gran número de conjuntos de entrenamiento (todas las posibles combinaciones entre las relaciones existentes en la BD). Por ello, es necesario saber como seleccionar aquellos conjuntos que representen de manera proporcional cualquier tendencia que pueda existir en la BD.

Si partimos de la premisa que no contamos con ningún pre – procesamiento que nos indique cualquier patrón existente, la mejor manera de conformar nuestro conjunto de entrenamiento es elegir al azar un subconjunto de la BD original. Por ello, se pueden crear varios pares de conjuntos (de entrenamiento y de prueba) que serán utilizados para buscar el mejor modelo que represente a los patrones. A pesar de esto, no podremos afirmar con certeza que nuestros conjuntos elegidos incluyan cualquier patrón. Por ello, siempre se busca que estos conjuntos conformen una buena muestra (número de tuplas o registros) de la BD original, tratando de disminuir la probabilidad de excluir algún patrón.

5.4 TIPOS DE APRENDIZAJE

De acuerdo a la Figura 5.1, el aprendizaje es la parte medular de la Minería de Datos, que puede estar basada en conceptos muy diversos como: la entropía de la información, la predicción numérica, la asociación, entre muchos otros. Dependiendo de la forma de operación, el aprendizaje puede verse como un proceso claro y entendible como la clasificación, o como un proceso poco comprensible, como se realiza en las Redes Neuronales. Básicamente podemos distinguir dos tipos de algoritmos de Minería de Datos: de clasificación y de agrupamiento. En las siguientes secciones se describirán algunos de los métodos más comunes.

5.4.1 Métodos de clasificación

5.4.1.1 Árboles de clasificación

Es una de las técnicas de aprendizaje más sencillas, pero muy versátil y utilizada. Como su nombre lo indica, aquí se construye un árbol que contiene una serie de pruebas que permiten clasificar un conjunto de datos en clases. Antes de proseguir, definamos el concepto de clase.

En un conjunto de entrenamiento $\mathfrak{I}$ cualquiera, suele presentarse que grupos de registros tienen características comunes entre si, en los cuales los valores de los respectivos atributos coinciden. A estos grupos que cumplen ciertas condiciones se les denomina clases. Formalmente, una clase se define como [14]:

> **Definición 7: Clase**
> *Una clase C_i, es un subconjunto de $\mathfrak{I}$, consistente de todos las tuplas que satisfacen cierta condición $cond_i$ (alguna prueba específica sobre las tuplas), es decir, una clase estará dada por:*
>
> $$C_i = \{x \in \mathfrak{I} \mid cond_i(x) \text{ se cumple}\}$$

Los árboles de clasificación son construidos de tal forma que en cada rama del árbol se asocien exclusivamente registros de una sola clase. La forma de construirlos depende del algoritmo seleccionado, los cuales finalmente determinan la estructura de los árboles.

Un algoritmo muy representativo de los árboles de clasificación es ID3, desarrollado por Quinlan durante la década de los 70's [16], el cuál opera utilizando la *entropía* de los datos. Un ejemplo de un árbol que se construye con este método es el mostrado por la Figura 5.2, donde cada nodo A_x representa a un atributo x y las ramas que emanan de dicho nodo son los valores de su dominio. El objetivo que persiguen estos árboles es agrupar en los nodos de mayor profundidad registros de una misma clase, los cuales deben de cumplir todas las condiciones dictadas por el camino desde la raíz del árbol hasta el nodo donde se encuentran asociados. Los nodos que no tienen un subárbol asociado (los de mayor profundidad) son conocidos como *hojas del árbol*. Analicemos a detalle como trabaja este método.

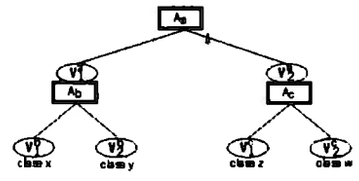


Figura 5.2. Ejemplo de un árbol ID3

El paso inicial consiste en determinar las clases que se usarán como base para la construcción. Estas clases van a provienen del dominio de un atributo "*P*", el cuál es determinado por el usuario. A este atributo se le denomina *atributo clasificador*, el cuál lo denotaremos por A_p, donde existirán β_p clases. Una vez establecido este atributo, se debe de determinar al conjunto de atributos bajo los cuales se va a construir el árbol. A este conjunto de atributos lo representaremos por $\Psi = \{A_1, A_2, \ldots, A_n\}$, donde $A_p \notin \Psi$. Es claro que el conjunto de entrenamiento $\Im$ estará conformado por los atributos de Ψ, donde cada uno de estos atributos A_i podrá tomar un conjunto finito de valores distintos, denominado *dominio del atributo*. Al dominio del atributo A_i lo denotaremos por D_i, y con β_i al número de valores distintos que pertenecen a este dominio.

Con la información anterior, el algoritmo empieza a buscar un atributo que, comparado con el resto de los atributos que se trabajan, demuestre que reduce en mayor grado la entropía de los registros. A éste atributo lo llamaremos atributo ganador A_g, el cuál al ser calculado, pasará a formar parte del árbol de clasificación que se construye como un nodo. En caso que se trate del primer atributo que pasa a formar parte del árbol, éste será su raíz. Notemos que el atributo A_g tiene asociado un dominio D_g con β_g valores distintos. Para agrupar los registros en basa a estos valores distintos, al nodo que se ha creado se le asocia una rama para cada uno de los valores del dominio, con el objetivo de que en cada rama sólo queden registros que coincidan en el valor del atributo A_g. Por notación, a cada uno de estos valores del dominio lo representaremos por medio de v_j^i, el cuál representa al j-ésimo valor en el dominio del atributo i-ésimo, esto es, $v_j^i \in D_i$ y $j \in [1, \ldots, \beta_i]$ donde $\beta_i = |D_i|$. Por medio de $\Im_j^i$ representaremos al grupo de atributos asociados a la rama correspondiente a v_j^i.

Establezcamos como calcular el atributo ganador. Definamos al atributo sobre el cuál trabajaremos por medio de $A_a \mid A_a \in \Psi$ y $A_a \neq A_p$. Este tendrá β_a posibles valores v_j^a ($j = 1, \ldots, \beta_a$), los cuales formarán un igual número de conjuntos $\Im_j^a$, donde los elementos (registros) de

cada uno de estos conjuntos tienen como valor común al valor *j*-ésimo en el campo correspondiente al atributo A_a. En ID3, para calcular el atributo ganador se debe elegir aquél que presente la mayor ganancia. La ganancia es un cálculo que depende directamente de la entropía que cada atributo presenta en relación al atributo clasificador. Este cálculo se realiza de la siguiente manera.

Cada elemento de los conjuntos $\Im_j^a$ tiene una relación directa con el atributo A_p. Esto se debe a que cada registro perteneciente a $\Im_j^a$ tiene asociado una clase v_s^p en su campo correspondiente a A_p. Asociemos a todos los registros de $\Im_j^a$ en subgrupos que coincidan en su clase. Notemos que dependiendo del número de clases distintas v_s^p presentes en los registros será la cantidad de subgrupos que se obtengan (es claro que este valor no será el mismo para cada conjunto $\Im_j^a$). Al número de subgrupos distintos que se obtengan lo representaremos por "*t*", donde $1 < t \leq \beta_p$ para cualquier conjunto, mientras que por medio de $C_{j(s)}^{a(p)}$ representaremos al conjunto de registros que tienen el valor "*j*" del atributo A_a con la clase "*s*" correspondiente al atributo A_p.

Con la información anterior se puede calcular la probabilidad de cada clase basada en el conjunto $\Im_j^a$, relacionando la cantidad de elementos de cada conjunto $C_{j(s)}^{a(p)}$ entre el número de registros del conjunto $\Im_j^a$. Esta probabilidad queda expresada por:

$$\Pr\left(C_{j(s)}^{a(p)}\right) = \frac{|C_{j(s)}^{a(p)}|}{|\Im_j^a|}$$

La expresión anterior nos proporciona la probabilidad de la clase "*s*" correspondiente al atributo A_p en relación a los registros con el *j*-ésimo valor del atributo A_a.

Así como se ha expuesto la probabilidad de cada clase en cada grupo, se puede obtener la probabilidad de cada grupo $\Im_j^a$ en relación al total de registros que se estén manejando en el proceso, es decir, $\Im$. Esta probabilidad se expresa como:

$$\Pr\left(\Im_j^a\right) = \frac{|\Im_j^a|}{|\Im|}$$

Con las probabilidades que se han presentado, se puede definir la expresión que permite calcular la entropía del atributo A_a dada por:

$$E(A_a) = \sum_{j=1}^{\beta_a} \Pr(\Im_j^a) * I(\Pr(C_{j(1)}^{a(p)}), \Pr(C_{j(2)}^{a(p)}), \ldots, \Pr(C_{j(t)}^{a(p)}))$$

donde:

$$I(\Pr(C_{j(1)}^{a(p)}), \Pr(C_{j(2)}^{a(p)}), \ldots, \Pr(C_{j(t)}^{a(p)})) = -\sum_{s=1}^{t} \Pr(C_{j(s)}^{a(p)}) * \log_2 \Pr(C_{j(s)}^{a(p)})$$

La última expresión es conocida como el grado de información que un atributo A_a aporta con respecto al atributo A_p.

Haciendo uso de las expresiones ya expuestas, se puede obtener la ganancia de seleccionar el atributo A_a por medio de:

$$G(A_a) = I(\Pr(\Im_1^p), \Pr(\Im_2^p), \ldots, \Pr(\Im_{\beta_p}^p)) - E(A_a)$$

La expresión anterior se aplica para cada atributo en Ψ, obteniendo un total de $|\Psi|$ ganancias. Por tanto, obtendremos finalmente un conjunto de ganancias, de la cuál debemos obtener la de mayor valor. Lo anterior indica que el atributo ganador A_g se obtendrá de la expresión siguiente:

$$A_g = \max\{G(A_x) | \forall A_x \in \Psi\}$$

Una vez calculado A_g, ID3 lo añade en forma de nodo al árbol que se construye. De forma recursiva, para cada una de las ramas que éste nodo ha formado se aplica el mismo método, eliminando el atributo A_g del conjunto Ψ. Cuando una rama es analizada y se determina que el conjunto de registros asociada a ella es de una sola clase, el proceso de crecimiento de está rama termina, rematándose a la misma con la clase encontrada y continuando el crecimiento de otra rama hermana. Por tanto, ID3 no termina hasta que no se haya desarrollado cada una de las ramas y todas hayan concluido en un nodo donde los registros son de una sola clase. En algunos casos no es necesario que la totalidad de registros de una rama sea puro para detener el proceso de crecimiento de la misma, ya que se pueden implementar ciertos criterios de tolerancia, es decir, se remata a una rama con la clase que presente la mayor probabilidad, la cuál sobrepasa un valor establecido. Lo anterior contribuye a evitar problemas de crecimiento en el árbol, conocidos como *sobre – ajuste*, en donde por más que se desarrolle una rama, ya no es posible obtener grupos de atributos con una sola clase.

Un aspecto sobre la ganancia que se ha definido es el hecho de favorecer a aquellos atributos donde la cardinalidad del dominio del atributo es cercano al número de atributos totales de $\Im$. Una solución a este problema es dar una proporción a cada ganancia que se obtiene con respecto al número de atributos que se están manejando en ese momento. Para ello, en trabajos posteriores, Quinlan sugirió manejar la siguiente expresión:

$$G'(A_a) = \frac{I(P(\Im_1^p), P(\Im_2^p), \ldots, P(\Im_{\beta_p}^p)) - E(A_a)}{I(P(\Im_1^a), P(\Im_2^a), \ldots P(\Im_{\beta_a}^a))}$$

Notemos que lo anterior no afecta en ninguna manera al proceso de construcción que ya se ha descrito, simplemente se ha modificado la forma de calcular la ganancia de cada atributo. Por tanto y basándonos en los conceptos ya descritos, el pseudocódigo del algoritmo ID3 [11], [15] se ilustra en el Algoritmo 1.

Function ID3 (Ψ: conjunto de atributos $\neq$ de A_p,
s: valor de default,
$\Im$: conjunto de entrenamiento)
begin
Si $\Im$ es vacío
return un nodo simple con un valor de error
Si $\Im$ consiste de registros con la misma clasificación
return un nodo con la clasificación
Si Ψ es vacío
return un nodo con el valor mayoritario de clasificación de $\Im$.
Sea A_α el atributo con mayor ganancia G'(A_α).
Crea un nuevo árbol δ con A_α como raíz.
Para cada v_j^a hacer:
$\Im_j$: conjunto de registros correspondientes a v_j^a
Crea un subárbol δ' de **ID3**(Ψ - {A_α},
Valor mayoritario de clasificación de $\Im_j$,
$\Im_j$)
Asigna a δ' como hijo de δ correspondiente a v_i^a

Algoritmo 1. Inductive Desicion Tree (ID3)

Un ejemplo de un árbol ID3 es el que se muestra a continuación, tomando como base los datos de la Tabla 2 con "clase" como atributo clasificador:

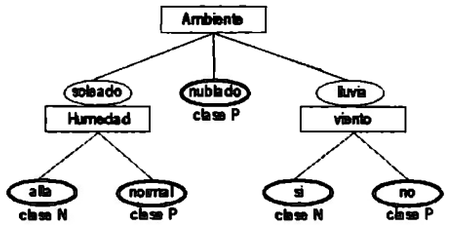


Figura 5.3. Árbol ID3 correspondiente a la Tabla 2.

En está figura aparecen remarcados en negro los nodos de mayor profundidad de cada rama, denominados hojas. Notemos que las hojas tienen una clase asociada, y está es la clase que se asignará a los registros que se asocien a una hoja en particular. Por ejemplo, supongamos un registro que en el atributo *Ambiente* tiene el valor *soleado* y en el atributo *Humedad* tiene el valor *normal*. Al aplicarse el árbol de la Figura 5.3, este registro cumple el camino *Ambiente – soleado – Humedad – normal*, con lo cuál el registro se clasifica con la clase P.

Este algoritmo ha gozado de una gran aceptación por lo simple y potente que es. Por ello, se volvió objeto de un gran estudio que ha derivado en algunas variantes del mismo. Ejemplo de ello es el algoritmo C4.5 desarrollado igualmente por Quinlan [9], [33] donde se introducen técnicas de podado, de sustitución de valores, de predicción de valores faltantes, manejo de valores continuos, entre otras mejoras. Otros algoritmo relacionado es ***Classification and Regresión Trees*** (CART) [9], [33], desarrollado por Leo Breiman, Jeorme Friedman, Richard Losen, y

Charles Stone. Este método añade ventajas sobre el ID3, funcionando bajo jerarquías y predictores. Las jerarquías permiten establecer una relación entre valores que no tienen relación alguna, permitiendo construir árboles que comparen a estos valores de forma directa. Por otra parte, los predictores funcionan bajo técnicas de regresión, permitiendo manejar el problema de valores faltantes, por ejemplo.

Los diversos algoritmos de árboles de clasificación básicamente difieren en la estructura que estos le dan a cada árbol. Sin embargo, la estructura de los árboles continua siendo muy similar.

Existen otros algoritmos que construyen árboles de clasificación, sólo que estos operan exclusivamente sobre valores numéricos. Estos árboles son conocidos como *árboles de decisión*, los cuales son descritos en la siguiente sección.

5.4.1.2 **Árboles de decisión**

Los árboles de decisión se enfocan principalmente a trabajar con valores numéricos. Cada nodo de estos árboles constituye una prueba de comparación sobre el valor numérico de un atributo, utilizando los operadores de orden ($<$, $\leq$, $=$, $>$, $\geq$). Al igual que en los árboles de clasificación, el camino desde la raíz hasta la hoja de un árbol forman las condiciones necesarias para que un registro pertenezca a la clase que dicta la hoja. Generalmente, cuando se trabaja con operadores de orden se construyen árboles binarios, sin embargo, se pueden establecer rangos para las pruebas que se definen para cada nodo, permitiendo formar nodos con más de dos hijos. Un ejemplo de un árbol de decisión es el mostrado por la Figura 5.4.

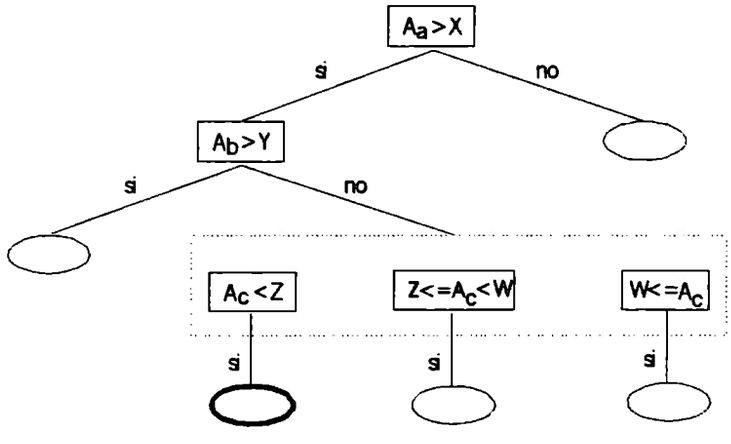


Figura 5.4. Estructura básica de un árbol de decisión.

Este árbol ilustra como algunos nodos como la raíz tiene la estructura de un nodo binario, mientras que el nodo encerrado en el cuadro punteado establece pruebas por intervalos.

En los métodos basados en los árboles de decisión es posible implementar técnicas estadísticas que permitan identificar problemas de ruido o solucionar problemas de valores faltantes. Por otra parte, si se trabaja con valores nominales, lo ideal es establecer una relación de éstos a valores numéricos. Si lo anterior no es posible, se puede emplear directamente el valor nominal, observando que lo anterior genera tantas ramas como elementos en el dominio del atributo existan.

5.4.1.3 **Reglas de clasificación**

Una estructura que es muy entendible para el usuario final son las ***reglas de clasificación***. Las reglas de clasificación tienen la estructura *antecedente* → *consecuente*, donde el antecedente está conformado por una o varias pruebas sobre los atributos, relacionadas entre sí por el conectivo lógico ∧. Por otra parte, el consecuente representa la clase que cumple las condiciones establecidas en el antecedente. Las reglas de clasificación pueden ser generadas a partir de los árboles de decisión o clasificación. Para lograrlo, se siguen todos los posibles caminos a cada hoja del árbol, formando tantas reglas como hojas en el árbol existan. Esta forma de derivar reglas es muy eficiente, ya que si el árbol del que se derivan fue construido correctamente, el conjunto de reglas será óptimo, es decir, no será redundante (reglas innecesarias).

Sin embargo, no es necesario contar con un árbol como premisa para construir este tipo de reglas. Uno de los algoritmos más simples para generar reglas de clasificación es el llamado "**1 - rule**" [2], en el cuál, las reglas se obtienen de seleccionar aquel atributo o conjunto de atributos que presentan un menor error acumulado.

Los errores acumulados se calculan de la siguiente manera: considere que se tienen "*n*" atributos en la relación o tabla, donde A_a representa al atributo que deseamos obtener su error acumulado y A_p representa al atributo clasificador. Recordemos que la probabilidad basada en la clase "*c*" del atributo clasificador A_p en el conjunto de registros $\Im_j^a$ (registros con el valor j en el campo correspondiente al atributo A_a) se representa por $P(C_{j(c)}^{a(p)})$, por lo que su error se expresa por 1 - $P(C_{j(c)}^{a(p)})$. El objetivo de este método es seleccionar aquellas clases "*c*" que presentan el menor error en cada conjunto de registros $\Im_j^a$ derivados del atributo A_a. Representemos por medio de c^x a la clase con menor error 1 - $P(C_{j(c^x)}^{a(p)})$ de la totalidad de clases presentes en $\Im_j^a$. La suma de los errores de cada clase c^x presentes en cada conjunto $\Im_j^a$ para j = 1, ..., β_a representa al error acumulado del atributo A_a. Este error se expresa de la siguiente manera:

$$E(A_a) = \sum_{k=1}^{\beta_a} 1 - P(C_{k(c^x)}^{a(p)})$$

Para cada $A_x \in \Re$, $1 \le x \le n$, hacer:
  Para cada v_i^x, $1 \le i \le \beta_x$ hacer:
  - Obtener la frecuencia de cada clase que aparece asociado a cada v_i^x.
  - Seleccionar la clase más frecuente.
  - Formar una regla asociada a la clase de la forma $A_x = v_i^x \rightarrow$ clase

  Calcular el radio de error de las reglas obtenidas

Algoritmo 2. Algoritmo 1-Rule

Como se tienen $|\Psi|$ atributos, se elige aquel atributo que presente el menor error acumulado, representado por A_g. Una vez calculado este atributo, las reglas de clasificación se forman de la siguiente forma:

$$A_g = v_j^g \rightarrow c^x \text{ para } j = 1, \ldots, \beta_g \text{ y } x \text{ no necesariamente igual}$$

Las reglas que se generan sólo contienen una prueba en el antecedente. Ello provoca que los errores asociados a las reglas tiendan a ser grandes, provocando que estas no sean muy precisas. El proceso descrito queda expresado por el Algoritmo 2.

Por ejemplo, basándonos en los datos de la Tabla 2, los resultados parciales de ejecutar este algoritmo son:

ATRIBUTO	VALORES DEL ATRIBUTO						ERROR
Ambiente	Soleado		nublado		Lluvia		4/14
	P	N	P	N	P	N	
	2	3	4	0	3	2	
	Si ambiente = soleado→N Error: 2/5		Si Ambiente = nublado → P Error: 0/4		Si Ambiente = lluvia → P Error: 2/5		
Temperatura	Alta		media		Baja		5/14
	P	N	P	N	P	N	
	2	2	4	2	3	1	
	Si Temperatura = alta → P Error: 2/4		Si Temperatura = media → P Error: 2/6		Si Temperatura = baja→P Error: ¼		
Humedad	Alta		Normal				4/14
	P	N	P	N			
	3	4	6	1			
	Si Humedad = alta → N Error: 3/7		Si Humedad = normal → P Error: 1/7				
Viento	Si		No				5/14
	P	N	P	N			
	3	3	6	2			
	Si Viento = si → P Error: 3/6		Si Viento = no → P Error: 2/8				

En base a lo anterior, las reglas resultantes son las siguientes:

Si ambiente = soleado→N
Si Ambiente = nublado → P
Si Ambiente = lluvia → P
Si Humedad = alta → N

Si Humedad = normal → P

Otro algoritmo que genera reglas de clasificación, pero en base a una clase en particular es **PRISM.** En este algoritmo, se selecciona en primera instancia la clase c^x deseada, así como todos los registros de está clase, denotada por $\Im^c$. Se crea una regla de la forma ? → c, donde ? será sustituido por el conjunto de pruebas que se determinarán en el proceso de construcción.

La idea del algoritmo es calcular la probabilidad de cada valor del dominio de cada atributo, seleccionando aquél valor que presente la mayor probabilidad. Acto seguido, dicho valor con su correspondiente atributo pasa a formar parte del antecedente de la regla que se construye. En caso que ya se hayan clasificado correctamente todos los registros de la clase buscada, el proceso termina. De no ser así, el proceso se repite trabajando ahora sobre las instancias que no fueron clasificadas en el proceso anterior. En caso de terminar con el conjunto de atributos, pero sin haber abarcado el total de las instancias originales que son de la clase buscada, se eliminan de este conjunto original las instancias que ya se han identificado correctamente y el proceso vuelve a iniciar desde el principio. Lo anterior queda expresado formalmente en el Algoritmo 3.

ENTRADA
$\Im^c$: conjunto de atributos con la clase c.
c: clase buscada, donde c ∈ A_p
Ψ: Conjunto de atributos
A: Conjunto que almacena la estructura de el antecedente de cada regla.
SALIDA
Conjunto de reglas que identifican las características de los registros de entrada.

1. Se hace $\Im' \leftarrow \Im^c$, $\Psi \leftarrow \Psi - \{A_p\}$, $A \leftarrow \emptyset$.
2. $\forall A_a \in \Psi$ calcular: $P(\Im^{a(p)}_{j(c)}) = \dfrac{|\Im^{a(p)}_{j(s)}|}{|\Im^{a}_{j}|}$, $j = 1,\ldots, \beta_a$
3. Se selecciona el ***max*** $\{P(\Im^{a(p)}_{j(s)}), j = 1, \ldots, \beta_a\}$. Supongamos que el valor máximo correspondió al atributo A_x con el valor v^x_y.
4. Se realizan las operaciones siguientes:
   $A \leftarrow A \cup \{A_x = v^x_y\}$, $\Psi \leftarrow \Psi - \{A_x\}$
5. Si $\Gamma = \phi$ hacer:
   Si $\forall d_x \in \delta \mid d_x$ cumple $A \rightarrow s$, se concluye con la regla $A \rightarrow s$
   De otra manera, se da como resultado parcial $A \rightarrow s$.

Algoritmo 3. PRISM

Es importante señalar que en caso de que dos o más probabilidades tengan el mismo valor, se suele seleccionar aquella probabilidad que tiene una mayor cobertura, es decir, la que considere un mayor número de registros.

Por ejemplo, supongamos que nos basamos en los datos de la Tabla 1, en la cuál un usuario desea obtener reglas que clasifiquen a los registros de la clase "hard". En primea instancia, el método construye una regla de la forma:

Si ? → hard

Una corrida bajo las condiciones planteadas queda de la siguiente forma:

REGLA DE ENTRADA	PRUEBAS	ERROR
Si ? → hard	Age = young	2/8
	Age = pre - presbyopic	1/8
	Age = presbyopic	1/8
	Spectacle prescription = myope	3/12
	Spectacle prescription = hypermetrope	1/12
	Astigmatism = yes	**4/12**
	Astigmatism = no	0/12
	Tear production = normal	4/12
	Tear production = reduce	0/12
Si astigmatism = yes y ? → hard	Age = young	2/4
	Age = pre – presbyopic	¼
	Age = presbyopic	¼
	Prescription = myope	3/6
	Prescription = hypermetrope	1/6
	Tear production = reduce	0/6
	Tear production = normal	**4/6**
Si astigmatism = yes y tear prduction = normal y ? → hard	Age = young	2/2
	Age = pre – presbyopic	½
	Age = presbyopic	½
	Prescription = myope	**3/3**
	Prescription = hypermetrope	1/3

Del proceso anterior deducimos la regla siguiente:

Si astigmatism = yes y tear prduction = normal y prescription = myope → hard

Como en la regla anterior solo se consideran 3 de los 4 registros de clase "hard", se repite el proceso eliminando los registros ya clasificados de $\mathfrak{I}^{hard}$ hasta que la totalidad de los registros sean clasificados. En la segunda iteración ya se clasifica el atributo faltante, con lo cuál el resultado final es el siguiente:

Si astigmabtism = yes y tear prduction = normal y prescription = myope → hard
Si age = young y astigmatism = yes y tear production = normal → hard

Este método tiene la ventaja de permitir al usuario construir reglas para una clase muy específica, pero en caso de que se desee obtener reglas para varios tipos de clases, se debe ejecutar el método para cada una de ellas.

Existen otros tipos de reglas denominadas ***reglas con excepciones***, las cuales permiten considerar valores "especiales" que se salen del patrón marcado por la mayoría de los registros. Esto solo es posible realizarse cuando se conocen de manera explícita estas excepciones a la regla. Para crear reglas con excepciones, se puede utilizar primeramente cualquier otro método que genere reglas de clasificación. En un proceso posterior, se ordenan las reglas integrando una consideración que nos permita excluir aquellos registros que cumplan la regla que se está considerando pero además, cumpla la excepción. La estructura de éstas reglas se ilustra a continuación.

1. *Si* $D_k = x$ [...$\wedge D_l = y$] $\rightarrow$ *clase i (concecuente)*
2. *Excepto Si* $D_g = z$ [...$\wedge D_h = w$] $\rightarrow$ *clase i (concecuente)*
3. *Excepto Si ...*

Es importante señalar que las reglas deben de aplicarse de manera secuencial, es decir, para que un conjunto de registros verifique la primera excepción que aparece en el recuadro anterior (línea 2), debe primero cumplir el conjunto de antecedentes expresados en la línea 1. La excepción que aparece en la línea 3 solo se verifica sobre aquellas tuplas que cumplieron con las condiciones de las líneas 1 y 2 y así sucesivamente.

5.4.1.4 **Tablas de decisión**

Una de las primeras formas en las cuales se realizó la Minería de Datos fue mediante las ***tablas de decisión*** [9]. El método es simple: consiste en reunir en una sola tabla el conjunto de instancias totales de la BD, para buscar las condiciones necesarias con las cuales identificar una clase. Estas condiciones eran representadas generalmente por reglas de la forma *antecedente* $\rightarrow$ *consecuente*, que es la estructura utilizada por las reglas de clasificación. Esta tarea era rudimentaria, debido a que no tenía un proceso formal para realizarlo, por lo cuál cayó en desuso.

5.4.1.5 **Métodos basados en la estadística**

Otro técnica utilizada por la Minería de Datos es el *modelo estadístico*. Aquí se hacen uso de los conceptos matemáticos de las probabilidades. Una propiedad importante de la probabilidad de un evento es la siguiente: la probabilidad de un evento A, representado por Pr(A), cumple que $0 \leq \Pr(A) \leq 1$ (en [11] se pueden consultar propiedades adicionales). Un concepto importante dentro de la estadística es la *verosimilitud*. La verosimilitud de un evento A es obtenida de multiplicar las probabilidades independientes de los eventos A_i que la conforman, donde $1 \leq i \leq n$. Formalmente la verosimilitud del evento A la expresamos por:

$$\Pr_V(A) = \prod_{i=1}^{n} \Pr(A_i)$$

Si pensamos que existen "*m*" eventos (caso general), donde cada evento lo denotamos por medio de A^t con $0 < t \leq m$, la probabilidad del evento A^k en base a las verosimilitudes de cada uno de los eventos es:

$$\Pr(A^k) = \frac{\Pr_v(A^k)}{\sum_{t=1}^{m} \Pr_v(A^t)}$$

Otro modelo estadístico utilizado es la *probabilidad condicional de Bayes*, que se obtiene de calcular la probabilidad de un evento H dado un evento E. Esta probabilidad se expresa en la siguiente ecuación:

$$\Pr[H \mid E] = \frac{\Pr[E \mid H] * \Pr[H]}{\Pr[E]}$$

donde $\Pr[H \mid E]$ represente a la probabilidad del evento H dado el evento E.

En el caso general, no solo se considera un evento E, sino un conjunto de ellos. Por ello, si consideramos un conjunto de eventos E_1, E_2, ..., E_n, la probabilidad condicional de un evento H dado estos eventos se representará por medio de:

$$\Pr[H \mid E] = \frac{\Pr[E_1 \mid H] * \Pr[E_2 \mid H] * \cdots * \Pr[E_x \mid H] * \Pr[H]}{\Pr[E]}$$

Cabe hacer mención que está expresión es utilizada para encontrar la probabilidad condicional de eventos con valores nominales. En caso de manejar valores numéricos, las probabilidades de los eventos se podrán obtener de la siguiente manera:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

donde:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}, \quad \sigma = \frac{\sum_{i=1}^{n} (\mu - x_i)^2}{n-1}$$

y x_i es el valor del atributo del cuál deseamos obtener su probabilidad condicional.

De la probabilidad condicional de Bayes se derivó el *aprendizaje en redes Bayesianas* [11] que, en general, es un grafo donde los nodos representan una condición para un atributo en particular (un evento) y los arcos representan la dependencia probabilística entre los eventos.

Tanto el modelo estadístico como la probabilidad condicional de Bayes han sido ampliamente utilizadas en procesos de clasificación. Sin embargo, el usuario final debe de saber interpretar correctamente la probabilidad dada, ya que, al final de cuentas, solo se obtiene como resultado un número en el intervalo [0, 1].

5.4.1.6 **Redes Neuronales Artificiales**

El primer trabajo formal sobre redes neuronales (RNA) data de 1943 [22], [23], cuando Warren McCulloch y Walter Pitts propusieron un modelo basado en el concepto de neurona. Estos trabajos se basaron en la idea de simular el complejo funcionamiento del cerebro humano a partir del elemento básico que lo compone: la neurona. Lo anterior no ha sido sencillo, ya que el número de neuronas, 100,000 millones aproximadamente, hacen sumamente compleja su simulación [22].

Desde la contribución de McCulloch y Pitts, se han desarrollado una gran cantidad de trabajos [22][23][24][25][26][27][28][30]. Éstos, dependiendo de su naturaleza, se agrupan en dos modelos básicos [22]: las "*redes biológicas*", que buscan simular al cerebro y las redes "*dirigidas a la aplicación*", donde el problema dicta la estructura que deberá adoptar la red.

Ambos modelos están compuestos de elementos básicos llamadas *neuronas artificiales*, a las cuales se les ha tratado de dar un funcionamiento similar a las neuronas biológicas. Las neuronas biológicas son células que funcionan por medio de impulsos eléctricos (estímulos), los cuales son utilizados para comunicarse con neuronas vecinas. Estos impulsos viajan por medio del axón de la neurona hasta pequeños bulbos que casi se encuentran en contacto con las dentritas (finos filamentos que rodean a cualquier neurona) de neuronas vecinas. Dependiendo de la cantidad de corriente que una neurona reciba, determinará si está transmite cierta corriente por medio de su axón. Si la neurona transmite corriente, se dice que recibió una señal excitatoria, de lo contrario, la señal recibida se denomina inhibitoria [28], [24]. Lo anterior es lo que se busca simular en las RNA.

Una RNA la podemos ver como se ilustra en la Figura 5.5.

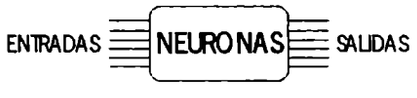


Figura 5.5. Estructura básica de una RNA.

A través de las entradas, la RNA es alimentada. Estos datos de entrada son entonces procesador por el conjunto de neuronas que constituyen a la red, las cuales finalmente dan un resultado por medio de las salidas.

Las neuronas artificiales tienen una estructura simple: un conjunto de entradas, un núcleo donde se calcula la salida y la salida, así como se ilustra en la Figura 5.6.

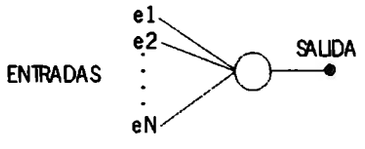


Figura 5.6. Estructura básica de una neurona

Estas neuronas generalmente cuentan con un sumador lineal que toma los valores recibidos por las entradas de la neurona. El resultado de está suma es pasada a una función no lineal, llamada *función de activación*. Esta función determina si las señales recibidas son lo suficientemente fuertes para activar a la neurona, transmitiendo unas señal por la salida. Las RNA's emplean muy diversas funciones de activación (polinomiales, sigmoidales, etc), dependiendo de la aplicación a la cuál se destine la red. Una vez establecidas las funciones, estas no varían. La salida de la neurona, a su vez sirve como valor de entrada a otras neuronas, estableciéndose así un alto grado de conexión entre un gran número de neuronas. Por otra parte, al valor de salida de la neurona se conoce como *peso de la neurona*.

Los pesos de las neuronas finalmente son las que dictan el comportamiento de una RNA, los cuales son ajustados en un proceso de aprendizaje. Existen dos tipos de aprendizaje [26]: el *supervisado* y el *no supervisado*. En el aprendizaje supervisado, se le proporcionan los datos correctos de entrada y salida a una RNA, buscando que ajuste sus pesos de tal manera que coincidan sus respuestas lo más posible con los valores de salida proporcionados. En el aprendizaje no supervisado, a la RNA solo se le proporcionan los datos de entrada, y en base a estos, la red ajusta sus pesos. Una vez calculados la totalidad de los pesos de las neuronas, la red refleja lo aprendido mapeando la salida correspondiente a los datos de entrada.

Las neuronas que integran a una RNA son agrupadas en capas. Dependiendo de la forma de conexión de las capas, se determina que arquitectura tiene una RNA. Actualmente, las arquitecturas de las RNA generalmente se clasifican en [25], [24]:

- *Redes de capa simple con búsqueda hacia delante*: En este esquema solo se cuenta con una capa de entrada y otra capa de salida, fluyendo la información de atrás hacia adelante.

- *Redes multicapa con búsqueda hacia adelante*: Este tipo de RNA está compuesto por varias capas de neuronas, donde una capa "*s-1*" estimula a la capa "*s*". Existe una capa de salida, la cuál muestra el resultado final, siendo una capa visible, mientras que el resto de las capas se denominan capas ocultas o prohibidas. El flujo de información es de atrás hacia adelante. Se han desarrollado muy diversas redes multicapa, las cuales han sido empleadas con éxito en el reconocimiento y tratamiento de idiomas, predicción de datos en el tiempo (por ejemplo, en la demanda de gas y electricidad), variaciones en el sector financiero, etc. También encontramos las llamadas redes de "*backpropagation*", donde las neuronas de cada capa no se interconectan entre si; solo existe interconexión entre neuronas de diferentes capas, en la cuál, una neurona proporciona una entrada a todas las neuronas de la siguiente capa [26], [28].

- *Redes recurrentes*: Caracterizadas por tener "alimentación hacia atrás" (feedback), en el cuál las salidas sirven de entrada (retroalimentación). Este tipo de redes suelen ser más complejas que los otros dos modelos. Uno de los trabajos más relevantes son las redes de "*Hopfield*", las cuales reconstruyen los patrones de entrada que aprendieron durante el aprendizaje. La interconexión es total (no existen restricciones). Hopfield introdujo conceptos que hicieron a este tipo de redes estables (debido a la recurrencia, se volvían redes inestables). Se dice que una red es estable cuando se mantienen los valores de excitación ante nuevos estímulos. Si es

el caso, la red busca una configuración parecida a los patrones ya aprendidos, siendo el resultado que presenta la red ante el patrón de entrada [26], [27].

Las RNA's se han empleado con éxito en actividades como la clasificación, asociación y reconocimiento de patrones. Dentro de la clasificación, las RNA son entrenadas para identificar las características de cada grupo que se desea clasificar. Una vez entrenada la RNA, el usuario sólo introduce los datos de entrada y la RNA le proporcionará su clasificación. Sin embargo, la principal desventaja es que el usuario no tiene la certeza de cómo se clasifica su muestra, ya que generalmente es un proceso complejo de seguir. Además, el diseño de las RNA's es complejo y una vez construidas, modificarlas implica un gran esfuerzo.

5.4.2 Métodos de agrupamiento (Clustering)

5.4.2.1 Métodos basados en el concepto de distancia entre vecinos

En estos métodos, los registros de una relación son agrupados por medio de alguna medida basada en el concepto de distancia [33]. Generalmente se hace uso de la media, la moda, etc., para establecer la distancia. Lo anterior nos induce a trabajar exclusivamente con valores numéricos pero, al igual que en otros casos, se puede realizar un mapeado de valores nominales a numéricos. Desde está perspectiva, se trabaja sobre un espacio de dimensión *n*, donde *n* es menor o igual al número total de atributos de la relación.

En este espacio serán ubicadas todas las tuplas de $\Re$, identificadas por su posición mediante un vector $<x_1, ..., x_n>$, que es obtenido de los valores correspondientes de cada registro. Existen dos formas de llevar acabo la agrupación[16]:

- ***Métodos de división***: En este tipo de estrategia se puede pensar que se parte de un gran conjunto que contiene a todos los registros y que, en pasos sucesivos, será dividido en dos o más pequeños conjuntos o clusters.

- ***Métodos aglomerativos***: Por el contrario, estos métodos parten del hecho que cada registro forma un grupo, y el objetivo es crear un menor número de grupos, donde cada uno de ellos este conformado por un número mayor de registros, buscando formar un número de clusters menor al original, agrupados en base a sus características comunes.

Los métodos más utilizados son los de división, donde el algoritmo k-medias (*k – means)* goza de una amplia aceptación. Este algoritmo se basa en la fórmula general de la distancia euclidiana entre dos puntos (X y Y), dada por la siguiente ecuación:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

donde $X = <x_1, ..., x_n>$ y $Y = <y_1, ..., y_n>$, y ambos son dos registros cualesquiera distintos entre sí.

El algoritmo *k - medias* requiere que se especifique *k*, que es el número de grupos en que se van a concentrar las instancias. *k* es un número arbitrario que dependerá del criterio del usuario. La forma de operar de este método se muestra en el Algoritmo 4.

1. El proceso inicia eligiendo "*k*" registros llamados semillas, los cuales son seleccionados al azar del conjunto inicial de registros.
2. Se obtiene la distancia de cada registro a cada una de las semillas.
3. Cada registro va a ser asociado a aquella semilla con la cuál presentó la menor distancia, formándose "*k*" grupos de registros. En caso de que ningún registro cambie de grupo, el proceso termina.
4. Para cada grupo de registros se obtiene el punto central o medio. Este es calculado por la media entre cada uno de los componentes correspondientes de cada vector $<x_1, ..., x_n>$ de los registros que integran a cada grupo.
5. Estos puntos se toman como las nuevas semillas, repitiéndose el proceso desde el punto 2.

Algoritmo 4. K medias ("K means").

El algoritmo *k - medias* es simple y efectivo a la vez, pero es propenso al ruido. Cuando se construye un cluster, este agrupa a aquellos registros que se encuentren más cercanos entre sí, pero, también puede incluir registros que no se encuentren cerca de la mayor concentración (son incluidos por el hecho de encontrarse más cerca del cluster en cuestión que de otros clusters). Por ello, algunos métodos consideran la desviación estándar, ayudando a determinar cuando un registro pertenece o no a un cluster determinado. Estos métodos se basan en la probabilidad y estadística, donde podemos encontrar, a EM [7], por ejemplo. Otra estrategia considera la densidad por regiones en el espacio, como lo hace DBSCAN [18], en el cuál un grupo será formado de acuerdo a la cantidad de registros próximos entre ellos. Esta última técnica es más eficiente que *k-medias*, debido a que puede encontrar un cluster con mayor precisión, disminuyendo considerablemente el factor de ruido.

5.4.2.2 **Reglas de asociación**

Este método se basa en la idea de asociar registros que coincidan en los valores correspondientes a los campos de los atributos. El número de campos que deben de coincidir para formar las reglas es variable, por lo que se pueden formar reglas de 1 condición, 2 condiciones, hasta *n* condiciones, donde *n* es el número de atributos de la relación $\Re$ considerada.

Es importante señalar que debido a que se puede considerar cualquier combinación de registros, el número de reglas de asociación que se pueden obtener incluso en BD's pequeñas es grande. Por ello, una de las estrategias que generalmente se ha tomado es establecer un mínimo de cobertura para las reglas. Esto es, el usuario debe definir por lo menos cuantos registros deben estar asociados a una regla cualquiera, y en caso de no cumplir con está consideración, la regla es descartada.

Expongamos un ejemplo de reglas de asociación. Supongamos que deseamos construir un conjunto de reglas basadas en los datos de la Tabla 2, considerando que las reglas deben de cumplir una cobertura mínima de 2 registros. En la tabla siguiente se muestran algunas de las reglas generadas hasta 3 condiciones [9].

REGLAS DE 1 CONDICION	REGLAS DE 2 CONDICIONES	REGLAS DE 3 CONDICIONES
Ambiente = soleado (5)	Ambiente = soleado y Temperatura = media (2)	Ambiente = soleado y Temperatura = alta y Humedad = alta (2)
Ambiente = nublado (4)	Ambiente = soleado y Temperatura = alta (2)	Ambiente = soleado y Temperatura = alta y Clase = N (2)
Ambiente = lluvia (5)	Ambiente = soleado y Humedad = normal (2)	Ambiente = soleado y Humedad = normal y Clase = P (2)
Temperatura = baja (4)	Ambiente = soleado y Humedad = alta (3)	Ambiente = soleado y Humedad = alta y Viento = no (2)
Temperatura = media (6)	Ambiente = soleado y Viento = si (2)	Ambiente = soleado y Humedad = alta y Clase = N (3)
Temperatura = alta (4)	Ambiente = soleado y Viento = no (3)	Ambiente = soleado y Viento = no y Clase = N (2)
Humedad = normal	Ambiente = soleado y Clase = P (2)	Ambiente = nublado y Temperatura = alta y Clase = P (2)
⋮	⋮	⋮

El ejemplo anterior da una idea clara de cómo las reglas de asociación sirven para identificar registros que coinciden en diversos valores correspondientes a los atributos que los integran.

Una forma efectiva de obtener un conjunto de reglas de clasificación, donde este conjunto sea mínimo, es decir, no contenga reglas innecesarias, es por medio de la estrategia siguiente: supongamos que se tienen un conjunto de registros que cumplen con la cobertura mínima deseada para las reglas de condición 1. Supongamos que ahora deseamos obtener las reglas de 2 condiciones con la misma cobertura. Como son reglas de 2 condiciones, se debe de elegir registros que en 2 atributos coincidan sus valores. Para ello, se deben buscar registros con la estructura <..., A, B, ...>, <..., A, C, ...>, <..., B, C, ...>. Esto es, primero se deben elegir pares de registros del conjunto de registros de cobertura 1 en los cuales, un valor de un atributo x coincidan y además, en el valor correspondiente a otro atributo específico difieran. Por ejemplo, <..., A, B, ...>, <..., A, C, ...> coinciden con el valor A en un atributo y en el siguiente atributo difieren en su valor. Además, se debe cerciorar que la combinación de los valores de los atributos que no coinciden si existan en el conjunto de registros de cobertura 1. Para nuestro ejemplo, se

debe verificar que los registros con la estructura <..., B, C, ...> existan. Además, cada estructura de registros ya expuesta debe de cumplir con la cobertura mínima señalada. El procedimiento anterior se puede extender para reglas de 3 condiciones, 4 condiciones, etc.

Por su naturaleza, las reglas de asociación funcionan perfectamente con valores nominales, no así con valores numéricos continuos como los números reales, ya que por tratarse de valores que incluso pueden no coincidir para todos los registros de que integran a una $\Re$, puede darse el caso de no encontrase ninguna regla de asociación.

5.5 MINERIA DE DATOS EN SISTEMAS PARALELOS Y DISTRIBUIDOS

Un sistema paralelo es aquel que nos permite ejecutar un conjunto de instrucciones, no necesariamente las mismas, al mismo tiempo, donde cada instrucción es asignada a algún procesador del conjunto finito de procesadores disponibles. Las máquinas paralelas tienen dos esquemas básicos de memoria: memoria compartida o memoria distribuida. Independientemente del esquema de memoria utilizada, estas pueden ejecutar múltiples instrucciones sobre distintos datos (Multiple Instruction stream, Multiple Data stream, MIMD) o una misma instrucción sobre múltiples datos (Single Instruction stream, Multiple Data stream, SIMD)[42] [43].

Las máquinas con memoria compartida son las de mayor costo, debido al hardware especial que es necesario para la sincronización del equipo. Sin embargo, presentan un esquema de programación más simple, debido a que el programador se desentiende de los aspectos de sincronización entre los procesadores, concentrándose exclusivamente en los algoritmos a desarrollar. Por otra parte, un sistema con memoria distribuida es más económico, debido a que se pueden enlazar sistemas de cómputo independientes para trabajar como un sistema paralelo (en este esquema entran los sistemas distribuidos). Sin embargo, la problemática que presentan es la sincronización: está es implementada por el programador por medio de paso de mensajes, es decir, el control es por medio de software.

Se pueden distinguir dos tipos de sistemas capaces de realizar cómputo paralelo: los sistemas multiprocesadores y los sistemas distribuidos. Ambos tienen una característica común: realizar distintas operaciones al mismo tiempo (paralelismo). Sin embargo, distinguiremos a un sistema multiprocesador de uno distribuido en lo siguiente[44]: una máquina multiprocesador es un sistema de alto rendimiento con un conjunto finito de procesadores, donde existe una interconexión especial que optimiza la comunicación entre los procesadores, los cuales son sincronizados por medio de hardware y con un esquema de memoria compartida. Por otra parte, un sistema distribuido es aquel que interconecta a un conjunto de computadoras independientes por medio de una red (ya sea exclusiva o no), en el que la sincronización del sistema en conjunto es por medio de software (paso de mensajes) con un esquema de memoria distribuida. Por lo anterior, podemos concluir que ambos son sistemas paralelos, debido a que tienen la capacidad de realizar cálculos de manera paralela, radicando la diferencia en la forma de interconexión, sincronización y modelo de memoria.

Un punto fundamental tanto en los sistemas paralelos como en los distribuidos es la repartición de recursos y tareas. De hecho, es fácil deducir que la asignación de recursos y tareas se considera como un problema NP Completo. Considere que se tienen X_P procesadores y X_t tareas donde lo que se desea es distribuir el trabajo entre los procesadores de la forma más eficiente. Ello implica analizar todas las posibles formas de asignar las tareas a los procesadores, que está dado por X_P^{Xt}, siendo un problema de orden exponencial. Existen dos esquemas de asignación de tareas: *estática*, en la cuál desde un principio se dividen las tareas a los procesadores y no cambia con el transcurso de la ejecución y, *dinámicas*, en las cuales, de acuerdo a la disponibilidad y eficiencia, se asignan las tareas a los procesadores.

En la asignación de tareas, tres criterios son básicos:

- Tiempo de ejecución, esto es, se busca minimizar el tiempo necesario para ejecutar el algoritmo.
- Tiempo de comunicación, donde se determina que tanto volumen de información va a ser intercambiada por la red, buscando minimizar el tiempo empleado en el intercambio de está información.
- Equilibrio de cargas, donde se busca que los procesadores mantengan un índice de ocupación similar.

La asignación de tareas también debe de considerar el hardware utilizado, tomando como base:

- La arquitectura de interconexión entre los procesadores.
- El tipo de procesadores (su rendimiento).

Una correcta repartición de recursos y tareas dará como resultado una ejecución más rápida de las tareas. Por ello, se han desarrollado estudios muy amplios para obtener esquemas que permitan aprovechar al máximo el cómputo paralelo en sistemas multiprocesador y distribuidos [19], [20], [21], [41], [42], [43]. Sin embargo, la forma de diseñar los algoritmos para un sistema multiprocesador y para un sistema distribuido tienen algunas diferencias.

El flujo de información entre los procesadores de una máquina multiprocesador es considerado como un cuello de botella, ya que entre mayor sea el flujo de datos entre los procesadores, la velocidad total del sistema dependerá más de la eficiencia con la cuál el sistema de comunicación transporte los datos de un procesador a otro. Sin embargo, estos sistemas de comunicación suelen ser muy óptimos en la transferencia de datos, por lo que el diseño de los algoritmos para máquinas multiprocesador toma en cuenta la transferencia de datos entre los procesadores, pero no es un punto tan crítico como lo es en un sistema distribuido. En un sistema distribuido, la comunicación de las computadoras dependen de una red de computadoras. Las redes de computadoras han avanzado considerablemente pero, pese a estos avances, su capacidad de transmisión sigue siendo limitada. Por ejemplo, si la comunicación de los sistemas se lleva a cabo vía telefónica por medio de un proveedor de servicio de Internet (ISP), que comúnmente es a 56Kbps, deduciremos inmediatamente que el ancho de banda disponible es limitado, funcionando

bien para transferencias de datos pequeñas. Sin embargo, para transferencias de datos muy grandes, está velocidad dará como resultado grandes retrasos en el tiempo de comunicación.

A pesar de las limitantes de comunicación que puede presentar un sistema distribuido, el bajo costo en la implementación física de estos ha favorecido enormemente a su proliferación. Sin embargo, estos requieren invertir más esfuerzo en el desarrollo de software que aproveche al máximo los recursos de cómputo de cada sistema, tratando de minimizar lo más posible la cantidad de datos a transmitir.

La Minería de Datos en sistemas paralelos y distribuidos persigue aprovechar el paralelismo que ofrecen buscando reducir los tiempos de respuesta de los algoritmos. Como estos algoritmos dependen directamente de los datos, una de las primeras consideraciones a hacer es determinar como se encuentran almacenados los datos a analizar, en una sola base o un conjunto de bases, y a partir de este hecho, considerar el tipo de sistema paralelo a utilizar, ya sea un sistema multiprocesador o distribuido.

Notemos un hecho interesante antes de continuar. Un sistema multiprocesador se puede ver como una sola computadora con un conjunto finito de procesadores, la cuál generalmente tiene sus unidades de almacenamiento exclusivos. En estas unidades de almacenamiento se puede albergar tanto a una BD centralizada como a una BD fragmentada (en ocasiones los fragmentos son ubicados en las distintas unidades de almacenamiento del sistema multiprocesador, debido al gran tamaño que la BD puede tener). Por otra parte, una BDD está ubicada sobre un conjunto de sistemas de cómputo independientes comunicados por una red, que de hecho forman un sistema distribuido. Por ello, los algoritmos diseñados para una BDD necesariamente deben de contemplar como un punto crítico el volumen de datos a transmitir, debido a las limitaciones ya expuestas de los sistemas distribuidos.

Por lo anterior, los algoritmos de Minería de Datos para una BDD deben de aprovechar al máximo el poder de cómputo de cada computadora que integra al sistema distribuido, tratando de minimizar lo más posible el flujo de información que requiera el algoritmo y, por supuesto, sin afectar la veracidad de los resultados, donde los tiempos de respuesta del algoritmo serán influenciados fuertemente por la eficiencia de la red sobre la cuál este montada la BDD. Por otra parte, los algoritmos diseñados para un sistema multiprocesador tienden a tener tiempos de respuesta menores a los de un algoritmo diseñado para una BDD, por las consideraciones ya planteadas. Además, el volumen de información a transmitir entre los procesadores no es un punto tan crítico como lo es en una BDD, aunque no deja de ser importante.

El trabajo que se ha realizado en Minería de Datos para adaptar los algoritmos tanto a sistemas multiprocesador como distribuidas es amplio. Algunos de estos trabajos se han enfocado principalmente a buscar esquemas que trabajen eficientemente en sistemas multiprocesador, donde la prioridad es la rapidez de los resultados [17][35]. Estos estudios son de muy diversa índole, y consideran desde la eficiencia en el uso de los recursos, la comunicación, hasta aspectos como la saturación que provocan los algoritmos en las computadoras [36].

Por ejemplo, se han desarrollado trabajos enfocados a la representación de conocimiento por medio de redes Bayesianas en datos distribuidos [37]. Otros trabajos se han desarrollado

basándose en las reglas de asociación sobre una BDD, donde el objetivo fue minimizar la cantidad de mensajes requeridos sobre la red [38]. En sistemas multiprocesador se han propuesto diversos esquemas. Por ejemplo, se han hecho trabajos basados en los árboles de clasificación tratando de equilibrar la carga de trabajo entre los procesadores disponibles sin sobrecargar el flujo de datos entre los mismos [39].

Todos estos trabajos se esfuerzan por aprovechar al máximo la naturaleza del sistema sobre el cuál trabajan. De hecho, por las enormes cantidades de datos que actualmente manejan las BD's, sería imposible realizar trabajos de Minería de Datos basándose en un solo procesador, debido a la gran cantidad de tiempo que se requeriría. Por tanto, la tendencia actual es el desarrollo de algoritmos basados en el concepto de cómputo paralelo, ya sea en sistemas multiprocesador o sistemas distribuidos.

En este sentido, el presente trabajo de tesis presenta un algoritmo de clasificación capaz de trabajar en una BDD con fragmentación horizontal primaria. Para ello, en el siguiente capítulo se exponen las consideraciones que se han hecho para diseñar el algoritmo. Posteriormente, se presentará dicho algoritmo de forma detallada.

6 ALGORITMO ID3 EN UN AMBIENTE DISTRIBUIDO

En éste capítulo se propone un algoritmo ID3 modificado para trabajar sobre una BDD. El algoritmo tiene como idea fundamental aprovechar el poder de cómputo de los sitios que integran a la BDD, tratando de minimizar el paso de mensajes entre los sitios, evitando así sobrecargar el flujo de datos en la red que comunica a la base.

La exposición inicia explicando las características que hacen al algoritmo ID3 un método factible de ser modificado para un ambiente distribuido. Posteriormente, se plantean las consideraciones necesarias para la modificación de dicho algoritmo considerando una BDD con "Y" sitios (caso general). Con los elementos anteriores, finalmente se presenta el algoritmo propuesto, haciendo un análisis detallado de las secciones que lo componen.

El análisis que se plantea, comprende tres aspectos fundamentales: la estructura de los datos de entrada, el desempeño del algoritmo y el aprovechamiento de recursos.

El primer aspecto se asocia a los datos de entrada. Ya se comentó que se va a trabajar bajo un esquema de fragmentación horizontal primaria, la cuál mantiene la estructura de la relación original. Sin embargo, se debe considerar como afecta está fragmentación a las posibles relaciones que se desean encontrar. Es decir, se debe analizar si la fragmentación afecta o no a las relaciones existentes entre los dominios de cada atributo y el atributo clasificador.

Con respecto al desempeño, se han identificado dos puntos fundamentales a tratar:

- *Desempeño temporal:* se obtendrá el grado del algoritmo, estableciendo su comportamiento en el peor de los casos, caso promedio y el mejor de los casos.

- *Esquema del algoritmo:* que tipo de algoritmo es el ideal, uno que construya de manera independiente resultados en cada sitio para reunirlos en un proceso final o con intercambio de información, para construir de manera conjunta el resultado.

Con respecto al aprovechamiento de los recursos, se plantean los siguientes puntos:

- *Comunicación del sistema:* El volumen de información a intercambiar entre los sitios es sumamente importante, debido a que se trata de un sistema distribuido en el cuál, se puede

contar con medios de transmisión muy eficientes o no tan eficientes. Por ello, mantener un bajo intercambio de información entre los sitios favorecerá a un mejor desempeño del algoritmo.

- *Equilibrio de cargas:* Es recomendable que los recursos de procesamiento de cada sitio sean aprovechados al máximo. Con ello, se persigue que el mayor número de cálculos se realicen en cada sitio, y sólo los resultados de estos cálculos sean intercambiados, minimizando así la comunicación del sistema.

6.1 Selección del algoritmo

En el planteamiento del trabajo se fijo la postura de encontrar un algoritmo que, por sus características, fuera factible modificar para adaptarlo a trabajar sobre una BDD con fragmentación horizontal primaria.

En base a los estudios realizados, se determinó que el algoritmo ID3 desarrollado por Quinlan tiene características que lo hacen un buen candidato para ser elegido. Estas características involucran varios aspectos, los cuales se describen a continuación.

El primer punto a considerar es referente a la facilidad en el uso e interpretación de resultados que el usuario final obtendrá del método. ID3 es un algoritmo basado en la búsqueda guiada. Lo anterior significa que el usuario debe establecer tanto los atributos con los cuales se va a trabajar así como el atributo clasificador. Una ventaja de elegir un atributo clasificador es el hecho de que mediante este atributo el usuario establece de un solo paso un conjunto de clases en las cuales se van a clasificar los registros, a diferencia de otros algoritmos como 1 – rule, en el cuál el usuario establece de entrada una sola clase. Fuera de estas especificaciones, ID3 no requiere que el usuario interactúe más con el sistema. Lo anterior es muy importante si se piensa en una ejecución sobre una BDD, ya que sus sitios generalmente son ubicados en lugares geográficamente distintos, lo cuál exige que si se va a ejecutar un algoritmo distribuido sobre está BDD, la interacción del usuario sea mínima, ya que lo contrario requeriría grandes recursos humanos para sincronizar las decisiones y operaciones. Notemos que de hecho, un usuario podría establecer en un solo sitio los atributos a trabajar y el atributo clasificador requeridos por ID3 y, una vez iniciado el proceso de construcción, replicarse está información a cada sitio, sincronizando desde un principio el proceso de construcción. También debemos señalar que lo anterior es posible realizar debido a que la BDD que se está considerando tiene una fragmentación horizontal primaria, lo cuál garantiza que la estructura de los atributos en cada sitio es idéntica.

Con respecto a la interpretación de los resultados, ID3 obtiene un árbol de clasificación como el que se ilustra en la Figura 5.3. Para usuarios acostumbrados a trabajar con este tipo de árboles son resultados muy claros y entendibles, no así para usuarios poco habituados. Sin embargo, esto no es problema, ya que la estructura del árbol resultante puede ser llevada fácilmente a reglas de clasificación, las cuales son sumamente entendibles y claras. Por ejemplo, el árbol de la Figura 5.3 puede transformarse en el siguiente conjunto de reglas:

SI (Ambiente = soleado) y (Humedad = alta) ENTONCES es de la clase Clase: N.
SI (Ambiente = soleado) y (Humedad = normal) ENTONCES es de la clase Clase: P.
SI (Ambiente = nublado) ENTONCES es de la clase Clase: P.
SI (Ambiente = lluvia) y (Viento = no) ENTONCES es de la clase Clase: P.
SI (Ambiente = lluvia) y (Viento = si) ENTONCES es de la clase Clase: N.

Analicemos ahora las características propias del algoritmo. ID3 es un algoritmo que puede trabajar perfectamente con valores nominales. También se puede trabajar con valores numéricos no continuos, como los números naturales, no así con números continuos como los reales, ya que estos, a pesar de que se limiten a un intervalo muy pequeño, la cantidad de valores posibles en dicho intervalo es infinita, lo cuál provoca que el dominio del atributo sea difícil de manejar para el algoritmo. Sin embargo, desde la perspectiva del autor, manejar valores nominales y numéricos no continuos le da una versatilidad de uso muy amplia en distintos tipos de BD's.

Con respecto al número de operaciones, debemos recordar que ID3 es un método de Minería de Datos y como tal, el manejo del descubrimiento de los patrones es dirigida por los mismos datos. Entre más ordenados se encuentren los datos de acuerdo al atributo clasificador, la tendencia será requerir menor número de operaciones para construir el árbol. Ello conlleva a que si la cantidad de datos de la BD aumenta, no necesariamente aumentará la cantidad de cálculos a realizar, siendo una buena cualidad del algoritmo.

Un punto que se ha identificado modificar es la forma en que se determina el atributo ganador. En el algoritmo ID3 centralizado, está decisión depende exclusivamente de la base centralizada pero, en una BDD, está decisión debe considerar los resultados de otros sitios, por lo cuál, éste será uno de los puntos que se tendrán que rediseñar.

Las características ya expuestas ayudaron a determinar que el algoritmo ID3 es un buen método para ser analizado y adaptado para una BDD. En los siguientes puntos se analiza a detalle las consideraciones necesarias para llevar a cabo la modificación del algoritmo original.

6.2 Factores a considerar para las modificaciones del algoritmo ID3 en una BDD

Identificar los aspectos que intervienen para que ID3 funcione en una BDD, implica comprender dos factores básicos: los relacionados a la entropía (en la cuál se basa el algoritmo), y los relacionados a los datos (entender las posibles consecuencias que implica tener los datos distribuidos). Estos puntos se analizan a continuación.

6.2.1 Factores relacionados a la entropía

En esta sección se presenta como se ha pensado calcular la ganancia global de cada atributo considerando los resultados de cada uno de los sitios de la BDD. Para ello, se analizarán desde la

forma en la cual se construirá el árbol de clasificación hasta los tipos de resultados que cada sitio debe generar para poder calcular un atributo ganador.

Las ganancias de los atributos ganadores se basan en el concepto de la entropía. La entropía nos permite conocer el grado de orden (o desorden, según el enfoque) presente en una $\Re$ cualquiera. En ID3, cuando se elige un atributo ganador A_g, el grupo de registros que se están trabajando (representado por $\Im$) es dividido en β_g, los cuales son representados por $\Im_j^g$ y $1 \leq j \leq \beta_g$, donde cada grupo se asocia a una rama del nodo que forma el atributo A_g en el árbol.

Repartir los registros entre las ramas del árbol, provoca que la entropía proporcional de cada grupo $\Im_j^g$ sea menor o igual (en el peor de los casos) a la entropía del conjunto $\Im$. Lo anterior lo podemos mostrar de la siguiente manera: sea un conjunto de registros $\Im$ cualquiera con "*m*" registros, el cuál es dividido en "*t*" grupos $\Im_j^a$ en base al atributo A_a. Cada subconjunto $\Im_j$ con $1 \leq j \leq t$ tendrá m_j registros (no necesariamente iguales), donde:

$$\sum_{j=1}^{t} m_j = m$$

La entropía de está fragmentación estará dada por:

$$E(A_a) = \sum_{j=1}^{\beta_a} \Pr(\Im_j^a) * I(\Pr(C_{j(1)}^{a(p)}), \Pr(C_{j(2)}^{a(p)}), \ldots, \Pr(C_{j(t)}^{a(p)}))$$

Podemos notar que la entropía se compone de las sumas parciales $I(\Pr(C_{j(1)}^{a(p)}), \Pr(C_{j(2)}^{a(p)}), \ldots, \Pr(C_{j(t)}^{a(p)}))$, que a su vez representan la entropía proporcional de cada $\Im_j^a$. Si $\forall j \; |\Pr(\Im_j^a) * I(\Pr(C_{j(1)}^{a(p)}), \Pr(C_{j(2)}^{a(p)}), \ldots, \Pr(C_{j(t)}^{a(p)})) \neq 0$ y además, cualquier probabilidad nunca es negativa, entonces cada entropía es menor a $E(A_a)$. Algo importante que se debe rescatar de lo anterior es que fragmentar a un conjunto de registros, afectará a su entropía en mayor o menor grado.

Ahora, ¿qué pasa en un BDD? Una BDD está compuesta de "Y" sitios, donde cada sitio es independiente tanto a nivel de procesamiento como de datos. Como los datos de cada sitio tienen un comportamiento independiente entre sí, es de esperarse que la entropía de estos no sea exactamente la misma, provocando que los árboles locales no necesariamente coincidan. Esto conlleva a plantear la manera de construir un árbol general o global que permita clasificar los registros de todas las bases.

Estrategias para la construcción de un árbol ID3 en una BDD

La construcción de un árbol ID3 en una BDD se puede llevar a cabo de las siguientes formas básicas:

- Construir de manera independiente cada uno de los árboles en cada sitio y, en un proceso final, reunir está información para crear un árbol que represente todos los patrones posibles

dentro de la BDD. Esta forma presenta la disyuntiva de cómo conjugar árboles que pueden diferir en los nodos que los componen. De hecho, nada impide que desde la raíz del árbol difieran unos de otros, por lo que, las ramas derivadas de este nodo presentarán un fuerte problema de relación. Finalmente, como este proceso se basa en los árboles locales, que a su vez se fundamentan en la entropía de los mismos, el árbol global que se construya no necesariamente coincidirá con el árbol que se pudiera obtener si los datos estuvieran en un solo sitio, ya que la entropía puede variar.

- Construir el árbol global con intercambio de datos en el proceso. En éste esquema, se llevarán datos de un sitio a otro dependiendo si son requeridos para calcular una entropía de algún atributo. De hecho, se podría optar por asignar a cada sitio un atributo en particular, el cuál requerirá importar los valores de los campos de los registros de dicho atributo y una vez calculada su entropía, enviar está información a todos los sitios o sitio responsable del control de las acciones a seguir. Esta propuesta construirá el mismo árbol que en el caso centralizado, ya que los resultados se obtienen de reunir todos los datos referentes a una ganancia en particular. Sin embargo se tiene un problema muy fuerte: el gran flujo de información entre los sitios, ya que aquí, estos intercambian datos y, considerando que cada sitio tiene toda la estructura de una BD, la cantidad de información que en un momento dado se tendrá que llevar a otro sitio (o sitios) es muy grande.

- Construcción con intercambio de ganancias. Este esquema construye de manera conjunta el árbol global. El intercambio de ganancias induce a esperar un menor flujo de datos entre los sitios, ya que estos sintetizan los patrones presentes en cada uno de ellos. Notemos que este método se basará en los patrones de cada sitio, provocando que el árbol que finalmente se construya no necesariamente coincida con el árbol que se pudiera construir si los datos estuvieran en una sola BD.

La última estrategia expuesta presenta características muy interesantes. En primer lugar, los recursos de cada sitio serán mejor aprovechados, debido a que estos tienen que realizar los cálculos de las ganancias de los datos locales, que son las operaciones que exigen mayor procesamiento en la construcción de un árbol basado en ID3. Lo anterior también implica que se tendrá un alto grado de paralelismo al momento de calcular las ganancias, ya que estas serán realizadas en conjunto por los procesadores de los diversos sitios de la BDD. Aunado a esto, transferir las ganancias implica un menor flujo de datos en la red, debido a que estos sintetizan la entropía que un conjunto de datos puede tener. Lo anterior se apega a la ideología de un sistema distribuido, por lo cuál se adoptará como esquema de trabajo. Analicemos a continuación el punto relacionado al equilibrio de cargas, que determinan en que grado serán aprovechados los recursos de procesamiento de cada sitio.

Equilibrio de trabajo entre los sitios de la BDD

Explotar al máximo los recursos de cada sitio, implica considerar dos aspectos importantes: la capacidad de procesamiento y el equilibrio de cargas de cada sitio. Sobre el primer punto, cada sitio tiene la capacidad de procesar de manera independiente datos; sin embargo, lo anterior no indica el poder de procesamiento que puede presentar cada uno de ellos. Esto es, los sitios de una

BDD pueden tener el mismo tipo de hardware, pero también pueden diferir, provocando que algunos sitios tengan más poder de procesamiento que otros. En la teoría para el desarrollo de BDD's se considera este aspecto, persiguiendo mantener una relación equilibrada entre la cantidad de datos y las operaciones a realizar con el hardware presente en cada sitio. Basándonos en lo anterior, en este trabajo solo analizaremos que sucede con el equilibrio de cargas entre los sitios. En el equilibrio de cargas, lo ideal es mantener ocupados a los procesadores el mayor tiempo posible para explotar al máximo su poder de cómputo. Sin embargo, mantener un alto índice de ocupación en los procesadores implica tener disponibles datos para procesar. Sobre este aspecto, observemos la Figura 6.1.

Figura 6.1. Comparación entre tiempos de procesamiento

Supongamos tres procesadores con la misma capacidad (uno para cada sitio de una BDD cualesquiera), a los cuales se les asigna la misma cantidad de datos a procesar, finalizando en primera instancia el procesador 2, debido a que la entropía en el conjunto de datos que analizó es muy baja. En ese momento, el procesador es capaz de "cooperar" con el procesamiento que se realiza en los otros sitios que aún no han finalizado, pero ello implica importar información de los otros dos sitios. Ahora, ¿qué cantidad de datos es necesario importar para poder cooperar con el procesamiento? Las ganancias se basan en dos atributos: un atributo clasificador A_p y el atributo A_x del cuál se obtendrá su ganancia. Por ello, es necesario tener los valores de los registros de estos dos atributos. Supongamos que de acuerdo al diseño de la BDD, es más económico llevar datos del sitio 1 al sitio 2 que del sitio 3 al sitio 2.

Como se ilustra en la Figura 6.2, si se desea que el sitio 2 coopere con el procesamiento que se realiza en el sitio 1, es necesario transferir los valores de los registros del atributo clasificador y los correspondientes valores de algún atributo A_x no procesado. Notemos que al importar el atributo clasificador es posible realizar el cálculo de más de una ganancia, simplemente importando aquellos atributos que se desean calcular. Al obtener el(los) resultado(s) de la(s) ganancia(s), y dependiendo de la estructura del control de operaciones adoptado para la BDD (que se analiza más adelante), este debe retornar el resultado a uno o todos los sitios de la BDD. Si calculamos exclusivamente la cantidad de datos que se transmite de un sitio a otro, en base a "r" atributos más el atributo clasificador con "m" registros cada uno, veremos que se transmiten $m(r + 1)$ bytes, considerando que con un byte podemos identificar a cada valor de cada registro. Como "m" puede ser muy grande, la cantidad de información a transmitir será igualmente grande, siendo un punto muy negativo para considerar llevar a cabo una estrategia de equilibrio de cargas. Sin embargo, si consideramos que uno de los criterios para la construcción de una BDD

es mantener cierto equilibro entre el número de registros asociados a cada sitio, se puede pensar que los patrones de cada una de estas bases se comporte de forma similar, provocando que se espere un número similar de operaciones al momento de construirse un árbol ID3.

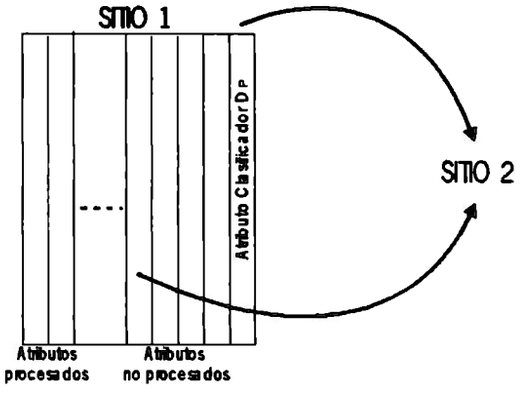


Figura 6.2. Flujo de datos para el procesamiento en otro sitio.

Como no sabremos a ciencia cierta como se comportarán los patrones en los sitios, podemos plantear tres posibles escenarios para el equilibro de las cargas:

- *Peor de los casos:* Cuando los dominios de los atributos son diferentes para cada sitio, los patrones existentes en los sitios son disjuntos, provocando que cuando se construya el árbol global, ciertos sitios tengan que trabajar más por tener patrones que en otros sitios no existen. Ello provocaría un desequilibrio de cargas de trabajo para cada sitio.

- *Caso promedio:* En un caso promedio, se espera que la distribución de los patrones en los diversos sitios esté equilibrada, provocando que en el proceso de construcción del árbol ID3, la mayoría de los sitios participen en el desarrollo de una rama y que solo algunos, para casos especiales, no participen en está construcción. La consecuencia inmediata es que la carga de trabajo se mantendrá casi equilibrada para cada nodo que se desarrolle en el árbol.

- *Caso óptimo:* El caso óptimo se presentará cuando cualquier patrón posible esté presente en cualquier sitio, por lo que estos intervendrán de manera equilibrada en el proceso de construcción, provocando que la carga de trabajo para cada sitio sea exactamente la misma.

El peor de los casos y el caso óptimo se pueden considerar como situaciones especiales que no se esperaría encontrar en una BDD, ya que estas persiguen mantener una distribución equitativa de los datos. Por ello, consideraremos como base de nuestro estudio al caso promedio.

Construcción del árbol ID3 distribuido: requerimientos para generar los resultados

Ahora analicemos como construir el árbol ID3 en una BDD. Una de los requerimientos que debe cumplir la BDD que analicemos es el hecho de estar basada en una fragmentación horizontal. Ello implica que Ψ es la misma para cada sitio, garantizando que un atributo que está presente en un sitio cualquiera, también estará presente en el resto de los sitios que componen a la BDD.

Denotemos con Ψ^k ($1 \leq k \leq Y$) al conjunto de atributos presentes en la relación $\Re^k$ del sitio S^k y con A_x^k a un atributo "x" de $\Re^k$. En cada $\Re^k$, obtener el atributo con mayor ganancia A_g^k significa analizar todos las $G(A_x^k)$ posibles del conjunto Ψ^k - {atributos ya procesados}. Al conjunto de atributos ya procesados lo denotaremos con Γ^k. Por tanto, el atributo ganador se obtiene de:

$$A_g^k = \max\left\{G(A_x^k) \middle| A_x^k \in \Psi^k - \Gamma^k \; y \; A_x^k \neq A_p^k\right\}$$

Notemos que cada A_g^k puede ser el mismo para cada sitio de la BDD. Si es así, este atributo puede pasar a formar parte del árbol global que se construye, pero esto no necesariamente se cumple. Ahora, ¿qué hacer para obtener el atributo que en conjunto presente la mayor ganancia tomando en cuenta cada tendencia de cada sitio?

Lo anterior implica analizar cada ganancia de cada sitio. Notemos que obtener una ganancia máxima en cada sitio, implica calcular las ganancias de todos los atributos presentes en el sitio. De hecho, se generan $\left|\Psi^k - \Gamma^k\right|$ ganancias, una ganancia para cada atributo procesado. Denotemos a estas ganancias por medio del siguiente vector:

$$G^k = \{G_1^k, G_2^k, \ldots, G_s^k\} \text{ donde } s = \left|\Psi^k - \Gamma^k\right|$$

Para identificar la ganancia respectiva de cada atributo, es necesario que cada elemento G_x^k conste de dos valores: el identificador del atributo A_x y la ganancia del mismo, que es un número flotante en el rango [0, 1] obtenido de $G(A_x^k)$. Si denotamos por T_D al tamaño con el cuál identificamos a cualquier atributo perteneciente a la BDD y con T_G al tamaño con el cuál podemos representar la ganancia, el tamaño del vector generado será (en bytes):

$$|G^k| = |\Psi^k - \Gamma^k| * (T_D + T_G) \text{ bytes}$$

Una vez que cada sitio ha calculado su vector de ganancias, el proceso siguiente consistirá en reunir estos para sacar un consenso sobre al atributo con mayor ganancia. Ya señalamos que A_g^k no va a coincidir necesariamente para todos los sitios, debido a que este valor depende de la distribución marcada por A_p^k. En este punto se abren dos interrogantes: ¿en donde reunir los resultados? Y ¿cómo calcular el atributo ganador?

El proceso de reunir los resultados para realizar el consenso se puede realizar básicamente de dos formas: propagar los resultados a cada uno de los sitios o reunirlos en un sitio. El primer caso implica intercambiar los resultados locales de todos los sitios pertenecientes a la BDD. En la Figura 6.3. se ilustra el caso para el sitio 1 en una BDD cualquiera.

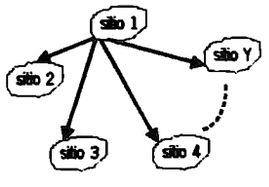


Figura 6.3. Intercambio de información entre todos los sitios.

Observamos que este sitio tiene que enviar a cada uno de los sitios de la BDD su resultado local, transmitiendo un total de "Y-1" vectores, lo que implica un flujo de $(Y\text{-}1)*|G^J|$ bytes. Como es necesario que cada sitio a su vez transmita los resultados que obtuvo en su cálculo local, veremos que para calcular un atributo ganador, la BDD tendrá que transmitir:

$$(\text{Y-1})*\sum_{i=1}^{Y}|G^i| \text{ bytes}$$

Antes de continuar, debemos señalar que el vector de ganancias transmitido por cada sitio decrece para cada nivel del árbol en $T_D + T_G$ bytes. Por ejemplo, si suponemos que la Figura 5.3 es obtenida de una BDD, el proceso que dio origen al nodo *Humedad* ya no requirió que los sitios calcularán la ganancia del atributo *Ambiente*, por lo cuál los vectores de ganancias fueron menores en dimensión que los que dieron origen al nodo *Ambiente*.

En este esquema, como cada sitio cuenta con la totalidad de los resultados obtenidos en el resto de los sitios, cada uno es capaz de obtener el mismo atributo ganador para cada iteración, evitando una transmisión adicional sobre el atributo ganador A_g. Sin embargo, notemos que si no existe un orden entre los dominios de los atributos, será necesario una transmisión adicional que sincronice que rama se seleccionará para trabajar en la siguiente iteración. Este esquema funcionará perfectamente si los dominios de los atributos locales son idénticos, sin embargo, esto no necesariamente se dará, provocando un problema de sincronización cuando se seleccione algún valor de un atributo el cuál no pertenezca al dominio local de uno o varios sitios. Ello conlleva a plantear esquemas de sincronización que prevengan este problema, pero a su vez, implicará aumentar el flujo de datos que se dé entre los sitios.

También observemos que el flujo de información para este esquema es de orden cuadrático con respecto al número de sitios que integran a la BDD, aumentando considerablemente si el número de sitios es grande. Sin embargo, tiene la ventaja de presentar una gran tolerancia a los fallos, ya que todos los sitios realizan las mismas tareas, por lo que no existe ningún sistema central del cuál dependa el cálculo global. Esto conlleva a que si un sitio llega a fallar, el cálculo de las ganancias podría proseguir, ajustándose exclusivamente a los sitios que continúen operando.

Ahora analicemos el esquema cliente – servidor. Este tiene la característica de que el control de los cálculos es llevado por un solo sitio, que llamaremos servidor. Esto sugiere que el buen término de los cálculos depende en gran parte del servidor, el cuál, entre otras cosas, se encargará de recopilar las ganancias de cada sitio para calcular el atributo ganador, y una vez calculado, elegir la rama por la cuál todos los sitios deberán continuar operando. Este esquema lo podemos visualizar en la Figura 6.4.

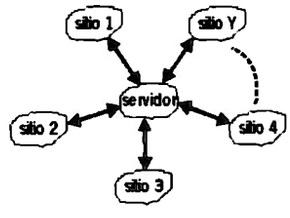


Figura 6.4. Intercambio de información en un esquema cliente – servidor.

En está figura podemos observar que, una vez calculadas las ganancias en cada sitio, estas serán transmitidas al servidor, el cuál se encargará de determinar cuál es el atributo con el que se debe trabajar. Acto seguido, el servidor enviará un mensaje a cada sitio notificando el resultado obtenido. Es importante notar que cada sitio en sí debe construir un árbol local correspondiente a sus datos, mientras que el servidor se encargará de construir el árbol global basado en los resultados de los árboles locales.

Analicemos cuál es el flujo de datos que esperaremos de éste esquema. Cada sitio generará un vector de ganancias G^k para cada iteración, transmitiendo este vector exclusivamente a un solo sitio, el servidor. Por ello, solo será necesario transmitir $|G^k|$ bytes por sitio (el servidor puede ser un sitio más en la base o puede estar dedicado exclusivamente a actividades propias de administración; para nuestro estudio, consideraremos el segundo caso). Por tanto, para calcular un solo atributo ganador A_g, los sitios transmitirán al servidor:

$$\sum_{i=1}^{Y} |G^i| \text{ bytes}$$

Una vez reunidos los resultados de cada sitio, el servidor calculará por medio de alguna estrategia el atributo ganador. Sin embargo, para mantener sincronizados los cálculos de cada sitio, no solo se debe de informar cuál fue el atributo ganador, también se les debe indicar por cuál de las ramas se debe de continuar. Por ejemplo, supongamos que la Tabla 2 es dividida para formar una BDD con Y sitios y en el proceso de construcción del árbol ID3 para la BDD se determina que el atributo ganador para la primera iteración es *Ambiente*. Como lo muestra la Figura 5.3, este atributo genera 3 ramas. Si no se indica por cuál de ellas trabajar, se puede presentar el caso en el que algunos de los sitios elijan trabajar con la rama "*soleado*", otros con la rama "*nublado*" y otros con la rama "*lluvia*". Esto provocaría que en la siguiente iteración no se pueda calcular un atributo ganador global, debido a que no están sincronizados los cálculos. Por ello, es necesario incluir en los mensajes del servidor a los sitios el valor correspondiente al dominio del atributo ganador con el cuál se trabajará en la siguiente iteración.

Suponiendo que el tamaño para identificar un valor en el dominio de un atributo cualquiera es el mismo que para identificar a un atributo, el flujo requerido para reunir los datos necesarios para calcular un atributo ganador en el servidor y este indique por cuál rama continuar será:

$$\sum_{i=1}^{Y} |G^i| + 2T_D * Y \text{ bytes}$$

Observemos que el flujo de datos que este esquema presenta es menor al expuesto en el caso anterior. En contraparte, la dependencia del servidor no lo hace tan tolerable a los fallos, ya que si por alguna circunstancia deja de operar, el total de los cálculos no podrá continuar. Sin embargo, generalmente en una red de computadoras, la cuál da soporte a una BDD, existe un sitio que es el encargado de administrar su funcionamiento, siendo sistemas que comúnmente son muy estables. Para minimizar la probabilidad del problema expuesto, se podría optar por situar al servidor de nuestro sistema en este equipo.

Considerando los puntos ya expuestos, se determina que el esquema cliente - servidor presenta mayores ventajas que el esquema de distribución de resultados para cada sitio. Determinado el esquema de control de operaciones para nuestro algoritmo, analicemos ahora como calcular el atributo ganador.

Calculo del atributo ganador

En primera instancia, notemos que cada vector G^k está compuesto de $\left|\Psi^k - \Gamma^k\right|$ elementos, cada uno de los cuales representa la ganancia de un atributo. Como nos basamos en una fragmentación horizontal primaria, el conjunto $\Psi^k - \Gamma^k$ es idéntico para cada sitio. Esto nos indica que:

$$\forall A_x^k \mid A_x^k \in \Psi^k - \Gamma^k, \forall u \mid u = 1,\ldots,Y \; y \; u \neq k, A_x^k \in \Psi^u - \Gamma^u$$

Antes de continuar, por notación se usara superíndice para referirse a un sitio "*k*" de la BDD, y sin superíndice se referirá a los conjuntos globales que trabaje el servidor de la BDD.

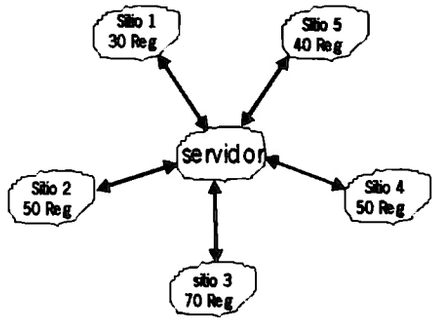


Figura 6.5. Ejemplo sobre la cobertura que aporta un sitio en base al número de registros.

Notemos que para cada sitio "*k*", los atributos pertenecientes a éstos generarán una ganancia $G(A_x^k)$, que no necesariamente es la misma. También es interesante notar que estas ganancias se obtienen de un número de registros que es variable para cada sitio "*k*", el cuál lo expresaremos por m^k. La ganancia en sí, no aporta información sobre el número de registros de la cuál fue obtenida, sólo representa el grado de orden de los registros conforme al atributo A_x^k. Sin embargo, una ganancia que proviene de un mayor número de registros va a ser más representativa que una ganancia que proviene de un menor número de registros, por tener una mayor cobertura. Esto lo podemos ejemplificar de la siguiente manera: consideremos la Figura 6.5., en la cuál el sitio 3 presenta la mayor carga de registros con 70, mientras que el sitio 1 sólo tiene 30 registros. Sería deseable que cada vector de ganancias tenga un mayor peso conforme al número de registros de la cuál se obtuvo. Con ello y de acuerdo a la figura, se buscaría que la importancia de cada vector sea (de mayor a menor): G^5, G^2, G^4, G^3, G^1.

Para jerarquizar a los vectores, podemos basarnos en la relación "número de registros considerados en cada sitio entre el número total de registros considerados en el sistema". Para el ejemplo citado, el sistema en conjunto trabaja sobre 240 registros, por lo que la proporción para cada vector se obtendrá del número de registros considerados en cada sitio entre los 240 registros. Jerarquizados los vectores, el siguiente paso es calcular la ganancia global correspondiente a un atributo en particular. Como cada atributo tiene una ganancia para cada sitio, una forma simple de

calcular la ganancia global de cada atributo es sumar las ganancias correspondientes de cada uno de los sitios. Si utilizamos está idea así como la de jerarquizar los vectores, calcular la ganancia global de un atributo "x" quedará expresado por:

$$G(A_x) = \frac{1}{\sum_{k=1}^{Y} m^k} * \sum_{k=1}^{Y} m^k G(A_x^k)$$

Ecuación 1. Cálculo de la ganancia de un atributo A_x en una BDD

Esta expresión, además de las cualidades ya consideradas, tiene la virtud de asignar una misma jerarquía a las ganancias de los atributos que provengan de un mismo número de registros, es decir, si m^k es la misma para dos o más sitios, la relación $m^k \Big/ \sum_{k=1}^{Y} m^k$ asignará la misma jerarquía a las ganancias consideradas. Esta situación se presenta en el ejemplo de la Figura 6.5. con los sitios 2 y 4.

Recordemos que en cada iteración no se está trabajando con un sólo atributo, sino se tienen $|\Psi - \Gamma|$ atributos, por lo que obtendremos $|\Psi - \Gamma|$ ganancias. Al igual que en el caso centralizado, se debe elegir aquél atributo con mayor ganancia, esto es, el atributo ganador para el caso distribuido estará dado por:

$$A_g = \max\{G(A_x) \mid 1 \le x \le |\Psi - \Gamma|\}$$

Ecuación 2. Cálculo de el atributo ganador global de una iteración en una BDD

Una vez calculado, el servidor informará a los sitios tanto del atributo ganador como del valor del dominio con el cuál se continuará trabajando.

6.2.2 Factores relacionados a la distribución de los datos

Esta sección presenta la forma en la que operará el sistema para llevar a cabo la construcción del árbol en el servidor. Para ello, se expone como se sincronizarán las operaciones entre los sitios y el servidor desde un inicio. También se determina como el servidor construirá el árbol, planteando el comportamiento de los sitios en base a sus datos. Se expondrá como los datos determinan cuando un sitio participará en el desarrollo de una rama del árbol y cuando ya no, así como el tipo de resultados requeridos en el servidor para cada caso. Por otra parte, se define que tipo de respuestas o indicaciones se emanarán del servidor hacia los sitios.

Ejemplo de un crecimiento no balanceado de un árbol

Los datos de cada sitio de una BDD tienen un comportamiento independiente entre sí. Ello implica que no se puede garantizar que los patrones presentes en un sitio estén presentes en otro sitio, provocando que en un proceso de construcción de un árbol ID3, algunos sitios tengan que analizar patrones que en otros sitios no existan. Lo anterior lo podemos ilustrar con el siguiente

ejemplo: supongamos que deseamos fragmentar los datos de la Tabla 2 en dos sitios y, después de haber tomado en cuenta las consideraciones hechas para la construcción de una BDD, se determina que los registros son divididos en las Tablas 3 y 4, teniendo en común el mismo número de registros, 7. Analicemos que patrones están presentes en cada uno de estos sitios utilizando el Algoritmo 1.

Ejecutando este algoritmo sobre la Tabla 3 ($\Re^1$) con el atributo clasificador "*Clase*", obtendremos el siguiente árbol:

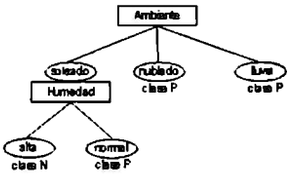


Figura 6.6. Árbol ID3 generado de la Tabla 3.

De igual forma, ejecutando el mismo algoritmo sobre la Tabla 4 ($\Re^2$) obtendremos lo siguiente:

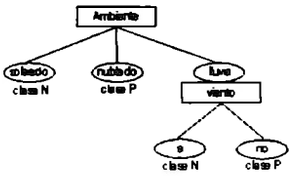


Figura 6.7. Árbol ID3 generado de la Tabla 4.

A pesar de que m^k =7 para k = 1,2, es claro que ambos árboles son distintos. Veamos porque los resultados difieren. Al iniciarse el proceso de construcción en $\Re^2$, ID3 empieza buscando el atributo que presenta mayor ganancia, utilizando los conjuntos de entrada siguientes (en nuestro ejemplo, Γ^2 se inicia con el conjunto vacío, pero en caso que no se desee considerar algunos atributos en el proceso, estos se eliminan de Ψ desde el principio, pasando a formar parte de Γ^2):

$\Psi^2 = \{ A_1^2 = \text{Ambiente}, A_2^2 = \text{Temperatura}, A_3^2 = \text{Humedad}, A_4^2 = \text{Viento}\}$
$\Gamma^2 = \{\}$

Calculada las ganancias de los atributos Ψ^2, se determina que el atributo ganador es A_g^2 = *Ambiente*, el cuál se convierte en la raíz del árbol y los valores de su dominio forman sus ramas (soleado, nublado, lluvia), donde $\beta_1^2 = 3$.

Para *Ambiente - soleado*, los registros asociados a está rama son de la clase "N" al 100%, condición suficiente para detener el proceso de construcción de la rama. Sin embargo, en $\Re^1$ no se presenta la misma situación: al iniciarse el proceso de construcción, ID3 trabaja con los siguientes conjuntos de atributos (observemos que son los mismos que en $\Re^2$ por tratarse de una fragmentación horizontal primaria):

$\Psi^1 = \{ A_1^1 = \text{Ambiente}, A_2^1 = \text{Temperatura}, A_3^1 = \text{Humedad}, A_4^1 = \text{Viento} \}$
$\Gamma^1 = \{\}$

En base a estos conjuntos, se determina que el atributo ganador es A_g^1 = *Ambiente*, el cuál coincide con el atributo ganador del otro sitio, aunque en general esto no es necesario que ocurra. Con éste atributo se crea el nodo raíz del árbol, asociándose una rama para cada valor de su dominio (notemos que el dominio del atributo *Ambiente* para ambos sitios es el mismo). Para este sitio, los registros asociados a la rama *Ambiente - soleado* no son en su totalidad de una clase: dos son de la clase "P" y uno de la "N". Por ello, el proceso de construcción de está rama continúa, basándose en los atributos:

$\Psi^1 = \{ A_2^1 = \text{Temperatura}, A_3^1 = \text{Humedad}, A_4^1 = \text{Viento} \}$
$\Gamma^1 = \{ A_1^1 = \text{Ambiente} \}$

Para la segunda iteración del algoritmo, la cuál desarrollará dicha rama, sólo se trabajará con los registros que tienen en su atributo *Ambiente* el valor de *soleado*. Al continuar el proceso, se determina que el atributo ganador para éste conjunto de entrada es *Humedad*, con $\beta_3^1 = 2$ (*alta, normal*). Los registros que se estaban trabajando en este ciclo son asignados a cada una de las dos ramas, en donde el valor del atributo coincida con el valor de la rama. Con este paso, los registros de cada rama resultan ser de una sola clase, finalizando el proceso de construcción. Con lo anterior, queda claro que el crecimiento de la rama *Ambiente – soleado* para ambos sitios no fue el mismo.

Un situación similar de desequilibrio en el crecimiento de las ramas se presenta para la rama *Ambiente - lluvia* , ya que en $\Re^1$ solo fue necesario generar está rama para clasificar correctamente a los registros de este sitio, mientras que en $\Re^2$ se tuvo que generar un subárbol para lograr el mismo objetivo.

Lo anterior nos muestra que el crecimiento de las ramas en los sitios no se va a comportar de igual manera, por lo que surgen varias preguntas: ¿cómo conjugar los subárboles de cada sitio para construir uno solo que clasifique correctamente a los registros de cada sitio?, ¿con qué clases etiquetar a las hojas que se obtienen en el árbol que se construye en el servidor?, ¿qué hacer cuando en sitios diferentes, el proceso de construcción para una rama termina en diferentes niveles? Utilizando el ejemplo expuesto como base, iniciemos el diseño de nuestro algoritmo distribuido.

Sincronización en el inicio de las operaciones

Notemos que en el ejemplo expuesto, el primer cálculo que realiza cada sitio es generar un vector de ganancias. Dicho vector se basa exclusivamente en los datos de cada sitio, por lo que no requiere de ningún tipo de resultado proveniente del servidor. Con la finalidad de que los cálculos se inicien de manera sincronizada, nuestro servidor enviará un mensaje a cada sitio indicando el comienzo de las operaciones, pasando a esperar el primer conjunto de resultados provenientes de los sitios.

Una vez que el servidor tenga todos los vectores G^k de los "Y" sitios de la BDD, o al menos de aquellos sitios que se estén considerando para la iteración, por medio de la Ecuación 2 calculará al atributo global con mayor ganancia A_g.

En nuestro ejemplo, cada sitio inicia calculando las ganancias para cada atributo en Ψ, enviando éstos resultados al servidor. Los vectores de ganancia que cada sitio genera para la primera iteración se ilustran en la Figura 6.8.

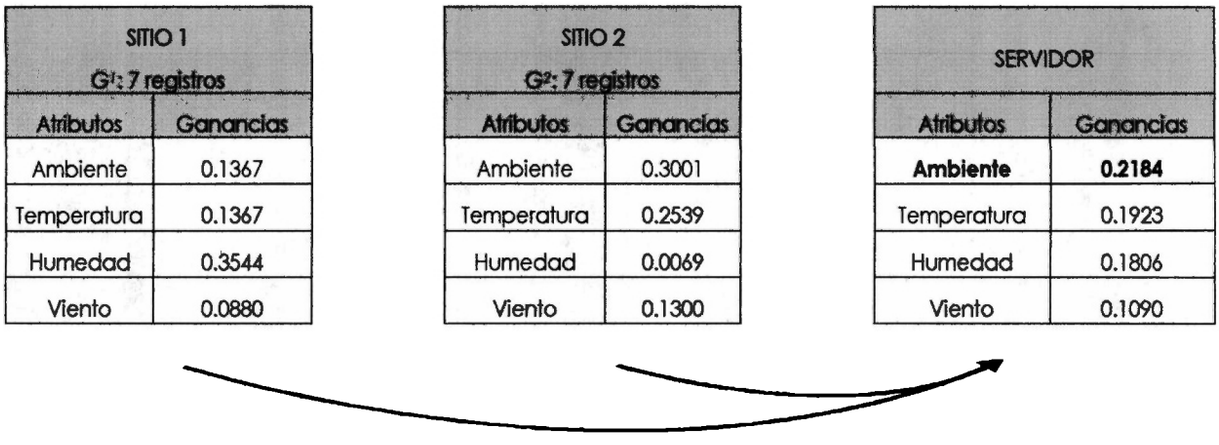


Figura 6.8. El servidor recibe y calcula la ganancia para la primera iteración.

En nuestro ejemplo, cada sitio inicia calculando las ganancias para cada atributo en Ψ, envíando estos resultados al servidor. Los vectores de ganancia que cada sitio genera para la primera iteración se ilustran en la Figura 6.8.

A la derecha se muestran los resultados que obtiene el servidor al calcular las ganancias globales de cada atributo. Para este caso, el atributo ganador es A_g = *Ambiente*. Este atributo se añade inmediatamente al árbol global que construye, siendo el nodo raíz por ser el primero que lo integra.

Ahora, para determinar el número de ramas que este nodo debe tener, se debe conocer el dominio del atributo ganador que se debe manejar para el mismo.

Manejo de los dominios de los atributos en el servidor

La arquitectura de construcción de una BDD nos permite conocer el dominio global para cada atributo (ver Figura 4.4). El dominio global de un atributo se puede obtener de la unión de los dominios locales de dicho atributo en cada sitio. Lo anterior significa que:

$$\forall A_x \mid D_x = \bigcup_{k=1}^{Y} D_x^k$$

Los dominios locales consideran la totalidad de los registros contenidos en cada sitio, por lo cuál, el dominio global de un atributo contendrá cualquier posible valor que pueda tomar cada atributo. Esta característica hace pensar que los dominios que debe considerar el servidor para determinar cuantas ramas debe tener cada nodo son precisamente los dominios globales. Sin embargo, lo anterior no es tan cierto.

Un árbol ID3 establece una serie de pruebas que determinan qué registros serán asociados a cada rama del árbol. Por ejemplo, para el árbol de la Figura 5.3, los registros asociados a la hoja que se

ilustra más a la izquierda deben cumplir que en el atributo *Ambiente* deben tener el valor de *soleado* y en el atributo *Humedad* el valor de *alta*. Con ello, sólo algunos registros de la Tabla 2 cumplirán con está condición. Si analizamos estos registros, el dominio del atributo *Temperatura* ya cambio con respecto al dominio del conjunto original de registros, ya que ahora no contendrá el valor de *baja*. Con lo anterior podemos afirmar que el dominio de cada atributo basado en el conjunto original de registros no necesariamente será válido para cada subconjunto de registros asociados a cada rama. Lo anterior debe tomarse muy en cuenta para nuestro algoritmo distribuido, ya que puede provocar que algunos valores en el dominio del atributo ganador ya no existan en ciertos sitios, producto de las fragmentaciones que el método realiza sobre los registros. Esto, a su vez, induce a un posible intercambio de mensajes inútiles entre el servidor y los sitios. Ilustremos este hecho mediante un ejemplo.

Consideremos una BDD integrada por 3 sitios, así como se ilustra en la Figura 6.9. Supongamos que se inicia el proceso de construcción del árbol global, donde después de finalizar la primera iteración, el atributo ganador resulta ser A_a. En la Figura 6.9 se ilustran, para cada sitio, el árbol que se genera al considerar este atributo como raíz, así como sus respectivas ramas que se crean en base a los dominios locales de dicho atributo. Supongamos que el servidor entra a la iteración que desarrollará a la rama A_a – *Valor2*. Si el servidor no tiene conocimiento de los dominios locales que se manejan en cada sitio, enviará un mensaje a cada sitio de la forma A_a – *Valor2*, indicando por cuál rama se continua el proceso de construcción del árbol. Sin embargo, notemos que el *Valor2* del atributo A_a sólo se encuentra en el sitio 1. Este sitio generará de forma normal el correspondiente vector de ganancias de la iteración, así como se ilustra en la figura, enviándolo al servidor. Sin embargo, los sitios 2 y 3 no tienen dicho valor, por lo que deben de informar al servidor del hecho por medio de un mensaje, que en el ejemplo se representa por medio del símbolo ∅. Estos mensajes no son fructíferos, ya que no producen resultados que aporten a la construcción del árbol.

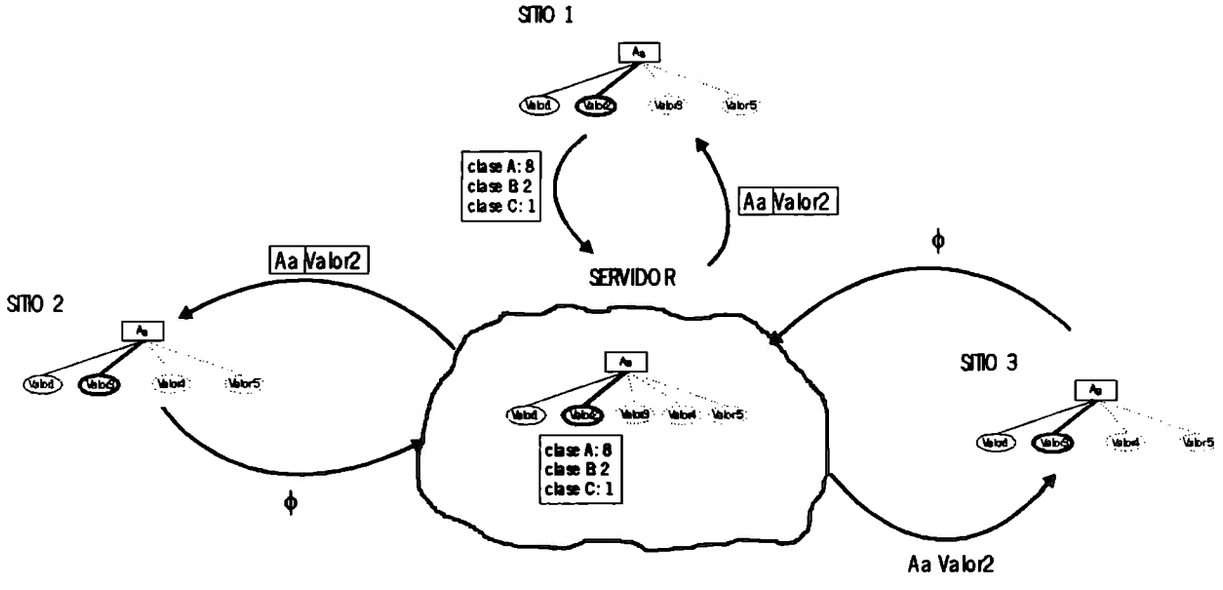


Figura 6.9. Consenso del servidor cuando un solo sitio participa en la clasificación de una rama.

Es evidente que la probabilidad de que el servidor envíe mensajes infructuosos a los sitios será mayor para nodos que tengan una considerable profundidad en un árbol ID3, ya que entre más

aumente la profundidad, mayor serán las pruebas exigidas para que un registro se asocie a una rama.

Lo ideal es evitar este tipo de mensajes. La pregunta es ¿cómo evitarlos? La solución al problema ya expuesto es que el servidor conozca de manera explícita el dominio del atributo ganador que se obtenga en la iteración en curso. Sin embargo, ello implica que los sitios le envíen de manera explícita dicho dominio. Con ello, al momento de estar desarrollando el árbol, podría verificar que el valor de la rama que va a desarrollar pertenezca a cada dominio local de cada sitio. De lo anterior, se podría determinar si es necesario enviar un mensaje al sitio para el desarrollo de la rama que se analiza o ahorrarse éste mensaje. La desventaja eminente de está opción es que enviar los dominios del atributo ganador al servidor aumenta el flujo de los datos.

Ahora ¿cuál de las dos estrategias inducirá un menor flujo de datos? La respuesta depende tanto de los patrones presentes en cada sitio como del tamaño final del árbol global. Es evidente que para cada nodo del árbol y a pesar de que se consideren nodos de la misma profundidad, la situación con respecto a los dominios será distinta. Por ello, analicemos que es lo que conviene para cada uno de ellos. Por una parte, podemos obtener la cantidad de datos que son necesarios transmitir si se opta que los sitios le informen al servidor del dominio del atributo ganador, dada por la expresión:

$$C_D = \sum_{i=1}^{Y'} | D_g^i |$$

Ecuación 3. Costo de transmitir los dominios locales de cada sitio con respecto al atributo ganador

Por otra parte, si denotamos por np^i al número de valores v que pertenecen al dominio global del atributo ganador A_g pero que no pertenecen al dominio local del mismo atributo en el sitio i considerando los registros asociados a la rama que se trabaja, el costo de los mensajes innecesarios entre el servidor y los sitios sería:

$$C_{MI} = 2\sum_{i=1}^{Y'} np^i$$

Ecuación 4. Costo de los mensajes innecesarios provocados por considerar el dominio global del atributo ganador

Suponiendo que el servidor tenga ambos costos, determinar que estrategia provoca un menor costo para cada nodo sería seleccionar el menor valor entre C_D y C_{MI}.

Como comentario, el autor considera que en una tendencia promedio para una BDD con sitios con un gran número de registros y un gran número de atributos, se esperaría que el árbol ID3 final sea de un tamaño considerable, tanto en el número de ramas por nodo como en la profundidad del mismo. Como a mayor profundidad en un árbol ID3, C_D tenderá a ser menor por contener menos valores en los dominios y C_{MI} tenderá a ser mayor por aumentar el número de mensajes innecesarios provocados por los valores que no pueden trabajar cada uno de los sitios,

se esperaría que considerar los dominios locales sería una mejor elección que manejar el dominio global en el servidor.

Por ejemplo, para el caso de la Tabla 2 que fue fragmentada en dos sitios, se puede calcular fácilmente que el esquema más apropiado para los nodos es el considerar los dominios globales en el servidor, el cuál requiere menos volumen de comunicación.

Rama a desarrollar en el proceso de construcción

Continuemos con nuestro análisis haciendo uso del ejemplo basado en la Tabla 2. Supongamos que el servidor recibe las ganancias correspondientes a la primera iteración, determinando que el atributo ganador es *Ambiente*. También establece que es mejor considerar los dominios globales del atributo, por inducir un menor flujo de datos en la red. Con estas consideraciones, el servidor genera el nodo raíz del árbol así como sus respectivas ramas, tal y como lo muestra la Figura 6.10:

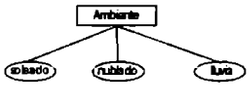


Figura 6.10. Árbol generado en el servidor después del primer paso.

Una vez generado el nodo raíz, el servidor debe determinar por cuál de las ramas se debe continuar el proceso de construcción. Para ello, este enviará a cada sitio un mensaje del tipo *atributo – valor*, donde *atributo* corresponde al atributo ganador y *valor* corresponde al valor del dominio a trabajar. Notemos que cuando se considere el dominio global, el mensaje que genere el servidor será enviado a todos los sitios y, en caso de que cada sitio haya enviado su dominio local, se puede verificar que el valor a trabajar pertenece o no al dominio de cada sitio. Para nuestro ejemplo, el servidor tendrá que analizar los valores *lluvia, nublado, soleado* asociados al dominio del atributo *Ambiente*. Si suponemos que analizaremos estos valores en orden alfabético, en primera instancia se generará el mensaje *Ambiente – lluvia*. Una vez recibido este mensaje en los dos sitios que integran a la BDD, estos generan la raíz del árbol local basado en el atributo *Ambiente* y empiezan a trabajar con la rama correspondiente a *lluvia*, así como se muestra en la Figura 6.11.

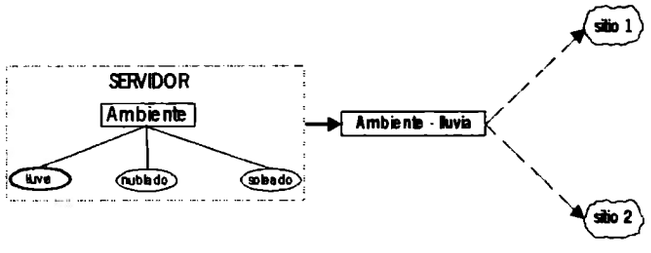


Figura 6.11. Mensaje del servidor a cada uno de los sitios

Por ello, en nuestro algoritmo, cuando un sitio reciba un mensaje del tipo *atributo – valor*, estos deben crear un nodo para el atributo en cuestión, si no ha sido creado, así como la rama correspondiente al valor indicado. Hecho esto, se debe calcular el vector de ganancias de los

atributos que no se han trabajado en base a los registros que se asocien a la nueva rama, para finalmente enviar un mensaje al servidor informándole de este resultado, así como del número de registros que se consideran en cada sitio.

Una estrategia que puede resultar útil es enviar en conjunto con el vector de ganancias, los costos de considerar el dominio global así como el local de cada atributo para cada sitio. Con ello, una vez que el servidor determine el atributo ganador, puede calcular inmediatamente el dominio que será asociado a dicho atributo.

Generación de resultados en el servidor: planteamientos

Para cada iteración, si el servidor recibe de cada sitio un vector de ganancias, este deberá calcular el atributo ganador en base a los vectores de ganancia recibidos. Sin embargo, no siempre se obtendrá un vector de ganancias de cada sitio, ya que cada vez que el algoritmo ID3 determina que todos los registros de una rama son de clase pura, se detiene el proceso de construcción de dicha rama. Para nuestro algoritmo distribuido, si un sitio presenta tal situación, de igual manera se debe detener el proceso de construcción de la rama e informar al servidor la situación. Ahora, ¿qué resultados son necesarios transmitir si un sitio ha finalizado la construcción de una rama? y ¿qué estrategia debe seguir el servidor cuando se presenta tal situación?

Resultados a transmitir por parte de los sitios que finalizan el crecimiento de una rama

Cuando un sitio finaliza el crecimiento de una rama, ID3 obtiene la clase asociada a dicha rama. Como los resultados finales son analizados por el servidor, es necesario enviar a éste la clase que se ha encontrado en el sitio "*k*". Sin embargo, ya se comentó que no siempre los registros asociados a la rama que se ha terminado de desarrollar son totalmente de una clase. En el sitio "*k*" se puede inmediatamente etiquetar al nodo de la rama con la clase de mayor cobertura, conociendo incluso su precisión, pero para el caso del servidor, es necesario que éste conozca el total de las clases que se asociaron a la rama, para que en un proceso posterior, a través de un consenso que involucre a todos los resultados de los sitios que participaron en el desarrollo de la rama, se obtenga la clase con mayor cobertura. Por ello, se considera necesario que cada sitio, al momento de finalizar el crecimiento de una rama, envíe al servidor un vector de coberturas de las clases que se asociaron a dicha rama. A dicho vector lo representaremos por medio de C^i, donde *i* indica el sitio donde se origina el vector, y cada elemento de dicho vector constará de un identificador que indique la clase y un valor asociado que represente la cobertura de dicha clase.

Ahora analicemos como debe operar el servidor si un sitio o sitios terminan el desarrollo de una rama. Aquí se van a presentar dos casos: todos los sitios de la BDD concluyen el crecimiento de la rama en el mismo nivel o, por otra parte, algunos concluyen el crecimiento pero otros continúan con el mismo. Consideremos en primer instancia que todos los sitios terminan en el mismo nivel el crecimiento de la misma rama.

Comportamiento del servidor cuando todos los sitios finalizan el crecimiento de una rama

Cuando la totalidad de los sitios considerados para una iteración finalizan el crecimiento de la rama indicada, éstos le enviarán sus correspondientes vectores de coberturas de clases al servidor.

Con dichos vectores, el servidor debe igualmente finalizar el crecimiento de la rama, etiquetandola con la clase de mayor cobertura, basada en los vectores que los sitios le han enviado. Para ilustrarlo, consideremos el ejemplo de la Figura 6.12, donde la BDD está compuesta de 3 sitios. Supongamos que el servidor entra en la iteración que desarrollará a la rama A_a – *Valor*$_2$. Después de considerar los resultados de cada sitio, se determina que el atributo ganador para está rama es A_b, asociándola a la misma. Supongamos que ya se han analizado todos los valores del dominio de este atributo, excepto *Valor4*. Por ello, el servidor envía un mensaje indicándole a cada sitio que este valor es el que a continuación se debe de analizar, el cuál está remarcado en negro en la Figura 6.12.

Consideremos que cada sitio determina que ya no se puede desarrollar más la rama en análisis. Por ello, éstos generarán como respuesta vectores de cobertura de clases. Notemos que la dimensión de cada uno de estos vectores C^i varía dependiendo de las clases que se encuentren asociadas al nodo, siendo el caso óptimo cuando sólo existe una clase asociada por el hecho de transmitirse al servidor un menor número de datos. Por el contrario, el peor de los casos se presenta cuando en la hoja queden asociados la totalidad de las clases presentes en A_p. Para nuestro ejemplo, el sitio 2 sólo transmite los datos de una clase pura "A" (que sería el caso óptimo), mientras que en los sitios 1 y 3 es necesario transmitir los datos de las clases "A", "B" y "C".

Una vez transmitidos los vectores, el servidor analizará cuál es la clase con mayor cobertura. Para el ejemplo, podemos observar que la clase con mayor cobertura es "A" con 28 registros, la cuál es utilizada para etiquetar la rama correspondiente, manejando un error de precisión de 8/34.

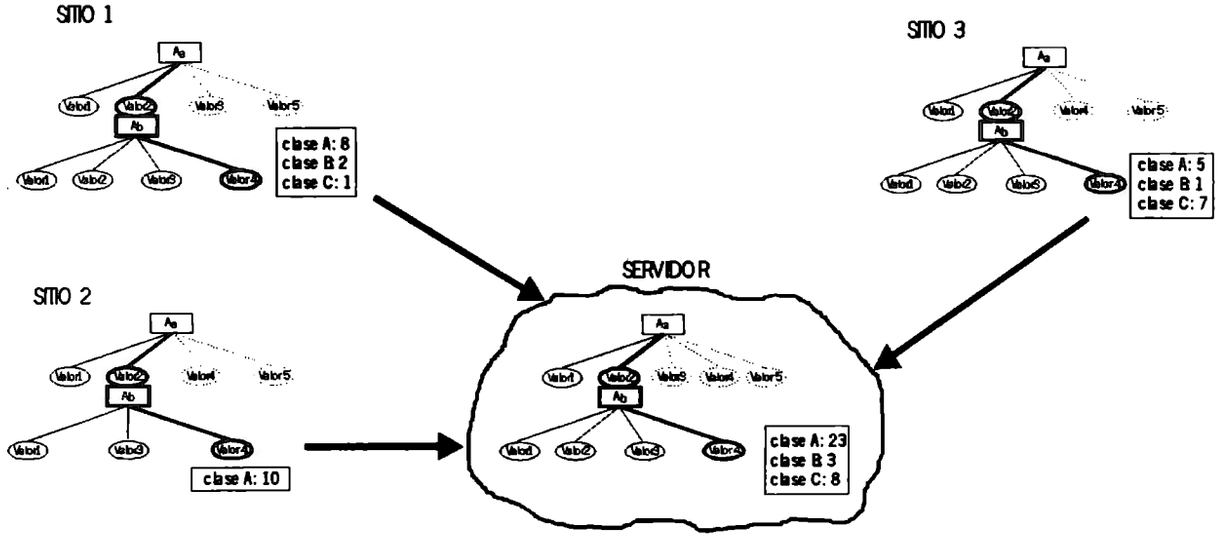


Figura 6.12. Consenso del servidor para determinar la clase de una rama si todos los sitios finalizaron en la misma iteración.

Comportamiento del servidor cuando un conjunto de sitios finaliza el crecimiento de una rama y otros no: caso con patrones disjuntos

Pasemos a analizar el caso cuando un conjunto de sitios finalizan el crecimiento de una rama, que denotaremos con $\aleph^F$, pero otro conjunto aún la puede desarrollar, denotada por $\aleph^N$. Lo que se

busca es integrar de alguna manera los resultados que ambos conjuntos de sitios generan. Integrarlos significa buscar la relación existente entre los patrones de ambos conjuntos. Es claro que mientras los sitios $\aleph^N$ no concluyan el crecimiento de la rama, no se puede conocer que estructura final tendrá el subárbol y con que clase se etiquetará a cada hoja de dicho árbol gnerado. Por otra parte, el subárbol que se construya se basará exclusivamente en los patrones de los sitios $\aleph^N$ y no contemplara a los sitios $\aleph^F$. Por tanto, debemos de pensar como incluir la clase que obtuvimos de los sitios $\aleph^F$ a la estructura del árbol que se derivó de los sitios $\aleph^N$.

Una estrategia que se puede adoptar es buscar algún patrón que identifique al conjunto de sitios $\aleph^F$ y que a su vez, no esté presente en los sitios $\aleph^N$. Notemos que está condición no siempre se dará, pero si es que existe, la mejor manera de separar unos de otros es por medio de un atributo A_x, el cuál permita agrupar en las distintas ramas a conjuntos exclusivos de los sitios $\aleph^F$ o $\aleph^N$, pero no ambos en la misma rama. Ilustremos la idea anterior por medio de un ejemplo.

Supongamos una BDD con 2 sitios representada por $\Re^1$ y $\Re^2$, las cuales contienen los datos de la Tabla 5 y la Tabla 6 respectivamente, donde el atributo clasificador A_p es *recommended lents*. Notemos que $\Re^1$ sólo contiene un tipo de clase para todos sus registros, *none*, lo que hace imposible construir un árbol para este sitio. Sin embargo, no sucede lo mismo para $\Re^2$, ya que este conjunto de datos tiene 3 tipos de clases distintas, permitiendo generar un árbol de clasificación ilustrado en la Figura 6.13.

Observemos que éste árbol contiene hojas etiquetadas con la clase *none*, la misma que se encontró en los sitios $\Re^1$. Si el recorrido de la raíz a estas hojas fuera el mismo que se requiriera para clasificar al total de los registros del sitio $\Re^1$, este árbol serviría para clasificar de manera global a todos los registros no importando el sitio. Sin embargo, lo anterior no siempre sucederá.

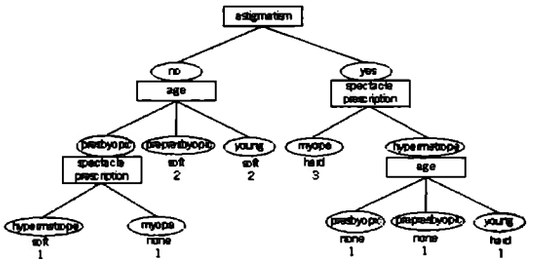


Figura 6.13. Árbol de clasificación para la Tabla 6 aplicando el algoritmo ID3 centralizado.

Por ello, el servidor deberá considerar al momento de la construcción la mejor forma de discernir los patrones de cada sitio. Usando este ejemplo, analicemos en que concepto el servidor se puede apoyar para identificar patrones de cada sitio. Al iniciarse la construcción del árbol para la BDD, los conjuntos de entrada para $x = 1, 2$ son los siguientes:

$\Psi^x = \{ D_1^x = age,\ D_2^x = spectacle\ prescription,\ D_3^x = astigmatism,\ D_4^x = tear\ production \}$

$\Gamma^x = \{\}$

Ambos sitios manejan exactamente los mismos conjuntos, y por ende, serán los que el servidor maneje. En la siguiente tabla se muestra el dominio de cada uno de estos atributos tanto para $\Re^1$ como para $\Re^2$.

Age		*spectacle prescription*		*Astigmatism*		*tear production*	
$\Re^1$	$\Re^2$	$\Re^1$	$\Re^2$	$\Re^1$	$\Re^2$	$\Re^1$	$\Re^1$
Young	Young	Myope	myope	No	No	Reduced	Normal
Pre Presbyoptic	Pre Presbyoptic	hypermetrope	hypermetrope	Yes	Yes		
Presbyoptic	Presbyoptic						

Al iniciarse el proceso de construcción en la BDD, el primer mensaje del servidor a los sitios consistirá en la sincronización de las operaciones. Cada sitio, al recibir la señal del servidor, tratará de construir el primer vector de ganancias de acuerdo a la entropía de los registros. Sin embargo, el sitio $\Re^1$ no envía un vector de ganancias, ya que todos los registros de éste sitio son de la clase *none*, por lo que la respuesta del sitio al primer mensaje del servidor será la clase que representa al sitio así como el número de registros del mismo, 12. En cambio, el sitio $\Re^2$ responde con un vector de ganancias G^2 dado por:

G^2: *12 registros*	
Age	0.1395
spectacle prescription	0.0954
Astigmatism	0.7704
Tear production	0

Es evidente que el servidor, al obtener de un sitio un vector de ganancias, este puede calcular inmediatamente un atributo ganador. Sin embargo, está no es la forma inmediata con la cuál debe responder el servidor, ya que debe considerar el vector de coberturas que generó uno de los sitios en la misma iteración.

Notemos un hecho interesante que ocurre en la tabla de dominios que se ilustró con antelación: para los atributos *age, spectacle prescription* y *astigmatism,* el dominio de estos en cada sitio es el mismo. Sin embargo, no sucede lo mismo para el atributo *tear production*: por una parte, todos los registros de $\Re^1$ solo tienen el valor *reduced*, mientras que para $\Re^2$ todos sus registros tienen el valor *normal*. Está diferencia o característica que presentan cada uno de los registros de cada sitio identifican de manera clara a cada registro de cada clase, existiendo un patrón que no es común, es decir, un patrón disjunto.

Para reflejar un patrón disjunto en un árbol ID3, es necesario incluir un nodo en este árbol correspondiente al atributo que identifica dichos patrones disjuntos. Este nodo debe ser incluido tanto en el árbol que construye el servidor, así como en los árboles de los sitios que están participando en el desarrollo de la rama en la cuál se presentó tal situación. Por ello, cuando se presenta la situación en la cuál los sitios son divididos en los conjuntos $\aleph^N$ y $\aleph^F$, será necesario

que el servidor le indique a los sitios $\aleph^N$ agregar a la rama que se trabaja el nodo correspondiente al atributo que ayudó a identificar a los patrones disjuntos.

Identificar un patrón disjunto depende directamente del análisis de los dominios de los atributos. Notemos que para este caso, es necesario que el servidor conozca los dominios de los atributos presentes en cada sitio y no considere el dominio global de los mismos. Ello conlleva a que, cuando un sitio o sitios finalicen el crecimiento de una rama, el total de los sitios deben de transferir los dominios locales asociados a cada atributo. Notemos que este esquema genera el mismo flujo de datos que cuando el servidor determina manejar los dominios locales de cada sitios.

El atributo que se elija para separar a los patrones disjuntos genera dos tipos de ramas: las que se asociarán a los sitios $\aleph^F$ y las correspondientes a los sitios $\aleph^N$. Las ramas asociadas a los conjuntos $\aleph^F$ se etiquetan inmediatamente con la clase que se obtenga de los vectores de coberturas de clases que estos sitios le han enviado al servidor. Por otra parte, ¿qué pasa con las ramas asociadas a los sitios $\aleph^N$? Para cada una de estas ramas, el servidor tendrá que enviar un mensaje del tipo *atributo – valor* a los sitios $\aleph^N$, con el fin de que estos desarrollen dichas ramas. Ilustremos las ideas expuestas por medio de un ejemplo.

En el ejemplo que iniciamos planteando para este análisis, al recibir el servidor los resultados de cada sitio, detecta que uno de ellos ha finalizado el crecimiento del árbol (de hecho, el árbol aún no contaba ni siquiera con un nodo) y el otro operó de manera normal, es decir, generó un vector de ganancias. Al suceder esto, se verifica si existen patrones disjuntos entre los sitios $\aleph^N$ y $\aleph^F$. Al estudiar los dominios de los atributos, encuentra que el atributo *tear production* separa los patrones de los sitios $\aleph^N$ y $\aleph^F$. Este pasa a formar parte del árbol que se construye, siendo la raíz por ser el primero que integra al árbol. En caso que ya hubiera existido un subárbol previo, este nodo se asociaría a la rama que se estaba trabajando. Lo anterior se ilustra en la Figura 6.14.

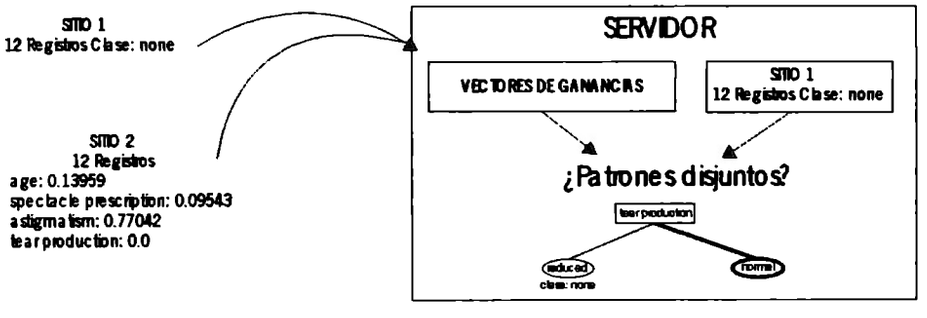


Figura 6.14. Ejemplo de cómo distinguir patrones disjuntos en el servidor, si es que existen

Inmediatamente el servidor propaga a los sitios $\aleph^F$ de la BDD este resultado, con el objetivo de que actualicen sus árboles locales.

Posteriormente, el servidor asigna a la hoja de la rama *reduced* la clase que se generó del sitio $\Re^1$, es decir *none*. Notemos que está rama en si representa a los patrones del sitio $\Re^1$, la cuál ya no se desarrollará por haber terminado el proceso de construcción del sitio. Por otra parte, para la rama

normal, remarcada en negro, se continuará el proceso de construcción basándose el servidor exclusivamente con el sitio 2.

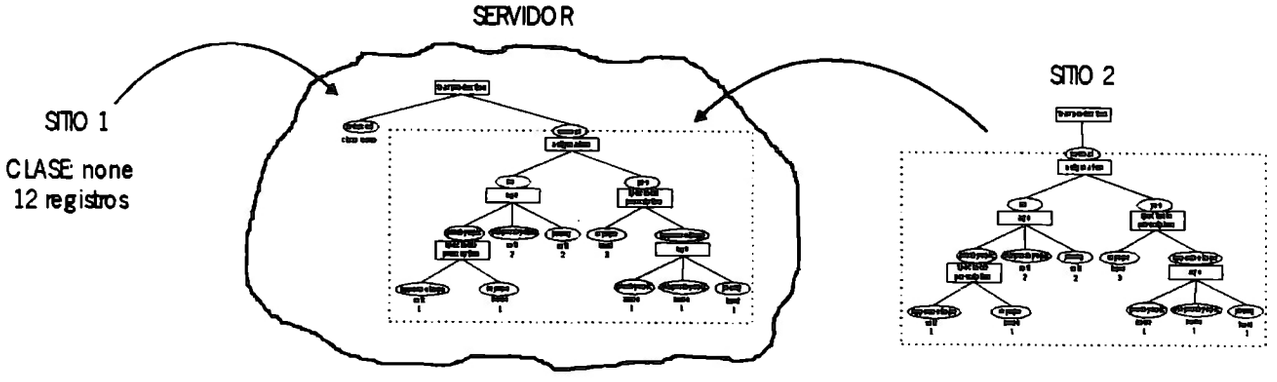


Figura 6.15. Árbol generado en el proceso distribuido basado en los datos de las tablas 5 y 6.

Notemos que este sitio es el único que desde ese punto participará en el desarrollo de la rama. Por ende, el subárbol que se genere en el sitio 2 va a ser el mismo que aparezca asociado a la rama *normal* del árbol del servidor, así como lo ilustra la Figura 6.15.

Comportamiento del servidor cuando un conjunto de sitios finaliza el crecimiento de una rama y otros no: caso sin patrones disjuntos

Ahora analicemos el caso cuando los registros de los sitios $\aleph^F$ tienen patrones comunes con los registros de los sitios $\aleph^N$. Los sitios $\aleph^N$ generarán a partir de un nodo λ un subárbol que denominaremos δ^N, en el cuál no participan los sitios $\aleph^F$ por haber concluido el crecimiento de la rama correspondiente al nodo λ. Sin embargo, los registros asociados a la rama en la cuál han finalizado los sitios $\aleph^F$ si podrán ser ubicados en diversas ramas del árbol δ^N. Por ejemplo, para la imagen de la Figura 6.15., se había dicho que los registros del sitio 1 sólo se ubicaban en la rama *reduced* del atributo *tear production*. Si suponemos que no existe ningún patrón disjunto entre los atributos de los sitios, y el nodo raíz continua siendo el mismo, los registros del sitio 1 no sólo se ubicarían en la rama *reduced*, también existirían registros de este sitio que se asociarán a la rama *normal*. Lo anterior obliga a plantear la forma en la cuál se deben etiquetar los nodos hoja del árbol δ^N.

La forma más sencilla y económica con respecto al flujo de datos, es hacer que el servidor almacene los vectores de cobertura de los sitios $\aleph^F$ (asociados al nodo λ) y, una vez que finalicen los sitios $\aleph^F$ de desarrollar el árbol, el vector de coberturas que obtengan se mezcle con el respectivo vector de los sitios $\aleph^F$, obteniendo una clase ganadora. Sin embargo, está estrategia tiene un fuerte problema.

Notemos que entre más aumenta la profundidad de un árbol de clasificación, generalmente el número de registros asociados a los nodos de más profundidad en el árbol será menor que el número de registros asociados a los niveles superiores del árbol. Para ilustrarlo, consideremos el árbol de la Figura 6.16, tomando como base la rama remarcada en negro. Aquí podemos observar claramente que entre más aumente la profundidad en el árbol, mayor será la cantidad de pruebas

que deben de cumplir los registros para asociarse a un nodo. Por ello, un nodo de profundidad *n* tenderá a tener menos registros asociados que un nodo de nivel superior.

Lo anterior también repercute al considerar los vectores de cobertura para cada nodo, ya que si comparamos los vectores asociados a un nodo de nivel superior e inferior, la clase predominante tenderá a ser la asociada al vector del nodo de nivel superior, por tener una mayor cobertura.

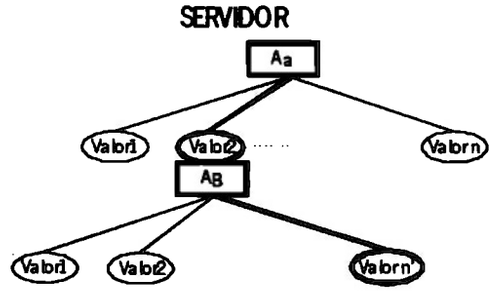


Figura 6.16. Entre más aumente la profundidad de un árbol, menor será el número de registros asociados a cada nodo en el nivel más profundo.

Una estrategia que evita el problema ya planteado, es que el servidor no se quede con el vector de coberturas de los sitios $\aleph^F$ asociados al nodo λ. En vez de ello, se espera a que los sitios $\aleph^N$ finalicen el crecimiento de la rama que desarrollan y una vez finalizado, el servidor le envíe a los sitios $\aleph^F$ la totalidad de las pruebas necesarias para llegar a cada nodo hoja del árbol δ^N. De está manera, los sitios $\aleph^F$ estarían aportando un vector de ganancias basado en registros que cumplan cabalmente las pruebas asociadas a cada hoja del árbol y con ello, la elección de la clase ganadora sería justa. Desde el enfoque asociado al flujo de datos, es obvio que está estrategia no induce a un aumento significativo en el flujo, ya que la cantidad de datos para representar la serie de reglas para cada hoja no es grande.

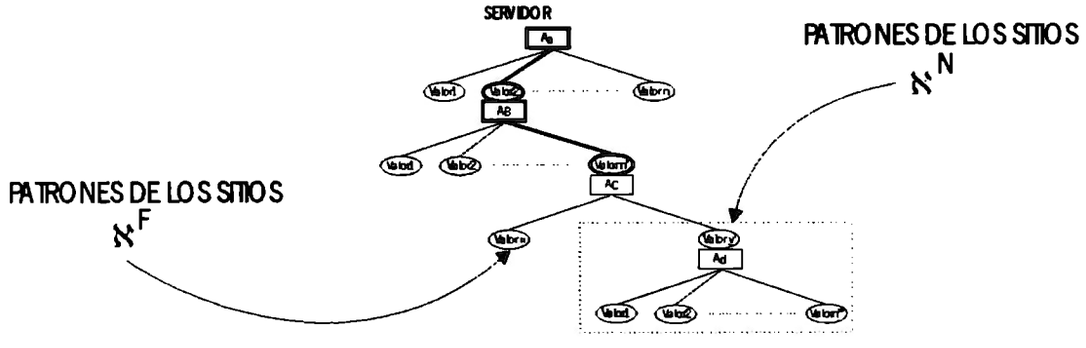


Figura 6.17. Representación de cómo después de cierta profundidad, aparecen patrones disjuntos entre los conjunto de los sitios $\aleph^N$ y $\aleph^F$.

Otra estrategia diferente a las dos ya expuestas es buscar, para cada nodo del árbol δ^N, patrones disjuntos, para separar los registros de los sitios $\aleph^F$ y $\aleph^N$ no hasta los nodos hoja, sino en nodos de nivel superior. Lo anterior se fundamenta en el hecho de que, entre más aumente la profundidad de un árbol, los registros de los sitios $\aleph^F$ y $\aleph^N$ pueden presentar patrones disjuntos que en ramas de nivel superior no existían, ya que los dominios de los atributos tienden a cambiar de un nivel *n* a un nivel *n+1*. Si para un nivel cualquiera no se detectan patrones disjuntos, se debe continuar el desarrollo de la rama, importando para cada iteración los dominios de los

atributos, con el objetivo de detectar patrones disjuntos. En caso de detectar patrones disjuntos, se podrá separar los patrones de los sitios $\aleph^F$ y $\aleph^N$, con lo cuál, el árbol que a partir de ese momento se desarrolle, ya no arrastrará los patrones de los sitios $\aleph^F$, así como se ilustra en la Figura 6.17.

Está última estrategia es más precisa que las dos anteriores, ya que para cada nodo, se está verificando los dominios de cada atributo en cada sitio, identificando con precisión cualquier variación en los patrones de los sitios. Sin embargo, para llevarla a cabo, será necesario, por una parte, que cada sitio $\aleph^F$ construya de manera paralela el árbol que desarrolla el servidor, con el objetivo de poder obtener el dominio asociado a cada nodo de dicho árbol, y por otra parte, tanto los sitios $\aleph^F$ y $\aleph^N$ deben enviar al servidor los vectores de dominios. Por tanto, a pesar de favorecer a la obtención de resultados más precisos, es palpable que el problema que presenta está estrategia es el aumento en el flujo de información.

Consecuentemente, la pregunta lógica es ¿cuánto aumenta? Esta pregunta no es tan sencilla de responder, ya que los vectores que envíen los sitios $\aleph^F$ y $\aleph^N$ con respecto a los dominios de los atributos pertenecientes a Ψ, variarán de tamaño dependiendo de cuantos valores distintos contengan los registros asociados a la rama en cuestión para cada uno de los atributos. El peor de los casos se presentará cuando, a pesar de pasar de un nivel "*n*" a un nivel "*n+1*" en profundidad, la cantidad de valores distintos se mantiene constante para cada atributo y además, no se encuentran patrones disjuntos en el desarrollo del árbol que permitan evitar el flujo de los dominios.

La expresión que permite calcular el flujo de datos para un nodo de está última estrategia se presenta a continuación. Esta expresión constará de dos partes: una correspondiente a los sitios $\aleph^N$ y otra para los sitios $\aleph^F$, donde la suma de ambos dará la cantidad de bytes a transmitir, como lo muestra la siguiente ecuación (en esta ecuación no se incluyen los costos de manejar los dominios):

$$\sum_{\Upsilon^j \in \aleph^F}^{j} \left[\sum_{A_i \in \Psi^j}^{i} (T_D + |D_i^j| * T_D) + 2 * |C^j| * T_D \right] + \sum_{\Upsilon^j \in \aleph^N}^{j} \left[|G^j| + T_N + \overline{\sum_{A_i \in \Psi^j}^{i} (1 + |D_i^j| * T_D)} \right] \text{ bytes}$$

Ecuación 5. Cálculo del flujo de información de los sitios de una BDD al servidor requerido para cada nodo del árbol que se construya.

La sumatoria correspondiente a los sitios $\aleph^F$ se compone de los siguientes elementos: la sumatoria correspondientes a los atributos $A_i \in \Psi^j$ calcula la dimensión en bytes de cada dominio de los atributos presentes en el sitio, enviando el identificador del atributo en cuestión así como su dominio. Además, cada sitio $\aleph^F$ debe enviar el vector de coberturas de clases de los registros asociados a la rama que se esté analizando, donde cada elemento del vector debe llevar el identificador de la clase así como su cobertura. Para los sitios $\aleph^N$ la expresión considera el costo de enviar el vector de ganancias al servidor así como el costo de informar del número de registros presentes en el sitio, representado por T_N. Notemos que en ocasiones será necesario enviar el vector de dominios (caso cuando no se han detectado patrones disjuntos a partir de un nodo λ), por lo cual, esta expresión considera el flujo para este caso, remarcado con la barra superior.

La estrategia que verifica los dominios en cada nivel será la utilizada en el presente trabajo, debido a que la precisión de sus resultados es buena y el flujo de los datos, tenderá a ser decreciente entre más aumente el tamaño del árbol (por disminuir el número de atributos a considerar para cada nivel).

Con todo lo anterior, estamos en condiciones de presentar el algoritmo que obtenga un árbol de clasificación ID3 de una BDD, el cuál tomará en cuenta todas las situaciones que ya se han citado. Este se explica en el siguiente apartado.

6.3 ID3 modificado para un sistema distribuido

El algoritmo que se presenta a continuación consta de dos partes: un código exclusivo para el servidor, y un código para los sitios de la BDD. Es importante establecer que el código para los sitios es el mismo para todos.

Estos códigos están basados en las consideraciones hechas a lo largo de la sección 6.2., las cuales se resumen a continuación:

- El algoritmo está diseñado bajo el esquema cliente- servidor, donde el servidor se encargará de dirigir la construcción del árbol ID3. El resultado final sólo lo tendrá el servidor.

- El servidor, para determinar el atributo ganador, debe realizar un consenso basado en los vectores de ganancia que cada sitio le ha enviado. Para ello, debe considerar la ganancia que cada atributo presentó en cada sitio, así como la cobertura de los mismos, para obtener finalmente una ganancia global que represente a dicho atributo, siendo la ganancia ganadora la de mayor valor entre todas las posibles ganancias.

- El servidor, para determinar cuantas ramas tendrá cada nodo del árbol, deberá considerar el dominio derivado de la estrategia que provoque un menor flujo de datos tomando como base la Ecuación 3 y la Ecuación 4.

- Para sincronizar las operaciones entre los diversos sitios, el servidor, una vez que determina el atributo ganador, debe enviar un mensaje con la estructura *atributo – valor* a cada sitio, donde *atributo* es el atributo ganador y *valor* es el valor del dominio con el cuál se continuará la construcción. Si el servidor conoce el dominio local de cada sitio, verificará que dicho valor se encuentre en el dominio del atributo del sitio y de no estar, no envía mensaje alguno a dicho sitio.

- Una vez que cada sitio ha recibido una instrucción del servidor del tipo *atributo – valor*, estos deben generar los resultados para la siguiente iteración. Se identifican dos posibles estructuras de resultados:

    a) Si el sitio determina que aún se puede desarrollar la rama que le ha indicado el servidor, este le regresará un paquete de resultados, conteniendo un vector de ganancias de los atributos, así como el número de registros analizados en la iteración.

b) Si el sitio determina que los registros asociados a la rama que se analiza cumple con cierto error o son de clase pura, éste detiene el crecimiento de dicha rama, enviando al servidor un vector de coberturas de clases, el cuál contiene cada clase asociada a la rama, así como su respectiva cobertura. Además, le informa del dominio de los atributos que aún no se han analizado, el cuál puede ser usado para determinar algún patrón disjunto en el servidor.

- Esta consideración se enfoca al caso cuando todos los sitios concluyen el crecimiento de la rama que se está trabajando. En tal situación, el servidor se enfocará a etiquetar al nodo de la rama trabajada con la clase de mayor cobertura global, basándose en los vectores de cobertura que cada sitio le ha enviado.

- Este punto hace referencia a cómo operar en el servidor cuando un conjunto de sitios finaliza el crecimiento de una rama ($\aleph^F$) y otro conjunto aún no finaliza ($\aleph^N$), existiendo entre ambos conjuntos algún patrón disjunto. En caso de ser así, el servidor añade a la rama que se está desarrollando el atributo que permite discernir ambos conjuntos de sitios, informándoles a todos los sitios cuál es el atributo elegido para que actualicen su árbol local. Acto seguido, y utilizando los vectores de cobertura que los sitios $\aleph^F$ le han enviado al servidor, éste etiqueta las ramas asociadas a estos sitios con una clase. El resto de las ramas se continuará desarrollando, solo participando los sitios $\aleph^N$ en su construcción.

- Esta consideración determina cómo operar cuando se tiene un conjunto de sitios $\aleph^N$ y $\aleph^F$ sin patrones disjuntos a partir de un nodo "*z*". Para este caso, el servidor desarrollará un árbol δ^N a partir de dicho nodo (que será la raíz del árbol), el cuál se basará exclusivamente en los vectores de ganancias que los sitios $\aleph^N$ generen para cada iteración. En el proceso de construcción del árbol, los sitios $\aleph^F$ estarán enviando al servidor tanto el vector de coberturas como de dominios para la rama que se desarrolle, con el fin de que el servidor verifique si no existen patrones disjuntos. En caso de existir patrones disjuntos a partir de un nodo "*w*", se operará de acuerdo a la consideración 7. En caso de que no se encuentre ningún patrón disjunto y el desarrollo de la rama llegue al final, la clase que se asocie a dicha rama tomará en cuenta los vectores de cobertura tanto de los sitios $\aleph^N$ como $\aleph^F$.

6.3.1 ID3 para una BDD: algoritmo correspondiente al servidor

Iniciemos la exposición con el algoritmo correspondiente al servidor. Este se dividirá en varios módulos, donde el módulo principal se presenta en el Algoritmo 5.

```
ENTRADA
Ψ: Conjunto de atributos globales
S = {S¹, S², ..., S^r}: Conjunto de sitios perteneciente a la BDD (identificador de cada sitio)
SALIDA:
δ: Árbol de clasificación para la BDD

1. Inicio()
2. { Arbol δ;
3.   Para todo i tal que S^i ∈ S:
4.     enviar un mensaje de inicio de operaciones a S^i.
5.   δ ← Recursivo(S, ∅, Ψ, 1);
6.   Para todo i tal que S^i ∈ S:
7.     enviar un mensaje de finalización de operaciones a S^i.
8. }
```

Algoritmo 5. ID3 para una BDD: algoritmo correspondiente al servidor. Módulo "Principal"

En este módulo se declara una variable δ, la cuál almacenará el resultado final del árbol que se construya. La función de este módulo es enviar un mensaje de sincronización a todos los sitios de la BDD, indicándoles el inicio de las operaciones, para luego, llamar a la función *Recursivo*, la cuál se encargará de la construcción del árbol en base a los resultados que los sitios le envíen. Finalizada la construcción, se envía un mensaje indicando la culminación de las operaciones.

La función recursivo maneja 4 parámetros. El primer parámetro es referente al conjunto de sitios $\aleph^N$ que pueden generar algún tipo de resultado, ya sea vector de ganancias o de coberturas. Al iniciarse las operaciones, se considera que todos los sitios de la BDD pueden participar en el desarrollo del árbol, por ello se le asigna a éste parámetro el valor de *S*. El segundo parámetro es referente a los sitios $\aleph^F$, es decir, aquellos sitios que ya no participarán en el desarrollo de la rama que se está desarrollando. Es claro que al iniciarse las operaciones, este conjunto es vacío, ya que todos los sitios podrán generar la primera serie de resultados. El tercer parámetro es referente al conjunto de atributos que se podrán considerar para la rama que se desarrolle. Para la generación del nodo raíz, la totalidad de los atributos podrán participar, representado por Ψ. El cuarto parámetro esta enfocado al control en la recepción de los resultados, en el cual con 1 se indica que todos los sitios continúan generando vectores de ganancia y con 0 se señala que algunos sitios ya han finalizado el desarrollo de una rama, sin embargo, estarán generando vectores de coberturas de clases.

Observemos que en el servidor no se hace referencia al atributo clasificador. Esto se debe a que el servidor no trabaja directamente con datos, simplemente recolecta resultados y determina como se construirá el árbol.

El desarrollo del árbol es realizado por la función *Recursivo*, lo que la convierte en la función relevante para el servidor. El código asociado a esta función se expone en el Algoritmo 6. Este algoritmo construirá de manera recursiva el árbol ID3 global. Un aspecto muy importante que se debe mencionar es que el árbol será desarrollado en *profundidad primero*, esto es, para un nodo cualquiera del árbol que tenga "*x*" hijos y suponiendo que se inicia de izquierda a derecha, primero se empieza desarrollando el nodo más izquierdo y hasta no haber terminado el crecimiento de este nodo, no se analiza otro nodo hermano del mismo.

Lo primero que va a realizar el algoritmo, es esperar los resultados de los sitios. La recepción de resultados se lleva a cabo por la función "*EsperaRespuesta*", la cuál tiene como parámetros $\aleph^N$, $\aleph^F$, *G, D, C, R, Ψ, N,* C_D, C_{MI}. Los dos primeros parámetros corresponden a los sitios que aún desarrollan una rama o no. *G* representa a un vector donde se almacenarán los vectores de ganancias generados de cada sitio. Se usará la notación *G[x][y]* para referirse a la ganancia del atributo *y* correspondiente al sitio *x*. *D* representa al vector que contendrá los dominios de cada uno de los atributos contenidos en el vector de ganancia. De igual forma, se usará la notación *D[x][y]* para referirse al dominio del atributo *y* proveniente del sitio *x*. Sin embargo, notemos que *D[x][y]* no necesariamente consta de un solo valor, ya que puede contener todo un conjunto de valores que representen al dominio del atributo correspondiente. *C* representa al vector de coberturas para las clases. Este vector no guardará de forma exclusiva las clases y sus coberturas para cada sitio. De hecho, la idea es que si se encuentra una clase *x* cualquiera, la cobertura de cada clase del mismo tipo de cada sitio se sume al valor correspondiente de la clase *x* en el vector *C*. Por ello, se usará la notación de *C[n][0]* para almacenar el identificador de una clase *x* en la posición *n* del vector y *C[n][1]* para referirse a la cobertura de dicha clase. Finalmente, C_D representa al vector de costos de transmitir los dominios de los sitios al servidor, así como C_{MI} corresponde al vector de costos de los mensajes innecesarios por considerar el dominio global. Es importante mencionar que la función *EsperarRespuesta* modificará los valores de las variables $\aleph^N$, $\aleph^F$, *G, D, C,* C_D, C_{MI} de la función *Recursivo*, actualizándolos dependiendo de los resultados recolectados.

Recolectados los resultados, el bloque de la línea 6 a la 11 verificará si todos los sitios han finalizado el desarrollo de la rama que se analiza. Si es así, se crea un nodo el cuál será etiquetado con la clase que se obtenga de analizar los vectores de cobertura de dichos sitios, Posteriormente, se envía un mensaje a los sitios que han concluido el desarrollo indicándoles que la rama ya no se desarrollará más, por lo que deberán esperar otro mensaje para desarrollar alguna otra rama. Finalmente, el nodo que se ha creado es el valor que devuelve la función.

En caso de que algunos sitios hayan generado vectores de ganancias, se verifica qué valor retorno la función *EsperarRespuesta*: si el valor de retorno es 0, indica que todos los sitios enviaron un vector de ganancias, esto es, todos continuarán operando en la siguiente iteración. En otro caso, indica que algunos sitios han finalizado el crecimiento de la rama y otros aún podrán desarrollar dicha rama.

De las líneas 14 a la 29 se analiza el caso cuando se generan exclusivamente vectores de ganancia. En este bloque, se crea un nodo que será etiquetado con el atributo de mayor ganancia, el cuál se obtiene al analizar los vectores de ganancia enviados por los sitios, operación que realiza la función *AtributoGanador*. Está función, a su vez, regresará en la variable D_g el dominio que resulte más económico utilizar para el atributo con respecto al flujo de datos. En base a este dominio, al nodo creado se le asocian tantas ramas como valores en el dominio existan. Cada vez que se crea una rama, se envía un mensaje a los sitios indicándoles el valor de la rama por la que continuará el desarrollo del árbol. El árbol que construya la función *Recursivo* será asociado a dicha rama. Recordemos que el dominio más económico con respecto al flujo puede ser el dominio global o el dominio resultante de la unión de los dominios locales de cada sitio. Cuando se maneje el dominio global, el servidor enviará a cada sitio un mensaje para cada valor de dicho

dominio. En cambio, si se considera el dominio local, la decisión de la línea 21 definirá a qué sitios no se les enviará ningún mensaje por no estar contenido en su dominio local el valor que se analiza. A los sitios que no les es enviado ningún mensaje, la función *Recursivo* ya no los considerará ni siquiera en el conjunto $\aleph^F$, debido a que estos no generarán ningún tipo de respuesta para la rama que se analice. Una vez que se haya concluido de analizar todas las ramas posibles, se envía a los sitios un mensaje en el cuál se les notifica que el desarrollo del nodo que se ha creado ha concluido, regresando la función el árbol que se construyó.

```
1.  Arbol Recursivo(Sitos ℵ^N, Sitos ℵ^F, Ψ, Entero R)
2.  { Entero b;              Atributo A_g;      Arbol δ;
3.    Dominio D, D^N, D^F;   Ganancias G;       Clases C;
4.    Clase cl;              VectorEntero N;    Sitios A;    VectorCostos C_D, C_Md;
5.    b ← EsperaRespuesta(ℵ^N, ℵ^F, G, D, C, R, Ψ, N, C_D, C_Md);
6.    si ℵ^N = φ hacer:
7.      { δ ← Crear un nuevo árbol (nodo);
8.        δ_clase ← ClaseGanadora(C);
9.        para cada i tal que S^i ∈ ℵ^F hacer:
10.           mandar mensaje que se ha terminado de construir la rama;
11.       regresar(δ);
12.     }
13.   si b = 0 hacer:
14.     { A_g ← AtributoGanador(ℵ^N, Ψ, G, N, C_D, C_Md, D_g, D, b);
15.       δ ← Crear un nuevo árbol (nodo);
16.       δ_atributo ← Etiqueta al nodo con A_g;
17.       para cada v ∈ D_g hacer:
18.         { asociar a δ una subrama para v;
19.           A ← ∅;
20.           para cada i tal que S^i ∈ ℵ^N hacer:
21.             si v ∈ D[i][g] hacer:
22.               enviar mensaje a S^i de continuar las operaciones con A_g - v;
23.             en caso contrario:
24.               A ← A ∪ {S^i};
25.           asociar a la rama δ – v el nodo ← Recursivo(ℵ^N - A, ℵ^F, Ψ-{A_g}, 1);
26.         }
27.       para cada i tal que S^i ∈ ℵ^N hacer:
28.         mandar mensaje que se ha terminado de construir la rama;
29.       regresar(δ);
30.     }
31.   en caso contrario:
32.   { A_g ← PatronesDisjuntos(D^N, D^F, Ψ, ℵ^N, ℵ^F, D);
33.     si A_g ≠ φ hacer:
34.       { cl ← ClaseGanadora(C);
35.         δ ← Crear un nuevo árbol(nodo);
36.         δ_atributo ← Etiqueta al nodo con A_g;
37.         para cada v ∈ D^F hacer:
38.           { asociar a δ una subrama para v;
39.             etiquetar al nodo de la rama δ – v con la clase cl;
40.           }
41.         para cada v ∈ D^N hacer:
42.           { asociar a δ una subrama para v;
43.             A ← ∅;
44.             para cada i tal que S^i ∈ ℵ^N hacer:
45.               si v ∈ D[i][g] hacer:
46.                 enviar mensaje a S^i de continuar las operaciones con A_g - v;
47.               en caso contrario:
48.                 A ← A ∪ {S^i};
49.             asociar a la rama δ – v el nodo ← Recursivo(ℵ^N - A, ∅, Ψ-{A_g}, 1);
50.           }
51.         para cada i tal que S^i ∈ {ℵ^N ∪ ℵ^F} hacer:
52.           mandar mensaje que se ha terminado de construir la rama;
53.         regresar(δ);
54.       }
```

```
55. en caso contrario:
56.     { A_g ← AtributoGanador(ℵ^N, Ψ, G, N, C_D, C_MS, D_g, D, b);
57.       δ ← Crear un nuevo árbol (nodo);
58.       δ_atributo ← Etiqueta al nodo con A_g;
59.       para cada v ∈ D_g hacer:
60.         { asociar a δ una subrama para v;
61.           A ← ∅;
62.           para cada i tal que S^i ∈ ℵ^N hacer:
63.               si v ∈ D[i][g] hacer:
64.                   enviar mensaje a S^i de continuar las operaciones con A_g – v y devuelve el vector de
                      dominios;
65.               en caso contrario:
66.                   A ← A ∪ {S^i};
67.           para cada i tal que S^i ∈ ℵ^F hacer:
68.               { enviar mensaje al sitio S^i de asociar a la rama que se está trabajando en el momento
                      un nodo correspondiente al atributo A_g, si es que no se ha creado,
                      asociando al mismo la rama A_g – v y pidiendo que el sitio S^i regrese el vector de
                      coberturas de clases y dominios asociadas a dicha rama
69.               }
70.           asociar a la rama δ – v el nodo ← Recursivo(ℵ^N - A, ℵ^F, Ψ-{A_g}, 0);
71.         }
72.         para cada i tal que S^i ∈ ℵ^N hacer:
73.           mandar mensaje que se ha terminado de construir la rama;
74.         regresar(δ);
75.     }
76.   }
77. }
```

Algoritmo 6. ID3 para una BDD: algoritmo correspondiente al servidor. Módulo "Recursivo"

Por otra parte, cuando algunos sitios producen vectores de ganancia y otros vectores de cobertura, se está en el caso donde algunos sitios continuarán el proceso de construcción y otros no. Para tal situación, se debe analizar si algún atributo permite separar a los patrones de los conjuntos de sitios $\aleph^N$ y $\aleph^F$, operación que realiza la función *PatronesDisjuntos*. Esta función toma como parámetros D^N, D^F, Ψ, $\aleph^N$, $\aleph^F$ y D. Tanto D^N como D^F son vectores que van a almacenar los valores del dominio del atributo que se asocien a los conjuntos $\aleph^N$ y $\aleph^F$, donde D^N corresponde a los valores asociados a los sitios $\aleph^N$ mientras que D^F corresponde a los valores asociados a los sitios $\aleph^F$. Los valores de estos vectores son actualizados por la función *PatronesDisjuntos*, mientras que el resto de sus parámetros son utilizados para el cálculo de las operaciones (de solo lectura).

En el bloque de las líneas 33 a la 54 se analiza el caso en el que existe un atributo que separa a los patrones. En éste, se obtiene la clase ganadora basándose en los vectores de cobertura que los sitios $\aleph^F$ han enviado. Además, se crea un nodo al que se etiqueta con el atributo separador y se le asocian dos tipos de ramas: aquellas para los valores del dominio correspondiente a los sitios $\aleph^F$ (las cuales son etiquetadas con la clase obtenida), y las correspondientes a los valores del dominio asociados a los sitios $\aleph^N$ (los cuales continuarán el desarrollo de dichas ramas, sin tomar en cuenta a los sitios $\aleph^F$). Por ello, cuando se llama a la función *Recursivo* para que vaya desarrollando cada una de dichas ramas, en su segundo parámetro se le da el valor de vacío.

De la línea 56 a la 75 se analiza el caso cuando no existen patrones disjuntos entre los sitios que finalizan el desarrollo de la rama y de los que aún la pueden continuar. En este caso y basándose en los vectores de ganancia que los sitios $\aleph^N$ han generado, se calcula un atributo ganador. En

seguida, se crea un nodo que es etiquetado con dicho atributo y al cuál se le añadirán, en un proceso recursivo, tantas ramas como valores en el dominio existan. Para cada valor v del dominio, se manda un mensaje al conjunto de sitios $\aleph^N$ para que continúen el desarrollo por el valor de la rama señalada, pero pidiendo que cada sitio devuelva el dominio asociado a cada atributo que se genere. Por otra parte, a los sitios $\aleph^F$ se les pide que añadan el nodo correspondiente al atributo ganador que se ha obtenido, si es que no lo han añadido, y devuelvan el dominio de los atributos correspondientes a la rama del valor v que se analiza. Con ello, se asegura que todos los sitios regresarán el vector de dominios de los atributos, los cuales serán utilizados en la siguiente recursión para buscar algún patrón disjunto. Además, al llamarse de forma recursiva la función *Recursivo*, al cuarto parámetro se le asigna el valor de 0, ajustando el comportamiento de la función *EsperarRespuesta* para que reciba los vectores de dominio de los sitios $\aleph^F$.

Notamos que este código hace uso de varias funciones adicionales, las cuales se definen a continuación. La función *ClaseGanadora* es la que calcula la clase que tiene mayor cobertura entre las clases almacenadas en el vector de coberturas de clases C. La clase con mayor cobertura es el valor que devuelve esta función.

```
1.  Clase ClaseGanadora(Clases C)
2.  { Entero n, j;
3.    n ←-1;
4.    para i = 0 hata i < |C| hacer:
5.      si C[i][1] > n hacer:
6.        { n ← C[i][1]; j ←i; }
7.    si n ≠-1 hacer:
8.      regresar(C[j][0]);
9.    en otro caso
10.     regresar(∅);
11. }
```

Algoritmo 7. ID3 para una BDD: algoritmo correspondiente al servidor. Módulo "ClaseGanadora"

Está función regresa el símbolo vacío si el vector de coberturas es vacío. Este caso se dará cuando el servidor considere el dominio global de un atributo y alguno de estos valores no pertenezca a ningún sistio. El código de ésta función se presenta en el Algoritmo 7

Continuemos con la función *EsperarRespuesta*, la cuál recolecta los resultados que los sitios van enviando al servidor en cada iteración. Esta función inicia recibiendo los vectores de dominio de los sitios $\aleph^F$, bloque de la línea 4 a la 10. Si es que existen sitios en éste conjunto, se reciben los vectores de dominio y cobertura que cada uno de ellos enviará, recordando que los vectores servirán para encontrar posibles patrones disjuntos. En seguida, se reciben los resultados de los sitios $\aleph^N$, bloque de las líneas 12 a la 30. Los sitios $\aleph^N$ fundamentalmente enviarán los datos asociados al vector de ganancias. En caso de que también se el envíen al servidor los vectores de dominios, estos deben de ser recibidos, situación que se presentará cuando se está en una iteración donde en un nodo previo no se encontró algún patrón disjunto. Además, si el sitio envía un vector de coberturas, indica que el desarrollo de la rama que se analiza ya concluyó para dicho sitio, recibiendo para este caso el vector de coberturas y dominios.

```
1.  Entero EsperarRespuesta(Sitios ℵ^N, Sitios ℵ^P, Ganancias G, Dominio D, Clases C, Entero R, Ψ
                            VectorEntero N, VectorCostos C_D, VectorCostos C_Ml)
2.  { Entero b;
3.    b ← 0; G ← ∅; D ← ∅; C ← ∅;
4.    si R = 0 hacer:
5.    { para todo i tal que S^i ∈ ℵ^P hacer:
6.        { para cada A ∈ Ψ hacer:
7.            D[i][A] ← dominio del atributo A del sitio S^i;
8.          C ← C ∪ {vector de coberturas de clases del sitio S^i};
9.        }
10.     b ← 1;
11.   }
12.   para todo i tal que S^i ∈ ℵ^N hacer:
13.   { si el mensaje de S^i contiene los vectores de ganancias hacer:
14.       { N[i] ← num. de reg. de S^i; G[G^i] ← ganancias de S^i;
15.         C_D[S^i] ← costo de transmitir los dominios locales del sitio S^i al servidor;
16.         C_Ml[S^i] ← costos de los mensajes inútiles derivados de considerar el dominio global;
17.         si el sitio S^i envia el vector de dominios hacer:
18.             D[D^i] ← dominio de S^i;
19.       }
20.     en caso contrario:
21.     si el mensaje de S^i es un vector de coberturas de clases hacer:
22.       { C ← C ∪ {vector de coberturas de clases de S^i}; D[D^i] ← dominios de S^i;
23.         ℵ^N ← ℵ^N − {S^i};
24.         ℵ^P ← ℵ^P ∪ {S^i};
25.         b ← 2;
26.       }
27.     en caso contrario:
28.     si el mensaje de S^i es el simbolo ∅ hacer:
29.       ℵ^N ← ℵ^N − {S^i};
30.   }
31.   si b=2 hacer:
32.     { para todos los sitios S^i ∈ ℵ^N hacer:
33.         si el sitio S^i no ha enviado el vector de dominios hacer:
34.         { enviar un mensaje a S^i para obtener el dominio de los atributos Ψ en S^i;
35.           para cada A ∈ Ψ hacer:
36.             D[i][A] ← dominio del atributo A del sitio S^i;
37.           C_Ml[S^i] ← ∞; C_D[S^i] ← 0; }
38.       b ← 1; }
39.   }
40.   regresa(b);
41. }
```

Algoritmo 8. ID3 para una BDD: algoritmo correspondiente al servidor. Módulo "EsperarRespuesta"

También se puede recibir un mensaje vacío del sitio, el cuál indica que el valor previo que ha enviado el servidor a dicho sitio no pertenece al dominio local de los registros asociados a la rama que se analiza. Finalmente, el bloque de las líneas 31 a la 38 sirve para indicarles a los sitios del conjunto $\aleph^N$ que envíen el vector de dominios asociado a la rama que se maneja. Esto se debe a que si algunos sitios del conjunto original $\aleph^N$ que recibió la función han terminado el desarrollo de la rama que se analiza, pero otros no, se debe buscar algún patrón disjunto, pero ello requiere que la totalidad de estos sitios envíen los vectores de dominios. Con este bloque, se asegura que si se presenta dicha situación, el servidor recolectará los dominios de cada sitio, si es que no lo habían enviado previamente.

Finalmente, a través de la variable *b* la función regresa dos posibles valores: 0, para indicar que la totalidad de sitios generó un vector de ganancias y, 1, en el caso cuando algunos sitios han finalizado el crecimiento de la rama que se analiza.

Continuemos la exposición de los métodos analizando la función *AtributoGanador*. Esta función realiza dos tareas: en primera instancia, entre las líneas 4 y 15 se calcula el atributo ganador en base a los vectores de ganancia que los sitios $\aleph^N$ hayan generado. Una vez calculado el atributo ganador, que es almacenado en la variable A_g, la función calcula cuál es el dominio que conviene manejar cuando no existen patrones disjuntos, representado cuando la variable b es 1. En el bloque de las líneas 16 a la 30 se determina si el dominio local de cada sitio o el dominio global es el que representa un menor flujo de datos para analizar las ramas del atributo A_g. Notemos que si se va a usar el dominio global, el servidor no importa ningún valor de los sitios. En cambio, si se determina usar el dominio local, es necesario importar cada dominio de cada sitio.

```
1.  Atributo AtributoGanador(Sitios ℵ^N, Ψ, Ganancias G, VectorEntero N, VectorCostos C_D,
                             VectorCostos C_M, Dominio D_g, Dominio D, Entero b)
2.  { Entero n, cd, cni;        Atributo A_g;        Flotante ga, gb;
3.    n ← 0;  D_g ← Ø;  cd ← 0;  cni ← 0;
4.    para cada i tal que S^i ∈ ℵ^N hacer:
5.      n ← n + N[i];
6.    para cada i tal que A_i ∈ Ψ hacer:
7.     { gb ← 0;
8.       para cada j tal que S^j ∈ ℵ^N hacer:
9.           gb ← gb + N[j]*G[j][i];
10.      gb ← gb/n;
11.      si gb > ga hacer:
12.       { A_g ← A_i;
13.         ga ← gb;
14.       }
15.    }
16.   si b = 0 hacer
17.   {
18.   para cada i tal que S^i ∈ ℵ^N hacer:
19.     { cd ← cd + C_D[i][A_g];
20.       cni ← cni + C_M[i][A_g];
21.     }
22.   si cd > cni hacer:
23.    { D_g ← dominio global del atributo A_g;
24.      para cada i tal que S^i ∈ ℵ^N hacer:
25.         D[D^i] ← D_g;
26.    }
27.   en otro caso
28.     { enviar un mensaje a los sitios ℵ^N para que devuelvan el dominio local del atributo A_g;
29.     D_g ← almacenar la unión de todos los dominios locales de los sitios ℵ^N; }
30.  }
31.  regresa(A_g);
32. }
```

Algoritmo 9. ID3 para una BDD: algoritmo correspondiente al servidor. Módulo "AtributoGanador"

Finalmente, analicemos el código de la función *PatronesDisjuntos*, el cuál se expone en el Algoritmo 10. En está función, primero se busca algún valor o valores pertenecientes a algún atributo A_j en Ψ que sean comunes al conjunto de sitios $\aleph^F$, pero que también cumplan que dichos valores no se encuentren en los dominios de los sitios $\aleph^N$. Si es que existen, estos valores formarán el conjunto D^F, y en el conjunto D^N se almacenarán los valores complementarios al dominio, es decir, aquellos valores que son comunes a ambos conjuntos de sitios. Si se encuentra un conjunto de valores D^N, la función regresa el atributo que permite diferenciar a los patrones disjuntos, representado por A_j. En caso de que no se haya encontrado ningún atributo el cuál, por medio de su dominio, permitiera diferenciar a algún patrón disjunto, la función devuelve el símbolo vacío. Con esta función se concluye la exposición de los códigos asociados al servidor. Analicemos ahora el código asociado a los sitios.

```
1.  Atributo PatronesDisjuntos(Dominio D^N, Dominio D^F, Ψ, N^N, N^F, D)
2.  { Sitios S^x; D^N ← Ø; D^F ← Ø;
3.    S^x ← seleccionar cualquier A_x ∈ N^F;
4.    para todo j tal que A_j ∈ Ψ hacer:
5.      { para todo v tal que v ∈ D[x][j] hacer:
6.          si para todo i tal que S^i ∈ N^F y v ∈ D[i][j] hacer:
7.              si para todo k tal que S^k ∈ N^N y v ∉ D[k][j] hacer:
8.                  D^F ← D^F ∪ {v};
9.        si D^F ≠ Ø hacer:
10.         { para todo i tal que S^i ∈ {N^N ∪ N^F} hacer:
11.             para todo v tal que v ∈ D[i][j] y v ∉ D^F hacer:
12.                 D^N ← D^N ∪ {v};
13.           regresar(A_j);
14.         }
15.     }
16.   regresar(Ø);
17. }
```

Algoritmo 10. ID3 para una BDD: algoritmo correspondiente al servidor. Módulo "PatronesDisjuntos"

6.3.2 . ID3 para una BDD: algoritmo correspondiente a los sitios

El código asociado a los sitios es más simple que el asociado al servidor, ya que éste operará de manera similar que el algoritmo ID3 centralizado clásico, difiriendo en que por si solo no determina el crecimiento del árbol ID3, decisión que es exclusiva del servidor.

```
ENTRADA
Ψ: Conjunto de atributos a trabajar
A_P : Atributo P de clasificación
ℑ: conjunto de registros de entrenamiento
SALIDA
δ: Árbol de clasificación local derivado de la construcción asociada con el servidor

1.  Inicio()
2.  { Árbol δ;      Ganancias G;      Dominio D;      Clases C;      Entero b;
3.    esperar mensaje de inicio del servidor;
4.    b ← CalcularResultados(G, D, C, Ψ, ℑ);
5.    si b = 0 hacer:
6.      { enviar al servidor(num. de Registros en ℑ, G, costo de transmitir los dominios de los atributos
                en Ψ al servidor, costo de los mensajes innecesarios de considerar el
                dominio global);
7.      δ ← Recursivo(Ψ, Ø, Ø);
8.      }
9.    en caso contrario:
10.     { enviar al servidor(C, D);
11.     δ ← Crear un nuevo árbol (nodo);
12.     δ_clase ← ClaseGanadora(C); }
13. }
```

Algoritmo 11. ID3 para una BDD: algoritmo correspondiente a los sitios. Módulo "Principal"

La primera función a exponer es la principal, la cuál se encarga de iniciar el proceso de construcción basada en las instrucciones enviadas por el servidor. Esta función recibe como entrada al conjunto Ψ de atributos con los cuales va a trabajar, el atributo clasificador A_p así como el conjunto de registros $\Im$. Debemos recordar que el atributo clasificador debe coincidir para todos los sitios, es decir, cada sitio va a trabajar con el mismo atributo clasificador.

La primera operación es esperar el mensaje del servidor que sincronice el inicio de los cálculos. Una vez recibido el mensaje del servidor, se llama a la función *CalcularGanancias* para obtener el primer conjunto de resultados. Notemos que se tienen dos posibles resultados: se puede obtener un vector de ganancias para el total de registros del conjunto $\Im$ o, se presenta el caso en el que desde un inicio todos los registros son de una sola clase. Esta función regresa un valor entero que es almacenado en la variable *b*, la cuál representa las dos posibles situaciones. En caso de que *b* sea 0, indica que se generó un vector de ganancias. En tal caso, se envía el número de registros, el vector de ganancias, así como el costo de transmitir los dominios de los atributos y el costo de los mensajes innecesarios derivados de considerar el dominio global. Posteriormente, se llama a la función *Recursivo*, la cuál regresará el árbol que finalmente se construya.

Si *b* es 1, indica que todos los registros analizados son de una sola clase. Si así ocurre, se envía al servidor el vector de cobertura de clases *C* así como los dominios de los atributos trabajados *D*, con el objetivo de que el servidor este conciente de que el sitio ya ha concluido sus operaciones. Posteriormente, se crea un solo nodo, el cuál será rematado con la clase mayoritaria mediante la función *ClaseGanadora*, la cuál hace uso del vector de coberturas de clases *C*. Esta función es la misma que se utiliza en el servidor, es decir, se llama a la función del Algoritmo 7.

```
1.  Entero CalcularResultados(Ganancias G, Dominio D, Clases C, Ψ, ℑ)
2.  { C ← obtener el vector de coberturas de las clases correspondientes al atributo clasificador en
        base a los registros analizados ℑ;
3.   si los registros ℑ son de una sola clase hacer:
4.     { para todo x tal que A_x ∈ Ψ hacer:
5.         D[x] ← obtener el dominio del atributo A_x en base a los registros ℑ;
6.       regresa(1);
7.     }
8.   en caso contrario:
9.   para todo x tal que A_x ∈ Ψ hacer:
10.    G[x] ← calcular la ganancia del atributo A_x en base a los registros ℑ;
11.  si Ψ ≠ ∅ regresar(0);
12.  en otro caso regresar(1);
13. }
```

Algoritmo 12. ID3 para una BDD: algoritmo correspondiente a los sitios. Módulo "CalcularResultados"

El código correspondiente a la función *CalcularResultados* se expone en el Algoritmo 12. Esta función calcula en primera instancia el vector de coberturas de clases *C* correspondientes al conjunto de registros $\Im$. Al construirse dicho vector, se puede determinar si la totalidad de los registros son de una sola clase. Si es así, se crea el vector de dominios para los atributos Ψ que se consideran en la iteración, retornando la función el valor de 1, con el cuál indica que la rama del sitio ya no será desarrollada.

Por otra parte, si los registros no son de una sola clase, se pasa a generar el vector de ganancias correspondientes al conjunto de atributos Ψ, retornando la función el valor de 0, con el cuál se indica que el desarrollo de la rama en el sitio continuará.

```
1.  Arbol Recursivo(Ψ, Arbol δ', Registros ℑ)
2.  { Arbol δ;      Atributo A_g;      Valor v;
3.    esperar mensaje del servidor;
4.    si el mensaje del servidor es del tipo atributo – valor hacer:
5.      { A_g ← atributo indicado por el servidor;
6.        v ← valor indicado por el servidor;
7.        si v pertenece al dominio local del atributo A_g en base a ℑ hacer:
8.          { si (δ' ≠ ∅ y δ'_atributo ≠ A_g) o δ' ≠ ∅ hacer:
9.              { δ ← crear un nuevo árbol (nodo);
10.               δ_atributo ← etiquetar al árbol con el atributo A_g;
11.             }
12.           en otro caso
13.             δ_atributo ← δ';
14.           añadir una rama a δ para el valor v;
15.           ℑ' ← registros de ℑ que cumplen con la condición A_g – v;
16.           b ← CalcularResultados(G, D, C, Ψ - {A_g}, ℑ');
17.           si b = 0 hacer:
18.             enviar al servidor (numero de registros en ℑ', G, costo de transmitir los dominios
                    de los atributos Ψ al servidor, costo de los mensajes innecesarios
                    de considerar el dominio global, - D -);
19.           en otro caso:
20.             enviar al servidor (C, D);
21.           asociar a la rama δ – v ← Recursivo(Ψ - {A_g}, δ, ℑ');
22.         }
23.       en otro caso:
24.         enviar al servidor(∅);
25.       Recursivo(Ψ, δ, v, ℑ);
26.       regresar(δ);
27.     }
28.   en otro caso:
29.   si el mensaje del servidor es del tipo "añade nodo y devuelve dominio" hacer
30.     { A_g ← atributo indicado por el servidor;
31.       v ← valor indicado por el ser[illegible]
32.       δ ← crear un nuevo árbol (n[illegible]);
33.       δ_atributo ← etiquetar al árbol con A_g;
34.       ℑ' ← registros de ℑ que cumplen con la condición A_g – v;
35.       D ← dominios de los atributos Ψ en base a los registros ℑ';
36.       C ← vector de coberturas de clases en base a los registros ℑ';
37.       enviar mensaje al servidor(D, C);
38.       asociar a la rama δ – v ← Recursivo(Ψ - {A_g}, δ, ℑ');
39.       Recursivo(Ψ, δ, v, ℑ');
40.       regresar(δ);
41.     }
42.   en otro caso:
43.   si el mensaje indica devolver el dominio de un atributo A hacer:
44.     { enviar mensaje al servidor(dominio del atributo A en base al conjunto de registros ℑ);
45.       Recursivo(Ψ, δ, v, ℑ);
46.     }
47.   si el mensaje del servidor es finalizar rama hacer:
48.     regresar(∅);
49. }
```

Algoritmo 13. ID3 para una BDD: algoritmo correspondiente a los sitios. Módulo "Recursivo"

Ahora analicemos la función *Recursivo*, la cuál se encarga de generar el árbol que se construirá de manera local en cada sitio. Esta función operará de acuerdo a las instrucciones que reciba de parte del servidor.

Si el servidor indica que se debe crear una rama, en el bloque de las líneas 4 a la 27 se especifica como el sitio desarrollará la rama correspondiente al valor v del atributo A_g. Notemos que si ya existe un nodo correspondiente al atributo, a dicho nodo se le añade una rama para al valor v. En caso de no existir, se crea un nodo para dicho atributo y se le añade la rama correspondiente al valor v. En ambos casos, se calcula si el desarrollo de la rama continuará o no, enviando las

correspondientes estructuras de resultados según el caso. Dicha función se llama a sí misma para continuar el desarrollo del árbol, ya sea con un nivel más profundo de la rama que se analizó o con una rama hermana de la ya analizada.

Otra posible instrucción del servidor a los sitios es que estos últimos añadan un nodo a la rama que se está analizando. Este caso solo sucederá cuando el sitio sea parte del conjunto $\aleph^F$ en el servidor, sin haber existido patrones disjuntos. Para tal caso, el sitio recibirá tanto el atributo ganador como el valor del dominio por el cuál se continuará el desarrollo. En seguida, se añade un nodo correspondiente a dicho atributo ganador, al cuál se le anexa una rama para el valor v indicado por el servidor. También se calcula el conjunto de registros que serán asociados a dicha rama, y una vez obtenidos, se calcula tanto el vector de cobertura de clases, como el de dominios para los registros asociados a la rama. Estos vectores son enviados al servidor, y la función se llama a sí misma de manera recursiva.

Otra indicación que puede recibir cada sitio del servidor es que estos le envíen el vector de dominios asociados a la rama que se analiza. Este mensaje sólo se generará cuando el servidor determine que es más barato considerar el dominio local de cada sitio que considerar el dominio global, con respecto al costo de los mensajes.

El otro posible mensaje que se puede recibir del servidor es el que indica la finalización del desarrollo de la rama.

Este mensaje lo recibirán los sitios cuando el servidor determine que ya ningún sitio puede desarrollar más la rama que se analiza, con lo cuál, cada sitio debe concluir su desarrollo y con ello, los códigos en cada sitio regresarán de un llamado recursivo para analizar otra posible rama de nivel superior.

Con lo anterior, se concluye la exposición de los algoritmos diseñados para la construcción del árbol ID3 en una BDD. El paso siguiente es comprobar que efectivamente arrojen árboles de clasificación que tengan una buena precisión, así como observar el flujo que se tendrá al aplicarse en ejemplos reales. Estas pruebas se exponen en el siguiente capítulo, el cuál intenta abarcar ejemplos que sean ilustrativos a la mayoría de posibles situaciones que se pueden presentar.

7 RESULTADOS

El presente capítulo expone los resultados que se obtuvieron de aplicar el algoritmo ID3 propuesto. Las pruebas se han dividido en dos secciones:

- Un conjunto de pruebas enfocadas a verificar la veracidad de los resultados, empleando BD's que de antemano se conocen los patrones existentes.
- Por otra parte, haciendo uso de una BD con un número considerable de registros, se verifica el flujo de información derivado de la comunicación entre los sitios. Es importante mencionar que siempre se partió de BD's centralizadas, las cuales fueron fragmentadas de manera aleatoria para simular una BDD.

Pruebas enfocadas a la veracidad de los resultados: 1^{er} ejercicio

Para esta sección usaremos el conjunto de datos ilustrados en la Tabla 2, los cuales han sido ampliamente trabajados dentro de la Minería de Datos, lo que permite conocer perfectamente sus patrones.

Hay que señalar que la cantidad de registros de está $\Re$ es pequeña. Desde el punto de vista de la teoría de las BDD's, está relación no tiene porqué ser dividida. Sin embargo, por tener muy pocos registros y por ser una relación de la cuál se conocen perfectamente sus patrones, se puede manipular de forma simple para construir una BDD que presente ciertas condiciones que prueben diversos casos de nuestro algoritmo.

Debido a que está $\Re$ solo cuenta con 14 registros, no es factible dividirla en un gran número de sitios. Por ello, solo se ha fragmentado (de manera aleatoria) en dos, correspondientes a la Tabla 3 o $\Re^1$ y la Tabla 4 o $\Re^2$. Aplicando el algoritmo centralizado (Algoritmo 1) a la relación de la Tabla 2, se obtiene el árbol ilustrado en la Figura 5.3., que representa a los patrones globales de la relación. Por otra parte, si se aplica este mismo algoritmo de manera independiente tanto a $\Re^1$ como a $\Re^2$, los árboles obtenidos son los ilustrados en la Figura 6.6 y la Figura 6.7 respectivamente. Como se comentó en la sección 6.2.2., el crecimiento de las subramas de los árboles no es equilibrada, ya que para $\Re^1$, los registros que quedan asociados a la rama *Ambiente – soleado* después de la primera iteración aún son de varias clases, por lo que éste sitio continúa el desarrollo de dicha rama. Sin embargo, todos los registros de la misma rama para $\Re^2$ son de una sola clase, como lo muestra su árbol local (Figura 6.7), concluyendo el crecimiento de la rama. Una situación análoga de crecimiento desequilibrado se presenta con la rama *Ambiente –*

lluvia, donde después de la primera iteración, el sitio que puede continuar desarrollando dicha rama es $\Re^2$ mientras que $\Re^1$ ya no podrá desarrollar a la rama en discurso.

También observamos que en ambos sitios, la rama *Ambiente – nublado* tiene la misma estructura, con un solo nivel en profundidad, lo cuál indica que terminarán de desarrollar dicha rama al mismo tiempo.

Pasemos a analizar como el algoritmo propuesto construirá el árbol en el servidor. Para ello, se expondrá a detalle el crecimiento de una rama, con el fin de tener clara la idea de cómo operará. Posteriormente, solo se expondrán resultados con el fin de analizarlos.

Al iniciarse el proceso de construcción, el módulo principal del servidor, Algoritmo 5, manejará como entrada los siguientes conjuntos:

Ψ = {*Ambiente, Temperatura, Humedad, Viento*}

S = {S^1 = *sitio que contiene a la* Tabla 3, S^2 = *sitio que contiene a la* Tabla 4}

Este módulo, envía un mensaje tanto a S^1 como a S^2 para sincronizar el inicio de las operaciones. Posteriormente se llama a la función *Recursivo*, Algoritmo 6, la cual a su vez, mediante el llamado de la función *EsperarRespuesta*, Algoritmo 8, esperará el primer conjunto de resultados que los sitios le envíen.

Analicemos que es lo que sucede tanto en S^1 como en S^2. Ambos sitios ejecutarán su correspondiente Algoritmo 11. Inicialmente, tanto el servidor como los sitios manejarán el mismo conjunto Ψ = {*Ambiente, Temperatura, Humedad, Viento*}. Además, S^1 como S^2 trabajan con el mismo atributo clasificador A_p = {*Clase*}. El conjunto de entrenamiento $\Im$ para S^1 es $\Re^1$ (Tabla 3), mientras que para S^2, $\Im$ es $\Re^2$ (Tabla 4).

Una vez recibido el mensaje del servidor, ambos sitios pasan a calcular el primer conjunto de resultados, llamando a la función *CalcularResultados*.

La función *CalcularResultados*, Algoritmo 12, inicia calculando el vector de coberturas de los registros correspondientes a $\Im$ (registros que se consideran en la iteración). Para S^1, el vector de coberturas será:

$$C^1 = \{[N, 1], [P, 6]\} \quad (1)$$

Tanto N como P son clases, y los números que los acompañan indican la cobertura de los mismos. Para S^2, el vector de coberturas estará dado por:

$$C^2 = \{[N, 4], [P, 3]\} \quad (2)$$

En ambos casos, se tiene un vector de coberturas conformado por más de una clase, por lo que la función *CalcularResultados* construye el vector de ganancias para cada atributo en Ψ. Para S^1 el vector obtenido se ilustra en (3), mientras que para S^1 se ilustra en (4).

Una vez calculados estos vectores y para ambos sitios, la función *CalcularResultados* retorna el valor de 0, indicando que se generó de manera local un vector de ganancias. Regresando al Algoritmo 11, se comprueba el valor de retorno de la función *CalcularResultados*. Como el valor devuelto es 0, S^1 envía el siguiente conjunto de datos:

Num. de Reg.	ATRIBUTO	G^1	*C.T.D.*	*C.M.I.*
7	*Ambiente*	0.1367	3	0
	Temperatura	0.1367	3	0
	Humedad	0.3544	2	0
	Viento	0.0880	2	0

(3)

C.T.D. significa el costo de transmitir los dominios y *C.M.I.* el costo de los mensajes inútiles. Por otra parte, el sitio S^2 genera su propio conjunto de resultados, dados por:

Num. de Reg.	ATRIBUTO	G^2	*C.T.D.*	*C.M.I.*
7	*Ambiente*	0.3001	3	0
	Temperatura	0.2539	3	0
	Humedad	0.0069	2	0
	Viento	0.1300	2	0

(4)

Por su parte, el servidor, mediante el Algoritmo 8, recibe los resultados ya citados. Esta función almacena en el vector N el número de registros de cada sitio, en el vector G los vectores de ganancia, en C_D los costos de transmitir los dominios de cada atributo en cada sitio y en C_{MI} el costo de los mensajes inútiles generados de considerar el dominio global. Como en ambos sitios se generó un vector de ganancias, la función retorna el valor de 0.

Acto seguido, el servidor retorna al código del Algoritmo 6, donde se comprueba si la función *EsperarRespuesta* retorno un valor 0. Como es el caso, se llama a la función *AtributoGanador*, Algoritmo 9, la cual basándose en los vectores de ganancia recibidos, obtiene un atributo ganador A_g, que para este caso es *Ambiente*. Una vez conocido el atributo ganador, esta función calcula que dominio es más económico usar: el dominio global o el dominio derivado de la unión de los dominios locales de cada sitio. Este cálculo se basa en los valores correspondientes al atributo *Ambiente* de los vectores C_D y C_{MI}. Para el problema que se analiza se determina que es más económico usar el dominio global, asignando dicho dominio a la variable D_g así como a cada vector de dominio $D[D^i]$ asociado a cada sitio. Finalmente, esta función regresa el atributo ganador A_g.

De regreso al Algoritmo 6, se crea un nuevo árbol δ, el cual es etiquetado con el atributo *Ambiente*. En seguida, se entra en un ciclo que recorrerá todos los posibles valores v almacenados en la variable D_g (que para este caso, son los valores del dominio global del atributo *Ambiente*), siendo los valores *soleado, nublado* y *lluvia*.

Para cada valor v que se analice en el ciclo, se crea una rama para dicho valor, la cual es asociada al nodo del atributo correspondiente, que en este caso es *Ambiente*. Por cuestiones didácticas, analicemos en primera instancia el caso para la rama *soleado*.

Para este valor se envía un mensaje a los sitios S^i pertenecientes a $\aleph^N$ indicándoles que los cálculos con los que deben de continuar corresponden a la rama *Ambiente – soleado*. Como se consideró el dominio global, a ambos sitios se les enviará dicho mensaje. Enviado el mensaje, a la rama *Ambiente – soleado* le será asociado un árbol derivado de llamar recursivamente a la función *Recursivo*, la cual recibirá los resultados que los sitios construyan derivados del mensaje que les acaba de enviar. Notemos que los parámetros que son utilizados para dicha función son actualizados, donde $\aleph^N = \{S^1, S^2\}$, $\aleph^F = \varnothing$, Ψ - {*Ambiente*}, donde Ψ = {*Ambiente, Temperatura, Humedad, Viento*} y 1 correspondiente al parámetro R, indicando que ningún sitio ha terminado. Este llamado da lugar a la segunda iteración del algoritmo.

Ya en la segunda iteración, el algoritmo llama a la función *EsperarRespuesta*, la cuál se encargará de recolectar los resultados enviados por los sitios.

Regresando a los sitios, tanto S^1 como S^2 reciben el mensaje *Ambiente – soleado*, Algoritmo 13. En ese momento, el algoritmo verifica que tipo de instrucción está enviando el servidor. En este caso, la estructura del mensaje es del tipo *atributo – valor*, por lo que se pasa a almacenar en la variable A_g el valor *Ambiente* y en *v* el valor de *soleado*. En seguida, se verifica si el nodo padre, que es δ', está etiquetado con el atributo *Ambiente*. Si no tiene está etiqueta o no existe un nodo padre, quiere decir que estamos en un ciclo en el cuál se va a crear un nuevo nodo δ que será etiquetado con el atributo A_g.

Una vez creado el nodo δ, se añade a este una rama para el valor *soleado*, la cuál se va a desarrollar en el presente ciclo. En seguida, ambos sitios obtendrán sus respectivos conjuntos de registros $\Im'$ de $\Im$, los cuales van a cumplir con la condición de que en el campo correspondiente al atributo *Ambiente* tienen el valor *soleado*. Posteriormente, los sitios llaman a la función *CalcularResultados*.

En ese punto, los sitios toman rumbos distintos. Por una parte, S^1 obtiene el vector de coberturas de las clases, dado por:

$$C^1 = \{[N, 1], [P, 2]\} \tag{5}$$

La estructura del vector indica que aún existen registros de distintas clases, por lo que el sitio construye el vector de ganancias para los atributos presentes en Ψ. Finalmente, ya en el Algoritmo 13, se envían al servidor tanto el vector de cobertura, el vector de ganancias y los costos respectivos de la transmisión de los dominios y de los mensajes inútiles (6).

Num. de Reg.	ATRIBUTO	G^1	*C.T.D.*	*C.M.I.*
7	*Temperatura*	0.5793	3	0
	Humedad	1.0	2	0
	Viento	0.2740	2	0

(6)

Por otra parte, para S^2, el vector de coberturas estará dado por:

$$C^2 = \{[N, 2]\} \tag{7}$$

Aquí notamos que ya solo existe una clase para los registros que se analizan. Lo anterior implica que ya no se generará un vector de ganancias, simplemente se construirá el vector de dominios D^2 correspondiente a los atributos presentes en Ψ, dado por:

ATRIBUTO	D^2
Temperatura	*alta, media*
Humedad	*Alta*
Viento	*No, si*

(8)

Tanto (7) como (8) son los resultados enviados por S^2 al servidor.

Por su parte, el servidor recibe el conjunto de resultados tanto de S^1 como de S^2. Sin embargo, como detecta que S^2 ha finalizado el crecimiento de la rama, le manda un mensaje al sitio S^1 para que le informe del dominio de los atributos asociados a la rama que se trabaja, debido a que serán necesarios para verificar algún posible patrón disjunto. Al enviarle este mensaje, el sitio S^1 devuelve el siguiente vector de dominios:

ATRIBUTO	D^1
Temperatura	*alta, media, baja*
Humedad	*Alta, normal*
Viento	*no, si*

(9)

Ya en el Algoritmo 6, se verifica que aun existan sitios que puedan seguir operando, es decir, $\aleph^N \neq \varnothing$. Como aún existe el sitio S^1, se pasa a ver cuál fue el valor de retorno de la función *EsperarRespuesta*. Como se comprueba que es 1, se llama a la función *PatronesDisjuntos* para analizar que no existan patrones disjuntos ente los conjuntos de sitios $\aleph^N$ y $\aleph^F$, Algoritmo 10. Esta función lo que va a tratar de hacer es encontrar algún valor de cualquier de los dominios de los atributos de (8) que no pertenezca al dominio de los atributos que se expresan en (9). Como no existe ningún valor que cumpla esta condición, se regresa el valor de vacío.

Por lo anterior, en el Algoritmo 6 se calcula el atributo ganador basado en el vector de ganancias (6) que se recibió en la iteración en curso, resultando como atributo ganador *Humedad*. Además, también se determina que lo más conveniente es manejar el dominio global de dicho atributo.

Acto seguido, se crear un nodo δ el cuál es etiquetado con dicho atributo ganador. Posteriormente, se entra a un ciclo en el cuál a éste nodo se le van a añadir tantas ramas como valores v existan en el dominio del atributo *Humedad*. Suponiendo que empezamos con el ciclo correspondiente al valor *alta*, se le añade a δ una rama para este valor. En seguida, al sitio S^1 (que pertenece a $\aleph^N$) se le envía el mensaje que continuar las operaciones con la rama *Humedad – alta*, mientras que al sitio S^2 se le notifica que debe añadir a la rama *Ambiente – soleado* el nodo *Humedad – alta*. A ambos sitios se les pide devolver los dominios de los atributos que se asocien a dicha rama.

Después de haber enviado estos mensajes, a la rama *Humedad – alta* se le asociará el árbol que regrese la función *Recursivo*, que es llamada con los parámetros $\aleph^N = \{S^1\}$, $\aleph^F = \{S^2\}$, $\Psi = \{$*Temperatura, Viento*$\}$ y $R = 0$. Una vez que se ha entrado a dicha función, el servidor esperará la respuesta de los sitios correspondientes al desarrollo de ésta rama.

Analicemos que pasa en S^1. Dicho sitio recibe el mensaje del servidor en el que se le indica desarrollar la rama *Humedad – alta* y además, devolver el dominio de los atributos considerados para dicha rama. Recibido el mensaje, el sitio guarda en las variables correspondientes el atributo y el valor indicado. Además, verifica que el valor del dominio señalado por el servidor pertenezca al dominio local. Como es el caso, se crea un nodo δ, el cual es etiquetado con el atributo *Humedad*, añadiéndole al mismo una rama correspondiente al valor *alta*. En seguida, se obtiene el conjunto de registros $\Im'$ de $\Im$ que tengan en el campo del atributo *Humedad* el valor de *alta*. En base a éste conjunto $\Im'$, se llama a la función *CalcularResultados*, la cual inmediatamente obtendrá el vector de coberturas indicado en (10).

$$C^1 = \{[N, 1]\} \tag{10}$$

Como el vector sólo contiene una clase, el proceso de construcción de la rama ya no continuará. Por ello, el sitio S^1 envía éste vector de coberturas en conjunto con el vector de dominios que se ha construido (11).

ATRIBUTO	D^1
Temperatura	*Alta*
Viento	*No*

(11)

Enviados estos resultados, se llama nuevamente de forma recursiva a la función *Recursivo,* con lo que se espera un nuevo mensaje del servidor.

Por su parte, S^2 recibirá la indicación de añadir a la rama *Ambiente – soleado* un nuevo nodo correspondiente al atributo *Humedad*, añadiéndole al mismo una rama para el valor *alta*. En seguida, se obtiene el conjunto de registros $\Im'$ de $\Im$ que tengan en el campo correspondiente al atributo *Humedad* el valor de *alta*. En base a éste conjunto $\Im'$, se calcula el vector de dominios y coberturas, dados por:

$$C^2 = \{[N, 2]\} \tag{12}$$

ATRIBUTO	D^2
Temperatura	*alta, media*
Viento	*no, si*

(13)

Enviados los resultados ya citados, el sitio S^2 llama a la función *Recursivo*, con el objetivo de esperar alguna nueva indicación del servidor correspondiente a la rama *Humedad – alta*.

Por su parte, el servidor recibe los resultados de los sitios en la función *EsperarRespuesta*. Como la función tiene el valor 0 en el parámetro *R*, indica que debe esperar resultados del conjunto de sitios $\aleph^N$, que está conformado hasta el momento únicamente por el sitio S^2. De este sitio, recibe el vector de coberturas (12) que es almacenado en *C* y el vector de dominios (13), almacenándolo en $D[D^2]$. Además, recibe los resultados del conjunto $\aleph^N$, en el cuál se encuentra S^1. Por una parte, recibe el vector de coberturas de clases (10), el cuál es añadido a *C*, quedando $C = \{[N, 3]\}$. Por otra parte, recibe el vector de dominios (11), el cuál es almacenado en $D[D^1]$.

Ya en el Algoritmo 6, se detecta que $\aleph^N = \emptyset$, es decir, todos los sitios han finalizado el crecimiento de la rama que se está trabajando, por lo que se crea un nuevo nodo δ, el cuál será etiquetado con la clase ganadora que obtengamos del vector C, que es N. Hecho lo anterior, se enviará un mensaje tanto a S^1 como a S^2 en el cuál se les informa que ya se ha terminado de construir la rama *Humedad – alta*, regresando la función *Recursivo* el nodo δ. Por tanto, el crecimiento de la rama *Ambiente – soleado – Humedad – alta* ha concluido.

Por su parte los sitios, al recibir el mensaje del servidor que les indica que ha concluido el desarrollo de la rama, finalizarán el desarrollo de la misma. Con ello, se regresa del llamado recursivo que desarrolló a la rama *Humedad – alta* y se entra a otro llamado recursivo para desarrollar cualquier posible rama hermana.

Finalizado el desarrollo de ésta rama, el servidor iniciará un nuevo ciclo para el valor *normal* del atributo *Humedad* (recordemos que este dominio se obtuvo de (9)). Primero se crea una rama para dicho valor, para luego enviar un mensaje al sitio S^1 pertenecientes a $\aleph^N$ indicarle que se pasa a analizar la rama *Humedad – normal*. Por otra parte, a S^2, que pertenece a $\aleph^F$, se le indica que debe añadir un nodo correspondiente a la rama *Humedad – normal* y regresar el vector de coberturas y dominios de la misma. Acto seguido, el servidor asociará a dicha rama el árbol que se obtenga de llamar a la función *Recursivo* con los parámetros $\aleph^N = \{S^1\}$, $\aleph^F = \{S^2\}$, $\Psi = \{Temperatura, Viento\}$ y $R = 0$.

Una vez que el sitio S^1 reciba el mensaje del servidor *Humedad – normal*, verifica si la etiqueta del nodo padre δ' coincide con el atributo que le está indicando el servidor. Como es el caso, ya no se crea un nuevo nodo, simplemente se actualiza la variable δ al valor del nodo δ'. En seguida, se crea una rama para el valor *normal*, la cual es añadida al nodo δ. Acto seguido, se calcula el conjunto de registros $\Im$' de $\Im$, los cuales deben tener en el campo correspondiente al atributo *Humedad* el valor de *normal*. Posteriormente, se llama a la función *CalcularResultados*, Algoritmo 12, la cuál construye el siguiente vector de coberturas C:

$$C^1 = \{[P, 2]\} \tag{14}$$

Como el total de registros manejados son de una sola clase, se construye el siguiente vector de dominios:

ATRIBUTO	D^1
Temperatura	*media, baja*
Viento	*no, si*

(15)

Dicha función finalmente retorna el valor de 1, enviando los vectores (13) y (14) al servidor. Posteriormente, se vuelve a llamar a la función *Recursivo* para desarrollar la rama *Humedad - normal*.

Por su parte, S^2 recibirá un mensaje del tipo añadir nodo y devolver dominio. Para ello, crea un nodo δ el cuál es etiquetado con el atributo *Humedad*, añadiendo al nodo una rama para el valor *normal*. Acto seguido, se calcula el conjunto de registros $\Im$' basado en $\Im$, donde los registros tengan en el campo correspondiente al atributo *Humedad* el valor de *normal*. Notemos que éste

conjunto no contendrá ningún registro, por lo que los vectores de coberturas de clases y de dominios serán vacíos, así como se muestra a continuación:

$$C^2 = \{\} \tag{16}$$

ATRIBUTO	D^2

(17)

Una vez transmitidos estos resultados, S^2 llama nuevamente a la función *Recursivo*, la cual esperará una nueva indicación del servidor.

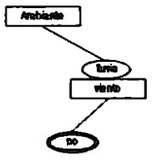
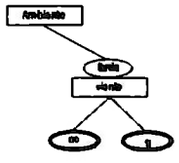
Al recibir los resultados, el servidor va a detectar que la totalidad de los sitios ha finalizado el crecimiento de la rama, con lo cual ya no desarrollará la rama, rematando a la misma con la clase de mayor cobertura contenida en el vector *C*. Para este caso, dicho vector estará conformado de la unión de los vectores (14) y (16), resultando ganadora la clase P, la cual es utilizada para etiquetar a un nuevo nodo δ. Acto seguido, se manda un mensaje a ambos sitios informando que se ha concluido el desarrollo de la rama *Humedad – normal*.

Al recibir los sitios el mensaje del servidor, estos finalizan el crecimiento de la rama *Ambiente – soleado - Humedad – normal*, regresando de un llamado recursivo de la función *Recursivo*. A partir de este instante, los sitios entraran en otro llamado recursivo para desarrollar cualquier rama hermana de *Humedad – normal* que no se haya analizada. Sin embargo, el servidor verifica que ya no existe ninguna rama por analizar para el nodo *humedad*, con lo cual, este envía un mensaje a los sitios informándoles de ello. Por tanto, los sitios como el servidor concluyen totalmente los llamados recursivos asociados a la rama *Ambiente – soleado*, entrando a otra iteración que desarrollará cualquiera de las dos ramas hermanas de esta que aún no se han desarrollado.

Una situación similar de desarrollo a la ya expuesta se presenta para la rama *Ambiente – lluvia*. En el desarrollo de está rama, solo participará activamente el sitio S^1, mientras que el sitio S^2 no participara debido a que desde un principio los registros asociados a está rama son de la clase P. Al igual que en el caso anterior, no se encuentran patrones disjuntos entre ambos sitios, por lo cuál, al etiquetarse las ramas con una clase, se ven influenciados estos resultados por los registros del sitio S^2.

En el cuadro siguiente, se resumen los mensajes que los sitios y el servidor intercambian en el proceso de construcción de la rama *Ambiente – lluvia*.

<table>
<tr><th>#</th><th>RAMA</th><th colspan="2">MENSAJE DEL SERV. A LOS SITIOS</th><th>RESPUESTA DEL SITIO S^1</th><th>RESPUESTA DEL SITIO S^2</th></tr>
<tr><td>1</td><td>  </td><td>S^1</td><td>Ambiente – lluvia</td><td>$C^1 = \{[P, 1]\}$
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr></table></td><td>Num. Reg.: 4
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th><th>G</th></tr><tr><td>Temperatura</td><td>baja, media</td><td>0.3836</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td>S^2</td><td>Ambiente – lluvia</td><td><table><tr><td>Temperatura</td><td>baja</td></tr><tr><td>Humedad</td><td>normal</td></tr><tr><td>Viento</td><td>no</td></tr></table></td><td><table><tr><td>Humedad</td><td>normal, alta</td><td>0</td></tr><tr><td>Viento</td><td>no, si</td><td>1</td></tr></table><table><tr><th>ATRIBUTO</th><th>C.T.D</th><th>C.M.I.</th></tr><tr><td>Temperatura</td><td>2</td><td>2</td></tr><tr><td>Humedad</td><td>2</td><td>0</td></tr><tr><td>Viento</td><td>2</td><td>0</td></tr></table></td></tr>
<tr><td>2</td><td>  </td><td>S^1</td><td>Agrega la rama Viento – no y devuelve el vector de coberturas y dominios de dicha rama</td><td>$C^1 = \{[P, 1]\}$<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td>Temperatura</td><td>baja</td></tr><tr><td>Humedad</td><td>normal</td></tr></table></td><td>$C^2 = \{[P, 2]\}$<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td>Temperatura</td><td>alta, media</td></tr><tr><td>Humedad</td><td>media, normal</td></tr></table></td></tr>
<tr><td></td><td></td><td>S^2</td><td>Viento - no</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>S^1</td><td colspan="3">Finaliza el crecimiento de la rama Ambiente – lluvia – Viento – no</td></tr>
<tr><td></td><td></td><td>S^2</td><td colspan="3">Finaliza el crecimiento de la rama Ambiente – lluvia – Viento – no</td></tr>
<tr><td>4</td><td>  </td><td>S^1</td><td>Viento – si</td><td>$C = \{\}$<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td></td><td></td></tr></table></td><td>$C = \{[N, 2]\}$<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td>Temperatura</td><td>baja, media</td></tr><tr><td>Humedad</td><td>alta, normal</td></tr></table></td></tr>
<tr><td></td><td></td><td>S^2</td><td>Viento - si</td><td></td><td></td></tr>
<tr><td>5</td><td></td><td>S^1</td><td colspan="3">Finaliza el crecimiento de la rama Ambiente – lluvia – Viento – si</td></tr>
<tr><td></td><td></td><td>S^2</td><td colspan="3">Finaliza el crecimiento de la rama Ambiente – lluvia – Viento – si</td></tr>
<tr><td>6</td><td></td><td>S^1</td><td colspan="3">Finaliza el crecimiento de la rama Ambiente – lluvia</td></tr>
<tr><td></td><td></td><td>S^2</td><td colspan="3">Finaliza el crecimiento de la rama Ambiente – lluvia</td></tr>
</table>

Hasta el momento se han expuesto dos ramas asociadas al nodo *Ambiente*. Analicemos la única rama que falta por estudiar, que es *Ambiente – nublado*. Para este caso, el servidor envía tanto a S^1 como a S^2 el mensaje de continuar con dicha rama. En S^1, el sitio encontrará que el total de sus registros asociados a la rama son de la clase P. Por ello, en respuesta, envía el vector de coberturas:

$$C^1 = \{[P, 3]\} \qquad (18)$$

Además le envía el correspondiente vector de dominios, dado por:

ATRIBUTO	D^1
Temperatura	*alta, baja*
Humedad	*alta, normal*
Viento	*no, si*

(19)

Por otra parte, el sitio S^2 descubre que solo un registro se asocia a la rama *Ambiente – nublado*, por lo que envía el vector de coberturas:

$$C^2 = \{[P, 1]\} \tag{20}$$

y el vector de dominios:

ATRIBUTO	D^2
Temperatura	*media*
Humedad	*alta*
Viento	*si*

(21)

Con los resultados anteriores, el servidor inmediatamente deduce que está rama ya no se desarrollará, rematando a la misma con la clase P, y enviando un mensaje a los sitios con el cuál finaliza el desarrollo de está rama.

Como ya no existe ninguna rama por analizar correspondiente al atributo *Ambiente*, finalmente envía un mensaje a todos los sitios finalizando el desarrollo de dicho nodo y con ello, se termina de construir el árbol global.

Notemos que la estructura del árbol que se construye en el servidor finalmente es la que se ilustra en la Figura 5.3, es decir, coincide con el árbol que se construye de analizar todos los datos en una sola base. Lo anterior nos da una buena referencia con respecto a la exactitud de los resultados, ya que el árbol de la Figura 5.3. refleja de manera precisa los patrones globales de los datos analizados.

Por otra parte, también podemos concluir para este ejemplo en particular que la partición que se realizó sobre los datos no afectó a los patrones presentes en los mismos. Sin embargo, esto no se podrá garantizar para cualquier ejemplo, es decir, no siempre se obtendrá de una BDD el mismo árbol que para el caso centralizado, si es que se conoce.

Como ya se tiene con certeza la totalidad de los mensajes que se requirió transmitir para construir el árbol en el servidor, se puede calcular el flujo que de está construcción se derivó. Basándonos en las ecuaciones Ecuación 3, Ecuación 4 y Ecuación 5, las cuales determinan los costos de manejar el dominio local o global de los sitios, así como los costos de transmitir los resultados que se generan para cada iteración, la siguiente tabla ilustra la cantidad de datos transmitidos para cada una de las tres ramas del árbol.

RAMAS / MENSAJES	*Inicio de operaciones*	*Ambiente - lluvia*	*Ambiente - nublado*	*Ambiente - soleado*
DEL SERVIDOR A LOS SITIOS	$2\ T_{MC}$	$12*T_D + 7*T_{MC}$	$4*T_D + 2*T_{MC}$	$12*T_D + 7*T_{MC}$
DE LOS SITIOS AL SERVIDOR	$0*T_D + 8*T_G + 10*T_N$	$23*T_D + 3*T_G + 11*T_N$	$11*T_D + 0*T_G + 2*T_N$	$29*T_D + 3*T_G + 3*T_N$

Donde los T_x significan:

- T_{MC}: tamaño de los mensajes de controlo del servidor a los sitios (inicio y finalización de alguna rama).
- T_D: tamaño necesario para identificar cualquier valor de los datos, ya sea atributos o valores de los dominios.
- T_G: tamaño necesario para representar la ganancia de un atributo. Como es un número flotante, éste puede ser representado por 4 bytes, aunque se puede idear alguna técnica para disminuir este tamaño, ya que el valor variará en el rango [0, 1].
- T_N: tamaño necesario para indicar el número de registros que se trabajan en cada sitio. Normalmente se podría representar con un entero, que son 4 bytes, aunque en algunos casos, puede ser necesario un mayor tamaño debido a las dimensiones de las BD's. Además, los costos de transmitir los dominios y de los mensajes inútiles también se pueden representar con esta cantidad.

De la tabla anterior obtenemos el flujo total del sistema dado por: $91T_D + 14T_G + 26T_N + 18T_{MC}$. Es evidente que el flujo de información en relación a la cantidad de datos que se están trabajando en la BDD es grande. Sin embargo, se puede deducir que esto se debe principalmente al número tan pequeño de atributos presentes en cada sitio, por lo que casi para cada registro, se generó un flujo de datos para generar una rama. Esto, por supuesto, se espera que no pase en BDD's donde el número de registros presentes en cada sitio sea grande.

Este ejercicio ha servido para verificar la precisión de los resultados, así como para comprobar el crecimiento de ramas de manera desequilibrada entre los dos sitios sin existir algún patrón disjunto. También sirvió para comprobar que la forma en que se rematan a las hojas fue correcta.

Ahora tomemos como base el siguiente ejemplo en el cual se presenta el caso de patrones disjuntos.

Pruebas enfocadas a la veracidad de los resultados: 2º ejercicio

Para este ejemplo, tomaremos como base a los datos de la Tabla 1, la cuál es dividida en dos fragmentos para simular a una BDD compuesta por 2 sitios, donde S^1 maneja el conjunto de registros de la Tabla 5, mientras S^2 contiene los registros de la Tabla 6.

Iniciemos presentando el árbol que se genera de aplicar el Algoritmo 1 (caso centralizado) a los registros de la Tabla 1:

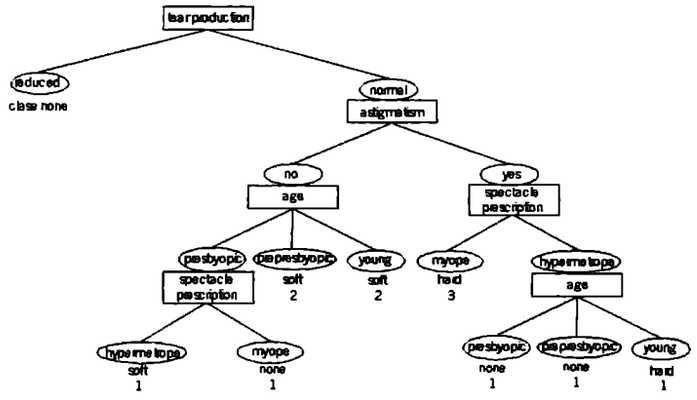


Figura 7.1. Árbol ID3 correspondiente a los registros de la Tabla 1.

Al aplicar el algoritmo distribuido propuesto tanto a S^1 como ha S^2, el árbol obtenido en el servidor resultó ser el mismo. Ello aumenta la veracidad en los resultados que este algoritmo esta arrojando. Lo interesante de esta prueba es que en el desarrollo del árbol se presenta el caso de los patrones disjuntos.

Notemos que S^1 es un sitio en el cual la totalidad de sus registros son de la clase *none*, mientras que el sitio S^2 maneja tres tipos de clases. Es evidente que S^1 solo emitirá para la primera iteración del algoritmo el vector de coberturas y de dominios, mientras que S^2 generará normalmente un vector de ganancias. Lo interesante de éste ejemplo es como el algoritmo desde un inicio separa a los patrones disjuntos de ambos sitios, con lo cuál se logra que los patrones de un sitio no interfieran con los patrones del otro sitio.

En las tablas siguientes podemos observar que a partir de la iteración 1, el servidor no vuelve a enviar mensajes al sitio S^1, por detectar que ya no participará en el desarrollo del árbol que se construye. Esto se debe a que en esta primera iteración, el servidor encontró patrones disjuntos entre los dominios del atributo *Tear Production* de cada sitio. Al identificar esto, el algoritmo asoció a la rama *reduced* la clase de los registros del sitio S^1, mientras que para la rama *normal* fue necesario desarrollar todo un árbol para clasificar a los registros del sitio S^2. De hecho, el árbol asociado a dicha rama coincide exactamente con el que se obtiene de aplicar el algoritmo centralizado ID3 al sitio S^2.

<table>
<tr><th>#</th><th colspan="2">MENSAJE DEL SERV. A LOS SITIOS</th><th>RESPUESTA DEL SITIO S^1</th><th>RESPUESTA DEL SITIO S^2</th></tr>
<tr><td rowspan="2">1</td><td>S^1</td><td>Mensaje de inicio de operaciones</td><td rowspan="2">C^1 = {[none, 12]}
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td>Age</td><td>Young, prepresbyopic, presbyopic</td></tr><tr><td>Spectacle Prescription</td><td>Miope, hypermetrope</td></tr><tr><td>Astigmatism</td><td>No, yes</td></tr><tr><td>Tear production</td><td>Reduced</td></tr></table></td><td rowspan="2">Num. Reg.: 12
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Age</td><td>0.1395</td><td>3</td><td>0</td></tr><tr><td>Spectacle Prescription</td><td>0.0954</td><td>2</td><td>0</td></tr><tr><td>Astigmatism</td><td>0.7704</td><td>2</td><td>0</td></tr><tr><td>Tear production</td><td>0.0</td><td>1</td><td>2</td></tr></table></td></tr>
<tr><td>S^2</td><td>Mensaje de inicio de operaciones</td></tr>
</table>

<table>
<tr><td>2</td><td>S^2</td><td>Mensaje para que regrese los dominios</td><td></td><td>
<table>
<tr><th>ATRIBUTO</th><th>DOMINIO</th></tr>
<tr><td>Age</td><td>Young, prepresbyopic, presbyopic</td></tr>
<tr><td>Spectacle Prescription</td><td>Miope, hypermetrope</td></tr>
<tr><td>Astigmatism</td><td>No, yes</td></tr>
<tr><td>Tear production</td><td>normal</td></tr>
</table>
</td></tr>
<tr><td rowspan="2">3</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">Num. Reg.: 12
<table>
<tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr>
<tr><td>Age</td><td>0.1395</td><td>3</td><td>0</td></tr>
<tr><td>Spectacle Prescription</td><td>0.0954</td><td>2</td><td>0</td></tr>
<tr><td>Astigmatism</td><td>0.7704</td><td>2</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Tear production - normal</td></tr>
<tr><td rowspan="2">4</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">Num. Reg.: 6
<table>
<tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr>
<tr><td>Age</td><td>0.1998</td><td>3</td><td>0</td></tr>
<tr><td>Spectacle Prescription</td><td>0.1908</td><td>2</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Astigmatism - no</td></tr>
<tr><td rowspan="2">5</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[soft, 2]}
<table>
<tr><th>ATRIBUTO</th><th>DOMINIO</th></tr>
<tr><td>Spectacle Prescription</td><td>Myope, hypermetrope</td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Age - young</td></tr>
<tr><td>6</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – no – Age - young</td></tr>
<tr><td rowspan="2">7</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[soft, 2]}
<table>
<tr><th>ATRIBUTO</th><th>DOMINIO</th></tr>
<tr><td>Spectacle Prescription</td><td>Myope, hypermetrope</td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Age - prepresbyopic</td></tr>
<tr><td>8</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – no – Age – prepresbyopic</td></tr>
<tr><td rowspan="2">9</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">Num. Reg.: 2
<table>
<tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr>
<tr><td>Spectacle Prescription</td><td>1.0</td><td>2</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Age - presbyopic</td></tr>
<tr><td rowspan="2">10</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[none, 1]}
<table>
<tr><th>ATRIBUTO</th><th>DOMINIO</th></tr>
<tr><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Spectacle Prescription - myope</td></tr>
<tr><td>11</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – no – Age – prepresbyopic – Spectacle Prescription - myope</td></tr>
<tr><td rowspan="2">12</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[soft, 1]}
<table>
<tr><th>ATRIBUTO</th><th>DOMINIO</th></tr>
<tr><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>S^2</td><td>Spectacle Prescription - hypermetrope</td></tr>
</table>

<table>
<tr><td>13</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – no – Age – presbyopic – Spectacle Prescription – hypermetrope</td></tr>
<tr><td>14</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – no – Age – prepresbyopic</td></tr>
<tr><td>15</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – no</td></tr>
<tr><td rowspan="2">16</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">Num. Reg.: 6
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Age</td><td>0.1587</td><td>3</td><td>0</td></tr><tr><td>Spectacle Prescription</td><td>0.4591</td><td>2</td><td>0</td></tr></table></td></tr>
<tr><td>S^2</td><td>Astigmatism - yes</td></tr>
<tr><td rowspan="2">17</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[hard, 3]}
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td>Age</td><td>Young, prepresbyopic, presbyopic</td></tr></table></td></tr>
<tr><td>S^2</td><td>Spectacle Prescription - myope</td></tr>
<tr><td>18</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – yes – Spectacle Prescription - myope</td></tr>
<tr><td rowspan="2">19</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">Num. Reg.: 3
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Age</td><td>0.5793</td><td>3</td><td>0</td></tr></table></td></tr>
<tr><td>S^2</td><td>Spectacle prescription - hypermetrope</td></tr>
<tr><td rowspan="2">20</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[hard, 1]}
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td></td><td></td></tr></table></td></tr>
<tr><td>S^2</td><td>Age - young</td></tr>
<tr><td>21</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – yes – Spectacle Prescription – hypermetrope – Age – young</td></tr>
<tr><td rowspan="2">22</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[none, 1]}
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td></td><td></td></tr></table></td></tr>
<tr><td>S^2</td><td>Age - prepresbyopic</td></tr>
<tr><td>23</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – yes – Spectacle Prescription – hypermetrope – Age - prepresbyopic</td></tr>
<tr><td rowspan="2">24</td><td>S^1</td><td></td><td rowspan="2"></td><td rowspan="2">C^2 = {[none, 1]}
<table><tr><th>ATRIBUTO</th><th>DOMINIO</th></tr><tr><td></td><td></td></tr></table></td></tr>
<tr><td>S^2</td><td>Age - presbyopic</td></tr>
<tr><td>25</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – yes – Spectacle Prescription – hypermetrope – Age - presbyopic</td></tr>
<tr><td>26</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – yes – Spectacle Prescription – hypermetrope</td></tr>
<tr><td>27</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal – Astigmatism – yes</td></tr>
<tr><td>28</td><td>S^2</td><td colspan="3">Finalizar la rama Tear production – normal</td></tr>
<tr><td rowspan="2">29</td><td>S^1</td><td colspan="3">Finalizar</td></tr>
<tr><td>S^2</td><td colspan="3">Finalizar</td></tr>
</table>

De la tabla anterior, podemos deducir que el flujo de datos derivado de la construcción del árbol es de $58T_D + 41T_N + 17T_{MC} + 13T_G$. Notemos que a pesar que el número de registros aumentó con respecto al ejemplo anterior, el volumen total de información intercambiada fue menor. Con ello, podemos corroborar que el flujo de datos estará dado directamente por los patrones que se encuentren en el proceso de construcción y no por el número de registros.

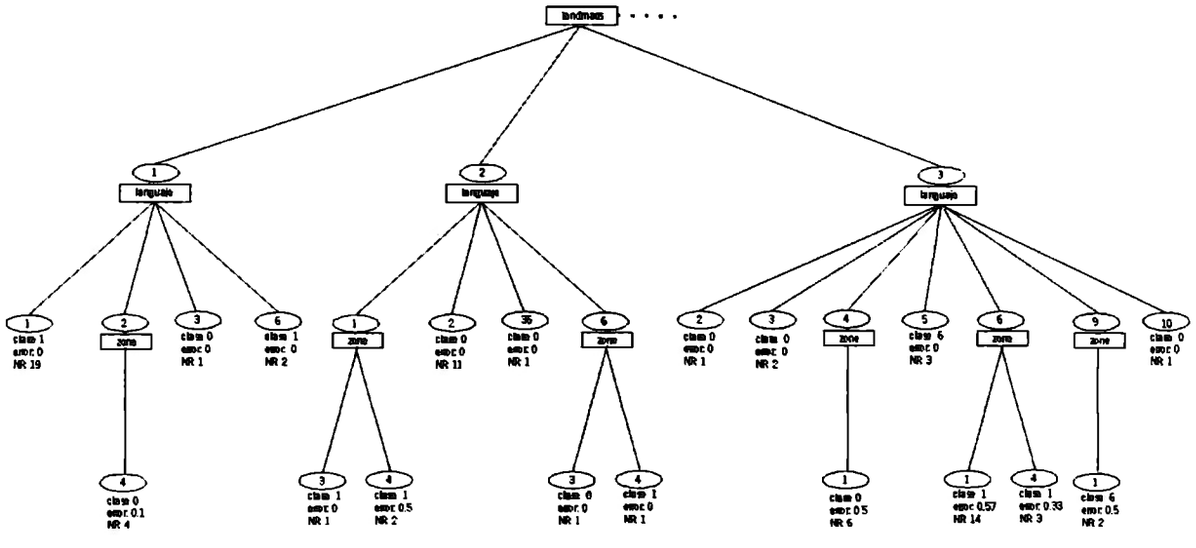
Prueba enfocada al flujo de datos

Ahora analizaremos parte de un ejemplo basado en una BD con un número mayor de registros [45]. En está BD se hace una recolección de características generales de diversos países del mundo[2], como: situación geográfica, población, idioma, religión, características de la bandera del país, como colores, tipos de figuras que maneja, etc. En conjunto, la BD tiene 30 campos de diversa índole, con un total de 194 registros, sin ningún valor faltante.

El objetivo de ésta base es establecer cualquier tipo de relación que al investigador le parezca interesante. En nuestro caso, al analizar la BD, nos pareció interesante clasificar a los países en base a su religión, la cuál puede ser: 0, católico; 1, algún otro tipo de Cristianismo; 2, Musulman; 3, Budista; 4, Indú; 5, étnico; 6, Marxista; 7, otros.

Para ello, se usaron los atributos *posición geográfica* (landmass), lenguaje (languaje) y zona (zone). Estos atributos manejan el siguiente dominio de valores:

- *Posición geográfica*: 1, Norteamérica; 2, Sudamérica; 3, Europa; 4, África; 5, Asia; 6, Oceanía.
- *Lenguaje*: 1, Inglés; 2, Español; 3, Francés; 4, Alemán; 5, Eslavo; 6, Indo-europeo, Chiño, Arábigo; 9, Japonés, Turco, Finlandés; 10, otros.
- *Zona*: cuadrante geográfico, basado en el meridiano de Greenwich y el ecuador: 1, NE; 2, SE; 3, SO; 4, NO.



[2] El trabajo de recolección de los datos fue realizado para: Collected primarily from the "Collins Gem Guide to Flags": Collins Publishers (1986). Investigación de: Richard S. Forsyth.

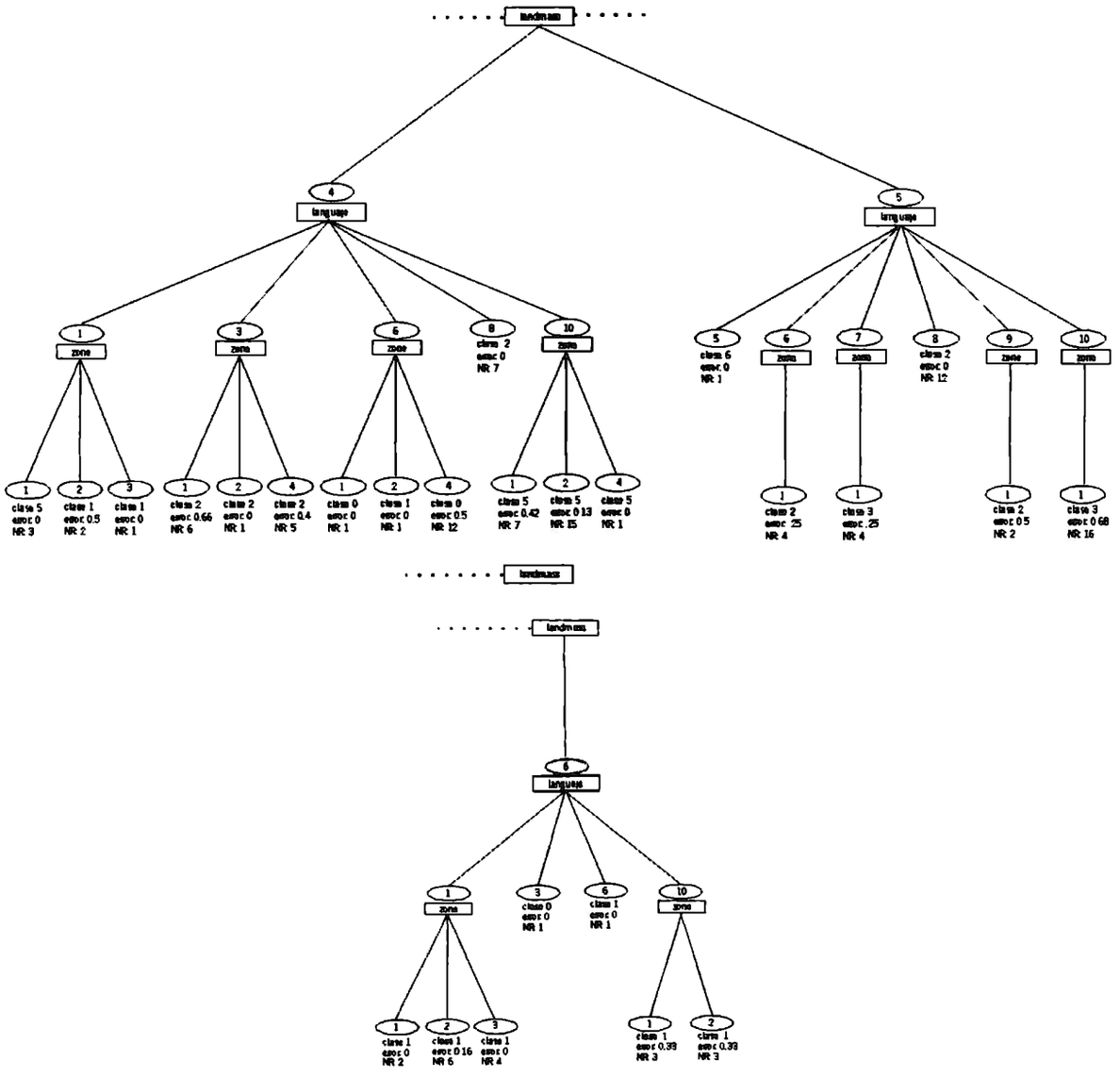


Figura 7.2. Árbol de clasificación basado en la BD banderas con el atributo clasificador religión y los atributos landmass, languaje y zone.

Desde el punto de vista del autor, no se usaron el resto de los atributos clasificadores por no presentar alguna relación lógica, aunque esto variará dependiendo del enfoque o visión de cada investigador.

Si se analiza el total de los registros basándonos en el atributo clasificador *religion* y los atributos *landmass*, *languaje* y *zone*, el árbol que se obtiene es el ilustrado en la Figura 7.2.

Para simular una BDD, el conjunto de registros fue dividido en dos bases. Para ello, se ordenaron los registros en orden alfabético según el nombre del país (notemos que estos no guardan un orden lógico entre sí). Una vez ordenados, se formó el sitio S^1 con los registros de los países de la letra A a la letra K y con el resto de los registros, de la letra L a la Z se formó el sitio S^2.

Una vez formados los sitios, a estos les fue aplicado el algoritmo propuesto, obteniendo un árbol casi idéntico al del caso centralizado. La diferencia entre ambos árboles radicó en la rama

landmass – 2 – languaje – 6. En el caso centralizado, como lo podemos observar en la Figura 7.2., aún se añade un subárbol a está rama correspondiente al atributo *zone*, la cuál genera dos ramas en las que se asocia un registro a cada una de ellas. En consecuencia, el error que se asocia a dichas ramas es 0. Sin embargo, en el caso distribuido, en S^1 sólo quedó un registro con la clase 0, mientras en S^2 quedó el otro registro de la case 1. Observemos que en ambos sitios el proceso de construcción finaliza, ya que cada uno de ellos determina que los registros asociados a está rama son de una sola clase. Por ende, envían los vectores de cobertura, finalizando el desarrollo de dicha rama. El servidor, al recibir estos resultados, busca la clase de mayor cobertura con la cuál rematar dicha rama. Como ambas clases tienen la misma cobertura, se elige cualquiera de las dos, manejando en cualquier caso un error de precisión de 0.5.

Otro aspecto interesante que no se había considerado en los ejemplos anteriores es que en este árbol se manejó tanto el dominio local de los atributos como el global. Por una parte, para generar el nodo raíz, se consideró el dominio global del atributo *landmass*. Sin embargo, para el resto de los atributos seleccionados, el algoritmo determinó importar los dominios, ya que estos implicaban un menor flujo de datos. Al analizar el árbol resultante nos podemos dar cuenta el porqué de ésta acción: a partir de la primera partición derivada del nodo raíz, los dominios de los atributos decrecen considerablemente, esto es, menos de la mitad de los valores originales de los registros quedan asociados a cada rama, por lo que, considerar el dominio global en el servidor, implicaría un flujo excesivo de mensajes inútiles o innecesarios.

Exponer a detalle el flujo total de mensajes es largo, por la gran cantidad de ramas que tiene este árbol, por lo cuál solo expondremos a detalle el desarrollo de la rama *landmass – 5 – languaje – 10 – zone – 1* (ésta rama fue elegida al azar). En la siguiente tabla se ilustran los mensajes intercambiados para su construcción.

<table>
<tr><th>#</th><th colspan="2">MENSAJE DEL SERV. A LOS SITIOS</th><th>RESPUESTA DEL SITIO S^1</th><th>RESPUESTA DEL SITIO S^2</th></tr>
<tr><td rowspan="2">1</td><td>S^1</td><td>Mensaje de inicio de operaciones</td><td rowspan="2">Num. Reg.: 96
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Landmass</td><td>0.4643</td><td>6</td><td>0</td></tr><tr><td>Languaje</td><td>0.4213</td><td>10</td><td>0</td></tr><tr><td>zone</td><td>0.4200</td><td>4</td><td>3</td></tr></table></td><td rowspan="2">Num. Reg.: 98
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Landmass</td><td>0.4218</td><td>6</td><td>0</td></tr><tr><td>Languaje</td><td>0.4532</td><td>10</td><td>0</td></tr><tr><td>zone</td><td>0.3348</td><td>4</td><td>3</td></tr></table></td></tr>
<tr><td>S^2</td><td>Mensaje de inicio de operaciones</td></tr>
<tr><td rowspan="2">2</td><td>S^1</td><td>Landmass – 5</td><td rowspan="2">Num. Reg.: 16
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Languaje</td><td>0.4888</td><td>5</td><td>5</td></tr><tr><td>zone</td><td>0.0</td><td>1</td><td>7</td></tr></table></td><td rowspan="2">Num. Reg.: 23
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>Languaje</td><td>0.4505</td><td>6</td><td>4</td></tr><tr><td>zone</td><td>0.0</td><td>1</td><td>7</td></tr></table></td></tr>
<tr><td>S^2</td><td>Landmass - 5</td></tr>
<tr><td rowspan="2">3</td><td>S^1</td><td>Envía dominio languaje</td><td rowspan="2"><table><tr><th>ATRIBUTO</th><th>Dominio</th></tr><tr><td>Languaje</td><td>6, 7, 8, 9, 10</td></tr></table></td><td rowspan="2">Num. Reg.: 23
<table><tr><th>ATRIBUTO</th><th>Dominio</th></tr><tr><td>Languaje</td><td>5, 6, 7, 8, 9, 10</td></tr></table></td></tr>
<tr><td>S^2</td><td>Envía dominio languaje</td></tr>
<tr><td rowspan="2">4</td><td>S^1</td><td>Landmass – 5</td><td rowspan="2">Num. Reg.: 16
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>zone</td><td>0.0</td><td>1</td><td>7</td></tr></table></td><td rowspan="2">Num. Reg.: 23
<table><tr><th>ATRIBUTO</th><th>GANANCIA</th><th>C.T.D.</th><th>C.M.I.</th></tr><tr><td>zone</td><td>0.0</td><td>1</td><td>7</td></tr></table></td></tr>
<tr><td>S^2</td><td>Landmass - 5</td></tr>
<tr><td rowspan="2">5</td><td>S^1</td><td>Envía dominio zone</td><td rowspan="2"><table><tr><th>ATRIBUTO</th><th>Dominio</th></tr><tr><td>Zone</td><td>1</td></tr></table></td><td rowspan="2">Num. Reg.: 23
<table><tr><th>ATRIBUTO</th><th>Dominio</th></tr><tr><td>Zone</td><td>1</td></tr></table></td></tr>
<tr><td>S^2</td><td>Envía dominio zone</td></tr>
<tr><td>6</td><td>S^1</td><td>Zone - 1</td><td>$C^1 = \{[3,3], [2,1], [7,1]\}$</td><td>$C^2 = \{[6,4], [2,2], [4,1], [7,1], [3,2]\}$</td></tr>
</table>

	S^2	*Zone - 1*		ATRIBUTO	Dominio			ATRIBUTO	Dominio	
7	S^1	Finaliza desarrollo rama *landmass – 5 – language – 10 – zone – 1*								
	S^2	Finaliza desarrollo rama *landmass – 5 – language – 10 – zone – 1*								
7	S^1	Finaliza desarrollo rama *landmass – 5 – language – 10*								
	S^2	Finaliza desarrollo rama *landmass – 5 – language – 10*								

De la tabla anterior, podemos deducir que el flujo para construir dicha rama es de $37T_D + 12T_G + 38T_N + 6T_{MC}$. Notemos que, a pesar de que el número de registros analizados fue mayor, no aumento el flujo de datos. Este mismo comportamiento esperaríamos si analizamos cada una de las ramas que constituyen al árbol.

Adicionalmente, se han realizado una serie de pruebas enfocadas a comparar la precisión de los resultados tanto del algoritmo que se ha propuesto como del centralizado, utilizando un conjunto de prueba. Para tal efecto, se usaron las bases Ambiente y Lentes de Contacto, separando el total de sus registros en dos conjuntos: uno de entrenamiento y de prueba. El conjunto de entrenamiento fue utilizado como entrada para la construcción del árbol, fragmentándolo a su vez en dos sitios para simular una BDD.

Los resultados que se obtuvieron tanto de aplicar el algoritmo ID3 centralizado como el algoritmo propuesto se resumen en la tabla siguiente.

Num. Reg.	Conj. Ent.	Conj. Prueba	Presc. Cent.	Presc. Dist.	No. Pruebas
14	14	0	-	-	1
14	10	4	62.5%	52%	5
14	12	2	75%	60%	7
194	185	9	49%	50%	6
194	189	5	72%	68%	9
194	192	2	94%	93%	15
194	172	20	50%	50%	2

Los resultados anteriores muestran que la precisión del árbol distribuido es muy cercana a la precisión que arroja el árbol centralizado. De hecho, en promedio la precisión del algoritmo ID3 centralizado fue de 67.08%, mientras que la precisión del árbol obtenido del algoritmo propuesto fue de 62.16%, existiendo una diferencia en la precisión de 4.92% a favor del algoritmo centralizado, siendo una muy aceptable diferencia de precisión.

Analizando un poco más de cerca los resultados, podemos notar que cuando se analizaron pocos registros, la precisión de los resultados tuvo una mayor diferencia a favor del algoritmo centralizado. Esto se esperaba que pasara, ya que por tratarse de un número muy pequeño de registros, los patrones presentes en cada sitio tienen una mayor probabilidad de ser disjuntos. Más sin embargo, cuando el número de registros aumentó, la cantidad alojada en cada sitio fue mayor, permitiendo que estos tuvieran o abarcarán un mayor número de patrones, con lo cual los patrones de los registros de cada sitio coincidieron en mayor número. Ello favoreció al algoritmo

propuesto, ya que permitió identificar con mayor precisión estos patrones, dando como resultado un árbol que tuvo una precisión muy similar al árbol centralizado.

De hecho, de la tabla anterior, si se toman en cuentea exclusivamente los resultados basados en la BD de 194 registros, notaremos que la precisión del árbol centralizado fue de 66.25%, mientras que del algoritmo propuesto fue de 65.25%, existiendo una diferencia mínima de 1% a favor del algoritmo centralizado.

Por otra parte, notemos que la complejidad temporal de los algoritmos es radicalmente distinta, ya que, mientras el algoritmo centralizado analiza el total de los registros, la complejidad del algoritmo propuesto para cada nodo dependerá de aquel sitio que analice más registros para la iteración, número que es mucho menor al total de los registros.

Consideramos que las pruebas ya expuestas son suficientes para demostrar que el algoritmo propuesto se comporta de una manera satisfactoria de acuerdo a los lineamientos que se plantearon desde el inicio.

8 CONCLUSIONES Y TRABAJO A FUTURO

El desarrollo del presente proyecto deja varias conclusiones. Un primer aspecto que podemos señalar es referente a la naturaleza distribuida del método que se ha propuesto, ya que permite construir un árbol de clasificación ID3 a pesar de que los datos se encuentren distribuidos en varios sitios. Ello puede resultar muy beneficioso en situaciones en las cuales la naturaleza descentralizada de las empresas obliga a adaptar BDD's para administrar su información. Con el algoritmo propuesto, se pueden realizar análisis de clasificación sin la necesidad de reunir estos datos en una sola base.

Por otro lado y basándonos en los resultados de las pruebas realizadas, concluimos que la precisión de los resultados será equiparable a los resultados del algoritmo ID3 centralizado. Sin embargo, se debe de señalar que la precisión de dichos resultados tiende a ser mejor cuando se cuenta con un buen conjunto de entrenamiento en cada sitio. Lamentablemente, en la mayoría de las ocasiones no se conoce ningún tipo de patrón en los datos, por lo cual no se puede asegurar que el conjunto de entrenamiento que se está empleando sea el más idóneo. Por tal motivo, al igual que en el algoritmo centralizado, es recomendable realizar varias pruebas con distintos conjuntos de entrenamiento, buscando aquél o aquellos que presenten la mayor precisión.

Otra conclusión que se obtuvo de realizar las pruebas es que, entre mayor fuera la muestra de registros en cada sitio, los patrones presentes en cada uno de ellos se comportaron de manera similar, es decir, disminuyo la probabilidad de que un conjunto de sitios finalizaran antes que otros el desarrollo de una rama. Lo anterior permitió que el algoritmo explotara de forma adecuada el procesamiento de cada uno de los sitios manteniéndolos ocupados el mayor tiempo. Sin embargo, esta situación no se presentó durante todas las pruebas.

Se observó que cuando un conjunto de sitios finalizan una rama de un nodo λ en un nivel n, estos sitios no apoyan al procesamiento de información cuando el resto de los sitios trabajan en el desarrollo de ramas hermanas derivadas del nodo λ. Por tanto, cuando se presentan situaciones como las ya planteadas, el algoritmo no aprovecha al máximo el poder de cómputo distribuido que podría ofrecer la BDD.

Relacionado con el punto anterior, se debe señalar que el flujo de datos requerido para la construcción puede aumentar si se presenta el caso en el que un conjunto de sitios $\aleph^F$ finaliza el desarrollo de una rama en un nivel n y otro conjunto de sitios $\aleph^N$ continúa desarrollando dicha rama y además, no se encuentran patrones disjuntos en los niveles inmediatos inferiores. Lamentablemente esta situación no

se puede controlar, ya que depende de los patrones de los datos. De acuerdo a las pruebas realizadas, si se desea obtener una buena precisión en los resultados, el precio que se puede pagar es que aumente el flujo de datos. Sin embargo, lo ya expuesto puede no llegar a ser tan negativo, ya que lo que se implementó para manejar dicha situación requiere informar al servidor de los dominios de los atributos que aún se estén considerando y los dominios, en general, no llegan a ser muy grandes (se restringen a un número modesto de posibles valores).

De acuerdo a las conclusiones anteriores, se plantea realizar el siguiente trabajo a futuro. En primer término, se propone implementar alguna estrategia en el equilibrio de cargas de trabajo, con el objetivo de mantener el mayor tiempo posible ocupados a los diversos sitios. Ello se debe a que cuando algunos sitios finalizan el desarrollo de una rama y otros no, los primeros no realizan ningún cálculo, desperdiciando su capacidad de procesamiento. Es claro que implementar una técnica debe de cuidar en mantener un bajo flujo de datos.

Otro aspecto a mejorar es el carácter centralizado del esquema propuesto, ya que si el servidor llega a detener sus operaciones, el resto de los sitios no podrían continuar el proceso de construcción. Para ello, se podrían implementar espejos del servidor, de tal manera que si éste llega a fallar, otro sitio tome su lugar, con el objetivo de que el proceso no se detenga.

También se observó que es conveniente manejar o implementar técnicas de podado en el proceso de construcción distribuido, ya que esto ayudaría a disminuir aún más el flujo de datos y además, evitar el sobreajuste (caso que se presenta cuando una rama, por más que se desarrolle, no se logra mejorar la clasificación de los registros) que se puede presentar. Ello conlleva a hacer estudios de las técnicas de podado ya existentes o de nuevas técnicas para implementarlas en el análisis de una BDD.

9 Apéndice A: Tablas

NumReg	age	spectacle prescription	astigmatism	tear production	recommended lents
1	young	myope	No	reduced	none
2	young	myope	No	normal	soft
3	young	myope	Yes	reduced	none
4	young	myope	Yes	normal	hard
5	young	hypermetrope	No	reduced	none
6	young	hypermetrope	No	normal	soft
7	young	hypermetrope	Yes	reduced	none
8	young	hypermetrope	Yes	normal	hard
9	prepresbyopic	myope	No	reduced	none
10	prepresbyopic	myope	No	normal	soft
11	prepresbyopic	myope	Yes	reduced	none
12	prepresbyopic	myope	Yes	normal	hard
13	prepresbyopic	hypermetrope	No	reduced	none
14	prepresbyopic	hypermetrope	No	normal	soft
15	prepresbyopic	hypermetrope	Yes	reduced	none
16	prepresbyopic	hypermetrope	Yes	normal	none
17	presbyopic	myope	No	reduced	none
18	presbyopic	myope	No	normal	none
19	presbyopic	myope	Yes	reduced	none
20	presbyopic	myope	Yes	normal	hard
21	presbyopic	hypermetrope	No	reduced	none
22	presbyopic	hypermetrope	No	normal	soft
23	presbyopic	hypermetrope	Yes	reduced	none
24	presbyopic	hypermetrope	Yes	normal	none

Tabla 1. Contact lents

Ambiente	Temperatura	Humedad	Viento	Clase
soleado	alta	alta	no	N
soleado	alta	alta	si	N
nublado	alta	alta	no	P
lluvia	media	alta	no	P
lluvia	baja	normal	no	P
lluvia	baja	normal	si	N
nublado	baja	normal	si	P
soleado	Media	alta	no	N
soleado	Baja	normal	no	P
lluvia	Media	normal	no	P
soleado	Media	normal	si	P
nublado	Media	alta	si	P
nublado	Alta	normal	no	P
lluvia	Media	alta	si	N

Tabla 2. Estado del tiempo

Ambiente	Temperatura	Humedad	Viento	Clase
soleado	alta	alta	no	N
nublado	alta	alta	no	P
lluvia	baja	normal	no	P
nublado	baja	normal	si	P
soleado	baja	normal	no	P
soleado	media	normal	si	P
nublado	alta	normal	no	P

Tabla 3. Estado del tiempo, fragmento 1

Ambiente	Temperatura	Humedad	Viento	Clase
soleado	alta	alta	si	N
lluvia	media	alta	no	P
lluvia	baja	normal	si	N
soleado	media	alta	no	N
lluvia	media	normal	no	P
nublado	media	alta	si	P
lluvia	media	alta	si	N

Tabla 4. Estado del tiempo, fragmento 2

NumReg	age	spectacle prescription	astigmatism	tear production	recommended lents
1	young	myope	No	reduced	none
3	young	myope	Yes	reduced	none
5	young	hypermetrope	No	reduced	none
7	young	hypermetrope	Yes	reduced	none
9	prepresbyopic	myope	No	reduced	none
11	prepresbyopic	myope	Yes	reduced	none
13	prepresbyopic	hypermetrope	No	reduced	none
15	prepresbyopic	hypermetrope	Yes	reduced	none
17	presbyopic	myope	No	reduced	none
19	presbyopic	myope	Yes	reduced	none
21	presbyopic	hypermetrope	No	reduced	none
23	presbyopic	hypermetrope	Yes	reduced	none

Tabla 5. Contact lents, fragmento 1

NumReg	age	spectacle prescription	astigmatism	tear production	recommended lents
2	young	myope	No	normal	soft
4	young	myope	Yes	normal	hard
6	young	hypermetrope	No	normal	soft
8	young	hypermetrope	Yes	normal	hard
10	prepresbyopic	myope	No	normal	soft
12	prepresbyopic	myope	Yes	normal	hard
14	prepresbyopic	hypermetrope	No	normal	soft
16	prepresbyopic	hypermetrope	Yes	normal	none
18	presbyopic	myope	No	normal	none
20	presbyopic	myope	Yes	normal	hard
22	presbyopic	hypermetrope	No	normal	soft
24	presbyopic	hypermetrope	Yes	normal	none

Tabla 6. Contact lents, fragmento 2.

10 REFERENCIAS

[1]. C. J: DATE. Introducción a los Sistemas de Bases de Datos. Addison Wesley Iberoamericana. 1986.

[2]. ROBERT GROTH. Data Mining. The data warehousing institute series. Prentice Hall,1997.

[3]. STEFANO CERI; GIUSEPPE PELAGATTI. Distributed Databases. Principles & Systems. McGraw Hill, 1987.

[4]. GROSSMAN, ROBERT. *et. al.* "Data Mining Research: Opportunities and Challenges", Enero de 1999. <http://www.ncdm.uic.edu/dmr-v8-4-52.htm> (4 de Abril del 2000).

[5]. The National Center for Data Mining <http://www.ncdm.uic.edu>. 20 de Abril del 2000.

[6]. Thearling, Kurt. "An introduction to Data Mining". <http://www3.shore.net/~kht/text/dmwhite/dmwhite.htm> (28 de Febrero del 2000)

[7]. MICHAEL J. A. BERRY; GORDON LINOFF. "Data Mining Techniques. For Marketing, Sales, and Customer Support". Wiley Computer Publishing. 1997.

[8]. PETER CABENA, PABLO, *et al.* "Discovering Data Mining. From concept to implementation". IBM, 1997.

[9]. IAN H. WITTEN; EIBE FRANK. "Data Mining". Morgan Kaufmann Publishers.1999.

[10]. DORIAN PYLE. "Data preparation for Data Mining. Morgan Kaufmann publishers". 1999.

[11]. MORALES, EDUARDO. "Descubrimiento de Conocimiento en Bases de Datos". ITESM. <http://w3.mor.itesm.mx/~emorales/emorales-esp.html> (27 de Febrero del 2000)

[12]. GUZMÁN ARENAS, ADOLFO. "Uso y diseño de Mineros de Datos". Revista Soluciones Avanzadas. <http://www.fciencias.unam.mx/revista/soluciones/30s/No34/mineria.html> (27 de Febrero del 2000)

[13]. PANDE JOSHI, KARUNA. "Analysis of Data Mining Algorithms". <http://www.gl.umbc.edu/~kjoshi1/data-mine/proj_rpt.htm> (4 de Marzo del 2000).

[14]. HOLSHEIMER, MARCEL, *et al.* Data Mining, The search for Knowledge in Databases.

[15]. http://yoda.cis.temple.edu:8080/UGAIWWW/lectures95/learn/C45/

[16]. MICHAEL J. A. BERRY, *et. al.* "Mastering Data Mining. The art and Science of customer relationship management". Wiley Computer Publishing.

[17]. REESE H., SARA. "Parallelism speeds data mining". IEEE Parallel & Distributed Technology

[18]. ESTER, MARTIN, *et. Al.* "Incremental Clustering for Mining in a Data Warehousing Enviroment". Institute of Computer Science, University of Munich.

[19]. JÁJÁ, JOSEPH. "An Introduction to Parallel Algorithms". Addison Wesley, 1992.

[20]. K. HWANG. "Advanced Computer Arcitecture. Parallelism, Scalability, Programmability". Mc Graw Hill. 1993.

[21]. SAPE MULLENDER. "Distributed Systems". Addison Wesley, 1993.

[22]. Andina de la Fuente, Diego, A. Sandham, William. Politécnica de Madrid-UPM. <http://www.gc.ssr.upm.es/inves/neural/ann2/anntutorial.html >. (27 de Diciembre 2000)

[23]. R. P. LIPPMANN. "An introduction to Computing with Neural Nets". IEEE Computer Society Press. Vol. 4. No 2., Abril 1987, pp 4 – 22.

[24]. HAYKIN, SIMON. "Neural Networks. A Comprehensive foundation". Prentice Hall, 1999.

[25]. T. HAGAN, MARTÍN, *et. al.* . "Neural Network Design". PWS Publishing Company, 1996.b

[26]. FU, LIMIN. "Neural Networks in Computer Intelligence". McGraw – Hill, 1994.

[27]. R. BEALE, T. JACKSON. "Neural Computing: An Introduction". Institute of Physics Publishing, 1990.

[28]. A. FREEMAN JAMES, M. SKAPURA DAVID. "Neural Networks. Algorithms, Applications, and Programming Techniques". Addison – Wesley. 1992.

[29]. Enciclopedia Microsoft Encarta 99. Microsoft Corporation. 1999.

[30]. <http://webserver.pue.udlap.mx/~pgomez/cursos/rna/notas/is466_not.html> (27 de Diciembre 2000)

[31]. BELL, DAVID. GRIMSON, JANE. "Distributed Database Systems", Addison Wesley.

*[32]. M. TAMER ÖZSU, VALDURIEZ PATRICK. "Principles of Distributed Database Systems". Prentice Hall, 1984.

[33]. BERSON, ALEX. J. SMITH, STEPHEN. "Data Warehousing, Data Mining & OLAP". Mc. Graw Hill, 1997.

[34]. MANNILA, HEIKKI. "Methods and problems in Data Mining". Departament of Computer Science, University of Helsinki.

[35]. SKILLICORN, DAVID. "Strategies for Parallel Data Mining". IEEE, 1999.

[36]. LI, BIN. SHASHA, DENNIS. "Free Parallel Data Mining". Departament of Computer Science, New York University.

[37]. LAM, WAI. SAGRE, ALBERTO MARÍA. "Distributed Data Mining of Probabilistic Knowledge". IEEE, 1997.

[38]. W. CHEUNIG, DAVID. *et. al.* "Efficient Mining of Association Rules in Distributed Databases". IEEE, 1996.

[39]. SRIVASTAVA, ANURAG. *et. al.* "Parallel Formulations of Decisión – Tree Clasification Algorithms". Data Mining and Knowledge discovery, 1999.

[40]. DEAN, THOMAS. *et. al.* "Artificial Intelligence. Theory and Practice". Addison – Wesley, 1995.

[41]. BOFFEY, BRIAN. "Distributed Computing". Blackwell Scientific Publications. 1992.

[42]. KUMAR, VIPIN. *et. al.* "Introduction to Parallel Computing. Design and Analysis of Algorithms". The Benjamin/Commings Publishing Company, Inc. 1994.

[43]. ANDREWS R. GREGORY. "Multithreaded, Parallel, and Distributed Programming". Addison – Wesley, 2000.

[44]. CODENOTTI, BRUNO. LEONCINI, MAURO. "Introduction to Parallel Processing". Addison – Wesley, 1993.

[45]. Information and Computer Science. University of California, Irvine. <http://www.ics.uci.edu/~mlearn/MLRepository.html>