UNIVERSIDAD DE GRANADA
E.T.S. INGENIERÍA INFORMÁTICA

Departamento de
Ciencias de la Computación
e Inteligencia Artificial

TESIS DOCTORAL

ART
Un método alternativo para la construcción de árboles de decisión

Fernando Berzal Galiano

Granada, junio de 2002

ART
Un método alternativo para la construcción de árboles de decisión

memoria que presenta

Fernando Berzal Galiano

para optar al grado de

Doctor en Informática

Junio de 2002

DIRECTOR

Juan Carlos Cubero Talavera

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
E INTELIGENCIA ARTIFICIAL

E.T.S. INGENIERÍA INFORMÁTICA UNIVERSIDAD DE GRANADA

La memoria titulada “ART: Un método alternativo para la construcción de árboles de decisión”, que presenta D. Fernando Berzal Galiano para optar al grado de Doctor, ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección del Doctor Juan Carlos Cubero Talavera.

Granada, junio de 2002.

El Doctorando

El Director

Fdo. Fernando Berzal

Fdo. Juan Carlos Cubero

Agradecimientos

En primer lugar, he de reconocer el esfuerzo, tesón y dedicación de una persona muy especial para mí, mi madre Adelaida, que siempre me ha apoyado en mis decisiones y ha estado ahí en los buenos momentos y en los no tan buenos.

En segundo lugar, pero no por ello de menor importancia, tengo que agradecerle a mi director, Juan Carlos, el interés que ha mostrado por mí desde que hace ya algunos años fui a pedirle una carta de recomendación, tras lo cual acabé siendo "su" becario e hice con él mi Proyecto de Fin de Carrera. Al fin y al cabo, la mera existencia de esta memoria se debe a su persistencia (y también a su paciencia). Durante estos años me ha demostrado su calidad como tutor y, sobre todo, su valía como persona. Espero que en el futuro, pase lo que pase, tenga en él a un gran amigo.

Así mismo, les debo mucho a las personas con las cuales he desarrollado mi trabajo en el seno del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, en particular a mis compañeros de mudanzas de Mecenas y a los integrantes del grupo de investigación IDBIS, desde su diligente y solícita directora hasta el artista del grupo. Mención especial merece mi compañero de despacho, amigo y socio. En cierto sentido, Nicolás se ha convertido en algo así como un hermano mayor para mí (entre otras cosas, por su perspectiva desde el año de ventaja que me lleva).

Por otro lado, tampoco puedo olvidar a los muchos profesores que han ido guiando mi desarrollo académico y personal. Aunque de pequeño quería ser profesor de Geografía e Historia como mi abuelo, del que heredé mi fascinación por los libros, mi inclinación por las Ciencias no tardó demasiado en aparecer. De hecho, mis primeros devaneos con las Matemáticas provienen de mi etapa en EGB con un profesor excepcional, Fernando Barranco, Sch.P., alguien inolvidable para muchos estudiantes que pasamos por los Escolapios de Granada. Allí conocí a profesores inigualables que son, además, bellísimas personas. En concreto, me estoy refiriendo a Mari Carmen López del Amo y a Fernando Martín, dos profesores a los que siempre recordaré con cariño.

Va por todos ellos...

PD: Aparte de las personas mencionadas y de aquéllas a las que haya podido omitir, he de confesar la colaboración de ELVEX*, que ha cumplido sobradamente a pesar de sus frecuentes idas y venidas, y también del viejo* SHERLOCK*, el cual ha realizado su trabajo a duras penas, si bien es cierto que nunca me ha fallado. ¡Ah! Casi me olvido de mi obsoleto CPC, al cual le debo mi pasión por la Informática ;-)*

Índice general

Capítulo 1

Introducción

Hasta hace no demasiado tiempo se utilizaba el término "procesamiento de datos" para describir la utilización de los ordenadores en distintos ámbitos. Hoy se utiliza otro término, IT [Information Technology], que se refiere a lo mismo pero implica un cambio de enfoque. Se hace énfasis no únicamente en el procesamiento de grandes cantidades de datos, sino en la extracción de información significativa de esos datos.

Los datos son información cruda, colecciones de hechos que deben ser procesados para que sean significativos. La información se obtiene asociando hechos (en un contexto determinado). El conocimiento utiliza la información obtenida en un contexto concreto y la asocia con más información obtenida en un contexto diferente. Finalmente, la sabiduría (término que nadie utiliza en IA) aparece cuando se obtienen principios generales de fragmentos de conocimiento.

Hasta ahora, la mayor parte del software ha sido desarrollada para procesar datos, información a lo sumo. En el futuro se trabajará con sistemas que procesen conocimiento. La clave reside en asociar elementos de información provenientes de distintas fuentes y sin conexión obvia de tal forma que la combinación nos proporcione beneficios. Este es uno de los desafíos más importantes de la actualidad: la construcción de sistemas que extraigan conocimiento de los datos de forma que sea práctico y beneficioso.

ROGER S. PRESSMAN
Ingeniería del Software: Un Enfoque Práctico

El aprendizaje en Inteligencia Artificial [53] [92] [94] [140] se entiende como un proceso por el cual un ordenador acrecienta su conocimiento y mejora su habilidad. En él se resaltan dos aspectos complementarios: el refinamiento de la habilidad y la adquisición de conocimiento. Tal como lo definió Simon, el aprendizaje denota cambios en el sistema que son adaptativos en el sentido de que permiten al sistema hacer la misma tarea a partir de la misma posición de un modo más efectivo.

Muchas de las técnicas de aprendizaje usadas en IA están basadas en el aprendizaje realizado por los seres vivos. Para ellos, la experiencia es muy importante, ya que les permite no volver a cometer los mismos errores una y otra vez. Además, la capacidad de adaptarse a nuevas situaciones y resolver nuevos problemas es una característica fundamental de los seres inteligentes. Por lo tanto, podemos aducir varias razones de peso para estudiar el aprendizaje: en primer lugar, como método de comprensión del proceso de aprendizaje (desde el punto de vista de la Psicología) y, en segundo término, aunque no por ello sea menos interesante, para conseguir programas que aprendan (desde una perspectiva más propia de la Inteligencia Artificial).

Una primera clasificación de las técnicas de aprendizaje existentes se puede realizar atendiendo a la filosofía seguida en el proceso de adquisición del conocimiento:

- En el *aprendizaje supervisado* (o aprendizaje a partir de ejemplos, con profesor), los ejemplos de entrada van acompañados de una clase o salida correcta. Esta familia de técnicas engloba al aprendizaje memorístico [*rote learning*], a los modelos de aprendizaje por ajuste de parámetros y a una amplia gama de métodos de construcción de distintos tipos de modelos de clasificación, desde árboles de decisión hasta listas de decisión.

- En el *aprendizaje no supervisado* (aprendizaje por observación, sin profesor) se construyen descripciones, hipótesis o teorías a partir de un conjunto de hechos u observaciones sin que exista una clasificación a priori de los ejemplos. Este tipo de aprendizaje es el que realizan los métodos de agrupamiento o *clustering*.

El aprendizaje con profesor o aprendizaje supervisado, también conocido como clasificación, es uno de los problemas más estudiados en Inteligencia Artificial. En particular, el objetivo de cualquier algoritmo de aprendizaje supervisado es construir un modelo de clasificación a partir de un conjunto de datos de entrada, denominado conjunto de entrenamiento, que contiene algunos ejemplos de cada una de las clases que pretendemos modelar. Los casos del conjunto de entrenamiento incluyen, además de la clase a la que corresponden, una serie de atributos o características que se utilizarán para construir un modelo abstracto de clasificación. El objetivo del aprendizaje supervisado es la obtención de una descripción precisa para cada clase utilizando los atributos incluidos en el conjunto de entrenamiento. El modelo que se obtiene durante el proceso de aprendizaje puede utilizarse para clasificar nuevos ejemplos (casos cuyas clases se desconozcan) o, simplemente, para comprender mejor los datos de los que disponemos. Formalmente, un modelo de clasificación se puede definir de la siguiente manera [39]:

> Si suponemos que todos los ejemplos que el modelo construido ha de reconocer son elementos potenciales de J clases distintas denotadas ω_j, llamaremos $\Omega = \{\omega_j | 1 \leq j \leq J\}$ al conjunto de las clases. En determinadas ocasiones extenderemos Ω con una clase de rechazo ω_0 a la que asignaremos todos aquellos casos para los que no se tiene una certeza aceptable de ser clasificados correctamente en alguna de las clases de Ω. De este modo, denotamos $\Omega^* = \Omega \cup \{\omega_0\}$ al conjunto extendido de clases. Un clasificador o regla de clasificación es una función $d : P \rightarrow \Omega^*$ definida sobre el conjunto de posibles ejemplos P tal que para todo ejemplo X se verifica que $d(X) \in \Omega^*$.

Un modelo de clasificación concreto puede construirse entrevistando a expertos en el tema. De hecho, la construcción de muchos sistemas basados en el conocimiento se basa en la extracción manual del conocimiento de los expertos, a pesar de la dificultad que entraña este proceso. Cuanto mejor es el experto peor suele describir su conocimiento (la paradoja de la Ingeniería del Conocimiento). Además, los expertos en un tema no siempre están de acuer-

do entre sí (la Ley de Hiram: *"Si se consultan suficientes expertos, se puede confirmar cualquier opinión"*).

No obstante, si se dispone de suficiente información registrada (almacenada en una base de datos, por ejemplo), el modelo de clasificación se puede construir generalizando a partir de ejemplos específicos mediante alguna técnica de aprendizaje automático. De hecho, podemos encontrar numerosos ejemplos de algoritmos de aprendizaje automático, como los empleados en la construcción de árboles de decisión (como C4.5 o CART) o los que siguen la metodología STAR de Michalski (INDUCE o AQ, por ejemplo).

Los casos de entrenamiento utilizados en la construcción del modelo de clasificación suelen expresarse en términos de un conjunto finito de propiedades o atributos con valores discretos o numéricos, mientras que las categorías a las que han de asignarse los distintos casos deben establecerse de antemano (al tratarse de aprendizaje supervisado). En general, estas clases serán disjuntas, si bien pueden establecerse jerarquías de clases en las cuales algunas clases son especialización de otras, de modo que las clases no son siempre disjuntas (aunque sí lo suelen ser aquéllas que se encuentran en el mismo nivel de la jerarquía de conceptos empleada para definir las clases del problema). Además, las clases suelen ser discretas, pues si son continuas nos encontramos ante un problema de regresión cuya resolución se puede realizar utilizando técnicas estadísticas. En ocasiones, no obstante, para predecir atributos con valores continuos se definen categorías discretas utilizando términos imprecisos propios del lenguaje natural (esto es, etiquetas lingüísticas que representan conjuntos difusos de valores).

Para que el aprendizaje automático sea correcto, entendiendo éste como un proceso de generalización a partir de ejemplos concretos, hemos de disponer de suficientes casos de entrenamiento (bastantes más que clases diferentes). Si las conclusiones obtenidas no están avaladas por bastantes ejemplos, entonces la aparición de errores en los datos podría conducir al aprendizaje de un modelo erróneo que no resultaría fiable. Por tanto, cuantos más datos obtengamos, más fácilmente podremos diferenciar patrones válidos de patrones debidos a irregularidades o errores.

Por suerte, hoy en día es relativamente fácil recopilar grandes cantidades de

datos relativos al problema que deseemos resolver, pues es sencillo digitalizar información y no resulta excesivamente caro almacenarla. Sin embargo, cuando el tamaño de los conjuntos de datos aumenta considerablemente, muchas de las técnicas tradicionalmente utilizadas en Inteligencia Artificial no resultan adecuadas por ser ineficientes y poco escalables. La necesidad de trabajar eficientemente con grandes conjuntos de datos ha dado lugar al desarrollo de las técnicas de *Data Mining* [17] [75], una rama de las Ciencias de la Computación actualmente en auge [96].

Las técnicas de *Data Mining* se enmarcan dentro del proceso de extracción de conocimiento denominado KDD [59], acrónimo de *Knowledge Discovery in Databases*. Se entiende por KDD la extracción no trivial de información potencialmente útil a partir de un gran volumen de datos en el cual la información está implícita (aunque no se conoce previamente). Su objetivo final es la interpretación de grandes cantidades de datos y el descubrimiento de relaciones o patrones existentes en los datos. Para alcanzar dicho objetivo se emplean algoritmos clásicos de aprendizaje, métodos estadísticos [73] y técnicas de bases de datos [86] [87], lo cual hace del KDD un área de conocimiento eminentemente multidisciplinar.

El proceso de extracción de conocimiento incluye la preparación de los datos y la interpretación de los resultados obtenidos, además de los algoritmos de *Data Mining* propiamente dichos, puesto que de la simple aplicación de técnicas de *Data Mining* sólo se obtienen patrones. Tales patrones no son más que expresiones que describen subconjuntos de los datos de entrada; esto es, son modelos aplicables a los datos de los que se obtuvieron.

Los algoritmos tradicionales de construcción de modelos de clasificación se suelen basar en el descubrimiento de patrones en los datos de entrenamiento y los algoritmos de *Data Mining* proporcionan las técnicas necesarias para poder construir clasificadores de forma eficiente incluso cuando los conjuntos de datos son enormes.

> *Los desarrollos más provechosos han surgido siempre donde se encontraron dos formas de pensar diferentes.*
>
> HEISENBERG

Los árboles de decisión, clasificación o identificación constituyen uno de los modelos más utilizados en aprendizaje supervisado y en aplicaciones de *Data Mining* [68]. Su principal virtud radica en que son modelos de clasificación de fácil comprensión. Además, su dominio de aplicación no está restringido a un ámbito concreto, sino que los árboles de decisión pueden utilizarse en áreas de diversa índole [125], desde aplicaciones de diagnóstico médico hasta sistemas de predicción meteorológica o juegos como el ajedrez.

Los métodos usuales de construcción de árboles de decisión, aun habiéndose utilizado con éxito en incontables ocasiones, carecen de la flexibilidad que ofrecen otras técnicas de inducción de reglas. Éstas, por su parte, aunque pueden llegar a conseguir modelos de clasificación más simples, suelen ser demasiado ineficientes para poder aplicarse con éxito en la resolución de problemas de *Data Mining*.

En esta memoria se pretende diseñar un método que permita construir modelos de clasificación simples, inteligibles y robustos de una forma eficiente y escalable. El objetivo final es lograr un algoritmo eficiente computacionalmente que sea capaz de trabajar con grandes conjuntos de datos, a semejanza de los modernos algoritmos de construcción de árboles de decisión. Estas cualidades se han de conseguir sin olvidar la necesidad de construir clasificadores que destaquen por su simplicidad, tal como los obtenidos mediante algoritmos de inducción de reglas. Por otro lado, el método propuesto ha de ser robusto; es decir, debe ser capaz de funcionar correctamente ante la presencia de ruido en el conjunto de entrenamiento que se utilice para construir el clasificador.

Para cumplir los objetivos identificados en el párrafo anterior, en esta memoria se presenta un método de construcción de modelos de clasificación que combina las mejores cualidades de los árboles de decisión con las de las técnicas de inducción de reglas. El método propuesto da lugar una nueva familia de algoritmos de inducción de árboles de decisión basada en técnicas de extracción de reglas de asociación [19], si bien también puede interpretarse como un algoritmo de inducción de listas de decisión. El uso de técnicas de extracción de reglas de asociación para construir un árbol de decisión da nombre al método propuesto en esta memoria: ART, acrónimo de *Association Rule Trees*.

El modelo de clasificación ART se caracteriza por dotar de mayor flexi-

bilidad a las técnicas tradicionales de construcción de árboles de decisión, al permitir la utilización simultánea de varios atributos para ramificar el árbol de decisión y agrupar en una rama 'else' las ramas del árbol menos interesantes a la hora de clasificar datos, sin descuidar por ello cuestiones relativas a la eficiencia del proceso de aprendizaje. Para lograr dicho propósito se emplean técnicas de extracción de reglas de asociación que nos permiten formular hipótesis complejas de una forma eficiente y métodos de discretización que nos ofrecen la posibilidad de trabajar en dominios continuos.

Como se verá en los capítulos siguientes, la búsqueda de mecanismos complementarios que permitan la aplicación real de ART en distintas situaciones ha dado lugar al desarrollo paralelo de técnicas cuya aplicación no está limitada al modelo de clasificación propuesto, ni siquiera al ámbito del aprendizaje supervisado. En este sentido, además del modelo de clasificación ART (capítulo 3), en esta memoria se presenta un algoritmo eficiente para la extracción de reglas de asociación en bases de datos relacionales (TBAR [19], capítulo 4), un método jerárquico de discretización supervisada (capítulo 5) y una arquitectura distribuida basada en componentes apta para cualquier aplicación de cálculo intensivo (capítulo 6).

Contenido de esta memoria

En el siguiente capítulo se presentan algunos conceptos básicos relacionados con las técnicas en que se basa el modelo de clasificación ART. En concreto, se estudia la construcción de árboles de decisión, la inducción de listas de decisión y la extracción de reglas de asociación, así como el uso de estas últimas en la resolución de problemas de clasificación.

Los algoritmos de construcción de árboles de decisión, estudiados en la sección 2.1, suelen construir de forma descendente los árboles de decisión, comenzando en la raíz del árbol. Por este motivo se suele hacer referencia a este tipo de algoritmos como pertenecientes a la familia TDIDT [*Top-Down Induction of Decision Trees*]. En este sentido, ART es un algoritmo TDIDT más, si bien el criterio de preferencia utilizado para ramificar el árbol de decisión y la topología del mismo difieren notablemente de las propuestas anteriores. Es precisamente la topología del árbol construido por ART la que permite interpretar el modelo de clasificación construido por ART como una lista de decisión. Las listas de decisión, como caso particular de los algoritmos de inducción de reglas, se analizan en la sección 2.2.

El capítulo 2 se cierra con la sección 2.3, en la que se estudian las técnicas de extracción de reglas de asociación y se comenta su posible uso en la construcción de modelos de clasificación.

Tras analizar los tres pilares en los que se asienta el modelo propuesto en esta memoria (árboles de clasificación, listas de decisión y reglas de asociación), el capítulo 3 se dedica por completo a la presentación del modelo de clasificación ART.

En la sección 3.1 se describe el algoritmo ART para la construcción de árboles de decisión. Acto seguido, en el apartado 3.2, aparece un ejemplo detallado que ayuda a comprender el funcionamiento del proceso de inducción llevado a cabo por ART. A continuación, la utilización de ART en la resolución de un problema real es objeto de la sección 3.3. Tras estos ejemplos concretos, que ayudan a entender mejor el modelo de clasificación ART, en la sección 3.4 se analiza el uso de los clasificadores ART (esto es, los modelos de clasificación obtenidos como resultado de aplicar el algoritmo ART).

El método ART obtiene modelos de clasificación excelentes a la vez que es capaz de trabajar adecuadamente con los enormes conjuntos de datos que suelen utilizarse para resolver problemas de extracción de conocimiento en bases de datos y *Data Mining*. Esta cualidad se debe a las buenas propiedades de escalabilidad de los algoritmos de extracción de reglas de asociación que utiliza internamente para ramificar el árbol de decisión. Éstas y otras propiedades del modelo de clasificación ART se comentan en la sección 3.5.

El tercer capítulo de esta memoria se cierra en el apartado 3.6 con la presentación de los resultados experimentales que se han obtenido con el modelo de clasificación propuesto. Como caso particular, se han obtenido empíricamente resultados interesantes al utilizar ART para clasificar uniones de genes en secuencias de ADN, problema en el que ART descubre y aprovecha las relaciones existentes entre los nucleótidos de una secuencia de ADN (véase la figura 3.1 de la página 63, que muestra el clasificador ART obtenido para este problema, cuya descripción aparece en el apartado 3.3 de esta memoria).

Si bien ART no siempre mejora los porcentajes de clasificación obtenidos al utilizar otros clasificadores existentes, como C4.5 [131] o RIPPER [38], los clasificadores ART suelen comportarse bien en términos de porcentaje de clasificación, simplicidad y robustez ante la presencia de ruido en los datos de entrenamiento. Además, ART dota de mayor flexibilidad al proceso tradicional de construcción de árboles de decisión y ofrece una alternativa escalable a las técnicas existentes de inducción de listas de decisión.

Como ya se ha comentado, el modelo de clasificación propuesto en esta memoria emplea internamente un eficiente algoritmo de extracción de reglas de asociación. Dicho algoritmo, denominado TBAR [19], es el núcleo en torno al cual gira el capítulo 4 de esta memoria.

Este algoritmo de extracción de reglas de asociación, como parte de ART, ofrece un mecanismo simple y efectivo para tratar una amplia variedad de situaciones sin necesidad de recurrir a otras técnicas más específicas, complejas y artificiales. Además, su eficiencia y escalabilidad permiten que ART sea perfectamente capaz de trabajar con los enormes conjuntos de datos comunes en problemas de *Data Mining*.

El algoritmo TBAR, como método general de extracción eficiente de reglas

de asociación en bases de datos relacionales, es objeto de estudio en la sección 4.2, mientras que su uso en ART se analiza en la sección 4.3.

El capítulo dedicado a la extracción de reglas de asociación también incluye un apartado, 4.4, en el que se describen distintas medidas que se pueden emplear para evaluar las reglas obtenidas por algoritmos como TBAR.

El algoritmo TBAR, en concreto, está diseñado para mejorar el rendimiento de otros métodos alternativos de extracción de reglas de asociación. De hecho, TBAR permite reducir al máximo los recursos computacionales necesarios para llevar a cabo este proceso, tanto el tiempo de ejecución requerido como el espacio de almacenamiento consumido.

Las reglas que se utilizan para construir el modelo de clasificación ART se obtienen a partir de conjuntos de datos que usualmente contendrán atributos de tipo numérico. Para no limitar innecesariamente la aplicabilidad de técnicas de aprendizaje como ART, se necesitan mecanismos que permitan construir modelos de clasificación con atributos continuos. El capítulo 5 de esta memoria se centra precisamente en el estudio de las técnicas que nos permiten construir árboles de decisión con atributos numéricos.

En la sección 5.1 se analizan las técnicas de discretización existentes, las cuales permiten que cualquier técnica de aprendizaje pueda trabajar con atributos continuos tratándolos como si fueran categóricos. Dado que muchos de los métodos de discretización existentes no aprovechan la información de que disponen, en la sección 5.2, se propone un nuevo método de discretización, el discretizador contextual, que resulta especialmente adecuado cuando se utiliza durante la construcción de modelos de clasificación.

La sección 5.3 está dedicada a la construcción de árboles de decisión con atributos de tipo numérico y en ella se describe cómo puede utilizarse el discretizador contextual de la sección 5.2 para dotar de mayor flexibilidad al proceso de construcción del árbol de decisión.

Los resultados experimentales que se han obtenido al utilizar distintos métodos de discretización se discuten en la sección 5.4, donde se analiza el uso de estas técnicas en la construcción de árboles de decisión TDIDT (apartado 5.4.1) y su aplicación en el modelo de clasificación ART (apartado 5.4.2).

Las principales aportaciones del capítulo 5 son, en primer lugar, la apli-

cación de un método de discretización jerárquico al proceso de construcción de árboles de decisión con atributos continuos y, en segundo lugar, la presentación de un método de discretización alternativo que procura hacer uso de la información disponible para conseguir una partición óptima del dominio de un atributo continuo.

Tras haber estudiado el modelo de clasificación ART y las dos técnicas que lo hacen útil en problemas reales (esto es, la extracción eficiente de reglas de asociación y la discretización de valores continuos), el capítulo 6 de esta memoria se centra en el estudio de la infraestructura computacional que hace factible la aplicación de técnicas de *Data Mining* a gran escala (como puede ser la construcción de clasificadores ART). En concreto, en este capítulo se propone la implementación de un sistema distribuido basado en componentes.

El modelo conceptual de un sistema general de *Data Mining* se presenta en la sección 6.1. Tal sistema sería conveniente que fuese distribuido dadas las necesidades computacionales de las técnicas que trabajan con enormes cantidades de datos. En el apartado 6.2 se propone un sistema descentralizado que pueda hacer frente a dichas necesidades.

La infraestructura planteada en el capítulo 6 debe proporcionar servicios de forma dinámica, por lo cual es recomendable implementarla como un sistema basado en componentes, fácilmente adaptable y reconfigurable, con la arquitectura propuesta en la sección 6.3.

El capítulo dedicado a cuestiones de infraestructura incluye, además, una descripción en el apartado 6.4 de los criterios que se habrían de seguir al diseñar un sistema como el descrito y de cómo se pueden implementar los subsistemas de los que se compone utilizando tecnologías existentes en la actualidad.

La infraestructura propuesta resulta de interés, no sólo para resolver problemas de *Data Mining*, sino para cualquier tipo de aplicación de cálculo intensivo para las que usualmente se recurre al uso de costosos supercomputadores.

Esta memoria se cierra con el capítulo 7, en el cual se exponen algunas de las conclusiones a las que se ha llegado durante el desarrollo de este trabajo, así como algunas sugerencias encaminadas a la realización de futuros trabajos.

En cuanto a los resultados que aparecen reflejados en esta memoria, además del modelo de clasificación ART analizado en el capítulo 3, resulta digno de

mención el hecho de que se hayan desarrollado algunas ideas y técnicas cuyo ámbito de aplicación va más allá de su uso específico en la construcción de clasificadores ART. Entre ellas destacan el algoritmo TBAR de extracción de reglas de asociación en bases de datos relacionales, el método de discretización contextual y la arquitectura distribuida basada en componentes propuesta para la resolución de problemas de cálculo intensivo. Descripciones detalladas de dichas propuestas pueden encontrarse en los capítulos 4, 5 y 6 de la presente memoria, respectivamente.

Capítulo 2

Propedéutica

Propedéutica:
Enseñanza preparatoria para el estudio de una disciplina.

Diccionario de la Real Academia Española de la Lengua

Los pilares sobre los que se asienta el modelo de clasificación propuesto en este trabajo son los árboles de decisión o clasificación, la inducción de listas de decisión y la extracción de reglas de asociación.

La construcción de árboles de decisión, también denominados árboles de clasificación o de identificación, es un conocido método de aprendizaje supervisado que se utiliza a menudo para resolver problemas de clasificación de todo tipo. A pesar de carecer de la expresividad de las redes semánticas o de la lógica de primer orden, la sencillez de los árboles de decisión los convierte en una alternativa muy atractiva de cara al usuario final de un sistema de extracción de conocimiento: el conocimiento obtenido durante el proceso de aprendizaje supervisado se representa mediante un árbol en el cual cada nodo interno contiene una pregunta acerca de un atributo particular (con un nodo hijo para cada posible respuesta) y en el que cada hoja se refiere a una decisión (etiquetada con una de las clases del problema).

Por otro lado, conforme el tamaño los árboles de decisión aumenta, su inteligibilidad disminuye. Tal como se comenta en [129], Shapiro propuso descomponer un árbol de decisión complejo en una jerarquía de pequeños árboles de decisión para obtener un modelo más comprensible, modelo al que denominó inducción estructurada. Sin embargo, es mucho más sencillo expresar el árbol de decisión construido como un conjunto de reglas de producción, un mecanismo de representación del conocimiento más inteligible que los árboles de decisión.

Si bien se pueden derivar reglas de producción a partir de un árbol de decisión con facilidad, también existen métodos de construcción de clasificadores que obtienen reglas de producción directamente, sin tener que construir previamente un árbol de decisión. Estas técnicas, más ineficientes computacionalmente, suelen emplear estrategias de búsqueda heurística como la búsqueda dirigida, una variante de la búsqueda primero el mejor.

En determinadas ocasiones, sin embargo, no se pueden construir modelos de clasificación completos que nos permitan clasificar todos los posibles casos con los que uno se pueda encontrar. A veces, hay que conformarse con descubrir modelos aproximados, los cuales contemplan algunas características de las distintas clases sin que el modelo abarque todas las clases posibles ni todos los casos particulares de una clase determinada. La construcción de un modelo de clasificación completo puede no ser factible cuando hemos de tratar con una gran cantidad de atributos, cuando muchos valores son desconocidos, cuando unos atributos deben modelarse en función de otros o cuando el número de casos de entrenamiento es excesivamente elevado.

Un modelo de clasificación parcial intenta descubrir características comunes a los distintos casos de cada clase sin tener que construir un modelo predictivo completo. En este contexto, la extracción de reglas de asociación puede ser útil para resolver problemas de clasificación parcial en situaciones donde las técnicas de clasificación clásicas no son efectivas.

El problema de la construcción de un modelo de clasificación parcial se puede abordar de dos formas diferentes utilizando reglas de asociación: dividiendo el conjunto de casos de entrenamiento (es decir, creando un subconjunto para cada clase) o considerando la clase como un atributo más. Para cual-

quiera de las dos aproximaciones mencionadas, ha de definirse alguna medida de interés que nos permita seleccionar las reglas de asociación más prometedoras.

En este trabajo se propone la construcción de un modelo de clasificación híbrido que, utilizando árboles de decisión como mecanismo de representación subyacente, intenta aprovechar las mejores cualidades de las reglas de asociación como modelo de clasificación parcial para conseguir un método eficiente de construcción de modelos completos de clasificación que, además, destacan por su robustez.

En las siguientes secciones se realiza un recorrido por las técnicas de aprendizaje supervisado en las que se basa el modelo ART. En primer lugar, describiremos la familia de algoritmos de inducción de árboles de decisión para después comentar algunas técnicas de inducción de reglas. Finalmente, nos centraremos en la extracción de reglas de asociación como método de particular interés a la hora de trabajar con grandes conjuntos de datos y concluiremos este capítulo reseñando algunos trabajos relacionados con el modelo propuesto en el capítulo 3 de esta memoria.

2.1. Árboles de decisión

Los árboles de decisión constituyen probablemente el modelo de clasificación más popular y utilizado (véanse, por ejemplo, las referencias [68] y [130]). Un árbol de decisión puede utilizarse para clasificar un ejemplo concreto comenzando en su raíz y siguiendo el camino determinado por las respuestas a las preguntas de los nodos internos hasta que se llega a una hoja del árbol. Su funcionamiento es análogo al de una aguja de ferrocarril: cada caso es dirigido hacia una u otra rama de acuerdo con los valores de sus atributos al igual que los trenes cambian de vía según su destino (las hojas del árbol) en función de la posición de las agujas de la red de ferrocarriles (los nodos internos).

Los árboles de clasificación son útiles siempre que los ejemplos a partir de los que se desea aprender se puedan representar mediante un conjunto prefijado de atributos y valores, ya sean éstos discretos o continuos. Sin embargo, no resultan demasiado adecuados cuando la estructura de los ejemplos es variable.

Tampoco están especialmente indicados para tratar con información incompleta (cuando aparecen valores desconocidos en algunos atributos de los casos de entrenamiento) y pueden resultar problemáticos cuando existen dependencias funcionales en los datos del conjunto de entrenamiento (cuando unos atributos son función de otros).

En principio, se busca la obtención de un árbol de decisión que sea compacto. Un árbol de decisión pequeño nos permite comprender mejor el modelo de clasificación obtenido y, además, es probable que el clasificador más simple sea el correcto, de acuerdo con el principio de economía de Occam (también conocido como navaja de Occam): "los entes no han de multiplicarse innecesariamente". Este principio, si bien permite la construcción de modelos fácilmente comprensibles, no garantiza que los modelos así obtenidos sean mejores que otros aparentemente más complejos [47] [48].

Por desgracia, no podemos construir todos los posibles árboles de decisión derivados de un conjunto de casos de entrenamiento para quedarnos con el más pequeño. Dicho problema es NP completo [157]. La construcción de un árbol de decisión a partir del conjunto de datos de entrada se suele realizar de forma descendente mediante algoritmos greedy de eficiencia de orden $O(n \log n)$, siendo n el número de ejemplos incluidos en el conjunto de entrenamiento.

Los árboles de decisión se construyen recursivamente siguiendo una estrategia descendente, desde conceptos generales hasta ejemplos particulares. Ésa es la razón por la cual el acrónimo TDIDT, que proviene de *"Top-Down Induction on Decision Trees"*, se emplea para hacer referencia a la familia de algoritmos de construcción de árboles de decisión.

Una vez que se han reunido los datos que se utilizarán como base del conjunto de entrenamiento, se descartan a priori aquellos atributos que sean irrelevantes utilizando algún método de selección de características y, finalmente, se construye recursivamente el árbol de decisión. El método de construcción de árboles de decisión mediante particiones recursivas del conjunto de casos de entrenamiento tiene su origen en el trabajo de Hunt a finales de los años 50. Este algoritmo "divide y vencerás" es simple y elegante:

- Si existen uno o más casos en el conjunto de entrenamiento y todos ellos corresponden a objetos de una misma clase $c \in Dom(C)$, el árbol de decisión es una hoja etiquetada con la clase c. Hemos alcanzado un nodo puro.

- Si no encontramos ninguna forma de seguir ramificando el árbol o se cumple alguna condición de parada (*regla de parada*), no se sigue expandiendo el árbol por la rama actual. Se crea un nodo hoja etiquetado con la clase más común del conjunto de casos de entrenamiento que corresponden al nodo actual. Si el conjunto de casos de entrenamiento queda vacío, la clasificación adecuada ha de determinarse utilizando información adicional (vg. C4.5 opta por la clase más frecuente en el nodo padre).

- Cuando en el conjunto de entrenamiento hay casos de distintas clases, éste se divide en subconjuntos que sean o conduzcan a agrupaciones uniformes de casos, entendiendo por éstas conjuntos de casos correspondientes a una misma clase. Utilizando los casos de entrenamiento disponibles, hemos de seleccionar una pregunta para ramificar el árbol de decisión. Dicha pregunta, basada en los valores que toman los atributos predictivos en el conjunto de entrenamiento, ha de tener dos o más respuestas alternativas mutuamente excluyentes R_i. De todas las posibles alternativas, se selecciona una empleando una regla heurística a la que se denomina *regla de división*. El árbol de decisión resultante consiste en un nodo que identifica la pregunta realizada del cual cuelgan tantos hijos como respuestas alternativas existan. El mismo método utilizado para el nodo se utiliza recursivamente para construir los subárboles correspondientes a cada hijo del nodo, teniendo en cuenta que al hijo H_i se le asigna el subconjunto de casos de entrenamiento correspondientes a la alternativa R_i.

En resumen, cuando se construye o expande un nodo, se considera el subconjunto de casos de entrenamiento que pertenecen a cada clase. Si todos los ejemplos pertenecen a una clase o se verifica alguna regla de parada, el nodo es una hoja del árbol. En caso contrario, se selecciona una pregunta basada en

los atributos predictivos del conjunto de entrenamiento (usando una regla de división heurística), se divide el conjunto de entrenamiento en subconjuntos (mutuamente excluyentes siempre y cuando no existan valores desconocidos ni se empleen, por ejemplo, conjuntos difusos [157]) y se aplica el mismo procedimiento a cada subconjunto del conjunto de entrenamiento.

2.1.1. Reglas de división

Cualquier pregunta que divida el conjunto de casos de entrenamiento en al menos dos subconjuntos no vacíos conducirá a la construcción de un árbol de decisión. No obstante, el objetivo del proceso de construcción de árboles de decisión es obtener un árbol que revele información interesante a la hora de realizar predicciones.

Por lo tanto, cada posible pregunta ha de evaluarse mediante alguna heurística y, dado que los algoritmos de construcción de árboles de decisión suelen ser de tipo greedy, esta heurística desempeña un papel esencial en la construcción del árbol: una vez que se ha escogido una pregunta para expandir un nodo, no se vuelven a considerar otras alternativas.

Las heurísticas estadísticas empleadas intentan favorecer las divisiones que discriminan mejor unas clases de otras. Ejemplos muy conocidos de estas heurísticas son la ganancia de información usada por ID3 [129], el criterio de proporción de ganancia de C4.5 [131] o el índice de diversidad de Gini empleado en CART [23].

Los criterios de división o ramificación generalmente están basados en medidas de la impureza de un nodo. Informalmente, se entiende por impureza de un nodo el grado en el que el nodo incluye casos de distintas clases del problema de clasificación que se pretende resolver con el árbol de decisión. Un nodo puro será, por tanto, aquél al que sólo correspondan casos pertenecientes a una de las clases del problema.

La bondad de una partición es el decrecimiento de impureza que se consigue con ella. La maximización de la bondad de una partición, por tanto, equivale a la minimización de la impureza del árbol generado por la partición (ya que el árbol de partida cuya impureza se quiere reducir es el mismo para las distintas particiones analizadas).

Una función de impureza ϕ mide la impureza de un nodo del árbol. Dado un problema de clasificación con J clases diferentes, la función de impureza suele ser no negativa y se define sobre el conjunto de las *J*-tuplas $(p_1, p_2, .., p_J)$, donde cada p_j indica la probabilidad de que un caso pertenezca a la clase j en el subárbol actual. Obviamente, $\sum p_j = 1$. Cualquier función ϕ ha de poseer las siguientes propiedades (adaptadas de [23]):

- La función ϕ tiene un único máximo en $(1/J, 1/J, .., 1/J)$. La impureza de un nodo es máxima cuando el número de ejemplos correspondientes a cada una de las clases del problema es el mismo para todas ellas; es decir, la distribución de las clases es uniforme.
- La función ϕ alcanza sus J mínimos en $\phi(1, 0, .., 0)$, $\phi(0, 1, .., 0)$ ... $\phi(0, 0, .., 1)$. Además, ϕ es igual a 0 en esos puntos. En otras palabras, un nodo es puro cuando sólo contiene ejemplos de una clase dada.
- La función ϕ es simétrica respecto a p_1, p_2, ... p_J.

La impureza de un árbol de decisión T puede obtenerse a partir de la impureza de sus hojas o nodos terminales $\tilde{T}$ de la siguiente manera:

$$\phi(T) = \sum_{t \in \tilde{T}} p(t)\phi(t)$$

donde $p(t)$ es la probabilidad de que un ejemplo dado corresponda a la hoja t y $\phi(t)$ es la impureza del nodo terminal t.

2.1.1.1. Ganancia de información: Entropía

ID3 [129] intenta maximizar la ganancia de información conseguida por el uso del atributo A_i para ramificar el árbol de decisión mediante la minimización de la función I:

$$I(A_i) = \sum_{j=1}^{M_i} p(A_{ij}) H(C|A_{ij})$$

donde A_i es el atributo utilizado para ramificar el árbol, M_i es el número de valores diferentes del atributo A_i, $p(A_{ij})$ es la probabilidad de que el atributo

A_i tome su *j*-ésimo valor y $H(C|A_{ij})$ es la entropía de clasificación del conjunto de ejemplos en los que el atributo A_i toma su *j*-ésimo valor. Esta entropía de clasificación se define como

$$H(C|A_{ij}) = -\sum_{k=1}^{J} p(C_k|A_{ij}) \log_2 p(C_k|A_{ij})$$

siendo J el número de clases del problema y $p(C_k|A_{ij})$ una estimación de la probabilidad de que un ejemplo pertenezca a la clase C_k cuando su atributo A_i toma su *j*-ésimo valor. En realidad, la estimación de la probabilidad $p(C_k|A_{ij})$ no es más que la frecuencia relativa $f(C_k|A_{ij})$ en el conjunto de entrenamiento utilizado.

La información transmitida en un mensaje depende de la probabilidad del mensaje p y puede expresarse en bits como $-\log_2 p$. Por ejemplo, si tenemos 256 mensaje diferentes, como el número de caracteres ASCII, cada mensaje transporta 8 bits. Si usásemos el logaritmo natural, la información se mediría en *nats*, mientras que serían *hartleys* si empleásemos logaritmos decimales.

La probabilidad de que un caso escogido aleatoriamente pertenezca a la clase C_k es $p(C_k)$ y la información que se obtiene es $-\log_2 p$. La información que esperamos obtener al clasificar un caso cualquiera del conjunto de datos de entrenamiento será igual a $-\sum p(C_k) \log_2 p(C_k)$, cantidad a la que se denomina entropía del conjunto.

La información necesaria para transmitir la división del conjunto de casos de entrenamiento T en M_i subconjuntos T_j es igual a $\sum p(T_j)H(T_j)$, donde $p(T_j)$ es la probabilidad de que un ejemplo pertenezca al subconjunto T_j y $H(T_j)$ es la entropía de clasificación del conjunto T_j.

La ganancia de información que se produce al dividir T en los subconjuntos T_j es igual a $H(T) - \sum p(T_j)H(T_j)$, donde $H(T)$ es la entropía de T. Para comparar las posibles particiones del conjunto T se evalúa la ganancia de información obtenida por cada una de ellas. Al ser $H(T)$ constante, nos basta con comparar el valor de la expresión $-\sum p(T_j)H(T_j)$, que es la utilizada por la regla de división del algoritmo ID3.

Esta heurística suele favorecer la construcción de árboles de decisión con un grado de ramificación elevado, hecho que propició el desarrollo de la siguiente regla de división.

2.1.1.2. El criterio de proporción de ganancia

Aunque usando la ganancia de información se obtienen buenos resultados al construir árboles de decisión, este criterio favorece a aquéllas preguntas que tienen más resultados posibles. Por ejemplo, si cada caso va acompañado de un atributo que lo identifica unívocamente, se elegirá este atributo en la raíz del árbol de forma que cada nodo hijo corresponderá a un único caso. Es decir, se tiende a construir árboles de decisión en los que se utilizan claves o casi claves para ramificar. Si bien se obtiene la máxima ganancia de información posible, el árbol de decisión construido no sirve como modelo de clasificación.

Podemos recurrir nuevamente a la Teoría de la Información para normalizar de algún modo la ganancia obtenida. El contenido de un mensaje que nos indique la respuesta a la pregunta realizada (no la clase a la que pertenece cada caso) es igual a $-\sum p(A_{ij}) \log_2 p(A_{ij})$. Utilizando este resultado podemos redefinir nuestro criterio de división:

$$R(A_i) = \frac{H(C) - \sum_{j=1}^{M_i} p(A_{ij}) H(C|A_{ij})}{-\sum_{j=1}^{M_i} p(A_{ij}) \log_2 p(A_{ij})}$$

Este criterio de división es el utilizado en C4.5 [131]. Cuando la división realizada del conjunto de casos de entrenamiento es trivial, el denominador de $R(A_i)$ es cercano a cero. Por tanto, se ha de escoger el atributo que maximice el cociente $R(A_i)$ siendo su ganancia, al menos, tan grande como la ganancia media de todas las alternativas analizadas.

Dado que en la práctica hemos de disponer de muchos más casos de entrenamiento que clases diferentes, el criterio de proporción de ganancia evitará la construcción de árboles de decisión que clasifiquen los casos utilizando sus claves.

Se ha observado que el criterio de proporción de ganancia tiende a la construcción de árboles poco balanceados, característica que hereda de la regla de división de la que se deriva (la ganancia de información). Ambas heurísticas se basan en una medida de entropía que favorece particiones del conjunto de

entrenamiento muy desiguales en tamaño cuando alguna de ellas es de gran pureza (todos los casos que incluye corresponden a una misma clase) aun siendo poco significativa (es decir, aun abarcando muy pocos casos de entrenamiento).

2.1.1.3. El índice de diversidad de Gini

El índice de diversidad de Gini trata de minimizar la impureza existente en los subconjuntos de casos de entrenamiento generados al ramificar el árbol de decisión. La función empleada es la siguiente:

$$G(A_i) = \sum_{j=1}^{M_i} p(A_{ij}) G(C|A_{ij})$$

$$G(C|A_{ij}) = -\sum_{k=1}^{J} p(C_k|A_{ij}) p(\neg C_k|A_{ij}) = 1 - \sum_{k=1}^{J} p^2(C_k|A_{ij})$$

donde A_i es el atributo utilizado para ramificar el árbol, J es el número de clases, M_i es el número de valores diferentes del atributo A_i, $p(A_{ij})$ es la probabilidad de que A_i tome su *j*-ésimo valor, $p(C_k|A_{ij})$ es la probabilidad de que un ejemplo pertenezca a la clase C_k cuando su atributo A_i toma su *j*-ésimo valor y $p(\neg C_k|A_{ij})$ es $1 - p(C_k|A_{ij})$.

Como se puede apreciar, la expresión es muy parecida a la que teníamos al calcular la entropía de clasificación: simplemente se ha sustituido el logaritmo de $-\log_2 p(C_k|A_{ij})$ por el factor $p(\neg C_k|A_{ij})$.

El índice de Gini es una medida de la diversidad de clases en un nodo del árbol que se utiliza. Al igual que las dos medidas heurísticas anteriores (ganancia de información y criterio de proporción de ganancia), el índice de Gini es una medida de impureza muy utilizada en distintos algoritmos de construcción de árboles de decisión. En concreto, es la medida que se utiliza en CART [23].

2.1.1.4. Otros criterios

López de Mantaras [105] propuso una alternativa a la normalización del criterio de proporción de ganancia utilizando una métrica de distancia que también evita la fragmentación del conjunto de entrenamiento característica de la regla de división de ID3. La métrica de distancia propuesta por López de Mantaras se define de la siguiente forma:

$$LM(A_i) = \frac{H(C) - \sum_{k=1}^{M_i} p(A_{ij}) H(C|A_{ij})}{-\sum_{j=1}^{M_i}\sum_{k=1}^{J} \frac{n(C_k|A_{ij})}{N} \log_2 \frac{n(C_k|A_{ij})}{N}}$$

En otro trabajo independiente, Taylor y Silverman [150] presentaron el criterio MPI *[mean posterior improvement]* como alternativa al índice de Gini. Si bien este criterio se definió para árboles de decisión binarios (como los de CART), se puede generalizar de forma que sea aplicable a árboles de decisión n-arios:

$$MPI(A_i) = \prod_{j=1}^{M_i} p(A_{ij}) * \left(1 - \sum_{k=1}^{J} \frac{\prod_{j=1}^{M_i} p(C_k|A_{ij})}{P(C_k)} \right)$$

Durante los últimos años se han propuesto otras reglas de división en la literatura especializada. La mayor parte de ellas son medidas de impureza como las ya vistas en esta sección y pretenden resolver situaciones particulares en las cuales los criterios de división existentes no se comportan adecuadamente.

Si bien las reglas de división propuestas tienden a ser cada vez de formulación más compleja desde el punto de vista matemático, en [18] se proponen dos criterios de división que obtienen resultados interesantes y mantienen al mínimo su complejidad matemática: el criterio MAXDIF y el Índice Generalizado de Gini.

Ambas reglas de división se idearon con el objetivo de facilitar la comprensión del proceso de construcción de árboles de decisión al usuario final

de los modelos de clasificación construidos, que bien puede ser un ejecutivo o analista no familiarizado con las ideas en que se basan las reglas de división expuestas en párrafos anteriores. De esta forma, al entender mejor cómo se obtuvo el árbol de decisión, al usuario le resulta más fácil depositar su confianza en el modelo de clasificación obtenido.

Matemáticamente, el criterio MAXDIF y el Índice Generalizado de Gini se definen de la siguiente manera de acuerdo con la notación empleada para definir las demás reglas de división:

- MAXDIF

$$D(A_i) = \sum_{j=1}^{M_i} p(A_{ij}) D(C|A_{ij})$$

$$D(C|A_{ij}) = \max_k \{p(C_k|A_{ij}) - p(\neg C_k|A_{ij})\}$$

- Índice Generalizado de Gini

$$GG(A_i) = \sum_{j=1}^{M_i} p(A_{ij}) GG(C|A_{ij})$$

$$GG(C|A_{ij}) = 1 - \max_k p(C_k|A_{ij})$$

Como algunas de las reglas de división anteriores, tanto MAXDIF como el Índice Generalizado de Gini realizan una suma ponderada de las medidas de impureza de cada uno de los subárboles resultantes de ramificar el nodo actual del árbol. No obstante, a diferencia de propuestas anteriores, ambos criterios dependen únicamente de la estimación de la probabilidad de la clase más común en cada subárbol. De hecho, dicha estimación es la única evidencia tangible de la que disponemos al comparar ramificaciones alternativas para construir árboles de decisión.

Pese a su mayor sencillez, las dos reglas de división propuestas en [18] obtienen resultados satisfactorios en la práctica: la precisión del modelo obtenido no se ve afectada por utilizar reglas de división de formulación más sencilla.

Además, el uso exclusivo de la probabilidad de la clase más común en cada subárbol facilita la interpretación de las dos reglas de división propuestas. De este modo, el usuario final no necesita conocimientos previos específicos

para comprender el proceso completo de construcción del árbol. No es necesario que recuerde las propiedades del índice de diversidad de Gini ni tiene por qué entender los conceptos de Teoría de la Información en que se basan el criterio de ganancia de información y todos sus derivados.

En el mismo trabajo en el que se definen MAXDIF y el Índice Generalizado de Gini, se propone además el uso de un umbral de soporte mínimo con el objetivo de mejorar el comportamiento de los algoritmos TDIDT clásicos en presencia de claves y de ruido en el conjunto de entrenamiento, de forma análoga a como se emplea en la extracción de reglas de asociación (sección 2.3). Este umbral de soporte sirve para no tener en cuenta ramas del árbol muy poco pobladas y eliminar así uno de los sesgos más comunes de las reglas de división, el de tender a ramificar el árbol de decisión utilizando los atributos que tengan más valores diferentes.

A pesar de que todos los criterios expuestos en esta memoria hasta el momento hacen énfasis en la pureza de los nodos resultantes de ramificar el árbol de decisión, existen criterios que se pueden adscribir a otras categorías [108]:

- Algunos, definidos usualmente para árboles binarios, miden la diferencia entre los distintos subconjuntos generados utilizando distancias o ángulos. De esta forma acentúan la disparidad de tales subconjuntos.

- Otros son medidas estadísticas de independencia (como un test χ^2, por ejemplo) entre las proporciones de las clases y los subconjuntos de entrenamiento, de manera que se intenta subrayar la fiabilidad de las predicciones.

En [108] se puede encontrar un estudio exhaustivo sobre distintas reglas de división, la correlación entre ellas y resultados experimentales con árboles de decisión binarios. Los criterios de división existentes para árboles de decisión binarios también se analizan en [143].

Hay que destacar, no obstante, que la mayor parte de las reglas de división propuestas mejoran marginalmente la precisión de los árboles de decisión construidos y lo hacen únicamente en situaciones concretas.

2.1.2. Reglas de parada

Cuando se detiene la construcción del árbol de decisión, se construye una hoja a la que se le puede asignar una distribución de probabilidad (según los casos que recoja) o simplemente la clase más común de las existentes en los casos recogidos. Empíricamente se ha demostrado que esta última técnica es mejor a la hora de minimizar el error de clasificación.

Las reglas de parada tratan de predecir si merece la pena seguir construyendo el árbol por la rama actual o no. Estas reglas también se denominan reglas de pre-poda porque reducen la complejidad del árbol durante su construcción, en contraste con las reglas usuales de post-poda que simplifican el árbol de decisión una vez éste ha sido construido por completo (sección 2.1.3).

Lo usual es detener la construcción del árbol de decisión cuando se ha llegado a un nodo puro, entendiendo por nodo puro aquél que contiene ejemplos de una única clase. No obstante, se pueden utilizar otros criterios de parada además del anterior. A continuación se describen tres posibles reglas de prepoda:

- *Pureza del nodo*

  Cuando un nodo solamente contiene ejemplos de una clase, obviamente, el proceso de construcción del árbol de decisión ha finalizado. Sin embargo, también puede utilizarse un umbral de pureza para detener la construcción del árbol de decisión cuando la ramificación del árbol no suponga una disminución significativa de la impureza del mismo (según alguna medida estadística de impureza). En la práctica, esto no suele resultar totalmente satisfactorio y se suele optar por construir el árbol de decisión completo para después realizar una poda a posteriori.

- *Cota de profundidad*

  Independientemente de lo anterior, se puede establecer de antemano una cota de profundidad para no construir árboles excesivamente complejos. Cuando un nodo se halle a más de cierta profundidad, se detiene el proceso de generación del árbol de clasificación.

- *Umbral de soporte*

  Por otro lado, cuando nos encontramos un nodo con menos de X ejemplos también podemos detener el proceso de construcción del árbol de decisión, ya que no consideramos fiable una clasificación avalada por menos de X casos de entrenamiento. En otras palabras, menos de X ejemplos se consideran insuficientes para estimar probabilidades adecuadamente.

2.1.3. Reglas de poda

Los algoritmos TDIDT usualmente presuponen que no existe ruido en los datos de entrada e intentan obtener una descripción perfecta de tales datos. A veces, esto resulta contraproducente en problemas reales, donde se requiere un tratamiento adecuado de la incertidumbre y de la información con ruido presentes en los datos de entrenamiento. Por desgracia, el método recursivo de construcción de árboles de decisión continúa dividiendo el conjunto de casos de entrenamiento hasta que encuentra un nodo puro o no puede encontrar ninguna forma de seguir ramificando el árbol actual.

La presencia de ruido suele ocasionar la generación de árboles de decisión muy complejos que sobreajustan los datos del conjunto de entrenamiento. Este sobreaprendizaje limita considerablemente la aplicabilidad del modelo de clasificación aprendido. Para evitarlo, las técnicas de poda (como las utilizadas en ASSISTANT y C4.5) han demostrado ser bastante útiles. Aquéllas ramas del árbol con menor capacidad de predicción se suelen podar una vez que el árbol de decisión ha sido construido por completo.

> El siguiente ejemplo ilustra la necesidad de la poda [131]:
>
> Supongamos que queremos construir un clasificador con datos aleatorios para las clases X (con probabilidad p) e Y (probabilidad $1 - p$), siendo $p \geq 0{,}5$. Si el clasificador siempre dice que los casos son de la clase X el error será, obviamente, $1 - p$.
>
> Si el clasificador asigna un caso a la clase X con probabilidad p y a la clase Y con probabilidad $1 - p$, el error estimado sería la

suma de la probabilidad de que un caso de X se asigne a la clase Y, $p(1-p)$, y de la probabilidad de que un caso de Y se asigne a la clase X, $(1-p)p$. Es decir, el error estimado será igual a $2p(1-p)$, error mayor que $1-p$ si $p \geq 0{,}5$.

En esta situación, el clasificador más sencillo posible, aquél que por defecto asigna siempre la clase más común a cualquier ejemplo, es mejor que cualquier otro clasificador cuando la clase y los atributos de los casos son estadísticamente independientes.

En problemas reales, lo anterior sucede cuando los atributos no recogen toda la información necesaria para realizar la clasificación o cuando se ha dividido el conjunto de entrenamiento en conjuntos tan pequeños que la elección de un test u otro no supone ninguna mejora notable.

Tras haber construido un árbol de decisión, por tanto, es necesario podarlo para mejorar su precisión y evitar situaciones como la descrita.

Un árbol de decisión se puede simplificar eliminando un subárbol completo en favor de una única hoja. También se puede sustituir un subárbol por una de sus ramas (vg: la rama del subárbol más usada).

La poda se suele aplicar después de construir el árbol completo (*post-poda*), ya que la correcta estimación a priori del beneficio obtenido al simplificar un árbol durante su construcción (*pre-poda*) resulta difícil y sólo se ha empleado en algoritmos recientes como PUBLIC [134]

A continuación se comentan algunos de los métodos de poda de árboles de decisión más comunes: la poda por coste-complejidad y la poda pesimista.

2.1.3.1. Poda por coste-complejidad

Esta técnica de poda, utilizada en CART, intenta llegar a un compromiso entre la precisión y el tamaño del árbol.

La complejidad del árbol viene dada por el número de nodos terminales (hojas) que posee. Si T es el árbol de decisión usado para clasificar N casos de entrenamiento y se clasifican mal M ejemplos, la medida de coste-complejidad

de T para un parámetro de complejidad α es

$$R_\alpha(T) = R(T) + \alpha l(T)$$

donde $l(T)$ es el número de hojas del árbol T y $R(T) = M/N$ es un estimador del error de T. Es decir, $R_\alpha(T)$ es una combinación lineal del coste del árbol y de su complejidad.

El árbol podado será el subárbol que haga mínima la medida de coste-complejidad $R_\alpha(T)$. Hay que resaltar que conforme el parámetro de complejidad α crece, el tamaño del árbol que minimiza $R_\alpha(T)$ decrece.

La medida de coste-complejidad involucra un parámetro α, cuyo valor adecuado no se conoce a priori. A la hora de implementar esta técnica de poda, se genera una secuencia finita y única de árboles podados incrementando gradualmente el valor de α, comenzando con $\alpha = 0$. Partiendo de un árbol T_1 con $\alpha_1 = 0$, se encuentra las ramas más débiles de T_i y se podan para crear T_{i+1} cuando α alcanza α_{i+1}. Conforme el parámetro α aumenta, se tienden a podar menos nodos, hasta llegar a un árbol final con un único nodo. De todos los árboles de la secuencia, se escoge aquél que tenga asociado el menor error, utilizando para estimar este error un conjunto de prueba independiente del conjunto de entrenamiento o validación cruzada, tal como se describe en [23].

2.1.3.2. Poda pesimista

La poda pesimista de C4.5 [131] utiliza sólo el conjunto de casos de entrenamiento con los que se construye el árbol, con lo que nos ahorramos tener que reservar casos para realizar la simplificación del árbol.

Cuando una hoja del árbol cubre N casos de entrenamiento, de los cuales E casos son clasificados incorrectamente, su error de resustitución es E/N. El estimador del error de resustitución asociado a un subárbol será, como es lógico, la suma de los errores estimados para cada una de sus ramas.

La probabilidad real del error cometido en un nodo del árbol no se puede determinar con exactitud, y menos aún a partir del conjunto de entrenamiento que se emplea para construir el árbol de decisión. Sin embargo, se puede

considerar que los E errores de un nodo corresponden a E "éxitos" en N experimentos aleatorios, por lo que, de forma heurística, se le asocia al nodo del árbol una distribución de probabilidad binomial. Dado un grado de confianza CF, se puede establecer un intervalo de confianza para el valor de una distribución binomial y se puede considerar el límite superior de este intervalo $U_{CF}(E, N)$ como una estimación del error en el nodo. Si bien esta estimación carece de una base sólida, se emplea para predecir el número de errores de un nodo del árbol: $N \times U_{CF}(E, N)$.

Al utilizar poda pesismista, se poda un subárbol si el intervalo de confianza del error de resustitución (generalmente de amplitud dos veces el error estándar) incluye el error de resustitución del nodo si se trata como hoja. De esta forma se eliminan los subárboles que no mejoran significativamente la precisión del clasificador. El método es tan cuestionable como cualquier otra heurística sin base teórica, pero suele producir resultados aceptables.

2.1.4. Algoritmos TDIDT

Una vez que se han analizado los distintos componentes necesarios para la construcción de un árbol de decisión, en esta sección se describen brevemente algunos de los algoritmos de construcción de árboles de decisión más representativos.

La familia de algoritmos TDIDT incluye clásicos como CLS , ID3 [129], C4.5 [131] o CART [23], además de otros algoritmos más recientes tales como SLIQ [111], SPRINT [141], QUEST [104], PUBLIC [134], RainForest [69] o BOAT [67]. En las tablas 2.1 y 2.2 que aparecen al final de este apartado se enumeran algunos de los más conocidos. Las propuestas más recientes suelen ser simples adaptaciones de los algoritmos clásicos que permiten trabajar con los grandes conjuntos de datos utilizados en problemas de *Data Mining*.

- CLS *[Concept Learning System]* es el origen de familia TDIDT de algoritmos para la construcción de árboles de decisión. CLS construye árboles binarios utilizando una técnica similar al minimax: se explora el espacio de posibles árboles de decisión (estableciendo una cota de profundidad) y se elige la acción que minimice la función de costo en el

conjunto de árboles construidos.

- ID3 *[Iterative Dichotomizer 3]* es un algoritmo greedy de Quinlan que prefiere árboles sencillos frente a árboles más complejos ya que, en principio, aquéllos que tienen sus caminos más cortos hasta las hojas son más útiles a la hora de clasificar. En cada momento se ramifica por el atributo de menor entropía y el proceso se repite recursivamente sobre los subconjuntos de casos de entrenamiento correspondientes a cada valor del atributo por el que se ha ramificado.

- ASSISTANT es otro derivado de ID3 que construye árboles de decisión binarios y permite manejar atributos con valores continuos (reales), si bien no es muy eficiente porque para cada atributo han de comprobarse $2^{v-1} - 1$ posibles tests, siendo v el número de valores diferentes del atributo.

- CART, acrónimo de *Classification and Regression Trees*, también construye árboles binarios, usa el índice de diversidad de Gini y permite realizar una poda por coste-complejidad con validación cruzada.

- C4 es otro descendiente más de ID3 que permite atributos continuos, sobre los que se aplican tests de la forma $atributo \leq valor$.

- Algunos algoritmos de construcción de árboles de decisión están destinados al aprendizaje incremental, como ID4 e ID5, que son descendientes de ID3, o propuestas más recientes como ITI o DMTI.

- C4.5 es un híbrido entre CART y C4 que permite usar como regla de división la ganancia de información, el índice de diversidad de Gini o el criterio de proporción de ganancia. Además, incluye la posibilidad de realizar una post-poda pesimista del árbol.

- QUEST trata de solucionar algunos problemas tradicionalmente asociados con CART mediante la utilización de métodos estadísticos que eviten el sesgo de CART a la hora de seleccionar atributos para ramificar el árbol de decisión.

En los párrafos anteriores se han comentado brevemente algunos algoritmos tradicionales de construcción de árboles de decisión. Las propuestas más recientes, no obstante, suelen estar orientadas hacia la resolución de problemas de *Data Mining* y hacen énfasis en la escalabilidad de los algoritmos. Cuando el conjunto de entrenamiento no cabe en memoria, se ha de recurrir a técnicas que permitan construir un árbol de decisión a partir del conjunto de entrenamiento completo accediendo lo menos posible a disco:

- SLIQ *[Supervised Learning in Quest]*, por ejemplo, construye el árbol de decisión en anchura, a diferencia de los demás algoritmos citados (que lo hacen en profundidad) y emplea un criterio de poda basado en el principio MDL *[Minimum Description Length]* de la Teoría de la Información.

- SPRINT *[Scalable PaRallelizable INduction of decision Trees]* mejora las propiedades de escalabilidad de SLIQ utilizando estructuras de datos diferentes.

- PUBLIC, en vez de construir el árbol de decisión completo y después podarlo, detiene la construcción del árbol estimando una cota inferior del coste MDL de cada subárbol, ahorrándose de esta forma gran parte del esfuerzo computacional que supone construir un árbol hasta el final.

- RAINFOREST calcula unos estadísticos suficientes en cada nodo del árbol de forma que éstos permiten seleccionar cómo ramificar el árbol de decisión en cada instante. Estos estadísticos, denominados conjuntos AVC *[attribute-value, class label]*, no almacenan más que la frecuencia de aparición de cada uno de los valores de los atributos del conjunto de entrenamiento para cada una de las clases del problema.

- BOAT, a diferencia de los anteriores, construye un árbol de decisión aproximado a partir de una muestra del conjunto de entrenamiento y, posteriormente, ajusta ese árbol aproximado utilizando el conjunto de entrenamiento completo.

Algoritmo	*Referencia*
CLS	Hunt, Martin & Stone (eds.): *Experiments in Induction*, Academic Press 1966
THAID	Morgan & Messenger: *THAID: A Sequential Analysis Program for the Analysis of Nominal Scale Dependent Variables*, TR Univ. Michigan, 1973
CART	Breiman, Friedman, Olshen & Stone: *Classification and Regression Trees*, Wadsworth 1984
ID3	Quinlan: *Induction on Decision Trees*, Machine Learning 1986
ID4	Schlimmer & Fisher: *A case study of incremental concept induction*, NCAI 1986
ASSISTANT	Cestnik, Kononenko & Bratko: *Assistant86: A knowledge-elicitation tool for sophisticated users*, EWSL-87, 1987
FACT	Loh & Vanichsetakul: *Tree-structured classification via generalized discriminant analysis (with discussion)*, Journal of the American Statistical Association, 1988
ID5R	Utgoff: *Incremental Induction of Decision Trees*, Machine Learning 1989
IDL	Van de Velde: *Incremental Induction of Topologically Minimal Trees*, ICML 1990
IC	Agrawal, Ghosh, Imielinski, Iyer & Swami: *An Interval Classifier for Database Mining Applications*, VLDB'92
CDP	Agrawal, Imielinski & Swami: *Database Mining: A Performance Perspective*, TKDE'93
C4.5	Quinlan: *C4.5: Programs for Machine Learning*, Morgan Kaufmann 1993
CHAID	Magidson: *The CHAID approach to segmentation modeling*, Handbook of Marketing Research, 1993

Tabla 2.1: Algoritmos de construcción de árboles de decisión

Algoritmo	*Referencia*
SLIQ	Mehta, Agrawal & Rissanen: *SLIQ: A fast scalable classifier for data mining*, EBDT'1995
SPRINT	Shafer, Agrawal & Manish: *SPRINT: A scalable parallel classifier for data mining*, VLDB'1996
SONAR	Fukuda, Morimoto, Morishita & Tokuyama: *Constructing Efficient Decision Trees by Using Optimized Numeric Association Rules*, VLDB'1996
MSC	Lovell & Bradley: *The multiscale classifier*, IEEE TPAML, 1996
GUIDE	Loh: *Unbiased regression trees*, University of Wisconsin, 1997
QUEST	Loh & Shih: *Split Selection Methods for Classification Trees*, Statistica Sinica, 1997
ITI	Utgoff, Berkman & Clouse: *Decision Tree Induction Based on Efficient Tree Restructuring*, Machine Learning, 1997
DMTI	Utgoff, Berkman & Clouse: *Decision Tree Induction Based on Efficient Tree Restructuring*, Machine Learning, 1997
PUBLIC	Rastogi & Sim: *PUBLIC: A Decision Tree Classifier that integrates building and pruning*, DMKD, 2000
RAINFOREST	Gehrke, Ramakrishnan & Ganti: *Rainforest - a framework for fast decision tree construction of large datasets*, DMKD, 2000
BOAT	Gehrke, Ganti, Ramakrishnan & Loh: *BOAT - Optimistic Decision Tree Construction*, SIGMOD'1999

Tabla 2.2: Algoritmos de construcción de árboles de decisión (continuación)

2.1.5. Paso de árboles a reglas

Una característica interesante de los árboles de decisión es la facilidad con la que se puede derivar, a partir de un árbol de decisión, un conjunto de reglas de producción (del tipo IF-THEN) completamente equivalente al árbol original. Este hecho es fundamental para facilitar la comprensión del modelo de clasificación construido cuando el árbol de decisión es complejo, ya que conforme el tamaño los árboles de decisión aumenta, su inteligibilidad disminuye. Cuando el problema de clasificación es complejo, el árbol de decisión generado es tan grande que ni siquiera tras su poda un experto es capaz de comprender el modelo de clasificación completo.

El algoritmo que nos permite realizar este cambio de modelo de representación es casi trivial: de cada camino desde la raíz del árbol hasta un nodo hoja se deriva una regla cuyo antecedente es una conjunción de literales relativos a los valores de los atributos situados en los nodos internos del árbol y cuyo consecuente es la decisión a la que hace referencia la hoja del árbol (la clasificación realizada).

Al convertir el árbol de decisión en una colección de reglas se obtiene una regla por cada hoja del árbol. Dichas reglas serán, por tanto, mutuamente excluyentes (salvo, claro está, que se utilicen conjuntos difusos en los tests de los nodos internos del árbol [157]).

Algunas de las reglas generadas, no obstante, pueden contener condiciones irrelevantes en su antecedente. Tales reglas pueden generalizarse eliminando dichas condiciones superfluas sin que disminuya la precisión del clasificador asociado, si bien entonces las reglas pueden dejar de ser mutuamente excluyentes [131].

Obviamente, también existe la posibilidad de conseguir directamente un conjunto de reglas a partir del conjunto de datos de entrenamiento, sin tener que construir un árbol de decisión. El conjunto de reglas así generado constituirá nuestro modelo de clasificación.

Tales conjuntos de reglas se pueden obtener de muy diversas maneras y suelen conducir a modelos de clasificación parcial que han de ser completados de algún modo más o menos artificial para conseguir un modelo de clasificación completo que nos permita clasificar datos.

Este enfoque es el utilizado por el algoritmo CN2 [34] [35] y, en general, por todos los algoritmos desarrollados a partir de la metodología STAR de Michalski, tales como INDUCE o AQ. Este tipo de algoritmos será objeto de estudio en la sección siguiente de esta memoria.

2.2. Inducción de reglas y listas de decisión

Una regla, en su sentido más amplio, particiona el dominio del problema en aquellas instancias del mismo que satisfacen la regla y aquéllas que no. Por tanto, una regla puede utilizarse como bloque de construcción de clasificadores.

Si bien las reglas IF-THEN se pueden extraer fácilmente a partir de un árbol de decisión, existen numerosos algoritmos que inducen conjuntos de reglas directamente a partir del conjunto de entrenamiento.

Algunos de estos algoritmos realizan una búsqueda dirigida en el espacio de posibles hipótesis. La búsqueda dirigida es una generalización de la búsqueda 'primero el mejor' que se caracteriza por mantener una lista limitada de soluciones parciales durante el proceso de exploración del espacio de búsqueda. La búsqueda dirigida no realiza una exploración exhaustiva del espacio de búsqueda y, por tanto, es un método incompleto, si bien es cierto que no resulta factible una búsqueda exhaustiva en el espacio de todas las reglas posibles. De hecho, este espacio incluye como caso particular el espacio definido por los árboles de decisión y enumerar todos los árboles de decisión posibles es un problema NP, como se mencionó en la sección anterior.

En los siguientes apartados se describen tres tipos de algoritmos que utilizan variantes de la búsqueda dirigida para guiar el proceso de construcción de modelos de clasificación basados en reglas: la metodología STAR, las listas de decisión y los algoritmos genéticos.

2.2.1. Metodología STAR

Michalski y sus colaboradores desarrollaron, bajo el nombre de metodología STAR [140], un conjunto de técnicas de aprendizaje inductivo incremental basadas en la utilización de expresiones lógicas en forma normal disyuntiva (modelo de representación más expresivo que el empleado por los algoritmos de construcción de árboles de decisión). Estas expresiones lógicas describen conceptos y se pueden utilizar directamente como reglas de clasificación. Entre los algoritmos desarrollados en el entorno de Michalski destacan INDUCE y AQ, del que existen múltiples variantes.

El algoritmo INDUCE se diseñó para su aplicación en situaciones estructuradas, aquéllas en las cuales los ejemplos de entrenamiento poseen estructura interna. INDUCE utiliza siempre todos los ejemplos del conjunto de entrenamiento para intentar conseguir reglas consistentes y completas. Se entiende por consistente aquella regla que no cubre ejemplos negativos de la clase que se desea modelar, mientras que se considera completa aquella regla que engloba a todos los ejemplos positivos. Durante el proceso de búsqueda, las reglas inconsistentes se especializan para eliminar falsos positivos y las reglas consistentes se generalizan para intentar conseguir reglas más completas.

Por su parte, el algoritmo AQ utiliza un ejemplo positivo como semilla e intenta encontrar la expresión más general que indique una condición necesaria para pertenecer a la clase del ejemplo seleccionado como semilla. Es decir, a partir de un ejemplo concreto, se busca una regla consistente con el conjunto de entrenamiento. A diferencia de INDUCE, AQ algoritmo funciona de forma incremental: mientras queden ejemplos positivos por clasificar, se repite el proceso de búsqueda de soluciones consistentes con los ejemplos restantes.

Tanto INDUCE como AQ utilizan una función de evaluación lexicográfica para evaluar las posibles reglas. Esta función consiste en una secuencia ordenada de criterios acompañados por márgenes de tolerancia (entre 0 % y 100 %) que controlan hasta qué punto dos soluciones diferentes se consideran equivalentes. Los elementos de la función de evaluación pueden ser medidas simples como la complejidad de la regla o el número de ejemplos positivos que cubre. La función de evaluación se utiliza de la siguiente manera:

1. Inicialmente, comenzamos empleando el criterio más relevante para el dominio del problema que deseamos resolver: C_1 ($i := 1$).

2. Se evalúan todas las alternativas de acuerdo con el criterio C_i. Nos quedamos con la mejor opción y con todas aquellas cuya evaluación de acuerdo a C_i quede dentro del rango delimitado por la tolerancia T_i asociada al criterio i.

3. Cuando sólo queda una solución, la devolvemos como mejor opción posible.

4. Si queda más de una opción y aún disponemos de otros criterios con los cuales evaluarlas, hacemos $i := i + 1$ y volvemos al punto 2.

5. Si se nos acaba la secuencia de pares criterio-tolerancia y aún nos quedan varias opciones, todas ellas se consideran equivalentes (igual de adecuadas para resolver el problema que nos ocupa).

Con posterioridad a los trabajos originales de Michalski, Peter Clark y su equipo propusieron el algoritmo CN2 en 1989 [34] [35]. Este algoritmo intenta combinar adecuadamente las mejores cualidades de los árboles de decisión y de los algoritmos AQ: CN2 trata de combinar la eficiencia de la familia de algoritmos de construcción de árboles de decisión con la flexibilidad de la familia AQ en su estrategia de búsqueda de reglas.

El algoritmo CN2 utiliza la búsqueda dirigida de los algoritmos de la familia STAR, pero elimina la dependencia de AQ respecto al orden de presentación de los ejemplos del conjunto de entrenamiento, ya que no toma ningún ejemplo concreto como semilla de la solución final. Además, CN2 permite utilizar reglas que no sean del todo consistentes (esto es, reglas que cubran algún ejemplo negativo del conjunto de entrenamiento).

Durante el proceso de especialización de reglas, en vez de emplear una función de evaluación lexicográfica, el algoritmo CN2 emplea dos heurísticas: una medida de entropía para evaluar la calidad de una regla (de forma análoga a la forma en que algoritmos como ID3 consideran qué atributo utilizar para ramificar un árbol de decisión) y un test estadístico para comprobar si la regla

obtenida es significativa. La entropía se emplea para determinar si una regla tiene una buena capacidad predictiva mientras que el test estadístico se usa para asegurarnos de que la regla sea fiable.

Curiosamente, la versión original de CN2 [34] obtiene una lista ordenada de reglas del estilo de las generadas por los algoritmos de inducción de listas de decisión que se analizarán en la sección siguiente, mientras que su versión revisada [35] mantiene la estrategia de búsqueda tradicional de los algoritmos AQ y sustituye la medida de entropía por una estimación laplaciana del error.

Los algoritmos de la familia STAR (como INDUCE, AQ o CN2) obtienen reglas para discriminar elementos de una clase de los que no pertenecen a dicha clase. Dichos métodos, especialmente INDUCE y AQ, carecen de un modelo que guíe la generación de reglas y determine la finalización de la búsqueda, pues realizan una enumeración exhaustiva de todas las modificaciones simples que se pueden aplicar al conjunto de reglas candidatas hasta que se consigue un modelo consistente y completo.

Algunos de los miembros más importantes de la familia de algoritmos STAR se recogen en la tabla 2.3. Aparte de los ya comentados, destacan:

- FOIL [115], que extiende el ámbito de aplicación de algoritmos como CN2 para obtener reglas de primer orden utilizando un algoritmo greedy de búsqueda, una función de evaluación similar a la de ID3 y un criterio de parada basado en el principio MDL de Rissanen; y

- CWS [46], acrónimo de *Conquering without Separating*, que cambia el orden usual en el que se realizan las operaciones para mejorar el rendimiento de los algoritmos de inducción de reglas cuando el conjunto de entrenamiento es muy grande: en vez de evaluar cada regla por separado, se entrelaza la construcción del conjunto completo de reglas para evaluar cada regla en el contexto del conjunto de reglas actual. En cada iteración, se añade una regla de antecedente vacío y se intentan especializar las reglas existentes en el conjunto de reglas actual. Dada una regla candidata R', especialización de una regla R del conjunto de reglas actual RS, se sustituye la regla original R con la regla R' si la precisión de $(RS - \{R\}) \cup \{R'\}$ es mayor que la de RS.

Algoritmo	*Referencia*
INDUCE	Michalski: *Theory and methodology of inductive learning*, Artificial Intelligence, vol. 20, pp. 111-161, 1983
AQ	Michalski: *On the quasi-minimal solution of the general covering problem*, Proceedings of the 5th International Symposium on Information Processing (FCIP 69), Vol. A3, pp. 125-128, 1969
AQ11	Michalski & Larson: *Incremental generation of vl¡sub¿1¡/sub¿hypotheses: the underlying methodology and the description of program AQ11*, ISG 83-5, UIUC, 1983
AQ15	Michalski, Mozetic, Hong & Lavrac: *The AQ15 inductive learning system: An overview and experiments*, UIUC, 1986. Michalski, Mozetic, Hong & Lavrac: *The multipurpose incremental learning system AQ15 and its testing application to three medical domains*, AAAI-86, pp. 1041-1045, 1986
AQ15-GA	Thrun et al.: *The MONK's Problems: A performance comparison of different learning algorithms*, CMU-CS-91-197, December 1991
AQ17	Thrun et al.: *The MONK's Problems: A performance comparison of different learning algorithms*, CMU-CS-91-197, December 1991
AQ18	Michalski & Kaufman: *Learning Patterns in Noisy Data: The AQ Approach*, LNAI 2049, p. 22 ff., 2001
AQR	Clark & Nibblet: *The CN2 Induction Algorithm*, Machine Learning, 3:4, 261-283, Kluwer, 1989.
FOIL	Quinlan: *Learning logical definitions from relations*, Machine Learning, 5, pp. 239-266, 1990.
CN2	Clark & Boswell: *Rule Induction with CN2: some recent improvements*, EWSL'91, pp. 151-163, Springer Verlag, 1991
CWS	Domingos: *Linear-time rule induction*, 2nd International Conference on Knowledge Discovery and Data Mining, pp. 96-101, AAAI, 1996
RISE	Domingos: *Unifying instance-based and rule-based induction*, Machine Learning, 24, pp. 141-168, 1996

Tabla 2.3: Algunos algoritmos de inducción de reglas

2.2.2. Listas de decisión

Otra familia de algoritmos de aprendizaje inductivos es la constituida por las listas de decisión. Las listas de decisión pueden considerarse reglas IF-THEN extendidas y tienen la forma *if - then - else if - ... - else -*. Como se verá en el capítulo siguiente, nuestro modelo ART también puede considerarse un algoritmo de inducción de listas de decisión.

Si bien los métodos generales de inducción de reglas suelen ser computacionalmente muy costosos, existen algoritmos más eficientes para la construcción de listas de decisión. De hecho, se ha demostrado teóricamente que las listas de decisión, cuando se limita su longitud a un número máximo de cláusulas conjuntivas, se pueden aprender en tiempo polinómico [135].

Los algoritmos de inducción de listas de decisión generan un conjunto ordenado de reglas. Al clasificar un ejemplo, se va emparejando dicho ejemplo con cada una de las reglas de la lista hasta que se verifica el antecedente de una de ellas. Entonces, se le asigna al ejemplo la clase que aparece en el consecuente de la regla activada. Por si se diese el caso de que no se verificase ninguna de las reglas de la lista de decisión, usualmente se añade al final de la lista una regla por defecto con antecedente vacío que corresponde a la clase más común de los ejemplos del conjunto de entrenamiento no cubiertos por las reglas seleccionadas (o, en su defecto, la clase más común en el conjunto de entrenamiento completo).

Las listas de decisión tienen la ventaja de que nunca habrá conflictos entre las reglas que constituyen el clasificador, pues sólo se puede disparar una de ellas siguiendo el proceso descrito en el párrafo anterior, por lo que este tipo de clasificadores no necesita disponer de ningún mecanismo adicional de resolución de conflictos. De hecho, su estructura ordenada elimina el solapamiento entre las reglas al que se le suelen achacar la ineficiencias de los algoritmos de inducción de reglas como AQ o CN2.

Por otro lado, las listas de decisión son más difíciles de comprender que los conjuntos de reglas obtenidos por algoritmos como AQ porque las reglas que las componen no son independientes entre sí. El significado de una regla depende de su posición en la lista de decisión; más en concreto, de todas las

reglas que la preceden en la lista ordenada de reglas. Esta característica, que facilita su aprendizaje por medios automáticos, dificulta la comprensibilidad de los modelos obtenidos: cuando la lista de decisión incluye muchas reglas, ni siquiera un experto es capaz de discernir el verdadero significado de las últimas reglas de las listas. Puesto que los modelos de clasificación obtenidos con ayuda de técnicas de *Data Mining* han de ser validados por expertos en determinadas situaciones, en muchas ocasiones se emplean árboles de decisión en detrimento de las listas de decisión, que son potencialmente más compactas.

A diferencia de los algoritmos de construcción de árboles de decisión, que utilizan una estrategia "divide y vencerás" para construir un modelo de clasificación en forma de árbol, las técnicas de inducción de listas de decisión se caracterizan por emplear una estrategia "separa y vencerás". Dicha estrategia consiste en buscar una solución parcial al problema (una regla en nuestro caso) y una vez encontrada, reducir el problema eliminando todos los ejemplos cubiertos por la solución encontrada. Obviamente, esta estrategia da lugar a la secuencia *if - then - else if - ... - else* - característica de las listas de decisión.

Algunos algoritmos de inducción de listas de decisión están diseñados para eliminar el 'problema del solapamiento de las reglas' típico de cualquier algoritmo que obtiene reglas de forma independiente. Esta es la causa, por ejemplo, de que la precisión de una lista de decisión no sea necesariamente función directa de la precisión de las reglas que la componen, pues su precisión global depende además del solapamiento que exista entre las reglas individuales. Para evitar este supuesto problema, BruteDL [138] no construye la lista de decisión de forma incremental (como AQ, CN2, RIPPER, etc.) sino que, al igual que algoritmos como CWS [46], realiza la búsqueda completa sin podar el conjunto de entrenamiento. Al no ir depurando de forma incremental el modelo de clasificación obtenido, no obstante, se ve forzado a añadir una cota de profundidad a la búsqueda en profundidad que realiza, lo cual que limita artificialmente su estrategia de búsqueda de soluciones.

IREP [64], acrónimo de *Incremental Reduced Error Pruning*, construye una lista de decisión dividiendo el conjunto de entrenamiento en dos subconjuntos de crecimiento y poda, al primero de los cuales se asignan dos tercios de los ejemplos del conjunto de entrenamiento. IREP construye una lista de

reglas utilizando un algoritmo "separa y vencerás" de tipo greedy en el cual se generan reglas candidatas utilizando una versión proposicional del algoritmo FOIL sobre el subconjunto de crecimiento del conjunto de entrenamiento hasta que las reglas sean consistentes y se generalizan las reglas obtenidas eliminando la secuencia final de condiciones que maximice una función heurística de poda v. Estas reglas candidatas pasarán a formar parte de la lista de decisión siempre y cuando su porcentaje de error en el subconjunto de poda no exceda al correspondiente a la regla vacía (aquélla que asigna por defecto la clase más común en el conjunto de entrenamiento). Cohen [38] modificó esta heurística para admitir cualquier regla que no superase el 50 % de error en el subconjunto de poda.

Cohen también propuso en [38] el algoritmo IREP*, que contiene varias mejoras y modificaciones respecto al algoritmo IREP original. En IREP* se modifica la heurística utilizada en el proceso de poda de reglas individuales (v) y se vuelve a modificar el criterio de aceptación de reglas de IREP: en vez de fijar un porcentaje arbitrario (como el 50 %), se emplea una técnica basada en el principio MDL de Rissanen.

El algoritmo RIPPER [38], acrónimo de *Repeated Incremental Pruning to Produce Error Reduction*, no es más que la aplicación múltiple, durante k iteraciones, del algoritmo IREP* acompañado de un proceso de optimización de la lista de decisión que aparece descrito con detalle en [38]:

> Dada una lista de decisión formada por una secuencia ordenada de reglas R_1, R_2... R_k, se considera cada una de estas reglas en el orden en que se aprendieron (primero R_1). Para cada regla R_i se construyen dos reglas alternativas, el reemplazo R_i' (que se obtiene generando y podando una regla R_i' de forma que minimice el error del conjunto de reglas R_1, .., R_i', .. R_k) y la revisión de R_i (que se forma de forma análoga a partir de la regla R_i, añadiéndole condiciones a la regla original, tras los cual se generaliza de la misma forma que la regla R_i'). El principio MDL de Rissanen se vuelve a utilizar aquí para determinar cuál de las tres reglas resulta más adecuada para su inclusión en la lista de decisión (la original, el reemplazo o la revisión).

Algoritmo	*Referencia*
CN2	Clark & Nibblet: *The CN2 Induction Algorithm*, Machine Learning, 3:4, 261-283, Kluwer, 1989.
BruteDL	Segal & Etzioni: *Learning decision lists using homogeneous rules*. AAAI-94, 1994.
IREP	Fürnkranz & Widmer: *Incremental reduced error pruning*, In "Machine Learning: Proceedings of the 11th Annual Conference", New Brunswick, New Jersey, Morgan Kaufmann, 1994
IREP*	Cohen: *Fast effective rule induction*, In "Proceedings of the 12th International Conference on Machine Learning"(ML95), pp. 115-123, Morgan Kaufmann, 1995.
RIPPERk	Cohen: *Fast effective rule induction*, In "Proceedings of the Twelfth International Conference on Machine Learning"(ML95), pp. 115-123, Morgan Kaufmann, 1995
SLIPPER	Cohen & Singer: *A simple, fast, and efective rule learner*, AAAI-1999, pp. 335-342, 1999
PNrule	Joshi, Agarwal & Kumar: *Mining needles in a haystack: Classifying rare classes via two-phase rule induction*, ACM SIGMOD 2001

Tabla 2.4: Algunos algoritmos de inducción de listas de decisión

También existen versiones especializadas de algoritmos de inducción de listas de decisión para problemas de clasificación binarios (cuando sólo existen dos clases). Éste es el caso de PNrule [88]. En una primera fase, PNrule construye reglas P para predecir la presencia de la clase objetivo. En una segunda etapa, PNrule intenta mejorar la precisión del clasificador generando reglas N para que predicen la ausencia de la clase objetivo y sirven para eliminar falsos positivos obtenidos al aplicar el primer conjunto de reglas.

La tabla 2.4 recoge algunos ejemplos significativos de algoritmos de inducción de listas de decisión, desde la versión original de CN2 [35] hasta el algoritmo PNrule [88].

2.2.3. Algoritmos genéticos

Los algoritmos genéticos se pueden considerar como un caso particular de las estrategias de búsqueda dirigida y también se han utilizado para construir clasificadores basados en reglas [79]. A diferencia de las técnicas vistas en párrafos anteriores, que son determinísticas, los algoritmos genéticos son algoritmos probabilísticos.

Los algoritmos genéticos son métodos generales de optimización independientes del problema, lo cual los hace robustos, por ser útiles para cualquier problema, pero a la vez débiles, pues no están especializados en ninguno.

Igual que las redes neuronales y los clasificadores basados en medidas de similitud, los clasificadores construidos utilizando algoritmos genéticos suelen destacar porque su rendimiento no se ve excesivamente afectado por la aparición de ruido en el conjunto de entrenamiento (lo que sí puede ocurrir con otros modelos simbólicos).

Los algoritmos genéticos están inspirados en la Naturaleza, en la teoría de la evolución descrita por Charles Darwin en su libro “Sobre el Origen de las Especies por medio de la Selección Natural”, escrito 20 años después del viaje de su autor por las islas Galápagos en el Beagle. La hipótesis de Darwin (y de Wallace, que llegó a la misma conclusión de forma independiente) es que la selección natural y pequeños cambios heredables por los seres vivos son los dos mecanismos que provocan el cambio en la Naturaleza y la generación de nuevas especies. Fue Lendel quien descubrió que los caracteres se heredaban de forma discreta, y que se tomaban del padre o de la madre, dependiendo de su carácter dominante o recesivo. Estos caracteres, que pueden tomar diferentes valores, se denominan genes, mientras que los alelos son los distintos valores que pueden tomar los genes. En los seres vivos, los genes se encuentran en los cromosomas.

En la evolución natural, los mecanismos de cambio alteran la proporción de alelos de un tipo determinado en una población, y se dividen en dos tipos: los que disminuyen la variabilidad (la selección natural y la deriva genética), y los que la aumentan (la mutación, la poliploidía, la recombinación o cruce y el flujo genético).

A principios de los 60, en la Universidad de Michigan en Ann Arbor, las ideas de John Holland comenzaron a desarrollarse y a dar frutos. Leyendo un libro escrito por un biólogo evolucionista, R.A. Fisher, titulado “La teoría genética de la selección natural”, aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado. Los objetivos de su investigación fueron dos: imitar los procesos adaptativos de los sistemas naturales y diseñar sistemas artificiales (programas) que retengan los mecanismos de los sistemas naturales.

Los algoritmos evolutivos tratan de imitar los mecanismos de la evolución para resolver problemas. La aplicación de un algoritmo genético consiste en hallar de qué parámetros depende el problema, codificarlos en un cromosoma y aplicar los métodos de la evolución (selección y reproducción sexual con intercambio de información y alteraciones que generen diversidad). De hecho, la mayoría de las veces, una codificación correcta es la clave de una buena resolución del problema.

Los algoritmos genéticos se han utilizado para construir sistemas clasificadores como mecanismo de búsqueda en el espacio de posibles reglas, equiparando el proceso de aprendizaje a una actividad de búsqueda en un espacio complejo. Las distintas propuestas existentes se encuadran dentro de dos modelos principales:

- En los modelos de aprendizaje evolutivo tipo Pittsburgh, cada individuo de la población del algoritmo genético representa por sí mismo una solución completa; es decir, cada cromosoma codifica un conjunto de reglas. Este esquema tiene el inconveniente de que los cromosomas pueden resultar excesivamente largos, si bien es cierto que se puede utilizar un algoritmo genético clásico sin necesidad de establecer estrategias de colaboración entre los individuos de la población.

- En los modelos de aprendizaje evolutivo tipo Michigan, cada individuo de la población representa una parte de la solución (una regla). La solución global se obtiene promocionando la cooperación y competición dentro de la población. Esto permite tratar problemas más complejos que

con los modelos Pittsburgh, aunque también es cierto que se requieren estrategias más sofisticadas que permitan la cooperación entre las partes de la solución (esto es, un sistema de asignación de crédito).

Otros modelos más avanzados, como REGAL (*RElational Genetic Algorithm based Learner*) [71], intentan combinar las ventajas de ambos esquemas. El objetivo final de REGAL es la obtención de reglas completas, consistentes y simples utilizando para ello una estrategia de búsqueda efectiva en grandes espacios de hipótesis: el conjunto de todas las reglas válidas en Lógica de Primer Orden (al estilo de FOIL [115]). REGAL emplea un algoritmo genético distribuido en el cual se fomenta la formación de nichos para aprender conceptos disjuntos. Se puede considerar un modelo híbrido Michigan Pittsburgh en el cual la población es un conjunto redundante de individuos de longitud fija, cada uno de los cuales es una descripción parcial de un concepto que evoluciona separadamente. Como algoritmo genético, REGAL emplea la mutación clásica, cuatro operadores de cruce (uniforme y en dos puntos, además dos operadores de cruce específicos para la especialización y la generalización) y un operador especial, el cual, dado un ejemplo positivo, devuelve una fórmula que lo cubre.

Aunque todos los modelos reseñados son potencialmente útiles, su uso en la práctica está bastante limitado si no se dispone de hardware especializado con la capacidad de cómputo necesaria para proporcionar resultados en un tiempo razonable (como sistemas multiprocesadores y/o multicomputadores), especialmente si hay que tratar con grandes bases de datos, como sucede habitualmente en problemas de *Data Mining*. En estos casos resulta más adecuado emplear técnicas de extracción de reglas de asociación:

2.3. Reglas de asociación

Las reglas de asociación constituyen un mecanismo de representación del conocimiento simple y útil para caracterizar las regularidades que se pueden encontrar en grandes bases de datos.

La extracción de reglas de asociación se ha aplicado tradicionalmente a bases de datos transaccionales. En este tipo de bases de datos, una transacción *T* es un conjunto de artículos o items, junto a un identificador único y algún que otro dato adicional (fecha y cliente, por ejemplo). Una transacción contiene un conjunto de items *I* si *I* está incluido en *T*. Un conjunto de items se denomina *itemset*, en general, o itemset de grado k (*k-itemset*) cuando se especifica el número de items que incluye (k).

Una regla de asociación es una implicación $X \Rightarrow Y$ en la cual *X* e *Y* son itemsets de intersección vacía (esto es, sin items en común). El significado intuitivo de una regla de asociación $X \Rightarrow Y$ es que las transacciones (o tuplas) que contienen a *X* también tienden a contener a *Y*.

La confianza *[confidence]* de una regla de asociación $X \Rightarrow Y$ es la proporción de las transacciones que, conteniendo a *X*, también incluyen a *Y*. El soporte *[support]* de la regla es la fracción de transacciones en la base de datos que contienen tanto a *X* como a *Y*.

En la base de datos típica de un supermercado, por ejemplo, los items pueden ser cada uno de los productos o categorías de productos que el establecimiento en cuestión comercializa. Si analizamos la base de datos de transacciones del supermercado, podríamos encontrar que los jueves por la tarde se verifica la siguiente regla: $\{bebidas\} \Rightarrow \{pañales\}$ con confianza 66 % y soporte 2 %. Dicha regla nos indica que el 2 % de las cestas de la compra incluyen bebidas y pañales los jueves por la tarde. Además, dos de cada tres clientes que se llevan bebidas también compran pañales (quizá porque el fin de semana tienen que pasarlo en sus casas cuidando a sus bebés).

La extracción de reglas de asociación también puede realizarse en bases de datos relacionales. En ellas, un item es un par *(atributo,valor)*. Además, se puede añadir una restricción adicional como consecuencia directa de la Primera Forma Normal, la cual establece que todo atributo de una relación debe

contener únicamente valores atómicos. Por tanto, todos los items de un itemset deben corresponder a atributos diferentes.

Dada una base de datos concreta, el proceso habitualmente seguido consiste en extraer todas las reglas de asociación que tengan un soporte y una confianza por encima de unos umbrales establecidos por el usuario, *MinSupport* y *MinConfidence* respectivamente. Este problema de extracción de reglas de asociación se suele descomponer en dos subproblemas utilizando la estrategia "Divide y Vencerás":

1. Encontrar todos los itemsets frecuentes [denominados *frequent*, *covering* o *large itemsets* en la literatura], que son aquellos itemsets cuyo soporte es mayor que un umbral de soporte mínimo establecido por el usuario. Este problema puede resolverse construyendo un conjunto candidato de itemsets potencialmente frecuentes e identificando, en dicho conjunto de candidatos, aquellos itemsets que realmente lo son. El tamaño de los itemsets considerados se va incrementando progresivamente hasta que no quedan itemsets frecuentes por descubrir.

2. Generar las reglas de asociación que se derivan de los itemsets frecuentes. Si $X \cup Y$ y X son itemsets frecuentes, la regla $X \Rightarrow Y$ se verifica si el cociente entre el soporte de $X \cup Y$ y el soporte de X es, al menos, tan grande como el umbral de confianza mínima. La regla superará el umbral de soporte porque $X \cup Y$ es frecuente.

El descubrimiento de los itemsets frecuentes es la parte del proceso de extracción de reglas de asociación más costosa computacionalmente, mientras que la generación de las reglas de asociación a partir de los itemsets frecuentes es casi inmediata. Por este motivo la mayor parte de los algoritmos de extracción de reglas de asociación han centrado su diseño en la enumeración eficientemente de todos los itemsets frecuentes presentes en una base de datos, tal como se comenta en la siguiente sección.

2.3.1. Algoritmos de extracción de reglas de asociación

Desde las propuestas originales, tales como AIS [4], SETM [84] o, sobre todas ellas, Apriori [7], se han propuesto muchos algoritmos para resolver el problema de la extracción de reglas de asociación. Entre ellos se encuentran, por citar algunos ejemplos representativos, los algoritmos DHP [119], DIC [24], CARMA [80], FP-Growth [74] y TBAR [19].

Las tablas 2.5 a 2.7 recogen distintos algoritmos de extracción de reglas de asociación que se han propuesto en la literatura. En [81] se puede obtener una visión general de las soluciones propuestas, mientras que en [76] se puede encontrar una comparación detallada de algunos de los algoritmos mencionados.

AIS [4] fue el primer algoritmo que se desarrolló para obtener reglas de asociación, si bien sólo contemplaba reglas de asociación con un item en su consecuente.

SETM [84], acrónimo de *SET-oriented Mining of association rules*, se caracteriza porque fue diseñado para utilizar SQL en la generación de itemsets relevantes y, por lo demás, es completamente equivalente a AIS.

Agrawal y Skirant, trabajando para el proyecto Quest de IBM en el Almaden Research Center de San José en California, propusieron en 1994 dos algoritmos notablemente más eficientes que los algoritmos anteriormente conocidos, los ya mencionados AIS y SETM. Los dos algoritmos propuestos, *Apriori* y *AprioriTID*, constituyen la base de muchos algoritmos posteriores. En el mismo trabajo también propusieron *AprioriHybrid*, que combina las mejores características de ambos y consigue un tiempo de ejecución proporcional al número de transacciones de la base de datos.

De ahora en adelante, los *k*-itemsets con soporte igual o mayor al umbral *MinSupport* se consideran pertenecientes al conjunto $L[k]$, el conjunto de k-itemsets frecuentes. En el conjunto $C[k]$, por su parte, se incluyen aquellos k-itemsets que, en principio, podrían pertenecer al conjunto $L[k]$. Los itemsets de $C[k]$ se denominan k-itemsets candidatos, pues son itemsets potencialmente frecuentes.

Los algoritmos de la familia *Apriori* realizan múltiples recorridos secuenciales sobre la base de datos para obtener los conjuntos de itemsets relevantes.

Algoritmo	*Referencia*
AIS	Agrawal, Imielinski & Swami: *Mining Association Rules between Sets of Items in Large Databases*, SIGMOD'93
OCD	Mannila, Toivonen & Verkamo: *Eficient Algorithms for Discovery Association Rules*, KDD'94
Apriori, AprioriTID & AprioriHybrid	
	Agrawal & Skirant: *Fast Algorithms for Mining Association Rules*, VLDB'94
Partition	Savasere, Omiecinski & Navathe: *An Efficient Algorithm for Mining Association Rules in Large Databases*, VLDB'95
SEAR, SPTID, SPEAR & SPINC	
	Mueller: *Fast Sequential and Parallel Algorithms for Association Rule Mining. A Comparison*, University of Maryland - College Park, Master Thesis, 1995
DHP	Park, Chen & Yu: *An Effective Hash-Based Algorithm for Mining Association Rules*, SIGMOD'95
Eclat, MaxEclat, Clique & MaxClique	
	Zaki, Parthasarathy, Ogihara & Li: *New Algorithms for Fast Discovery of Association Rules*, KDD'97
DIC	Brin, Motwani, Ullman & Tsur: *Dynamic Itemset Counting and Implication Rules for Market Basket Data*, SIGMOD'97
MINER(X)	Shen, Shen & Cheng: *New Algorithms for Efficient Mining of Association Rules*, Information Sciences, 1999
CARMA	Hidber: *Online Association Rule Mining*, SIGMOD'99
FP-growth	Han, Pei & Yin: *Mining Frequent Patterns without Candidate Generation*, SIGMOD'2000
TBAR	Berzal, Cubero, Marín & Serrano: *TBAR: An efficient method for association rule mining in relational databases*, Data & Knowledge Engineering, 2001

Tabla 2.5: Algunos algoritmos de extracción de reglas de asociación

Algoritmo	*Referencia*
PDM	Park, Chen & Yu: *Efficient Parallel Data Mining for Association Rules*, CIKM'95
PEAR & PPAR	Mueller: *Fast Sequential and Parallel Algorithms for Association Rule Mining. A Comparison*, University of Maryland - College Park, Master Thesis, 1995
Count, Data & Candidate Distribution	
	Agrawal & Shafer: *Parallel Mining of Association Rules*, IEEE TKDE 1996
DMA	Cheung, Han, Fu & Fu: *Efficient Mining of Association Rules in Distributed Databases*, IEEE TKDE 1996
CCPD	Zaki, Ogihara, Parthasarathy & Li: *Parallel Data Mining for Association Rules On Shared-Memory Multiprocessors*, Supercomputing'96
FDM...	Cheung, Han, Ng, Fu & Fu: *A Fast Distributed Algorithm for Mining Association Rules*, IEEE PDIS 1996
HPA	Shintani & Kitsuregawa: *Hash-based Parallel Algorithms for Mining Association Rules*, IEEE PDIS 1996
Intelligent Data Distribution & Hybrid Distribution	
	Han, Karypis & Kumar: *Scalable Parallel Data Mining for Association Rules*, SIGMOD'97
Eclat	Zaki, Parthasarathy & Li: *A Localized Algorithm for Parallel Association Rule Mining*, SPAA'97
PAR_MINER(X)	Shen, Shen & Cheng: *New Algorithms for Efficient Mining of Association Rules*, Information Sciences, 1999
TreeProjection	Agarwal, Aggarwal & Prasad: *A tree projection algorithm for generation of frequent itemsets*, Journal of Parallel and Distributed Computing, 2000

Tabla 2.6: Algoritmos paralelos de extracción de reglas de asociación

Algoritmos incrementales	
FUP	Cheung, Han, Ng & Wong: *Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique*, ICDE'96
FUP* & MLUp	Cheung, Ng & Tam: *Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules*, KDD'96
Naive, Propagation & Delta	Feldman, Amir, Auman, Zilberstien & Hirsh: *Incremental Algorithms for Association Generation*, KDD Tech. & Appl. 1997
UWEP	Ayan, Tansel & Arkun: *An Efficient Algorithm to Update Large Itemsets with Early Prunning*, KDD'99
Algoritmos con muestreo	
Sampling	Toivonen: *Sampling Large Databases for Association Rules*, VLDB'96
DS & SH	Park, Yu & Chen: *Mining Association Rules with Adjustable Accuracy*, CIKM'97
Reglas de asociación con restricciones	
MultipleJoins, Reorder & Direct	Skirant, Vu & Agrawal: *Mining Association Rules with Item Constraints*, KDD'97
Apriori+, Hybrid(m) & CAP	Ng, Lakshmanan, Han & Pang: *Exploratory Mining and Pruning Optimizations of Constrained Association Rules*, SIGMOD'98
Reglas de asociación generalizadas	
Basic, Cumulate, Stratify, Estimate & EstMerge	Skirant & Agrawal: *Mining Generalized Association Rules*, VLDB'95
NPGM, HPGM y sus diversas variantes	Shintani & Kitsuregawa: *Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy*, SIGMOD'98

Tabla 2.7: Otros algoritmos de extracción de reglas de asociación

En una primera pasada se obtienen los items individuales cuyo soporte alcanza el umbral mínimo preestablecido, con lo que se obtiene el conjunto $L[1]$ de items relevantes. En las siguientes iteraciones se utiliza el último conjunto $L[k]$ de k-itemsets relevantes para generar un conjunto de $(k+1)$-itemsets potencialmente relevantes (el conjunto de itemsets candidatos $C[k+1]$). Tras obtener el soporte de estos candidatos, nos quedaremos sólo con aquéllos que son frecuentes, que incluiremos en el conjunto $L[k+1]$. Este proceso se repite hasta que no se encuentran más itemsets relevantes.

En los algoritmos AIS y SETM, los candidatos se generaban sobre la marcha, conforme se iban leyendo transacciones de la base de datos, lo cual implicaba la generación innecesaria de itemsets candidatos que nunca podrían llegar a ser frecuentes.

Por su parte, los algoritmos derivados de *Apriori* generan los itemsets candidatos única y exclusivamente a partir del conjunto de itemsets frecuentes encontrados en la iteración anterior. Estos algoritmos basan su funcionamiento en una propiedad de los itemsets frecuentes: dado un itemset frecuente, cualquier subconjunto suyo también es frecuente. De esta forma se pueden idear métodos para generar los k-itemsets candidatos del conjunto $C[k]$ pueden a partir del conjunto de $(k-1)$-itemsets relevantes $L[k-1]$. Además, se pueden eliminar de $C[k]$ aquellos itemsets que incluyan algún itemset no frecuente.

Apriori, por ejemplo, realiza la generación del conjunto de candidatos $C[k]$ a partir del conjunto de itemsets relevantes $L[k-1]$ de la siguiente manera:

- En primer lugar se generan posibles candidatos a partir del producto cartesiano $L[k-1] \times L[k-1]$, imponiendo la restricción de que los $k-2$ primeros items de los elementos de $L[k-1]$ han de coincidir.
- Acto seguido se eliminan del conjunto de candidatos aquellos itemsets que contienen algún $(k-1)$-itemset que no se encuentre en $L[k-1]$.

Independientemente del trabajo de Agrawal y Skirant en IBM, Mannila, Toivonen y Verkamo [107] propusieron un procedimiento alternativo para la generación de candidatos en su algoritmo OCD *[Offline Candidate Generation]*, si bien el conjunto de candidatos que obtiene su algoritmo es un superconjunto del obtenido por *Apriori*.

La segunda propuesta de Agrawal y Skirant, *AprioriTID*, se diferencia de *Apriori* en que no accede a la base de datos para obtener el soporte de los itemsets candidatos. Este soporte lo obtiene a partir de conjuntos auxiliares $CT[k]$ similares en cierta medida a los que anteriormente ya se habían utilizado en SETM. En tales conjuntos se incluye un elemento para cada transacción t en el que aparecen, aparte de su identificador TID, todos los *k*-itemsets candidatos presentes en la transacción t.

El tamaño de los conjuntos auxiliares $CT[k]$ puede llegar a ser mucho menor que el de la base de datos original tras unas cuantas iteraciones del algoritmo (para valores grandes de k), lo que se traduce en un ahorro de tiempo considerable al reducir el número de operaciones de entrada/salida realizadas en cada iteración. No obstante, para valores pequeños de k (especialmente cuando k vale 2 ó 3), las entradas correspondientes a cada transacción incluidas $CT[k]$ pueden ocupar más espacio que las propias transacciones en la base de datos original, hecho que dio lugar a la aparición del siguiente algoritmo: *AprioriHybrid*.

> *Belleza: Ajuste proporcional de todas las partes de tal modo que no se puede añadir, eliminar o cambiar algo sin estropear la armonía del conjunto.*
>
> LEON BATTISTA ALBERTI

El algoritmo *AprioriHybrid* consiste en combinar los algoritmos *Apriori* y *AprioriTID* de una forma adecuada, conservando lo mejor de cada uno de ellos. En las primeras iteraciones, *Apriori* se suele comportar mejor que *AprioriTID*, mientras que *AprioriTID* es mejor que *Apriori* cuando el conjunto auxiliar $CT[k]$ cabe en memoria principal. Por tanto, el algoritmo *AprioriHybrid* utiliza *Apriori* en sus primeras iteraciones y *AprioriTID* en las últimas. Si bien suele comportarse mejor que *Apriori*, la mejora puede no ser significativa ya que el verdadero cuello de botella de estos algoritmos se encuentra en las primeras iteraciones.

En este sentido, otro algoritmo derivado de *Apriori*, el algoritmo DHP *[Direct Hashing and Pruning]*, procura reducir el número de candidatos generados

durante las primeras iteraciones utilizando una tabla hash auxiliar para $(k+1)$-itemsets al generar $L[k]$. DHP introduce la utilización de una técnica heurística que permite generar únicamente aquellos itemsets candidatos con una probabilidad elevada de ser frecuentes. Ante la presencia en una transacción t de un itemset i perteneciente a $C[k]$, se incrementan los contadores de las entradas de la tabla hash $h(c)$ para todos los $(k+1)$-itemsets candidatos c que se derivan del itemset i. A continuación, al generar $C[k+1]$ a partir de $L[k]$, podemos descartar automáticamente cualquier candidato c y no incluirlo en $C[k+1]$ si $h(c)$ es menor que el umbral de soporte preestablecido (*MinSupport*), ya que sabemos que no puede pertenecer a $L[k+1]$.

Un factor importante que afecta al rendimiento de los algoritmos de obtención de itemsets relevantes es, evidentemente, el tamaño de la base de datos que ha de recorrerse en cada iteración. Algoritmos como *Apriori* recorren la base de datos completa en cada iteración pero, conforme k aumenta, no sólo disminuye el número de k–itemsets relevantes sino también la cantidad de transacciones que incluyen algún k-itemset frecuente. En consecuencia con lo anterior, se puede ir reduciendo el tamaño de la base de datos en cada iteración conforme se descubren transacciones que no pueden incluir itemsets frecuentes de grado $k+1$ *[transaction trimming]*. Para que una transacción contenga un *(k+1)*-itemset relevante, ésta deberá contener $(k+1)$ *k*-itemsets relevantes como mínimo. Por lo tanto, se pueden descartar para las siguientes iteraciones aquellas transacciones que contengan menos de $(k+1)$ *k*-itemsets relevantes [120]. No obstante, hemos de tener en cuenta que esta operación duplica parte de la base de datos y, cuando se trabaja con grandes bases de datos, en ocasiones no resulta viable.

Cuando el número de candidatos no sea excesivamente elevado, $C[k+1]$ se puede obtener de $C[k] \times C[k]$ directamente, en vez de utilizar $L[k] \times L[k]$. Así se puede reducir el número de veces que se ha de recorrer la base de datos, de forma que se obtienen tanto $L[k]$ como $L[k+1]$ en un único recorrido secuencial de los datos. En el mejor de los casos, con sólo dos recorridos sobre la base de datos podemos ser capaces de obtener todos los conjuntos de itemsets relevantes (el primero es necesario para obtener $L[1]$). Como es lógico, la aplicabilidad de esta técnica depende en gran medida de la memoria disponible

para almacenar conjuntos de itemsets candidatos [7] [107] [120].

El algoritmo *DIC [Dynamic Itemset Counting]* consigue reducir el número de veces que ha de recorrerse la base de datos utilizando una estrategia diferente [24]: los itemsets candidatos se van generando conforme se sabe que pueden llegar a ser frecuentes, no sólo al final de cada iteración. De este modo, el número total de iteraciones se ve drásticamente reducido y se mejora la eficiencia del algoritmo. *CARMA [Continuous Association Rule Mining Algorithm]* lleva hasta sus últimas consecuencias la estrategia de DIC y sólo necesita dos recorridos secuenciales del conjunto de datos para obtener todos los itemsets frecuentes [80], aparte de ingentes cantidades de memoria que lo hacen inaplicable en situaciones complejas.

FP-growth [74], por su parte, utiliza una estructura de datos en forma de árbol, a la que denomina *FP-Tree [Frequent-Pattern Tree]*. Esta estructura de datos permite representar la base de datos de una forma compacta, se construye recorriendo dos veces la base de datos y de él se pueden obtener todos los itemsets frecuentes de forma recursiva.

También existen otros algoritmos de extracción de reglas de asociación que abordan el problema de una forma distinta. MAXMINER [14], por ejemplo, es capaz de descubrir itemsets frecuentes de gran tamaño sin necesidad de generar todos sus subconjuntos, con el objetivo de permitir la exploración de secuencias que los demás algoritmos serían incapaces de analizar. En estas situaciones, los algoritmos derivados de Apriori no son adecuados porque un k-itemset frecuente tiene $2^k - 2$ subconjuntos que son también frecuentes, y como tales han de ser generados y tratados por el algoritmo de extracción de reglas de asociación.

El desarrollo de los algoritmos descritos en este apartado y, en particular, de TBAR [19], específicamente diseñado para extraer reglas de asociación en bases de datos relacionales (capítulo 4), permite que la extracción de reglas de asociación se pueda realizar de forma extremadamente eficiente. Esta propiedad motiva el uso de las reglas de asociación en la resolución de problemas de clasificación, tal como se discute en el siguiente apartado.

2.3.2. Construcción de clasificadores con reglas de asociación

La información obtenida durante el proceso de extracción de reglas de asociación puede servirnos como guía para decidir cómo construir un modelo de clasificación, si bien existen diferencias fundamentales entre los problemas de clasificación y los de extracción de reglas de asociación [63]. Las reglas de asociación no conllevan predicción alguna, ni incluyen mecanismo alguno para evitar el sobreaprendizaje, aparte de un rudimentario umbral mínimo de soporte especificado por el usuario. La resolución de problemas de clasificación, en cambio, requiere un sesgo inductivo: un criterio que nos sirva para seleccionar unas hipótesis frente a otras (el Principio de Economía de Occam, por ejemplo). Este sesgo, como cualquier otro sesgo, es y debe ser dependiente del dominio.

A pesar de las diferencias mencionadas en el párrafo anterior, las reglas de asociación se han utilizado directamente para resolver problemas de clasificación, pues cualquier conjunto de reglas constituye por sí mismo un modelo parcial de clasificación.

En [8], por ejemplo, se utilizan las reglas de asociación para construir modelos de clasificación en dominios en los que los clasificadores convencionales no son efectivos (vg: los árboles de decisión tradicionales resultan especialmente problemáticos cuando existen muchos valores desconocidos o cuando la distribución de clases está muy sesgada). Bayardo [13], por su parte, ha identificado distintas estrategias de poda que pueden utilizarse para mejorar la extracción de reglas de asociación en problemas de clasificación.

Por otro lado, en [158] se propone construir un árbol de reglas a partir de un conjunto de reglas de asociación, árbol al que se denomina ADT (*Association-based Decision Tree*). Dado que la capacidad predictiva de las reglas suele aparecer asociada a un elevado valor de confianza, los autores de este trabajo eliminan el umbral mínimo de soporte para las reglas y sólo tienen en cuenta la confianza de las mismas, aun a costa de ver seriamente afectada la eficiencia del proceso mediante el cual obtienen su ADT, para el cual ya no sirven los algoritmos de la familia de *Apriori*.

CBA [Classification Based on Associations] es un algoritmo para la cons-

trucción de modelos de clasificación propuesto en [100]. Dicho algoritmo extrae todas las reglas de asociación que contengan la clase en su consecuente y, de este conjunto de reglas, CBA selecciona las reglas más adecuadas para construir un "modelo de clasificación asociativo", al que añade una clase por defecto para que sea un modelo completo y no sólo parcial. CBA realiza una búsqueda global exhaustiva utilizando fuerza bruta y obtiene resultados excelentes en comparación con C4.5. Posteriormente, la eficacia de la versión original de CBA "se mejoró" [103] permitiendo la existencia de múltiples umbrales de soporte mínimo para las diferentes clases del problema y recurriendo a algoritmos TDIDT tradicionales cuando no se encuentran reglas precisas.

Una estrategia similar a la de CBA se puede utilizar para clasificar documentos utilizando jerarquías de conceptos [159]. En primer lugar se extraen todas las reglas de asociación generalizadas que incluyan la clase en su consecuente, se ordenan y algunas de ellas se seleccionan para construir un clasificador contextual (ya que la proximidad entre clases es importante al clasificar documentos por temas).

Otros enfoques híbridos también se han sugerido en la literatura sobre el tema. LB [112], abreviatura de "Large Bayes", es un clasificador bayesiano basado en el Naïve Bayes que utiliza un algoritmo de extracción de patrones frecuentes similar a Apriori para obtener patrones frecuentes junto con su soporte para cada clase del problema. Estos valores de soporte se utilizan a continuación para estimar las probabilidades de que el patrón aparezca para cada una de las clases. El algoritmo propuesto consigue buenos resultados pero, sin embargo, carece de la inteligibilidad característica de los modelos simbólicos discutidos en este capítulo.

Otros tipos de patrones se pueden utilizar para construir clasificadores siguiendo la filosofía de LB. Es el caso de los patrones emergentes, EPs [emerging patterns], que son itemsets cuyo soporte se incrementa significativamente de un conjunto de datos a otro [49]. CAEP [50], por ejemplo, descubre todos los patrones emergentes que superan para cada clase los umbrales de soporte y proporción de crecimiento (un parámetro adicional utilizado para evaluar los patrones emergentes). A continuación, CAEP calcula un índice discriminante que le sirve para determinar, dado un ejemplo concreto, cuál es la clase más

adecuada para él. Este índice está ideado para que el algoritmo CAEP funcione bien incluso cuando las poblaciones de ejemplos de cada clase estén muy desequilibradas. Sin embargo, como sucede con LB, CAEP no proporciona un modelo de clasificación inteligible.

Siguiendo otra línea de investigación, Liu, Hu y Hsu [101] proponen la utilización de una representación jerárquica que consiste en utilizar reglas generales y excepciones a esas reglas en vez del modelo plano tradicional en el cual la existencia de demasiadas reglas de asociación dificulta, si no imposibilita, la comprensibilidad del modelo de clasificación construido. El mismo enfoque se aplica en [102] para obtener un resumen de la información contenida en un conjunto arbitrario de reglas.

El modelo de clasificación propuesto en esta memoria, ART, que será objeto de estudio en el siguiente capítulo, también utiliza técnicas de extracción de reglas de asociación para, aprovechando las regularidades existentes en el conjunto de entrenamiento, construir un clasificador robusto y fiable en forma de árbol de decisión. Las reglas de asociación resultan especialmente adecuadas porque permiten caracterizar los patrones existentes en el conjunto de datos de entrenamiento y se pueden obtener eficientemente.

En vez de utilizar reglas generales y excepciones como en [101], ART emplea ramas 'else' intentando combinar las ventajas de los algoritmos de construcción de árboles de decisión con las asociadas a las técnicas de inducción de listas de decisión y a los eficientes algoritmos de extracción de reglas de asociación.

Como se verá en el capítulo siguiente, ART es capaz de utilizar combinaciones de distintos atributos para ramificar el árbol de decisión. En estudios previos [163] se ha demostrado que la utilización simultánea de varios atributos para ramificar un árbol de decisión resulta beneficiosa, algo que la topología del árbol generado por ART permite y fomenta. La estrategia de búsqueda empleada por ART, además, ofrece la posibilidad de obtener árboles n-arios utilizando simultáneamente varios atributos para ramificar el árbol, a diferencia de otras propuestas previas que sólo permiten la construcción de árboles binarios cuando se emplean múltiples atributos simultáneamente (p.ej. expresiones booleanas en BCT [27] y combinaciones lineales en CART [23]).

Capítulo 3

El modelo de clasificación ART

...
cincel que el bloque muerde
la estatua modelando,
...
atmósfera en que giran
con orden las ideas,
cual átomos que agrupa
recóndita atracción;

GUSTAVO ADOLFO BÉCQUER
Rimas

El acrónimo TDIDT hace referencia a todos los algoritmos "divide y vencerás" que construyen árboles de decisión desde la raíz hasta las hojas *recursivamente* (sección 2.1). En cierto modo, el modelo de clasificación que se presenta en este capítulo puede considerarse un algoritmo más de la familia TDIDT, pues construye un árbol de decisión comenzando por su nodo raíz.

El modelo aquí presentado, sin embargo, no utiliza una medida de impureza como la entropía, el criterio de proporción de ganancia o el índice de Gini. En vez de emplear una medida heurística, se utiliza la información obtenida a través de un algoritmo de extracción de reglas de asociación para decidir cómo ramificar el árbol. Esta estrategia da nombre al modelo propuesto en esta memoria: *Association Rule Tree* (ART).

ART construye un modelo de clasificación parcial con reglas de asociación y utiliza un subconjunto del conjunto de reglas obtenido para ramificar el árbol de decisión. Las reglas seleccionadas puede que no cubran algunos de lo ejemplos del conjunto de entrenamiento, por lo que dichos ejemplos se agrupan todos en una rama 'else' para ser procesados conjuntamente. El proceso se repite para descubrir reglas aplicables a los ejemplos de la rama 'else' mientras queden ejemplos de entrenamiento por clasificar.

Al finalizar la construcción del clasificador, el modelo obtenido por ART tiene la forma del mostrado en la figura 3.1. En esta figura se muestra un clasificador construido por ART cuyo objetivo es clasificar las uniones de genes en secuencias de ADN a partir de un conjunto de entrenamiento que contiene secuencias de nucleótidos. Este problema se describirá con mayor detalle en el apartado 3.3 de esta memoria.

En las siguientes secciones se analizará en profundidad el modelo de clasificación ART. La sección 3.1 estudia la construcción de clasificadores con ART, tras lo cual se muestra un ejemplo detallado en la sección 3.2. Las dos secciones siguientes describen el uso y propiedades de los clasificadores obtenidos utilizando el modelo propuesto en este capítulo. La sección 3.4 comenta algunos detalles relacionados con el uso de clasificadores ART, mientras que en la sección 3.5 se destacan las cualidades más interesantes. Finalmente, la sección 3.6 recopila los resultados experimentales que se han obtenido al utilizar ART en las resolución de distintos problemas de clasificación.

```
P30 = A : TYPE = N (473|62)
P30 = C : TYPE = N (441|24)
P30 = T : TYPE = N (447|57)
else
  P28 = A and P32 = T : TYPE = EI (235|33)
  P28 = G and P32 = T : TYPE = EI (130|20)
  P28 = C and P32 = A : TYPE = IE (160|31)
  P28 = C and P32 = C : TYPE = IE (167|35)
  P28 = C and P32 = G : TYPE = IE (179|36)
  else
    P28 = A : TYPE = N (106|14)
    P28 = G : TYPE = N (94|4)
    else
      P29 = C and P31 = G : TYPE = EI (40|5)
      P29 = A and P31 = A : TYPE = IE (86|4)
      P29 = A and P31 = C : TYPE = IE (61|4)
      P29 = A and P31 = T : TYPE = IE (39|1)
      else
        P25 = A and P35 = G : TYPE = EI (54|5)
        P25 = G and P35 = G : TYPE = EI (63|7)
        else
          P23 = G and P35 = G : TYPE = EI (40|8)
          P23 = T and P35 = C : TYPE = IE (37|7)
          else
            P21 = G and P34 = A : TYPE = EI (41|5)
            else
              P28 = T and P29 = A : TYPE = IE (66|8)
              else
                P31 = G and P33 = A : TYPE = EI (62|9)
                else
                  P28 = T : TYPE = N (49|6)
                  else
                    P24 = C and P29 = A : TYPE = IE (39|8)
                    else
                      TYPE = IE (66|39)
```

Figura 3.1: Un modelo de clasificación construido con ART.

3.1. Construcción del clasificador

El algoritmo ART de construcción de modelos de clasificación construye un árbol de decisión utilizando una estrategia de control irrevocable, igual que la mayor parte de los algoritmos TDIDT; es decir, emplea un algoritmo greedy en el cual una decisión no se revoca una vez que ha sido tomada.

A continuación se ofrece una visión general del algoritmo ART, describiendo a grandes rasgos el proceso de construcción del clasificador y omitiendo detalles que serán analizados en los apartados siguientes de este capítulo.

ART comienza el proceso de construcción del clasificador buscando reglas de asociación simples como $\{A_i.a_i\} \Rightarrow \{C.c_j\}$, donde A es un atributo, a es el valor de ese atributo, C es el atributo de la clase y c es una de las posibles clases del problema. Para ramificar el árbol, ART selecciona el atributo A con el *mejor conjunto de reglas adecuadas* $\{A_i.a_i\} \Rightarrow \{C.c_j\}$. Acto seguido, ART genera u nodo hoja para cada *regla adecuada* y todos los ejemplos del conjunto de entrenamiento no cubiertos por ninguna de esas reglas se agrupan en una rama 'else' para ser tratados a continuación aplicando recursivamente el mismo algoritmo sobre los datos que aún no han sido clasificados.

En este contexto, una *regla adecuada* es una regla precisa; esto es, una regla de asociación con confianza lo suficientemente alta para ser útil en la construcción de un buen clasificador. El *mejor conjunto de reglas adecuadas* es el conjunto de reglas más prometedor, el conjunto de reglas que parece ser mejor para construir un modelo predictivo desde un punto de vista heurístico, obviamente. Estos conjuntos de reglas se construyen de modo que todas las reglas de asociación incluidas en un conjunto dado compartan los atributos que aparecen en sus antecedentes. De este modo nos aseguramos de que las distintas ramas del árbol construido serán mutuamente excluyentes.

Cuando no se encuentra ninguna regla de asociación adecuada con un único par atributo-valor $A.a$ en su antecedente, ART busca reglas de asociación más complejas añadiendo atributos a sus antecedentes. Se empieza generando reglas de la forma $\{A_1.a_1\ A_2.a_2\} \Rightarrow \{C.c_j\}$. Si no se consiguen buenas reglas de asociación, ART prosigue su búsqueda con reglas que contengan tres pares atributo-valor en sus antecedentes, $\{A_1.a_1\ A_2.a_2\ A_3.a_3\} \Rightarrow \{C.c_j\}$, y

así sucesivamente.

En consecuencia, ART, a diferencia de otros algoritmos TDIDT, es capaz de utilizar varios atributos simultáneamente para ramificar el árbol de decisión. Esta característica posibilita una mejora en la capacidad del clasificador, incrementando su precisión y disminuyendo su complejidad [163].

Es aconsejable, no obstante, establecer una cota superior para el tamaño de los antecedentes de las reglas de asociación. Esta cota sirve para detener la búsqueda si no se encuentran reglas adecuadas y se designa por *MaxSize*. El máximo valor posible para esta cota es igual al número de atributos predictivos en el conjunto de datos de entrenamiento. Este valor máximo para el parámetro *MaxSize* viene dictado por la Primera Forma Normal, que establece que los valores de un atributo han de ser atómicos: un caso de entrenamiento no puede tener, por tanto, más pares atributo-valor que atributos predictivos haya en el conjunto de entrenamiento.

> *Que sea sencillo: lo más sencillo posible, pero no más.*
>
> ALBERT EINSTEIN

Tal como se ha comentado en los párrafos anteriores, ART comienza utilizando hipótesis sencillas para intentar clasificar los ejemplos del conjunto de entrenamiento y sólo recurre a hipótesis más complejas cuando no se pueden encontrar hipótesis más simples que se adapten a los datos. Por tanto, ART incorpora un criterio explícito de preferencia por modelos sencillos utilizando el Principio de Economía de Occam como sesgo inductivo. A pesar de las críticas que la "navaja" de Occam ha recibido en algunos trabajos de extracción de conocimiento en bases de datos [47] [48], este criterio que aún sigue siendo adecuado y útil para resolver problemas de clasificación de muy diversa índole. De cualquier forma, siempre es necesario algún tipo de sesgo inductivo para construir clasificadores, ya que el número de hipótesis que implican unos hechos dados es potencialmente infinito [63]. En este contexto, el criterio de Occam sigue siendo tan válido como cualquier otro.

El diagrama de la página siguiente muestra el flujo de control del algoritmo ART, cuyo pseudocódigo aparece esbozado en la figura 3.3 (página 68).

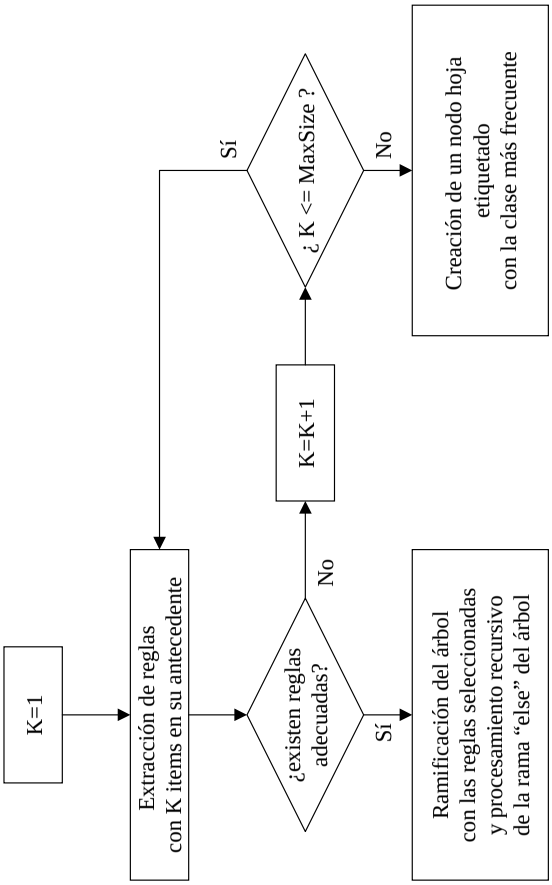

Figura 3.2: Algoritmo ART

En el pseudocódigo de ART mostrado en la figura 3.3 aparece indicado el uso de distintos parámetros. La tabla 3.1 describe la función de estos parámetros, que son necesarios para acotar el espacio de búsqueda explorado al construir el árbol de decisión. Las implicaciones del uso de estos parámetros serán objeto de estudio en los siguientes apartados, conforme se discutan los distintos aspectos relacionados con la construcción de clasificadores ART.

3.1.1. Selección de reglas: Criterio de preferencia

Cuando se construye un árbol de decisión, se ha de elegir cómo ramificar el árbol de entre las distintas particiones posibles del conjunto de datos de entrada, para lo que se han propuesto distintas reglas de división (apartado 2.1.1). Por ejemplo, ID3 y C4.5 utilizan reglas derivadas de la Teoría de la Información mientras que CART emplea el índice de Gini. De hecho, todos los algoritmos clásicos de construcción de árboles de decisión utilizan alguna medida de impureza o diversidad que tratan de minimizar.

El enfoque de ART es diferente. Una vez obtenidas algunas reglas adecuadas para clasificar los datos de entrada (un modelo de clasificación parcial), se seleccionan las mejores de ellas para construir un nivel del árbol de decisión (que una vez acabado será un modelo de clasificación completo por sí mismo).

Nuestro algoritmo comprueba la existencia de, al menos, un conjunto de atributos $\{A_1\ A_2\ ..\ A_k\}$ que sea adecuado para ramificar el árbol de decisión. El conjunto de atributos seleccionado debe proporcionar un conjunto de reglas precisas de la forma $\{A_1.a_1\ A_2.a_2\ ..\ A_k.a_k\} \Rightarrow \{C.c_j\}$. Cada regla de ese tipo se empleará para etiquetar los datos de entrada que verifiquen su antecedente como pertenecientes a la clase c_j.

El propósito de ART es aprovechar la información obtenida en forma de reglas de asociación para construir un buen clasificador. Todas las reglas extraídas en cada etapa del algoritmo incluyen *k*-itemsets en sus antecedentes. Cada *k*-itemset corresponde a un conjunto de k atributos diferentes (recuérdese nuevamente la Primera Forma Normal). Las reglas correspondientes a cada conjunto distinto de k atributos se agrupan y se recurre a heurísticas para seleccionar, de entre todos los conjuntos de k atributos, el mejor candidato para ramificar el árbol.

función ART (datos, MaxSize, MinSupp, MinConf): clasificador;

k = 1; *// Tamaño del antecedente*
árbol = $\varnothing$; *// Árbol resultante*

mientras ((árbol = $\varnothing$) **y** ($k \leq$ MaxSize))

// Extracción de reglas
Encontrar reglas en el conjunto de datos de entrada con
k items en el antecedente y la clase en el consecuente
v.g. $\{A_1.a_1$ .. $A_k.a_k\} \Rightarrow \{C.c_j\}$

si existen reglas buenas

// Selección de reglas
Seleccionar el mejor conjunto de reglas con el mismo
conjunto de atributos $\{A_1..A_k\}$ en el antecedente
de acuerdo con el criterio de preferencia.

// Ramificación del árbol
árbol = A partir de las reglas seleccionadas
- Cada regla $\{A_1.a_1$ .. $A_k.a_k\} \Rightarrow \{C.c_j\}$
se utiliza para crear una nueva hoja (etiquetada c_j).
- Todos los ejemplos de entrenamiento no cubiertos
por las reglas seleccionadas se agrupan en una rama
'else' que se construye recursivamente:
árbol.else = ART (datos, MaxSize, MinSupp, MinConf);

si no
k = k + 1;

si árbol = $\varnothing$ *// No se ha construido árbol alguno*
árbol = Nodo hoja etiquetado con la clase más frecuente;

devuelve árbol;

Figura 3.3: Pseudocódigo del algoritmo ART

Parámetro	*Descripción*
MaxSize	Tamaño máximo del antecedente de las reglas empleadas para construir el árbol de decisión. Por defecto es igual al número de atributos predictivos presentes en el conjunto de entrenamiento, si bien conviene establecer un valor menor para reducir el tiempo necesario para construir el árbol.
MinSupp	Umbral de soporte mínimo utilizado por el algoritmo de extracción de reglas de asociación. Su valor por defecto puede establecerse en torno a 0.1 (10 %). Este parámetro permite restringir el factor de ramificación del árbol de decisión y ajustar la granularidad del modelo construido (sección 3.5).
MinConf	Umbral de confianza mínimo para las reglas de asociación. Por defecto se utiliza una heurística que permite seleccionar automáticamente este parámetro, por lo que, normalmente, el usuario no tiene por qué especificarlo (véase la discusión sobre la selección del umbral en el apartado 3.1.3).

Tabla 3.1: Descripción de los parámetros de ART.

NOTA: A estos parámetros hay que añadirles el conjunto de datos de entrenamiento utilizado para construir el clasificador.

Por ejemplo, podríamos haber obtenido los siguientes conjuntos de reglas (todas las cuales satisfacen las restricciones de soporte y confianza en nuestro problema hipotético):

`Conjunto 1`:	`Regla 1`:	if W=w_4 and Y=y_2 then C=c_2
	`Regla 2`:	if W=w_3 and Y=y_3 then C=c_1
`Conjunto 2`:	`Regla 3`:	if X=x_1 and Z=z_1 then C=c_4
	`Regla 4`:	if X=x_1 and Z=z_5 then C=c_2

Como ART emplea una estrategia de control irrevocable, tenemos que seleccionar el `Conjunto 1` o el `Conjunto 2` y olvidarnos del otro.

Una heurística sencilla consiste en escoger el conjunto de atributos que clasifica correctamente más ejemplos del conjunto de entrenamiento (utilizando las reglas asociadas al conjunto de atributos seleccionado, obviamente). Dicha heurística hace máximo el número de ejemplos clasificados correctamente y, por tanto, hace mínimo el número de ejemplos que quedan por clasificar tras construir un nivel del árbol de decisión. Por tanto, indirectamente tiende a reducir la altura del árbol de decisión siguiendo el Principio de Economía de Occam.

Continuando el ejemplo anterior, si el primer conjunto de reglas (`Regla 1` y `Regla 2`) cubre a 200 ejemplos del conjunto de entrenamiento mientras que el segundo, formado por (`Regla 3` y `Regla 4`), sólo abarca 150 ejemplos, optaríamos por utilizar el `Conjunto 1` para ramificar el árbol.

3.1.2. Topología del árbol: Ramas 'else'

Una vez que hemos seleccionado un conjunto de reglas con el mismo conjunto de atributos en todos sus antecedentes, tenemos que ramificar el árbol utilizando la información obtenida durante el proceso de extracción de reglas de asociación:

- Cada una de las reglas seleccionadas conduce a una nueva hoja del árbol. Cada una de estas hojas se etiqueta con la clase que aparece en el conse-

cuente de la regla correspondiente. Siguiendo la notación de Quinlan en C4.5 [131], las etiquetas de las hojas pueden venir acompañadas por dos números $(n|e)$. El primero de ellos, n, indica el número de ejemplos de entrenamiento que caen en cada hoja (es decir, el soporte del antecedente de la regla correspondiente); mientras que el segundo, e, es el número de ejemplos del conjunto entrenamiento mal clasificados por la etiqueta de la hoja.

- Todos los ejemplos del conjunto de entrenamiento que no verifican el antecedente de ninguna de las reglas seleccionadas se agrupan en una rama 'else' para continuar por ella la construcción del árbol de decisión.

Utilizando el ejemplo de la sección anterior, se crearían dos nuevos nodos hoja al incluir dos reglas diferentes el conjunto seleccionado. Todos los ejemplos que verifiquen el antecedente de cualquiera de las dos reglas (`1` y `2`) se envían a las dos hojas recién creadas y los restantes casos del conjunto de entrenamiento se agrupan en una rama 'else':

W=w_4 and Y=y_2: C=c_2 (92|2)
W=w_3 and Y=y_3: C=c_1 (108|5)
`else`
...

ART difiere de los algoritmos TDIDT tradicionales porque utiliza ramas 'else' para agrupar todas aquellas tuplas que no correspondan a ninguna hoja de las generadas en cada nivel del árbol. En otras palabras, el conjunto de datos que queda sin cubrir tras ramificar el árbol se trata como un todo. Por tanto, ART no necesita crear una nueva rama para cada posible valor de los atributos seleccionados. ART centra su atención únicamente en aquéllos valores que generan resultados interesantes, reduciendo así el factor de ramificación del árbol. Este hecho, a su vez, permite que ART utilice simultáneamente varios atributos para ramificar el árbol, en vez del único atributo o test binario que usan casi todos los algoritmos TDIDT.

La utilización de ramas ‘else’ conduce a la construcción de un árbol de decisión cuya topología se asemeja a la de una lista de decisión. Este modelo de representación del conocimiento es más difícil de interpretar que los árboles de decisión tradicionales, pues el significado de una regla vendrá determinado por su posición en el modelo de clasificación construido. No obstante, el uso de ramas ‘else’ ayuda a construir mejores clasificadores agrupando pequeños subconjuntos de datos que, de otra forma, corresponderían a distintos subárboles en algoritmos como C4.5. Además, dichos conjuntos de datos llegarían eventualmente a ser tan pequeños que resultarían inútiles para construir modelos de clasificación exactos. Agrupando esos pequeños conjuntos de datos se puede conseguir una mayor precisión porque el efecto del ruido se reduce y se evita el problema del sobreaprendizaje.

En cierto sentido, ART sigue la filosofía de los algoritmos propuestos por Liu y sus colaboradores [101][102], quienes intentan organizar y resumir el conjunto de todas las reglas descubiertas mediante reglas más generales y excepciones a esas reglas. El objetivo es reemplazar un conjunto de potencialmente muchas reglas por una representación más concisa del conocimiento obtenido. El enfoque de ART a la hora de construir árboles de decisión permite alcanzar este objetivo conforme se crea el modelo de clasificación, mientras que Liu y sus colaboradores lo logran a posteriori.

Un efecto secundario del uso de ramas ‘else’ es que ART puede construir un árbol de decisión iterativamente, ya que la recursión de cola del algoritmo recursivo puede eliminarse para obtener un algoritmo iterativo equivalente, tal como se muestra en la figura 3.4. Los demás algoritmos TDIDT son inherentemente recursivos y requieren una implementación algo más compleja. De hecho, una implementación iterativa de nuestro algoritmo permite un aprendizaje más eficiente de árboles de decisión cuando el conjunto de datos de entrada es enorme.

Además, existen algoritmos eficientes de extracción de reglas de asociación y se pueden utilizar distintas técnicas para mejorar el rendimiento de ART. Por ejemplo, el tamaño del conjunto de entrenamiento se puede ir reduciendo conforme se profundiza en el árbol (e, independientemente, conforme se van generando itemsets [120]).

```
función ARTiterativo
        (datos, MaxSize, MinSupp, MinConf): clasificador;

var árbol: clasificador;                    // Árbol de decisión

nodo = árbol.raíz;                          // Comenzar por la raíz del árbol
entrenamiento = datos;                      // Conjunto de entrenamiento actual

mientras entrenamiento ≠ ∅                  // Construir nivel actual

        k = 1;                              // Tamaño del antecedente
        nuevoNodo = ∅;                      // Nuevo subárbol

        mientras ( (nuevoNodo=∅) y (k ≤ MaxSize) )

                Extraer reglas con k items en el antecedente

                si existen reglas con las cuales ramificar el árbol

                   Seleccionar del mejor conjunto de reglas
                   nuevoNodo = Nuevo nodo interno con:
                                - Una hoja por cada regla seleccionada.
                                - Una rama 'else' inicialmente vacía.
                   Reemplazar 'nodo' con 'nuevoNodo' en el árbol
                   nodo = nuevoNodo.else;
                   Eliminar los ejemplos de entrenamiento cubiertos
                             por alguna de las reglas seleccionadas.

                si no
                   k = k + 1;

        si nuevoNodo = ∅ // No se ha construido subárbol alguno
           Reemplazar 'nodo' con un nodo hoja
              (etiquetado con la clase más frecuente en 'entrenamiento')
           entrenamiento = ∅;

devuelve árbol;
```

Figura 3.4: Implementación iterativa de ART (completamente equivalente a la implementación recursiva del algoritmo que aparece en la figura 3.3).

3.1.3. Extracción de reglas: Hipótesis candidatas

La primera etapa de ART consiste en descubrir aquellas reglas derivadas del conjunto de entrenamiento que puedan ser potencialmente útiles para clasificar. Dichas reglas se utilizan para ramificar el árbol de decisión que ART construye de la forma que hemos visto en los apartados anteriores. Para completar el proceso de construcción del clasificador ART, por tanto, resulta imprescindible el uso de alguna técnica que nos permita descubrir tales reglas en el conjunto de datos de entrenamiento.

En el capítulo anterior se comentaron distintas técnicas de inducción de reglas y su uso en la construcción de clasificadores. Entre las técnicas analizadas se encontraban múltiples algoritmos de extracción de reglas de asociación, los cuales destacan por ser altamente eficientes y escalables. Los algoritmos tradicionales de extracción de reglas de asociación están ideados para extraer reglas de bases de datos transaccionales y no son adecuados para trabajar con conjuntos de datos densos (esto es, conjuntos de datos con un número relativamente pequeño de valores desconocidos). Desgraciadamente, los conjuntos de datos que se utilizan para resolver problemas de clasificación suelen ser densos al provenir de bases de datos relacionales. Por suerte, existen a nuestra disposición algoritmos diseñados para resolver este problema. Tal como se verá en el capítulo siguiente de esta memoria, un algoritmo como TBAR [19] resulta especialmente adecuado, pues está pensado explícitamente para extraer reglas de asociación en bases de datos relacionales.

El proceso de extracción de reglas de asociación usual consiste en descubrir todas las reglas de asociación que verifiquen las restricciones impuestas por el usuario mediante los umbrales de soporte y confianza, *MinSupp* y *MinConf* respectivamente. Evidentemente, a la hora de construir modelos de clasificación sólo nos interesan, de todas las reglas posibles, aquellas reglas que incluyan el atributo correspondiente a la clase en su consecuente.

Al emplear un algoritmo de extracción de reglas de asociación para construir un modelo de clasificación parcial, ART requiere, al menos en principio, que el usuario especifique los parámetros que guían el proceso de extracción de reglas de asociación: el umbral de soporte mínimo para los itemsets frecuen-

tes (*MinSupp*) y un valor mínimo para la confianza de las reglas de asociación (*MinConf*), aunque este último podrá omitirse, como se verá posteriormente.

3.1.3.1. El umbral de soporte mínimo: *MinSupp*

El umbral de soporte mínimo *MinSupp* puede fijarse como una cantidad determinada de tuplas o como un porcentaje sobre el tamaño del conjunto de entrenamiento (que va disminuyendo de tamaño conforme el algoritmo avanza). En el primer caso, se establece de antemano un umbral absoluto y se mantiene durante todas las etapas de construcción del árbol de decisión. Cuando se utiliza un umbral relativo, no obstante, el umbral real se va adaptando al tamaño del conjunto de datos.

Si, por ejemplo, el umbral de soporte relativo *MinSupp* se fija en 0.1 y comenzamos con 1000 tuplas en el conjunto de entrenamiento, el umbral de soporte absoluto será igual a 100 tuplas en la raíz del árbol e irá disminuyendo conforme el algoritmo avance. Cuando queden N tuplas en el conjunto de entrenamiento se utilizará 0.1*N como umbral de soporte para obtener los itemsets frecuentes en el conjunto de entrenamiento que nos queda, itemsets que quizá no sean frecuentes en el conjunto de entrenamiento completo.

Al establecer *MinSupp* como un porcentaje sobre el tamaño del conjunto de casos que nos queda por procesar, este parámetro se ajusta al tamaño de ese conjunto de forma que el usuario no tiene que afinarlo. Un valor de este umbral relativo en torno a 0.1 es razonable para construir árboles de decisión que no sean ni demasiado frondosos ni excesivamente escuetos.

Implícitamente, el umbral de soporte restringe el factor de ramificación del árbol. De esta forma, ART se asegura de que nunca se escogerán claves primarias ni claves candidatas para ramificar el árbol de decisión, un problema bastante común en otros algoritmos TDIDT.

En la sección 3.5 se puede encontrar una discusión algo más formal acerca del efecto que causa el umbral *MinSupp* en las propiedades del árbol de decisión construido por ART.

3.1.3.2. El umbral de confianza mínima: *MinConf*

> *...es característico de una mente instruida descansar satisfecha con el grado de precisión permitido por la naturaleza en cada asunto y no buscar la exactitud cuando sólo es posible una aproximación de la verdad...*
>
> ARISTÓTELES
> *330 a.C.*

Veamos, a continuación, cómo trabajar con el otro umbral que interviene en el proceso de extracción de reglas, el que establece la confianza mínima de las reglas de asociación: *MinConf*. Este parámetro influye decisivamente en el conjunto de reglas que se considerarán candidatas para ramificar el árbol y, por tanto, debería tener un valor cercano a 1 para asegurar que sólo se tendrán en cuenta reglas muy precisas (0.9 produce buenos resultados en muchos problemas). En nuestros experimentos hemos encontrado, no obstante, que el parámetro *MinConf* no debería ajustarlo el usuario. Como mostraremos, permitir que ART lo ajuste automáticamente suele conducir a mejores resultados.

De cualquier modo, si el usuario así lo exige, puede establecer manualmente un valor para el umbral *MinConf*. Un problema típico en el que el usuario puede estar interesado en fijar un umbral ad hoc es el conjunto de datos MUSHROOM del Machine Learning Repository de la Universidad de California en Irvine. Este conjunto de datos se utiliza para construir clasificadores que ayuden a determinar si una seta es venenosa o no a partir de sus propiedades organolépticas. Dado que un simple error de clasificación podría tener consecuencias dramáticas, un falso positivo no es permisible: Etiquetar una seta venenosa como comestible podrían conllevar graves problemas de salud para el degustador de setas, incluso su muerte. Por tanto, este conjunto de datos requiere un estricto umbral de confianza mínima igual a 1 (100 %).

En problemas como el mencionado en el párrafo anterior, es imprescindible la utilización de un umbral de confianza fijo para que el modelo de clasificación construido por ART tenga las propiedades deseadas por el usuario.

En tales situaciones, incluso, se podría plantear la conveniencia de establecer un umbral de confianza independiente para cada clase del problema, dependiendo de su semántica. En el problema concreto de las setas, que requiere la ausencia absoluta de falsos positivos para garantizar que no se ponga en peligro la salud de ningún aficionado a la micología, un falso negativo no tendría mayores consecuencias: etiquetar como venenosa una seta comestible sólo evita su posible degustación y no acarrea mayores complicaciones.

A pesar de la discusión recogida en los párrafos anteriores, un conjunto de datos como MUSHROOM es un caso extremo. Establecer a priori un umbral de confianza mínima puede ser contraproducente si dicho umbral no es realista, ya que impediría obtener modelo de clasificación alguno si su valor fuese excesivamente alto para un conjunto de datos particular. En esta situación, no se extraería ninguna regla de asociación del conjunto de datos y no se podría seguir ramificando el árbol de decisión utilizando ART (si bien siempre se puede recurrir a un algoritmo TDIDT tradicional es estos casos). Por otro lado, elevar el valor del umbral *MinConf* no implica necesariamente obtener un porcentaje de clasificación mayor (véase el apartado 3.6.4) y, desde un punto de vista práctico, el tiempo de ejecución del algoritmo se podría incrementar notablemente.

Visto lo anterior, resulta conveniente, si no necesario, incluir en ART algún mecanismo automático de selección del umbral de confianza. Una vez que se haya descubierto la regla de asociación más precisa posible (aquélla con mejor valor de confianza), sólo se deberían tener en cuenta reglas similares en cuanto a su precisión a la hora de construir el árbol de decisión, con el objetivo de que la precisión del modelo obtenido no se degrade innecesariamente.

Una heurística que hemos encontrado y parece funcionar bastante bien consiste en considerar en cada etapa del algoritmo sólo aquellas reglas cuya confianza supere el valor *MaxConf* - Δ, donde *MaxConf* es valor máximo de confianza obtenido de entre todas las reglas de asociación descubiertas. Esta heurística, parecida al criterio utilizado por las funciones de evaluación lexicográficas de los algoritmos STAR (página 37), nos sirve para considerar equivalentes aquellas reglas cuyo grado de cumplimiento es similar, estableciendo un "margen de tolerancia" dado por el intervalo [*MaxConf* - Δ, *MaxConf*]. En

concreto, se pueden obtener buenos resultados si se emplea el umbral de soporte mínimo como valor para el parámetro Δ (Δ = *MinSupp*) y 100 % como valor inicial de *MaxConf* (esto es, cuando aún no se ha descubierto ninguna regla, aspiramos a conseguir un modelo de clasificación perfecto).

Al utilizar la heurística anterior, se selecciona la mejor regla posible de las descubiertas y sólo se incluyen en el conjunto de reglas buenas aquéllas que son ligeramente peores que la mejor, las reglas cuya confianza queda dentro del intervalo [*MaxConf* - Δ, *MaxConf*]. Esta táctica restrictiva evita que el algoritmo ART tenga en cuenta reglas no muy buenas a la hora de construir hojas del árbol de decisión y deje abierta la posibilidad de encontrar mejores reglas en los siguientes niveles del árbol. Además, en el peor de los caso, aunque una regla adecuada no se seleccione como buena en el nivel actual del árbol, dicha regla se acabará seleccionando si no se encuentran reglas mejores al procesar los datos restantes (los de la rama 'else' del árbol de decisión).

La experimentación realizada, que aparece recogida en la sección 3.6 de este capítulo, nos ha mostrado que la selección automática del umbral de confianza obtiene soluciones casi óptimas. En nuestra opinión, el umbral de confianza sólo debería establecerlo manualmente el usuario cuando la semántica del problema lo requiera y sea estrictamente necesario (como sucedía con el conjunto de datos MUSHROOM). La incorporación de un umbral de confianza preestablecido en el mecanismo de construcción de árboles de ART es, en esos casos, positiva. En definitiva, el usuario normalmente no lo utilizará, aunque siempre tendrá la posibilidad de ajustarlo si su problema lo necesita.

3.2. Un ejemplo detallado

En esta sección utilizaremos el sencillo conjunto de datos de la tabla 3.2 para ilustrar el funcionamiento de nuestro algoritmo. Dicho conjunto de datos podría ser una versión simplificada del conjunto de datos usado para construir un sistema de ayuda a la decisión que permitiese conceder o denegar créditos. En este pequeño conjunto de datos, X, Y y Z son los atributos predictivos mientras que C indica la clase para nuestro problema.

	X	*Y*	*Z*	*C*
Cliente	¿desempleado?	¿con fondos?	¿con inmuebles?	¿crédito?
I1	0	0	0	0
I2	0	0	1	0
I3	1	1	0	0
I4	1	0	0	0
I5	1	1	1	1
I6	1	0	1	1
I7	0	1	0	1
I8	0	1	1	1

Tabla 3.2: Un conjunto de datos sencillo

Supongamos que utilizamos la selección automática de umbral de confianza de ART y establecemos el umbral de soporte relativo al 20 % (para requerir que, en la raíz del árbol, las reglas cubran al menos dos tuplas). Por último, el parámetro *MaxSize* es igual al número de atributos predictivos por defecto (3 en este sencillo ejemplo).

- *Nivel 1: Extracción de reglas*

  Comenzamos con el conjunto de datos completo e intentamos encontrar reglas con un único item en su antecedente (esto es, $k = 1$). Se obtienen los siguientes conjuntos de reglas:

```
S1: if (Y=0) then C=0 with confidence 75%
    if (Y=1) then C=1 with confidence 75%

S2: if (Z=0) then C=0 with confidence 75%
    if (Z=1) then C=1 with confidence 75%
```

  Si hubiésemos establecido un umbral de confianza rígido, aceptaríamos o rechazaríamos cualquiera de los conjuntos anteriores en función del valor que le hubiésemos asignado al umbral. Cualquier valor por encima

del 75 % haría que ART rechazase las reglas descubiertas al no considerarlas suficientemente buenas y buscase hipótesis más complejas para clasificar los datos. Si el umbral fuese igual o inferior al 75 %, ART escogería bien `S1` o bien `S2` para ramificar el árbol y el algoritmo terminaría en este punto.

Si embargo, estamos utilizando la selección automática de umbral proporcionada por ART y aspiramos a conseguir un clasificador mejor (con una precisión del 100 % a ser posible). Comprobamos que todas las reglas están por debajo del 80 % de confianza, valor de corte que proviene de 100 % (valor deseado inicialmente) - 20 % (margen de tolerancia fijado por el soporte mínimo). Al no haber ninguna regla lo suficientemente buena, ART busca conjuntos de reglas más complejas; esto es, reglas con más atributos en sus antecedentes. Con $k = 2$ se obtienen las siguientes reglas:

```
S1: if (X=0 and Y=0) then C=0 with confidence 100%
    if (X=0 and Y=1) then C=1 with confidence 100%

S2: if (X=1 and Z=0) then C=0 with confidence 100%
    if (X=1 and Z=1) then C=1 with confidence 100%

S3: if (Y=0 and Z=0) then C=0 with confidence 100%
    if (Y=1 and Z=1) then C=1 with confidence 100%
```

- *Nivel 1: Selección de reglas*

  A partir de los conjuntos de reglas de arriba, ART puede escoger cualquiera de los tres pares de reglas anteriores al producirse un empate técnico. Tanto `S1` como `S2` o `S3` tiene un 100 % de confianza y un soporte del 50 %, cubriendo el 50 % del conjunto de entrenamiento y dejando la otra mitad para otro nivel del árbol.

- *Nivel 1: Ramificación del árbol*

  Si se selecciona el primer par de reglas, el primer nivel del árbol quedaría como se muestra a continuación:

	X	*Y*	*Z*	*C*
Cliente	¿desempleado?	¿con fondos?	¿con inmuebles?	¿crédito?
I3	1	1	0	0
I4	1	0	0	0
I5	1	1	1	1
I6	1	0	1	1

Tabla 3.3: El conjunto de datos reducido

```
X = 0 and Y = 0 : C = 0 (2)
X = 0 and Y = 1 : C = 1 (2)
else
  ...
```

Recuérdese que la cantidad entre paréntesis indica el número de ejemplos de entrenamiento cubiertos por cada hoja del árbol. Los ejemplos de entrenamiento pueden eliminarse del conjunto de datos de entrenamiento una vez que han sido cubiertos por las reglas seleccionadas para ramificar el árbol, tal como se muestra en la tabla 3.3. A continuación se construye el siguiente nivel del árbol de decisión.

- *Nivel 2: Extracción de reglas*

  ART busca de nuevo reglas simples que permitan clasificar los datos que quedan en el conjunto de entrenamiento de la tabla 3.3 y encuentra el siguiente conjunto de reglas:

```
S1: if (Z=0) then C=0 with confidence 100%
    if (Z=1) then C=1 with confidence 100%
```

- *Nivel 2: Selección de reglas*

  Al descubrir un conjunto de reglas buenas, ART no necesita formular hipótesis más complejas para construir el segundo nivel del árbol.

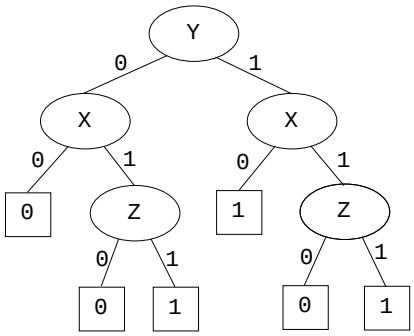

Figura 3.5: Clasificador TDIDT estándar para el conjunto de datos mostrado en la tabla 3.2

- *Nivel 2: Ramificación del árbol*

  Se ramifica el árbol y, como no quedan ejemplos del conjunto de entrenamiento por clasificar, el proceso de aprendizaje del clasificador concluye. El clasificador obtenido por ART es el siguiente:

```
X = 0 and Y = 0 : C = 0 (2)
X = 0 and Y = 1 : C = 1 (2)
else
  Z = 0 : C = 0 (2)
  Z = 1 : C = 1 (2)
```

ART ha logrado un árbol de decisión compacto mientras que cualquier otro algoritmo TDIDT estándar (tal como C4.5) construiría el árbol ligeramente más complejo que se muestra en la figura 3.5. Dicho árbol requeriría una etapa de post-procesamiento para igualar el resultado que ART consigue directamente.

En cualquier caso, en esta sección únicamente se ha presentado un sencillo ejemplo para ilustrar el funcionamiento de ART. Más adelante se discutirán los resultados experimentales obtenidos al aplicar ART en problemas reales.

3.3. Un caso real: SPLICE

Al comienzo de este capítulo se mostró, como adelanto, un ejemplo de clasificador construido por ART a partir del conjunto de datos SPLICE del UCI Machine Learning Repository (véase la página 63). El problema para cuya resolución se recopiló este conjunto de datos consiste en determinar el tipo de uniones de empalme de genes en secuencias de ADN (exon/intron, intron/exon, o ninguna de las dos) dada una secuencia de ADN de primates de 60 nucleótidos en torno a una posible unión de empalme. La siguiente descripción del problema es una traducción directa de la documentación que se puede encontrar en el UCI Machine Learning Repository:

> Las uniones de empalme son puntos en una secuencia de ADN en los cuales el ADN 'superfluo' se elimina durante el proceso de creación de proteínas en los organismos superiores. El problema planteado en este conjunto de datos consiste en reconocer, dada una secuencia de ADN, las fronteras entre exones (las partes de la secuencia de ADN que se conservan tras el empalme) y los intrones (las partes de la secuencia de ADN que quedan fuera del empalme). Este problema consiste en dos subproblemas: reconocer fronteras exon/intron (EI) y reconocer fronteras intron/exon (IE). En Biología, a las fronteras IE se les hace referencia como "receptoras" mientras que se alude a las fronteras EI como "donantes".

En este problema particular, el algoritmo ART construye un clasificador con una precisión bastante buena y un tamaño manejable. Otros algoritmos TDIDT, tal como C4.5, obtienen una precisión similar (entre el 85 y el 90 por ciento utilizando validación cruzada), si bien los árboles de decisión que construyen tienen siempre muchos más nodos y la complejidad adicional de esos árboles de decisión los hace más difíciles de comprender para el usuario final. Utilizando este conjunto de datos, el árbol de decisión construido por C4.5 posee cuatro veces más nodos que el generado por ART.

Si bien es cierto que C4.5 obtiene un porcentaje de clasificación ligeramente superior, ART consigue un modelo de clasificación más sencillo e inte-

	ART	*C4.5*
Precisión del clasificador	87.09 % ± 2.76 %	89.42 % ± 2.02 %
- Clase 1	76.64 %	77.82 %
- Clase 2	90.33 %	89.80 %
- Clase 3	90.41 %	94.60 %
Complejidad del árbol de decisión		
- Hojas	34.1 ± 3.7	143.1 ± 4.3
- Nodos internos	18.2 ± 3.4	62.0 ± 2.3
- Profundidad media	3.81 ± 0.67	3.13 ± 0.03
Tiempo de entrenamiento (en segundos)	84.81 ± 20.4	19.3 ± 0.7

Tabla 3.4: Resultados obtenidos por ART frente a C4.5 utilizando validación cruzada con 10 particiones del conjunto de datos SPLICE.

ligible para el usuario final. A pesar de que el número medio de nodos internos que un ejemplo del conjunto de entrenamiento ha de visitar antes de llegar a un nodo hoja (valor que aparece como profundidad media en la tabla 3.4) es ligeramente mayor en ART porque ART construye árboles de decisión nada balanceados, el número total de nodos internos, y también de hojas, es muy inferior en nuestro modelo al del árbol obtenido con C4.5.

En cuanto al tiempo requerido para construir el clasificador, C4.5 es más eficiente que ART porque no busca relaciones 'complejas' entre los atributos predictivos y el atributo que indica la clase. ART, sin embargo, explota las simetrías existentes en torno a las uniones (que se hallan entre los nucleótidos P30 y P31) para obtener un modelo más compacto Obsérvense, por ejemplo, las parejas P29-P31, P28-P32 y P25-P35 que se utilizan para ramificar el árbol de decisión. Los algoritmos TDIDT típicos no pueden utilizar esas relaciones y, por tanto, su aplicabilidad queda limitada en problemas para los cuales las interconexiones entre variables desempeñan un papel importante.

La tabla 3.4 compara los resultados obtenidos por ART y C4.5 sobre este conjunto de datos utilizando validación cruzada. En la sección 3.6 se pueden encontrar resultados adicionales relativos al conjunto de datos SPLICE, así como un estudio experimental más detallado del comportamiento de ART frente a otros métodos de construcción de clasificadores.

3.4. Notas acerca del clasificador ART

Antes de mostrar los resultados experimentales obtenidos con ART, resulta conveniente discutir algunos aspectos relacionados con el uso de los clasificadores construidos utilizando el modelo descrito en este capítulo: el proceso seguido por ART para clasificar datos, la forma en que ART maneja valores desconocidos y la interpretación del árbol construido por ART como un conjunto de reglas.

3.4.1. Clasificación con ART

El árbol de decisión que se obtiene con ART puede utilizarse para clasificar ejemplos no etiquetados como cualquier otro árbol. Comenzando en la raíz del árbol, un ejemplo dado sigue distintas ramas del árbol dependiendo de los valores de sus atributos. Cualquier ejemplo acabará alcanzando una hoja y será etiquetado con la clase más común en esa hoja.

Por tanto, cuando se nos presenta un caso concreto, se compara ese caso con cada una de las reglas empleadas para construir el nivel actual del árbol de decisión. Si el caso satisface el antecedente de alguna de ellas, entonces lo etiquetamos con la clase que aparece en el consecuente de la regla. Si, por el contrario, el caso no satisface ninguno de los antecedentes de las reglas del nivel actual, seguimos explorando el árbol por la rama 'else'.

Podría darse el caso, incluso, de que un ejemplo no corresponda a ninguna de las hojas del árbol ART porque no verifique el antecedente de ninguna de las reglas del último nivel del árbol. Esta situación puede aparecer cuando las reglas de asociación seleccionadas para ramificar el árbol cubren todos los ejemplos del conjunto de entrenamiento pero no todos los valores posibles de los atributos que aparecen en sus antecedentes. En tal escenario, el ejemplo se etiqueta utilizando la clase más común en el subárbol actual (más concretamente, la clase más común en el conjunto de entrenamiento que dio lugar a dicho subárbol).

El pseudocódigo correspondiente a los pasos seguidos por el método de clasificación ART para clasificar un ejemplo aparece en la figura 3.6.

función clasificar (art, ejemplo): clase;
// *Entrada: art = Árbol de clasificación ART*
// *ejemplo = Ejemplo no etiquetado*
// *Salida: clase = Clase asignada*

Emparejar el/los valor(es) de los atributos del ejemplo con
el/los valor(es) de los atributos utilizados para ramificar el árbol

si el/los valor(es) corresponde(n) a cualquiera
de las ramas que conducen a una hoja
devuelve la clase que etiqueta la hoja correspondiente
en otro caso, si existe una rama 'else' // *seguirla*
devuelve clasificar (art.else, ejemplo);
si no
devuelve la clase por defecto;

Figura 3.6: Clasificación de ejemplos no etiquetados

3.4.2. Manejo de valores nulos

Al construir el clasificador ART, los valores nulos pueden enviarse directamente a la rama 'else', ya que no están cubiertos por ninguna de las reglas de asociación descubiertas. Cuando lo que queremos es clasificar un caso para el cual no está definido un atributo de los usados en el nivel actual del árbol o dicho atributo tiene un valor desconocido, ART aplica la misma técnica que durante la etapa de construcción del árbol: al no poder emparejar el caso con el antecedente de ninguna de las reglas del nivel actual del árbol, se pasa dicho caso a la rama 'else' para buscar reglas aplicables al caso en los niveles inferiores del árbol.

No obstante, hay que mencionar que existen otras alternativas a la hora de clasificar datos que incluyan valores desconocidos:

- El algoritmo C4.5, por ejemplo, aplica una estrategia diferente a la de ART. Si nos encontramos en un nodo interno del árbol cuyo test incluye un atributo para el cual desconocemos su valor, se clasifica el caso actual utilizando todos y cada uno de los subárboles que cuelgan del nodo

interno. C4.5 construye una distribución de probabilidad de acuerdo a la frecuencia relativa de cada clase en los resultados provenientes de la clasificación del caso en cada uno de los subárboles del nodo. De hecho, al carecer de ramas 'else' en las que agrupar los datos que no se sabe muy bien a qué subárbol asignar, C4.5 se ve obligado a 'fraccionar' los casos utilizando pesos proporcionales a la frecuencia de cada rama del árbol (tanto al entrenar el clasificador como al utilizarlo para clasificar). De esta forma, un caso puede corresponder a múltiples caminos en el árbol, lo que dificulta la interpretación del modelo de clasificación cuando aparecen valores desconocidos.

- CART, por su parte, emplea una tercera estrategia que consiste en seleccionar tests alternativos cuya partición del conjunto de entrenamiento sea similar a la del test realizado en un nodo interno del árbol. Si el test original no es aplicable a un caso concreto, CART emplea el test alternativo (el cual, a su vez, puede tener otro test alternativo). Con este método las distintas ramas de un árbol siguen siendo mutuamente exclusivas, si bien puede ser difícil encontrar tests alternativos adecuados. Por desgracia, esta estrategia de búsqueda de tests alternativos es difícilmente aplicable en la construcción de árboles n-arios como los construidos por ART y C4.5. De hecho, en CART es viable porque este algoritmo se restringe a la construcción de árboles de decisión binarios.

La estrategia empleada en ART permite tratar adecuadamente la presencia de valores nulos (desconocidos o inaplicables), sin tener que añadir mecanismos adicionales (como en el caso de C4.5) ni restringir la topología del árbol (tal como sucede en CART), todo ello gracias a la utilización de ramas 'else' en el árbol de decisión.

3.4.3. Conversión del árbol en reglas

La popularidad de los árboles de decisión se debe, en gran medida, a que se pueden derivar reglas IF-THEN a partir de ellos de una forma trivial. Los árboles de decisión construidos por ART, de hecho, se pueden convertir en reglas empleando el mismo método que cualquier otro algoritmo TDIDT. Sólo

hay que tener en cuenta una salvedad: la utilización de ramas ‘else’ causa la aparición de negaciones en los antecedentes de las reglas.

Consideremos de nuevo el ejemplo de la sección 3.2. A partir del árbol obtenido en dicha sección, se puede obtener el siguiente conjunto de reglas:

```
if X=0 and Y=0 then C=0 (2)
if X=0 and Y=1 then C=1 (2)

if not(X=0) and Z=0 then C=0 (2)
if not(X=0) and Z=1 then C=1 (2)
```

Las reglas del conjunto anterior son independientes y mutuamente excluyentes, lo que permite su utilización inmediata en cualquier sistema basado en el conocimiento; esto es, el significado de cada regla es absoluto y no depende de su contexto.

De forma alternativa, se puede interpretar el modelo de clasificación construido por ART como una lista de decisión. En este caso, el árbol quedaría traducido a la siguiente lista ordenada de reglas:

```
1. if X=0 and Y=0 then C=0 (2)

2. if X=0 and Y=1 then C=1 (2)

3. if Z=0 then C=0 (2)

4. if Z=1 then C=1 (2)
```

Las reglas anteriores forman una lista de decisión en la cual el significado de cada regla individual depende de su posición dentro de la secuencia de reglas. Al final de esta secuencia se suele añadir una clase por defecto (para evitar la posibilidad de que un caso no verifique ninguna de las reglas).

La conversión del árbol ART en una lista de decisión nos permite reducir la complejidad de la formulación de las reglas (eliminando las negaciones correspondientes a las ramas ‘else’) a costa de empeorar la inteligibilidad del modelo de clasificación obtenido (pues las reglas hay que interpretarlas en su contexto, lo cual puede resultar difícil si la lista de reglas es larga).

En cualquier caso, ha de destacarse una diferencia de ART con respecto a los algoritmos tradicionales de inducción de listas de decisión: en vez de añadir reglas de una en una, ART añade directamente conjuntos de reglas en cada iteración (los conjuntos correspondientes a cada nivel del árbol de decisión). Este hecho permite que la implementación de ART sea más eficiente que la de otros algoritmos de inducción de listas de decisión.

3.5. Propiedades del clasificador ART

Esta sección complementa a la anterior tratando en profundidad varias propiedades clave del algoritmo ART como método de construcción de modelos de clasificación.

3.5.1. Estrategia de búsqueda

ART construye árboles muy poco balanceados porque intenta clasificar directamente tantos ejemplos como sea posible en cada nivel del árbol de decisión. No obstante, esta característica de ART también ayuda, al menos en parte, a tomar mejores decisiones locales al construir el árbol de decisión.

ART realiza una búsqueda exhaustiva de reglas potencialmente interesantes en el conjunto de datos de entrenamiento, aunque esa búsqueda se restringe al conjunto de datos que queda por clasificar conforme se profundiza en el árbol. De esta forma, la eficiencia, que caracteriza la estrategia de búsqueda greedy heurística de otros algoritmos TDIDT, se combina con la potencia ofrecida por la búsqueda exhaustiva realizada por el algoritmo de extracción de reglas de asociación.

En cierto sentido, ART sigue la metodología STAR de Michalski y sus colaboradores [140, capítulo 3] (igual que CN2 [35] [34]), ya que genera reglas iterativamente hasta que se construye un modelo de clasificación completo. Es digno de mención que, no obstante, ART elimina del conjunto de datos tanto los ejemplos positivos como los ejemplos negativos cubiertos por las reglas seleccionadas para ramificar el árbol, mientras que otros algoritmos STAR han de conservar todos los ejemplos negativos para construir nuevas hipótesis. Además, este hecho conduce a la obtención de reglas más complejas.

Aunque se ha escogido un algoritmo greedy para construir los clasificadores ART, también son factibles otros enfoques. Por ejemplo, una estrategia de búsqueda dirigida podría encontrar mejores clasificadores. Ese tipo de estrategia de búsqueda se utiliza en algoritmos como AQ o CN2, entre otros, los cuales siguen la filosofía de la metodología STAR. No obstante, la eficiencia es estrictamente necesaria en problemas de Data Mining y los algoritmos greedy constituyen la mejor alternativa en estos casos.

3.5.2. Robustez (ruido y claves primarias)

El uso de técnicas propias de la extracción de reglas de asociación ayuda a construir clasificadores más robustos al minimizar los efectos del ruido en los datos de entrada.

El umbral de soporte mínimo hace que sea completamente inofensiva la presencia de datos erróneos aislados (*outliers*), ya que no se tienen en cuenta a la hora de decidir cómo construir el clasificador. La aparición de este tipo de errores, usualmente debidos a problemas durante la adquisición de datos (p.ej. errores en la toma de medidas), es algo bastante común en cualquier ámbito y afecta negativamente a los algoritmos de aprendizaje supervisado. ART, no obstante, funciona adecuadamente incluso en presencia de ruido.

Por otra parte, el umbral de soporte también elimina los problemas que aparecen en otros algoritmos TDIDT cuando algunos atributos predictivos son casi claves (v.g. ID3 siempre escoge ese tipo de atributos para ramificar el árbol porque minimizan la entropía en los subárboles resultantes). Este problema se elimina fácilmente en ART sin necesidad de introducir mecanismos artificiales adicionales (como el criterio de proporción de ganancia en C4.5).

En definitiva, ART trata de una manera uniforme tanto a los outliers (que no contribuyen significativamente en las reglas de asociación) como a las claves primarias o candidatas (cuyos valores no tienen soporte suficiente para generar reglas de asociación por sí mismos).

El árbol construido por ART, a diferencia de los construidos por otros algoritmos TDIDT, no requiere poda alguna a posteriori porque los outliers no degradan su rendimiento. De hecho, los algoritmos de extracción de reglas de asociación funcionan perfectamente con datos que incluyen ruido.

3.5.3. Complejidad del árbol

Tanto la altura como la anchura del árbol de decisión construido por ART están restringidas por el umbral de soporte mínimo.

Dado un umbral absoluto de soporte mínimo *MinSupp* como un valor normalizado entre 0 y 1, el árbol de decisión construido por ART tendrá como máximo *1/MinSupp* niveles, porque en cada nivel se clasificarán al menos *#D*MinSupp* ejemplos, siendo *#D* el número de ejemplos en el conjunto de datos de entrenamiento.

Cuando se utiliza un umbral absoluto de soporte mínimo, nunca serán necesarias más de *MaxSize*(1/MinSupp)* pasadas sobre el conjunto de datos de entrada para construir el clasificador completo; ya que todas las reglas de asociación pueden obtenerse con, como mucho, *MaxSize* recorridos sobre el conjunto de datos de entrenamiento utilizando algoritmos como Apriori. En otras palabras, ART es *O(n)* sobre el tamaño del conjunto de datos de entrenamiento. Esta cualidad es esencial para realizar tareas de Data Mining.

El factor de ramificación del árbol también lo determina el umbral de soporte mínimo. *1/MinSupp* es una cota superior del factor de ramificación del árbol en cada nivel, independientemente de si el umbral de soporte *MinSupp* es absoluto o relativo, porque no puede haber más de *1/MinSupp* reglas de asociación seleccionadas simultáneamente. El caso extremo ocurre cuando todos los ejemplos del conjunto de entrenamiento se clasifican en el mismo nivel del árbol utilizando la máxima cantidad de reglas de asociación, todas ellas con el soporte mínimo permitido.

3.6. Resultados experimentales

Se ha implementado ART en el lenguaje de programación Java 2 utilizando el kit de desarrollo de Sun Microsystems. Durante los experimentos también se empleó AspectJ [90], una extensión “orientada a aspectos” del lenguaje Java. El uso de aspectos permite monitorizar la ejecución de un algoritmo, lo que resulta especialmente útil para tomar algunas medidas (en concreto, las relacionadas con la realización de operaciones de entrada/salida).

Para acceder a los datos, que se pueden almacenar en cualquier base de datos relacional, se emplea JDBC [Java Database Connectivity], el interfaz estándar de acceso a bases de datos de Java.

Para la implementación de ART se ha utilizado el algoritmo TBAR [19], un algoritmo de extracción de reglas de asociación de la familia de Apriori [7]. TBAR será objeto de estudio en el siguiente capítulo de esta memoria.

Los experimentos se realizaron en un PC con microprocesador Intel Pentium III a 800MHz con 128 MB de memoria RAM sobre el sistema operativo Windows NT 4.0 WorkStation de Microsoft. La base de datos a la que se accedía era InterBase 6, aunque cualquier otra base de datos podría haber sido utilizada dada la portabilidad de JDBC. De hecho, también se ha probado ART con datos almacenados en Oracle 8i e IBM DB2 UDB.

La tabla 3.5 recoge los conjuntos de datos utilizados en nuestros experimentos. Dichos conjuntos de datos se pueden conseguir gratuitamente accediendo al servidor web de la Universidad de California en Irvine. En concreto, se obtuvieron del UCI Machine Learning Repository, al cual se puede acceder a través de la siguiente URL:

http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Todos los resultados presentados en esta sección se han obtenido utilizando validación cruzada con 10 particiones sobre cada uno de los conjuntos de datos utilizados.

3.6.1. Precisión

La tabla 3.6 compara el rendimiento de ART frente a varios clasificadores ampliamente utilizados. En esta tabla, que recoge los porcentajes de clasificación obtenidos por distintos clasificadores, se agrupan los conjuntos de datos en función de su tamaño. Además, en la tabla se compara ART con cada uno de los demás clasificadores utilizados en los experimentos[*].

[*] Las ternas G-P-E indican el número de veces en que el porcentaje de clasificación obtenido con ART es mejor (G), el número de ocasiones en que ART es peor (P) y el número de conjuntos de datos en que la precisión de ART es similar (E). El comportamiento de dos clasificadores se considera similar cuando la diferencia entre ambos no llega al 1 %.

Conjunto de datos	*Tuplas*	*Atributos*	*Clases*
AUDIOLOGY	226	70	24
CAR	1728	7	4
CHESS	3196	36	2
HAYES-ROTH	160	5	3
LENSES	24	6	3
LUNG CANCER	32	57	3
MUSHROOM	8124	23	2
NURSERY	12960	9	5
SOYBEAN	683	36	19
SPLICE	3175	61	3
TICTACTOE	958	10	2
TITANIC	2201	4	2
VOTE	435	17	2

Tabla 3.5: Conjuntos de datos utilizados en los experimentos.

En estos experimentos se utilizó ART con selección automática del umbral de confianza (tal como se describe en la sección 3.1.3), un umbral de soporte del 5 % para las reglas de asociación (*MinSupp=5 %*) y una cota superior para el tamaño de los antecedentes de las reglas igual a 3 (*MaxSize=3*).

También se muestran en la tabla los resultados conseguidos por C4.5 [131], utilizando el criterio de proporción de ganancia como regla de división y poda pesimista (con CF=0.25). Las siguientes tres columnas incluyen resultados obtenidos con algoritmos basados en la metodología STAR de Michalski y sus colaboradores: AQR (una variante del conocido algoritmo AQ) y dos versiones del algoritmo CN2. La primera de ellas, CN2-STAR [34], obtiene un conjunto de reglas (como AQR), mientras que la segunda versión, CN2-DL [35], obtiene una lista de decisión, como los algoritmos IREP [64] y RIPPER [38] que aparecen a continuación. La tabla de resultados se completa con el clasificador Naive Bayes y el clasificador por defecto que resulta de asignar siempre la clase más común en el conjunto de entrenamiento, el cual se incluye a título meramente informativo.

Tamaño	*Conjunto de datos*	*ART*	*C4.5*	*AQR*	*CN2 (STAR)*	*CN2 (DL)*	*IREP*	*RIPPER k=2*	*Naive Bayes*	*Clasificador por defecto*
>1000	NURSERY	99.11 %	96.17 %	91.70 %	98.10 %	98.99 %	89.40 %	33.33 %	88.05 %	33.33 %
	MUSHROOM	98.52 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	99.95 %	94.31 %	51.80 %
	SPLICE	89.32 %	94.08 %	79.21 %	92.32 %	90.21 %	90.99 %	67.21 %	51.91 %	51.91 %
	CHESS	97.69 %	99.22 %	97.06 %	99.37 %	99.27 %	99.22 %	98.59 %	62.48 %	52.22 %
	TITANIC	78.56 %	79.05 %	67.70 %	75.78 %	79.05 %	78.33 %	70.42 %	68.01 %	67.70 %
	CAR	98.55 %	92.88 %	85.07 %	93.92 %	95.78 %	74.89 %	70.02 %	70.02 %	70.02 %
	Media	93.63 %	93.57 %	86.79 %	93.25 %	93.88 %	89.47 %	73.25 %	72.46 %	54.50 %
	Resultados		2-3-1	4-1-1	3-3-0	1-2-3	2-3-1	4-1-1	6-0-0	6-0-0
<1000	TICTACTOE	81.62 %	83.82 %	82.26 %	98.02 %	92.37 %	97.60 %	93.64 %	65.34 %	65.34 %
	SOYBEAN	91.52 %	93.70 %	83.17 %	92.09 %	93.41 %	88.15 %	14.35 %	58.71 %	13.18 %
	VOTE	95.86 %	95.86 %	93.32 %	94.71 %	92.19 %	94.02 %	91.71 %	89.19 %	61.36 %
	AUDIOLOGY	65.21 %	81.36 %	66.36 %	75.57 %	80.06 %	65.37 %	25.14 %	23.40 %	25.14 %
	HAYES-ROTH	84.38 %	73.75 %	65.00 %	76.25 %	76.25 %	68.75 %	48.75 %	61.88 %	29.38 %
	LUNG CANCER	40.83 %	43.33 %	25.83 %	39.17 %	35.83 %	35.00 %	31.67 %	43.33 %	40.00 %
	LENSES	70.00 %	81.67 %	65.00 %	76.67 %	73.33 %	56.67 %	63.33 %	63.33 %	63.33 %
	Media	75.63 %	79.07 %	68.71 %	78.93 %	77.63 %	72.22 %	52.66 %	57.88 %	42.53 %
	Resultados		1-5-1	5-1-1	3-3-1	3-4-0	5-1-1	6-1-0	6-1-0	6-0-1
Media		83.94 %	85.76 %	77.05 %	85.54 %	85.13 %	80.18 %	62.16 %	64.61 %	48.05 %
Mejor-Peor-Igual (1 %)			3-8-2	9-2-2	6-6-1	4-6-3	7-4-2	10-2-1	12-1-0	12-0-1

Tabla 3.6: Precisión del clasificador ART y de otros conocidos métodos de clasificación.

Aunque no aparecen recogidas en la tabla 3.6, también se realizaron pruebas con otros clasificadores TDIDT (utilizando distintas reglas de división) y algunas variantes del clasificador Naive Bayes (usando las leyes de Laplace o de Lidstone para estimar probabilidades). Dichos experimentos obtuvieron resultados similares a los ya mostrados, por lo cual se omiten en la tabla 3.6.

En los experimentos mostrados en la tabla 3.6 se utilizó una heurística de selección automática del umbral de confianza *MinConf* porque se ha observado experimentalmente que dicha heurística obtiene resultados cercanos al mejor resultado obtenido cuando se establece dicho umbral realizando una batería de pruebas (tal como se verá en la sección 3.6.4). De hecho, cuando se emplea un umbral de confianza preestablecido, aumentar su valor no implica necesariamente mejorar la precisión del clasificador.

Tal como se recoge en la tabla 3.6 y se muestra en la figura 3.7 de la página 97, la precisión del clasificador ART es comparable a la que obtienen otros clasificadores, especialmente en conjuntos de datos de cierto tamaño (como NURSERY o CAR). Sin embargo, el rendimiento de ART empeora ligeramente cuando los conjuntos de datos son pequeños (AUDIOLOGY o LENSES) debido a la propia naturaleza del proceso de extracción de reglas de asociación, el cual está basado en la obtención eficiente de patrones frecuentes en grandes conjuntos de datos.

En líneas generales, no obstante, ART mantiene los porcentajes de clasificación obtenidos por otros clasificadores como C4.5 o CN2, mejorando significativamente los resultados conseguidos con otros métodos (AQR, RIPPER, Naive Bayes y clasificador por defecto), como se recoge en la tabla 3.7.

La tabla 3.7 recoge los resultados obtenidos al realizar tests estadísticos para comparar la precisión obtenida por ART con los resultados logrados por los demás clasificadores analizados. Dado que el test t de Student, habitualmente empleado para comparar clasificadores, exhibe una probabilidad elevada de error de tipo I (esto es, detecta diferencias significativas no existentes en realidad) y asume que las diferencias existentes están normalmente distribuidas (algo que no podemos asegurar en nuestros experimentos), la tabla 3.7 incluye también los resultados obtenidos al realizar un test de Wilcoxon sobre los resultados recogidos en la tabla 3.6. Dicho test no presupone la normalidad de la

	Test t de Student	*Test de Wilcoxon*
ART vs. C4.5	$p \leq 0{,}3507$	$p \leq 0{,}3475$
ART vs. AQR	$p \leq 0{,}0032$ ++	$p \leq 0{,}0100$ ++
ART vs. CN2-STAR	$p \leq 0{,}3901$	$p \leq 0{,}5405$
ART vs. CN2-DL	$p \leq 0{,}3942$	$p \leq 0{,}5860$
ART vs. IREP	$p \leq 0{,}1645$	$p \leq 0{,}1548$
ART vs. RIPPER	$p \leq 0{,}0128$ +	$p \leq 0{,}0100$ ++
ART vs. Naive Bayes	$p \leq 0{,}0004$ ++	$p \leq 0{,}0100$ ++
ART vs. por defecto	$p \leq 0{,}0001$ ++	$p \leq 0{,}0100$ ++

Tabla 3.7: Tests estadísticos realizados para comprobar si las diferencias apreciadas en la precisión de los distintos clasificadores son significativas (+, $p \leq 0{,}05$) o estadísticamente significativas (++, $p \leq 0{,}01$).

población (técnica no paramétrica).

Como es lógico, ART es significativamente mejor que el clasificador por defecto. Además, ART es consistentemente mejor que AQR y Naive Bayes. Así mismo, ART mejora los resultados obtenidos por RIPPER porque este último no se comporta adecuadamente con algunos de los conjuntos de datos utilizados en los experimentos. Respecto a los demás métodos, ART consigue resultados similares en cuanto a la precisión de los modelos de clasificación construidos, si bien a continuación se verá cómo el modelo de clasificación ART propuesto en esta memoria es mejor que otros modelos en aspectos tales como la eficiencia del proceso de construcción del clasificador (apartado 3.6.2) o la complejidad de los modelos de clasificación construidos (apartado 3.6.3).

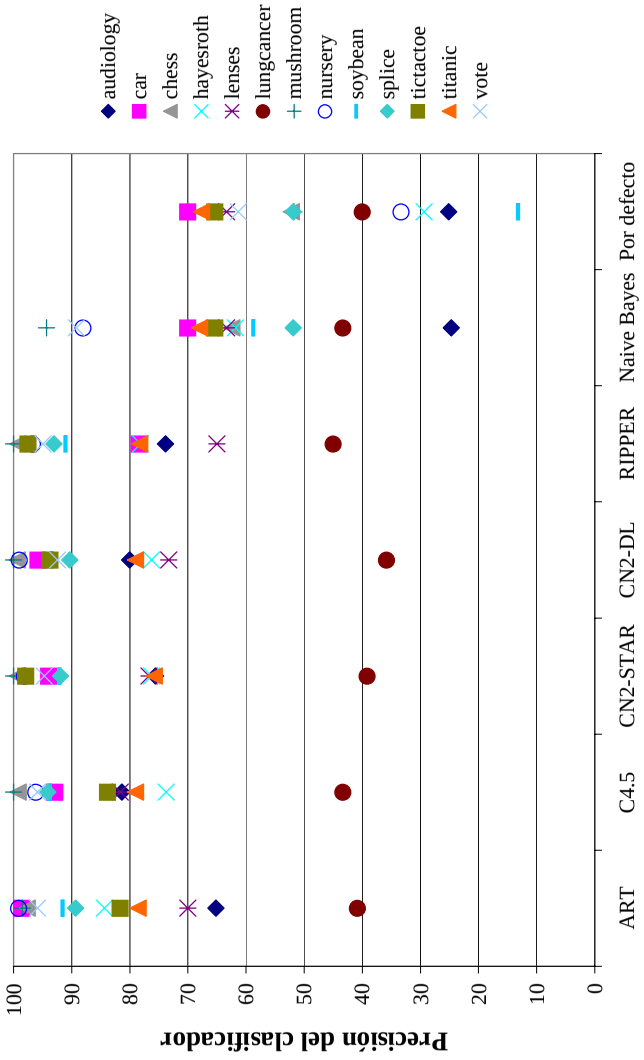

Figura 3.7: Precisión del clasificador ART frente a otros clasificadores comunes.

3.6.2. Eficiencia

La tabla 3.8 muestra el tiempo empleado por cada uno de los métodos de construcción de clasificadores utilizados en el apartado anterior. La tabla recoge los tiempos de entrenamiento para algunos de los conjuntos de datos de la tabla 3.5. En concreto, se muestran los resultados obtenidos con los seis conjuntos de datos de la tabla 3.5 que contienen más de mil tuplas.

Con el objetivo de hacer más notorias las diferencias existentes entre los distintos algoritmos de construcción de clasificadores, también se ha realizado una serie de experimentos en los cuales los clasificadores se construyen trabajando directamente sobre ficheros (sin almacenar los datos en memoria principal), de forma que se fuerza un acceso a disco cada vez que se accede al conjunto de datos de entrenamiento y se refleja de una forma más fidedigna el comportamiento de los distintos algoritmos en la resolución de problemas de *Data Mining* cuando los conjuntos de datos son demasiado grandes para caber en memoria principal. Los resultados obtenidos de este modo se resumen en la tabla 3.9.

De hecho, cuando los conjuntos de datos no caben en memoria principal, los algoritmos de inducción de reglas necesitan utilizar alguna técnica de muestreo para seguir siendo competitivos con ART y con la implementación de C4.5 que se ha utilizado en esta memoria.

La figura 3.8 muestra gráficamente los tiempos de entrenamiento correspondientes a todos los conjuntos de datos de la tabla 3.5. El clasificador Naive Bayes se incluye en esta figura para que sirva de referencia a los demás métodos de clasificación. Dicho clasificador se construye recorriendo secuencialmente el conjunto de datos una sola vez, por lo que su eficiencia se puede considerar óptima.

Nuestra implementación de C4.5, similar a RainForest [69], resulta especialmente adecuada para trabajar con grandes conjuntos de datos. Dicha implementación sólo requiere recorrer el conjunto de datos de entrenamiento dos veces en cada nodo interno del árbol y una sola vez en sus nodos hoja. El primer recorrido sirve para obtener la frecuencia de cada valor de los distintos atributos de los casos del conjunto de entrenamiento y la frecuencia de cada

Algoritmo	*NURSERY*	*MUSHROOM*	*SPLICE*	*CHESS*	*TITANIC*	*CAR*
C4.5	0.8s	0.8s	0.9s	0.9s	0.1s	0.2s
CN2-DL	15.9s	1.7s	23.7s	1.4s	0.2s	0.3s
IREP	61s	4.2s	19.7s	2.6s	0.2s	0.5s
AQR	63s	1.8s	64s	3.3s	0.1s	1.1s
ART	6.2s	1.6s	188s	6.8s	1.1s	4.8s
RIPPER	236s	7.0s	39.5s	4.6s	0.3s	1.5s
CN2-STAR	310s	8.0s	217s	14.1s	0.2s	4.2s

Tabla 3.8: Tiempo de entrenamiento requerido por distintos algoritmos de construcción de clasificadores.

Algoritmo	*NURSERY*	*MUSHROOM*	*SPLICE*	*CHESS*	*TITANIC*	*CAR*
C4.5	17s	4s	9s	7s	1.2s	4s
ART	101s	13s	605s	57s	4.7s	13s
RIPPER	45s	719s	3062s	819s	6.8s	9s
IREP	5634s	415s	4389s	505s	12.0s	101s
CN2-DL	1743s	129s	4710s	155s	12.7s	51s
AQR	5906s	112s	12297s	403s	0.6s	223s
CN2-STAR	29552s	836s	29257s	4528s	21.5s	1023s

Tabla 3.9: Tiempo de entrenamiento requerido por los distintos clasificadores cuando se trabaja directamente sobre disco.

clase para cada uno de esos valores (esto es, los conjuntos atributo-valor-clase utilizados en RainForest). Ésta es la única información que requiere el algoritmo TDIDT para ramificar el árbol de decisión. Cuando se decide no seguir ramificando el árbol (llegamos a un nodo hoja) no es necesario acceder más al conjunto de datos de entrenamiento, ya que simplemente se etiqueta la hoja con la clase más común en los datos de entrenamiento. Si, por el contrario, hay que ramificar el árbol de decisión, es necesario recorrer de nuevo el conjunto de entrenamiento para distribuir los casos de entrenamiento entre los distintos subárboles generados.

En cuanto al clasificador ART, hay que resaltar que también es muy eficiente cuando se trabaja sobre grandes conjuntos de datos, ya que ART sólo requiere recorrer, en el peor de los casos, *MaxSize*+1 veces el conjunto de datos de entrenamiento para decidir qué reglas se utilizan en la ramificación de cada nivel del árbol. El número de veces que se recorren secuencialmente los datos de entrenamiento viene dado por la implementación del proceso de extracción de reglas de asociación empleado para obtener hipótesis candidatas. Este proceso será estudiado con detalle en el capítulo siguiente de esta memoria.

Cuando el volumen de datos del conjunto de entrenamiento es elevado, la estrategia de búsqueda de ART es mucho más adecuada que la empleada por algoritmos anteriores de inducción de reglas para resolver problemas de extracción de conocimiento en bases de datos. Estos algoritmos, por lo general, suelen requerir un recorrido completo del conjunto de datos cada vez que se formula una hipótesis. Aunque la tabla 3.8 no refleja este hecho en toda su magnitud porque los conjuntos de datos utilizados caben perfectamente en memoria principal, hay que tener en cuenta que algoritmos como CN2 o RIPPER realizan un recorrido secuencial del conjunto de entrenamiento cada vez que se evalúa una regla para ver si resulta adecuado añadirla al modelo de clasificación correspondiente. Por su parte, ART realiza un máximo de *MaxSize*+1 recorridos sobre el conjunto de entrenamiento, durante los cuales obtiene en paralelo todas las reglas candidatas que serán consideradas para formar parte del modelo de clasificación.

Cuando el conjunto de datos de entrenamiento no se puede cargar en memoria principal, por su tamaño o por restricciones de espacio en un servidor

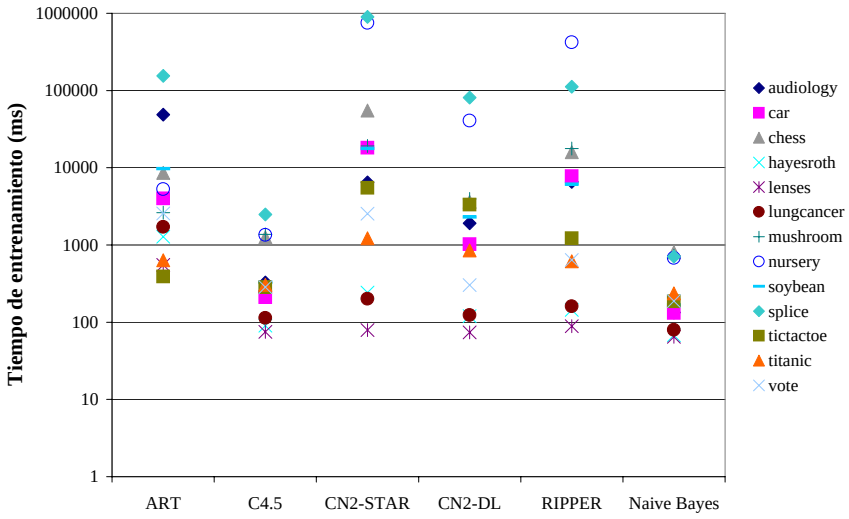

Figura 3.8: Tiempo de entrenamiento requerido para la construcción de los distintos clasificadores.

que ha de dar servicio a muchos clientes diferentes, ART puede llegar a ser uno o varios órdenes de magnitud más eficiente que los algoritmos AQR, CN2, IREP y RIPPER. En tales situaciones, estos últimos requieren la utilización de alguna técnica de muestreo porque, a diferencia de ART, no son escalables.

Las diferencias existentes entre las distintas técnicas se deben principalmente al ancho de banda requerido por las operaciones de E/S, el verdadero cuello de botella de los algoritmos de aprendizaje en KDD. En la figura 3.9 aparece el número de operaciones de entrada/salida necesarias para la construcción de cada tipo de clasificador[**].

[**]En realidad, las operaciones de E/S son únicamente de entrada, pues consisten exclusivamente en recorridos secuenciales del conjunto de entrenamiento. La gráfica de la parte superior muestra el número de veces que se recorre secuencialmente el conjunto de entrenamiento (o una parte de él cuando avanzamos en la construcción del modelo de clasificación). En la gráfica de debajo aparece el número de veces que se accede a una tupla del conjunto de entrenamiento.

Las dos gráficas que aparecen en la figura 3.9 ilustran claramente a lo que nos referimos cuando decimos que el algoritmo ART destaca por su escalabilidad.

3.6.3. Complejidad

Si bien el tiempo de entrenamiento requerido por ART es superior al necesario para construir un árbol de decisión utilizando un algoritmo TDIDT clásico (al realizar ART una búsqueda en un espacio se soluciones mayor), los modelos de clasificación obtenidos con ART tienden a ser más compactos que los frondosos árboles de decisión construidos por algoritmos como C4.5, incluso después de podarlos. La búsqueda adicional que realiza ART es precisamente la que le permite obtener modelos de clasificación más simples, que son potencialmente más fáciles de entender para el usuario.

La complejidad de los clasificadores obtenidos utilizando los distintos métodos de clasificación vistos se muestra en la figura 3.10. En esta gráfica aparece representado, en escala logarítmica, el número de reglas de cada clasificador como medida de su complejidad. En el caso de los árboles de decisión construidos por C4.5, se muestra su número hojas, ya que de cada hoja de un árbol de decisión se deriva una regla (como se vio en el apartado 2.1.5).

Los clasificadores más complejos son los generados por los algoritmos de inducción de reglas basados en la metodología STAR de Michalski; es decir, AQR y CN2-STAR. En el extremo opuesto se hallan los algoritmos de inducción de listas de decisión, CN2-DL y RIPPER, si bien hay que tener en cuenta que el significado de las reglas en una lista de decisión depende de su posición, lo que puede dificultar la interpretación del modelo construido. En el caso de los algoritmos TDIDT de construcción de árboles de decisión (C4.5), su complejidad en número de hojas se halla a medio camino entre las listas de decisión y los algoritmos STAR.

Tal como se aprecia en la figura 3.10, el algoritmo ART, a medio camino entre las listas y los árboles de decisión, destaca por conseguir modelos de clasificación de complejidad comparable o incluso menor que la de cualquier otro de los métodos analizados.

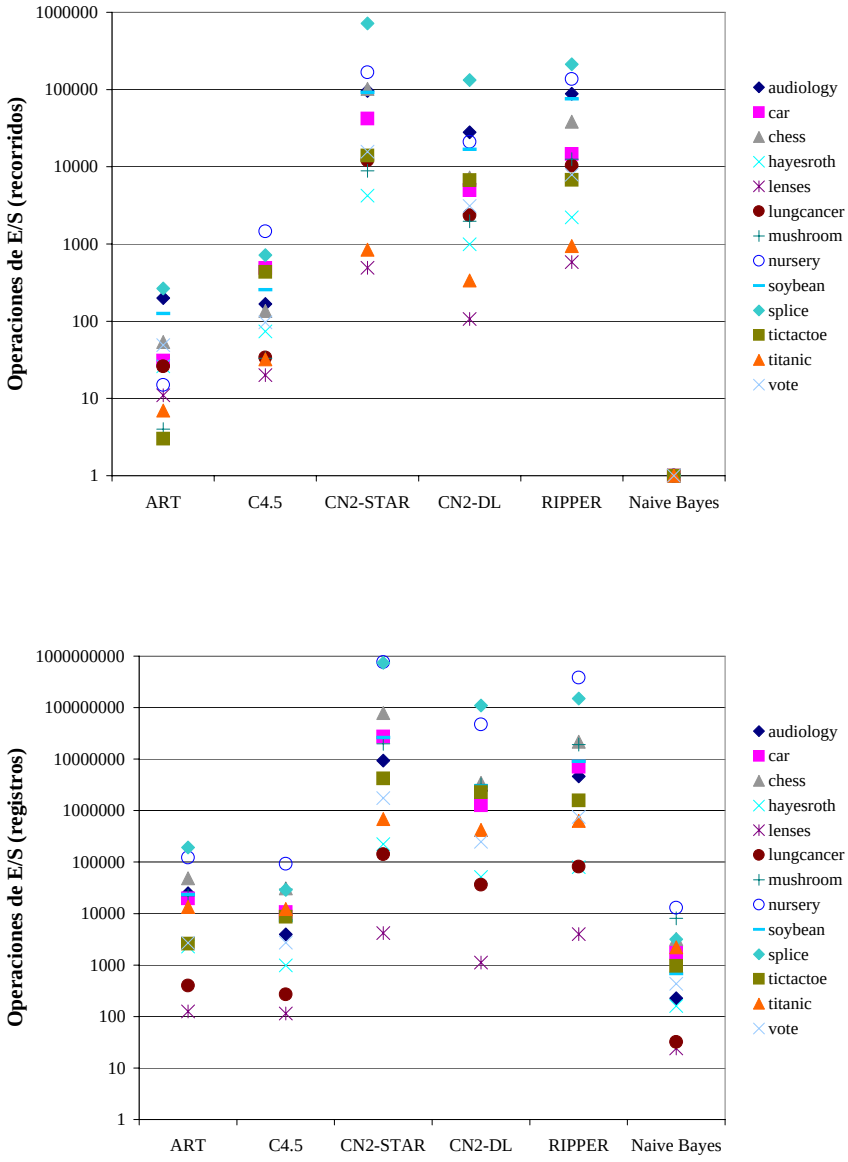

Figura 3.9: Operaciones de E/S requeridas para la construcción de los distintos clasificadores.

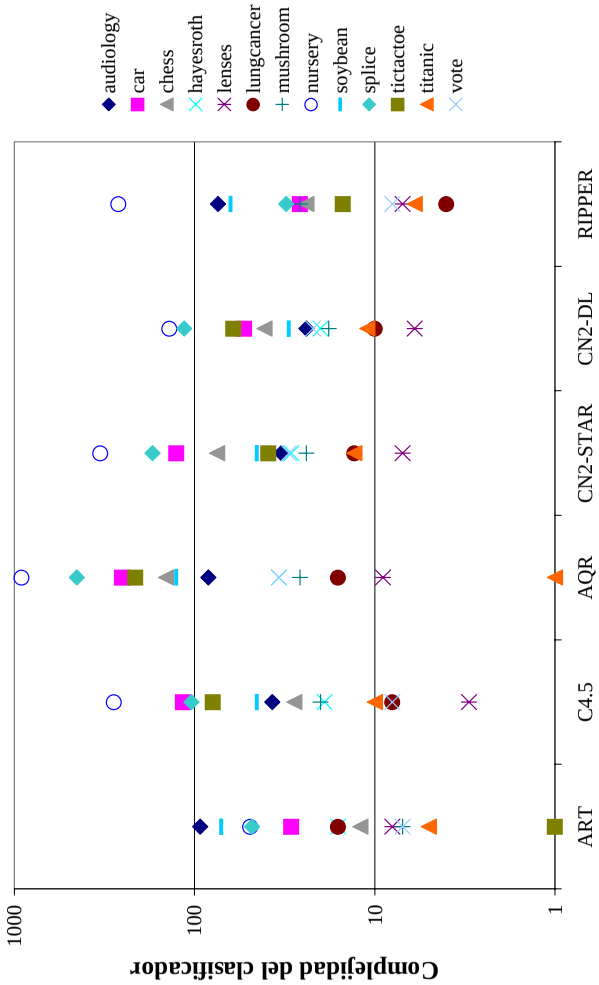

Figura 3.10: Complejidad media de los distintos clasificadores.

3.6.4. El umbral de confianza

En este apartado se resumen los resultados obtenidos en una serie de experimentos realizados con la intención de justificar el uso de una heurística que permita seleccionar automáticamente el umbral de confianza involucrado en el proceso de extracción de reglas de asociación utilizado por ART para construir hipótesis candidatas. Al emplear esta heurística, el usuario no ha de preocuparse en establecer el valor de este parámetro, salvo que lo estime oportuno para resolver su problema de clasificación adecuadamente (tal como se discutió en el apartado 3.1.3).

La tabla 3.10 resume los resultados obtenidos por ART sobre algunos de los conjuntos de datos empleados en los experimentos de las secciones anteriores. En la tabla se muestra el porcentaje de clasificación obtenido para cada conjunto de datos variando los umbrales de soporte y confianza utilizados para obtener las reglas de asociación que sirven como base del modelo de clasificación ART.

Dicha tabla muestra, además, cómo incrementar el umbral de confianza mínimo no implica necesariamente la obtención de un mejor porcentaje de clasificación. Si bien es cierto que este umbral puede utilizarse para establecer una precisión mínima deseada para las reglas extraídas del conjunto de entrenamiento, esta característica no es aplicable al conjunto de prueba con el que se mide la precisión del clasificador.

Los resultados obtenidos con el conjunto de datos NURSERY muestran que las variaciones en los parámetros de ART no siempre modifican los resultados obtenidos. Al trabajar con este conjunto de datos, llega un momento en el cual la precisión del clasificador queda estancada en el 66.25 %, cuando se logran clasificar perfectamente dos de las cinco clases del problema. Las otras tres ni tan siquiera aparecen en los modelos de clasificación construidos por ART, ya que no son lo suficientemente frecuentes como para aparecer en el consecuente de ninguna regla de asociación.

Los resultados obtenidos con la heurística de selección automática del umbral, utilizada en los experimentos mostrados en la tabla 3.6, suelen encontrarse cercanos al mejor valor obtenido realizando una serie completa de experi-

MinSupp	*MinConf*	*VOTE*	*SOYBEAN*	*NURSERY*
0.0	*0.7*	95.62 %	81.56 %	86.93 %
	0.8	95.39 %	84.78 %	**91.04 %**
	0.9	95.61 %	90.20 %	73.10 %
0.1	*0.7*	95.62 %	85.07 %	66.25 %
	0.8	95.39 %	85.37 %	66.25 %
	0.9	95.61 %	90.64 %	66.25 %
0.2	*0.7*	95.62 %	**91.37 %**	66.25 %
	0.8	95.62 %	**91.37 %**	66.25 %
	0.9	**95.85 %**	**91.37 %**	66.25 %
0.3	*0.7*	88.50 %	57.07 %	66.25 %
	0.8	88.74 %	57.07 %	66.25 %
	0.9	88.74 %	57.07 %	66.25 %

Tabla 3.10: Resultados obtenidos con ART variando los umbrales de soporte y confianza. Cuando aparece *0.0* como umbral de soporte mínimo, este umbral indica que se tendrán en cuenta, al construir el árbol de decisión, todas las reglas de asociación posibles (incluso aunque su soporte se limite a una única tupla en el conjunto de entrenamiento).

mentos con distintos valores para el umbral de confianza, lo que confirma que la heurística utilizada de selección automática del umbral resulta adecuada. En la práctica, resulta incluso preferible de cara al usuario, pues al utilizar dicho mecanismo automático se obtienen buenos resultados y se ahorra bastante tiempo (el tiempo necesario para ajustar los parámetros del clasificador[***]).

3.6.5. El umbral de soporte

Respecto al umbral de soporte mínimo, la tabla 3.10 también nos muestra que disminuir el valor del umbral de confianza no implica que ART consiga mejores porcentajes de clasificación, tal como indican los resultados obtenidos con los conjuntos de datos VOTE y SOYBEAN.

[***] Esta experimentación puede convertirse una ardua tarea no sólo para ART, sino para cualquier otro modelo de clasificación.

MinSupp	1 %	2.5 %	5 %	7.5 %	10 %	20 %
Precisión (10-CV)	77.18 %	79.04 %	79.22 %	80.86 %	81.08 %	80.92 %
Tiempo de entrenamiento	17.9s	21.8s	18.5s	13.9s	8.0s	5.6s
Topología del árbol						
- Hojas del árbol	36.8	44.1	37.0	35.9	27.8	19.6
- Nodos internos	19.0	20.4	18.2	17.5	14.2	11.2
- Profundidad media	8.08	7.88	7.41	6.88	6.28	5.46
Operaciones de E/S						
- Registros	38900	41400	36400	34800	28700	18200
- Recorridos	65.7	70.9	62.8	59.8	47.4	35.4

Tabla 3.11: Resumen de los experimentos realizados para distintos valores del umbral de soporte.

El parámetro *MinSupp* nos ayuda a establecer el nivel de granularidad que mejor se adapte a nuestro problema y, además, permite acotar el tiempo necesario para construir el clasificador, como se puede deducir de los resultados obtenidos empíricamente que se resumen en la tabla 3.11. En la figura 3.11 se muestra gráficamente la evolución de la precisión del clasificador ART conforme se varía el umbral de soporte mínimo *MinSupp*, mientras que la figura 3.12 recoge la complejidad de los árboles construidos y los gráficos de la figura 3.13 hacen referencia al coste del proceso de construcción del clasificador.

Un valor bajo del umbral de soporte mínimo tiende a favorecer el sobreaprendizaje y la construcción de clasificadores más complejos de lo necesario. Además, cuanto más bajo sea el umbral de soporte mínimo, la construcción del clasificador será más costosa en términos computacionales. Un umbral excesivamente elevado, en cambio, puede conducir a la construcción de un modelo de clasificación cuya precisión no alcance niveles aceptables al no considerarse lo suficientemente frecuentes algunos patrones potencialmente interesantes, especialmente si se utiliza la heurística de selección automática del umbral de confianza descrita en la sección 3.1.3.

Es cierto, no obstante, que, además de restringir el tiempo necesario para la construcción del clasificador, el umbral de soporte mínimo *MinSupp* nos ayuda a establecer el nivel deseado de granularidad de nuestro clasificador.

Para concluir esta sección, sólo falta mencionar el papel desempeñado por el tercer parámetro de ART, *MaxSize*, que también influye en el tiempo de entrenamiento necesario para construir el clasificador. Si utilizásemos *MaxSize=1* conseguiríamos un clasificador ART tan eficiente como cualquier otro clasificador TDIDT, si bien eliminaríamos de ART la capacidad de emplear combinaciones de atributos para ramificar el árbol de decisión. Por otro lado, cuando el valor *MaxSize* es mayor que 3, el rendimiento de ART podría deteriorarse ya que el número de patrones frecuentes presentes en un conjunto de datos puede aumentar exponencialmente con el tamaño de los patrones. Este hecho, especialmente preocupante en conjuntos de datos densos como los utilizados habitualmente para construir clasificadores, dio razón de ser al algoritmo TBAR, que se describirá detalladamente en el capítulo siguiente de esta memoria.

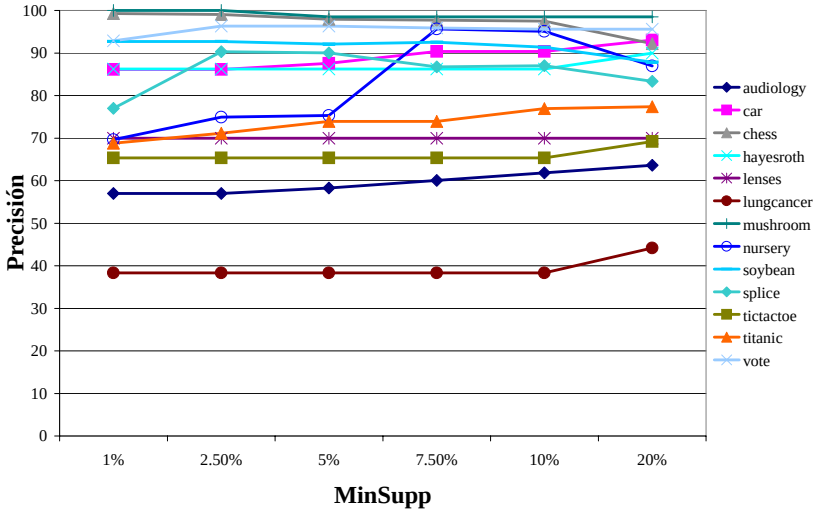

Figura 3.11: Variación de la precisión del clasificador ART en función del umbral de soporte mínimo.

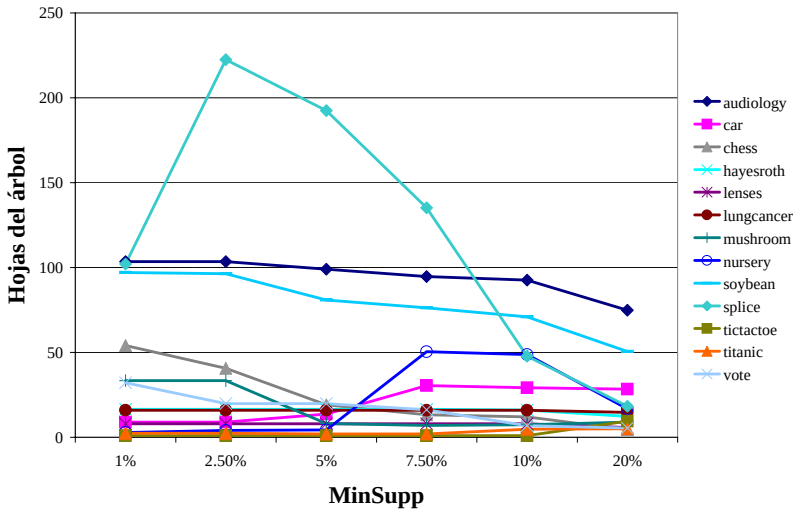

Figura 3.12: Variación del número de hojas del árbol ART en función del umbral de soporte mínimo.

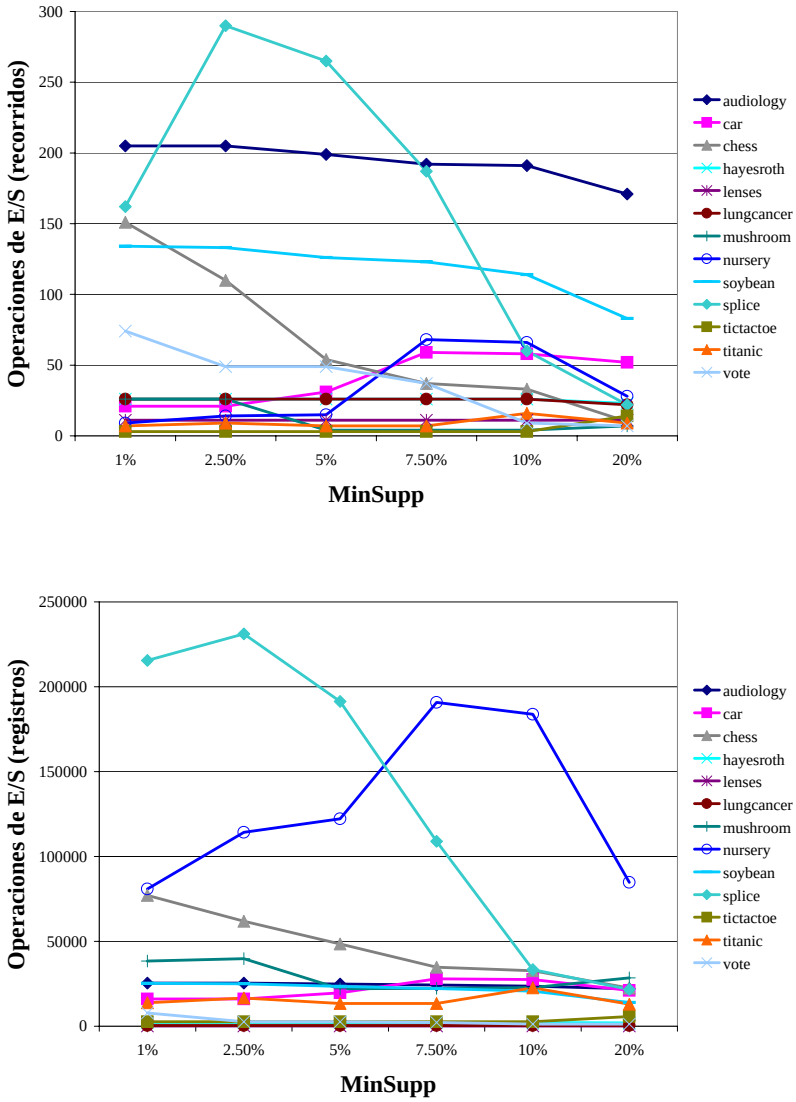

Figura 3.13: Variación del número de operaciones de E/S realizadas en la construcción de ART en función del umbral de soporte mínimo.

En resumen, las restricciones impuestas por los parámetros *MaxSupp* y *MaxSize* sobre el proceso de búsqueda realizado por ART permiten que éste pueda utilizarse eficientemente con grandes conjuntos de datos. Ambos parámetros acotan el espacio de búsqueda explorado por ART, haciéndolo competitivo con otros algoritmos de inducción de reglas. Resulta recomendable, pues, llegar a un compromiso a través del valor de estos umbrales entre el espacio de búsqueda explorado y el tiempo de entrenamiento requerido para construir el modelo de clasificación.

3.6.6. Otros experimentos

Aparte del mecanismo de selección automática del umbral de confianza y del método de selección de reglas descritos en las secciones anteriores de este capítulo, también se han realizado otros experimentos en los que se evalúan algunas heurísticas alternativas.

Entre las opciones analizadas relativas al ajuste automático de los parámetros empleados por ART, se realizaron algunas pruebas en las que se utilizaba un umbral de confianza adaptativo independiente para cada clase del problema. Esta técnica se ideó para intentar evitar que ART ignorase clases muy poco frecuentes en el conjunto de entrenamiento, aunque no se consiguieron mejoras notables a nivel global. Obviamente, si las clases son verdaderamente poco frecuentes, la precisión del clasificador no se verá afectada.

Tampoco se consiguieron mejoras destacables al utilizar métodos heurísticos más complejos para evaluar las reglas candidatas durante el proceso de construcción del clasificador (recuérdese el proceso de selección de reglas realizado a la hora de ramificar el árbol de decisión, sección 3.1.3). Una alternativa posible consiste, por ejemplo, en medir la divergencia Hellinger [98] asociada a las reglas candidatas en vez de tomar simplemente el número de ejemplos del conjunto de datos de entrada correctamente clasificados.

A pesar de no haberse conseguido resultados positivos hasta ahora con heurísticas alternativas, la búsqueda de heurísticas que mejoren la versión actual de ART sigue siendo una línea de investigación abierta que puede proporcionar resultados prometedores en el futuro.

3.6.7. Comentarios finales

Como se ha visto en las secciones anteriores, ART destaca por conseguir un buen compromiso entre la precisión de los clasificadores que construye, el tiempo necesario para construirlos y la complejidad de los modelos generados:

- ART consigue obtener clasificadores con una precisión aceptable, similar a la obtenida con métodos más complejos como C4.5 o RIPPER.
- ART es altamente escalable, característica que lo hace idóneo en la resolución de problemas de *Data Mining*. Aun siendo ligeramente más ineficiente que las versiones más recientes de algoritmos TDIDT (como la implementación de C4.5 similar a RainForest utilizada en los experimentos), es mucho más eficiente que otros algoritmos de inducción de reglas y listas de decisión.
- ART no sólo es más eficiente que dichos algoritmos, sino que, además, consigue modelos de clasificación tan compactos como los obtenidos por otros métodos, modelos que son especialmente simples si se comparan con los árboles de decisión obtenidos por C4.5.
- ART permite tratar con facilidad la presencia de valores desconocidos en los datos al utilizar ramas 'else', sin tener que recurrir a mecanismos complejos ni restringir la topología del árbol de decisión.
- ART realiza un proceso de búsqueda, guiado por los parámetros *MinSupp* y *MaxSize*, gracias al cual es capaz de aprovechar las relaciones existentes entre distintos atributos y ramificar el árbol utilizando los valores de varios atributos simultáneamente.
- ART, finalmente, dispone de un mecanismo automático de selección del umbral de confianza *MinConf* que simplifica su utilización por parte de usuarios no expertos sin suponer perjuicio alguno para las características del clasificador obtenido.

Capítulo 4

Construcción de hipótesis candidatas

> *...la respuesta errónea a un ejercicio puede mostrar una ausencia real de comprensión o revelar el hecho de que el alumno ha construido su propio modelo personal. Desde luego, todo el mundo estará de acuerdo en que, sea cual sea el modelo construido, éste ha de juzgarse por su coherencia, y no por su conformidad con el del profesor.*
>
> IVES KODRATOFF
> *Introduction to Machine Learning*

En el capítulo anterior se ha presentado un modelo de clasificación, al que hemos denominado ART, que se encuentra a medio camino entre los algoritmos TDIDT de construcción de árboles de decisión y los algoritmos de inducción de reglas que construyen listas de decisión. Es este capítulo nos centraremos en la etapa de construcción de hipótesis candidatas, el proceso mediante el cual ART genera conjuntos de reglas potencialmente útiles para construir el árbol de clasificación.

Como ya se ha comentado, ART emplea una de las técnicas más populares de *Data Mining* para llevar a cabo esta etapa de una forma eficiente: la extracción de reglas de asociación. El proceso de extracción de reglas de asociación, usualmente ligado a la exploración de grandes bases de datos transacciona-

les, se suele descomponer en dos etapas, tal como vimos en el apartado 2.3: encontrar todos los itemsets frecuentes presentes en la base de datos (esto es, aquellos patrones cuyo soporte es mayor o igual que un umbral mínimo establecido por el usuario, *MinSupp*) y, a partir de esos itemsets, derivar todas las reglas de asociación cuya confianza se encuentre por encima del otro umbral que ha de establecer el usuario (*MinConf*).

La implementación actual de ART utiliza TBAR [19], un algoritmo de la familia de Apriori [7] que permite obtener los itemsets frecuentes de un modo eficiente. El algoritmo TBAR resulta particularmente adecuado para descubrir patrones en conjuntos de datos densos, que son aquellos conjuntos de datos que presentan relativamente pocos valores nulos (frente a las bases de datos transaccionales en las cuales cada transacción sólo incluye una mínima fracción de los artículos o productos con los que trabaja una empresa).

En problemas de clasificación, los conjuntos de datos de entrenamiento suelen provenir de bases de datos relacionales, ampliamente utilizadas en sistemas informáticos de todo tipo. El conjunto de entrenamiento suele ser, por tanto, una tabla con un conjunto finito de atributos. Cada ejemplo de entrenamiento es, pues, un conjunto de pares *atributo:valor* y no una mera serie de items presentes en una transacción. TBAR, de hecho, se ideó con este tipo de conjunto de datos en mente para agilizar el proceso de extracción de itemsets frecuentes en bases de datos relacionales [19].

Por consiguiente, TBAR se ajusta perfectamente a las necesidades de generación de hipótesis candidatas de ART. Los itemsets frecuentes que se obtienen con TBAR se utilizan como punto de partida para construir reglas de asociación y las mejores reglas de asociación de las obtenidas se seleccionan para ramificar el árbol de decisión que construye ART.

En cualquier caso, ha de quedar claro que el algoritmo TBAR es un método general de extracción de reglas de asociación en bases de datos relacionales y como tal se estudiará en la sección 4.2. En la sección siguiente se mostrará cómo se puede aplicar TBAR a un caso particular: la construcción de modelos de clasificación con ART.

4.1. Extracción de reglas de asociación

En esta sección se recopilan las ideas que sirven de base al algoritmo TBAR, presentado en la sección 4.2. Este algoritmo logra mejorar notablemente la eficiencia de algoritmos previos como Apriori, tal como se verá en el apartado 4.2.3.

El proceso general de extracción de reglas de asociación ya se presentó en el capítulo 2 antes de describir distintos modelos de clasificación basados en reglas de asociación (véase la sección 2.3).

La parte que más tiempo consume del proceso de extracción de reglas de asociación es el descubrimiento de los itemsets frecuentes presentes en una base de datos, mientras que la generación de las reglas de asociación derivadas de esos itemsets es relativamente inmediata. Por tanto, centraremos nuestra atención en el proceso de obtención de los itemsets frecuentes.

Denotemos L_k al conjunto de todos los k-itemsets frecuentes (donde L proviene del adjetivo *large*) y C_k al conjunto de k-itemsets candidatos (esto es, los k-itemsets potencialmente frecuentes).

4.1.1. El algoritmo Apriori

El algoritmo Apriori [7] realiza múltiples recorridos secuenciales sobre la base de datos para encontrar los itemsets frecuentes. En el k-ésimo recorrido sobre el conjunto de datos, el algoritmo encuentra todos los k-itemsets frecuentes. De hecho, cada iteración del algoritmo incluye dos fases: la fase de generación de candidatos que obtiene C_k a partir de L_{k-1}, y el recorrido secuencial propiamente dicho que permite calcular el soporte de los itemsets candidatos y podar dicho conjunto para obtener el conjunto L_k de itemsets frecuentes. El algoritmo termina su ejecución cuando L_k, o C_k, se queda vacío. Para agilizar el proceso, se asume que los items en un itemsets están ordenados lexicográficamente y se utiliza una estructura de datos en forma de tabla hash multinivel que permite almacenar y gestionar los conjuntos de candidatos C_k de una forma eficiente.

Si bien la mayor parte de algoritmos de extracción de reglas de asociación siguen el proceso descrito para el algoritmo Apriori, existen alternativas en las

cuales no es necesario generar un conjunto candidato de itemsets potencialmente frecuentes para obtener los itemsets frecuentes [74].

A continuación se describen las dos fases en las que se descompone cada iteración del algoritmo Apriori:

- **Generación de candidatos**

  En la fase de generación de candidatos, el conjunto de todos los $(k-1)$-itemsets frecuentes descubiertos en la iteración $(k-1)$ se utiliza para generar el conjunto de candidatos C_k. C_k ha de ser necesariamente un superconjunto de L_k, el conjunto formado por todos los k-itemsets frecuentes. Dado que todos los subconjuntos de un itemset frecuente son también frecuentes, C_k puede obtenerse a partir de L_{k-1} en dos pasos:

  - REUNIÓN: Se crea un superconjunto C'_k de C_k a partir del producto cartesiano de L_{k-1} consigo mismo, lo cual puede realizarse de una forma eficiente cuando los items de los itemsets están ordenados lexicográficamente. En tal situación, el conjunto C'_k se obtiene realizando la reunión natural $L_{k-1} \bowtie L_{k-1}$ sobre los primeros $k-2$ items de L_{k-1}.
  - PODA: Se eliminan todos los k-itemsets $c \in C'_k$ con algún subconjunto propio de $k-1$ items que no esté en L_{k-1}. Tras la construcción de C'_k ya sabemos que dos de los subconjuntos de $k-1$ items son frecuentes, por lo que sólo tendremos que comprobar la presencia en L_{k-1} de $k-2$ subconjuntos de cada k-itemset candidato.

- **Recorrido secuencial de la base de datos**

  Una vez que hemos obtenido el conjunto candidato de itemsets potencialmente frecuentes, se recorre secuencialmente la base de datos para determinar el número de transacciones (tuplas en bases de datos relacionales) en que aparece cada uno de los itemsets candidatos. Para cada transacción, se incrementa en uno el soporte de los candidatos de C_k que estén incluidos en la transacción. Al finalizar el recorrido de la base de datos, se examina el soporte de cada candidato para determinar cuáles

de los candidatos son realmente itemsets frecuentes (miembros de pleno derecho de L_k). Apriori emplea una tabla hash multinivel para realizar la cuenta del número de ocurrencias de cada itemset en la base de datos.

4.1.2. El algoritmo DHP

El algoritmo DHP [119], *Direct Hashing and Pruning*, se ideó partiendo del algoritmo Apriori. Como este último, DHP utiliza una tabla hash multinivel para almacenar los itemsets candidatos. Además, DHP emplea una tabla hash adicional para $(k+1)$-itemsets al calcular el soporte de los k-itemsets. Cada entrada de esa tabla hash contiene la suma del número de ocurrencias de todos los itemsets de tamaño $k+1$ que tengan el mismo valor hash. Cuando se crea el conjunto de candidatos C_{k+1}, si el valor almacenado en la casilla correspondiente a un itemset candidato de tamaño $k+1$ es menor que el umbral de soporte mínimo (*MinSupp*), entonces ese itemset no se incluye en C_{k+1}, ya que no puede ser frecuente.

El uso de esta tabla hash adicional durante las primeras iteraciones de un algoritmo como Apriori permite reducir el tamaño de los conjuntos de candidatos generados. Como la parte que más tiempo consume de la extracción de reglas de asociación es la obtención de los itemsets frecuentes a partir de un conjunto grande de itemsets candidatos, DHP consigue reducir el tiempo consumido por el algoritmo de extracción de reglas de asociación.

En las últimas iteraciones del algoritmo, cuando el conjunto de candidatos ya no es tan grande, puede emplearse el algoritmo Apriori para eliminar la carga adicional que supone la utilización de la tabla DHP.

En cualquier caso, hay que resaltar que las mejoras de rendimiento obtenidas con la técnica DHP dependen en gran medida de la naturaleza del conjunto de datos con el que se esté trabajando. Las diferencias pueden ser enormes en las bases de datos transaccionales típicas o no existir siquiera en algunas de las bases de datos relacionales utilizadas para resolver problemas de clasificación.

Otra técnica interesante utilizada por el algoritmo DHP consiste en reducir el tamaño de la base de datos en cada iteración, lo que se suele conocer por el término inglés *transaction trimming*. Si una transacción contiene un itemset frecuente de tamaño $k+1$, debe contener al menos $k+1$ itemsets frecuentes de

tamaño k. Aquellas transacciones que no contengan $k+1$ itemsets frecuentes pueden eliminarse en la k-ésima iteración del algoritmo para reducir el tamaño del conjunto de datos utilizado en posteriores iteraciones. De esta forma se puede conseguir un ahorro considerable de tiempo en el proceso de extracción de reglas de asociación. Sin embargo, ha de tenerse en cuenta que esta técnica puede no ser aplicable si no se dispone del espacio suficiente para almacenar los conjuntos de datos temporales utilizados en cada iteración.

4.2. El algoritmo $\overline{T}$ (TBAR)

En esta sección se describe detalladamente el funcionamiento del algoritmo TBAR, acrónimo de *Tree-Based Association Rule mining*. TBAR es un algoritmo general de extracción de reglas de asociación que puede ser útil siempre que deseemos extraer itemsets frecuentes en un conjunto de datos. Los itemsets descubiertos por TBAR pueden utilizarse para analizar el contenido de grandes bases de datos utilizando reglas de asociación, estudiar secuencias y series temporales, construir perfiles de usuario en sistemas de recuperación de información a partir de los términos más comunes en un conjunto de documentos o, incluso, como base para otras aplicaciones más exóticas [152].

TBAR hace especial hincapié en la aplicación del proceso de extracción de reglas de asociación sobre bases de datos relacionales porque muchos de los sistemas de información que funcionan hoy en día almacenan la información en forma de tablas (en el sentido relacional del término). Este hecho hace de las bases de datos relacionales un objetivo de especial interés para las técnicas de *Data Mining*.

El algoritmo TBAR es un algoritmo de extracción de reglas de asociación de la familia de Apriori [7], que se caracteriza por almacenar todos los itemsets descubiertos en una única estructura de datos en forma de árbol a la que denominaremos árbol de itemsets. Además, TBAR incluye la posibilidad de emplear una versión generalizada de DHP [119], una técnica que permite podar aún más el conjunto de itemsets candidatos (apartado 4.1.2).

Por otro lado, TBAR también se diferencia de Apriori al adoptar una definición diferente de item que nos permitirá reducir el tamaño del conjunto de

itemsets candidatos (los itemsets potencialmente relevantes) cuando se utilizan bases de datos relacionales. En este contexto, un item es un par a:v, donde a es una atributo (una columna de una tabla) y v es uno de los valores que puede tomar el atributo a. Una tupla t contiene un item a:v si su atributo a toma como valor v.

Igual que en las bases de datos transaccionales, un itemset no es más que un conjunto de items. Un k-itemset es un itemset que contiene k items. Una tupla t de aridad m contiene un itemset I de grado $k \leq m$ si la tupla t contiene todos los items presentes en el itemset I.

Una propiedad fundamental de los itemsets derivados de una relación obtenida como resultado de realizar una consulta sobre una base de datos relacional es que un itemset no puede contener más que un item para cada columna de la tabla. En otras palabras, todos los items incluidos en un itemset deben corresponder a diferentes columnas de la tabla. Matemáticamente, si a_1:v_1 y a_2:v_2 pertenecen al itemset I, siendo $v_1 \neq v_2$, entonces $a_1 \neq a_2$. Esta propiedad es consecuencia directa de la Primera Forma Normal (1NF): una relación está en Primera Forma Normal si los dominios de todos sus atributos contienen únicamente valores atómicos.

La propiedad anterior de las bases de datos relacionales nos permitirá podar el conjunto de candidatos durante la generación de itemsets frecuentes y justifica nuestra distinción entre items de una base de datos transaccional e items en una base de datos relacional. Cuando se trabaja sobre bases de datos relacionales, la etapa de reunión de la fase de generación de candidatos del algoritmo Apriori puede modificarse para podar el conjunto de candidatos eliminando aquéllos itemsets que no estén en Primera Forma Normal.

4.2.1. Visión general de TBAR

El primer subproblema que afronta cualquier algoritmo de extracción de reglas de asociación es el descubrimiento de todos los itemsets potencialmente interesantes, denominados itemsets frecuentes cuando utilizamos el soporte como medida de relevancia (véase la sección 4.4.2). En el marco tradicional de extracción de reglas de asociación, el objetivo es encontrar todos los itemsets cuyo soporte iguale, al menos, al umbral *MinSupp*, aunque también se pueden

utilizar otras medidas de interés (el uso de los itemsets frecuentes, de hecho, ha sido muy criticado [2] [3]). Así pues, en general, nos referimos a los itemsets potencialmente interesantes como 'itemsets relevantes' en vez de 'itemsets frecuentes', dejando abierta la posibilidad de utilizar medidas de relevancia alternativas.

TBAR requiere que, si un conjunto de items dado es relevante, entonces todos sus subconjuntos también sean relevantes. Los itemsets frecuentes satisfacen esta propiedad de monotonía y otros criterios también lo hacen. De hecho, este requerimiento es esencial para cualquier algoritmo que se base en la generación de un conjunto de candidatos de itemsets potencialmente relevantes. Desde este punto de vista, Apriori y TBAR son completamente equivalentes.

El algoritmo TBAR, pues, sigue la misma filosofía que la mayor parte de los algoritmos de extracción de reglas de asociación: primero busca los itemsets relevantes y después genera las reglas de asociación que se derivan a partir de los itemsets obtenidos.

4.2.1.1. Obtención de los itemsets relevantes

El primer paso del algoritmo consiste en descubrir todos los itemsets potencialmente interesantes que se encuentren en la base de datos. TBAR, al igual que Apriori, implementa un algoritmo iterativo que, paulatinamente, va obteniendo itemsets de mayor tamaño a partir de los itemsets encontrados en la iteración anterior. El algoritmo TBAR se puede implementar utilizando el lenguaje de programación Java como se muestra a continuación:

```
set.Init (MinSupport);
itemsets = set.Relevants(1);
k = 2;
while (k<=columns && itemsets>=k) {
      itemsets = set.Candidates(k);
      if (itemsets>0)
         itemsets = set.Relevants(k);
      k++;
}
```

En el fragmento de código anterior, `set` denota la estructura de datos utilizada por TBAR para almacenar todos los itemsets descubiertos (el árbol de itemsets descrito en la sección 4.2.2) y la variable `k` indica en cada momento el tamaño de los itemsets sobre los que se está trabajando.

El método `Init` inicializa la estructura de datos indicándole los parámetros establecidos por el usuario (en este caso, el umbral de soporte mínimo). `Relevants(k)` sirve para generar L_k (una vez que tenemos C_k) mientras que `Candidates(k)` nos permite crear C_k a partir de L_{k-1}. La implementación de ambos métodos devuelve el número de itemsets incluidos en el último conjunto obtenido con la finalidad de poder finalizar el proceso cuando no haya itemsets candidatos. Obviamente, no se pueden obtener itemsets relevantes de un conjunto vacío de candidatos porque L_k es siempre un subconjunto de C_k. Además, el proceso también se detendrá cuando el número de itemsets relevantes de tamaño $k-1$ sea inferior a k, situación que nos indica que no puede haber ningún itemset relevante de tamaño k puesto que todos los subconjuntos de un itemset relevante han de ser también relevantes y un itemset de tamaño k incluye k itemsets de tamaño $k-1$. Esta propiedad permite a algoritmos como Apriori [7] y OCD [107] reducir el tamaño del conjunto C_k durante la fase de generación de itemsets candidatos (representada en TBAR por la llamada al método `Candidates(k)`).

Respecto al fragmento de código anterior, sólo falta mencionar que, en una base de datos relacional, el número máximo de items incluidos en un itemset es igual al número de columnas de la relación sobre la que estemos trabajando (la variable `columns`).

Los distintos métodos que permiten ir construyendo iterativamente el árbol de itemsets se describirán detalladamente en la sección 4.2.2. Para completar nuestra visión general de TBAR sólo nos falta ver cómo se obtienen las reglas de asociación a partir de los itemsets relevantes.

4.2.1.2. Generación de las reglas de asociación

Una vez que tenemos los itemsets relevantes, todas las reglas de asociación que se derivan de ellos pueden obtenerse recorriendo el árbol de itemsets. El recorrido adecuado del árbol lo realiza el método `Rules`, que se describirá en

la sección 4.2.2 de esta memoria.

Durante el recorrido del árbol, se emplea el umbral de confianza mínima establecido por el usuario, *MinConf*, para obtener sólo aquellas reglas cuya confianza sea lo suficientemente alta como para que las reglas sean consideradas potencialmente interesantes. Como es lógico, se puede utilizar cualquier otra medida de cumplimiento (sección 4.4.3) como criterio para restringir el conjunto de reglas devuelto.

Además, el formato de las reglas de asociación y de los itemsets que aparecen en sus antecedentes o en sus consecuentes puede establecerse de antemano para limitar el volumen del conjunto de reglas que se obtiene a partir del árbol de itemsets. Algunas restricciones de este tipo se imponen en ART para agilizar el proceso de generación de hipótesis candidatas (como se verá en la sección 4.3) y han sido en el pasado objeto de estudio en otros trabajos [8] [147].

4.2.2. El árbol de itemsets

TBAR emplea una estructura de datos a medida para representar tanto los conjuntos de itemsets candidatos como los frecuentes: el árbol de itemsets. Este árbol almacena todos los itemsets descubiertos en un árbol de prefijos, similar en cierta medida a un árbol de enumeración de subconjuntos.

A diferencia de Apriori, TBAR utiliza un único árbol para almacenar todos los itemsets encontrados. Apriori, sin embargo, construye una estructura de datos en forma de árbol (su tabla hash multinivel) para cada tamaño de itemset. La representación compacta de los itemsets conseguida por TBAR proporciona un ahorro substancial de espacio de almacenamiento en comparación con Apriori [7] e, incluso, con propuestas más recientes como el árbol de patrones frecuentes de Han, Pei y Yin [74]. Este último no es más que una representación alternativa de la base de datos completa, no sólo de los itemsets frecuentes.

Para ilustrar el funcionamiento de la estructura de datos empleada por TBAR utilizaremos un sencillo ejemplo, empleando el soporte como medida de relevancia de los itemsets. Supongamos que, tras algún tipo de preprocesamiento, obtenemos el pequeño conjunto de datos mostrado en la tabla 4.1.

Si el umbral de soporte mínimo establecido por el usuario es igual al 40 %

A	B	C
0	0	0
0	0	1
0	1	1
1	1	1
1	1	1

Tabla 4.1: Un sencillo conjunto de datos.

de las tuplas de nuestro conjunto de datos (esto es, dos de las cinco tuplas), a partir del conjunto de datos se obtienen los itemsets relevantes que aparecen en la tabla 4.2. Estos itemsets son los itemsets frecuentes de nuestro conjunto de datos ya que utilizamos el soporte como medida de relevancia (sección 4.4.2).

En TBAR se representan todos los itemsets mostrados en la tabla 4.2 mediante un árbol de enumeración de subconjuntos. Dicho árbol puede empaquetarse utilizando una representación compacta como la mostrada en la figura 4.1 con el objetivo de ahorrar espacio y disminuir la fragmentación de memoria durante la ejecución de TBAR.

Obsérvese que los items están ordenados lexicográficamente, igual que en Apriori. El número de k-itemsets representados en el árbol de itemsets es igual al número de items incluidos en los nodos situados en el nivel $L[k]$. Los k-itemsets pueden reconstruirse concatenando los items que se encuentran en el camino desde la raíz del árbol hasta el nodo situado en el nivel $L[k]$. El soporte de un k-itemset se almacena en el árbol acompañando al k-ésimo item del itemset.

Se puede utilizar una tabla hash interna en cada nodo con el objetivo de optimizar el acceso a la información contenida en él. Esta tabla se crea automáticamente cuando el número de items incluidos en el nodo supera cierto umbral dependiente de la implementación y permite indexar los items por el par *atributo:valor* que los representa ($a : v$). Esta sencilla técnica permite acceder eficientemente a los itemsets almacenados en el árbol de itemsets y es análoga a la utilizada en la tabla hash multinivel del algoritmo Apriori.

Como se comentó en el apartado 4.1.2, las tablas DHP permiten reducir

Conjunto $L[k]$	Itemsets {items} [soporte]	Cardinalidad $\sharp L[k]$
$L[1]$	{A:0} [3]	5
	{A:1} [2]	
	{B:0} [2]	
	{B:1} [3]	
	{C:1} [4]	
$L[2]$	{A:0, B:0} [2]	5
	{A:0, C:1} [2]	
	{A:1, B:1} [2]	
	{A:1, C:1} [2]	
	{B:1, C:1} [3]	
$L[3]$	{A:1, B:1, C:1} [2]	1

Tabla 4.2: Itemsets frecuentes derivados del conjunto de datos de la tabla 4.1.

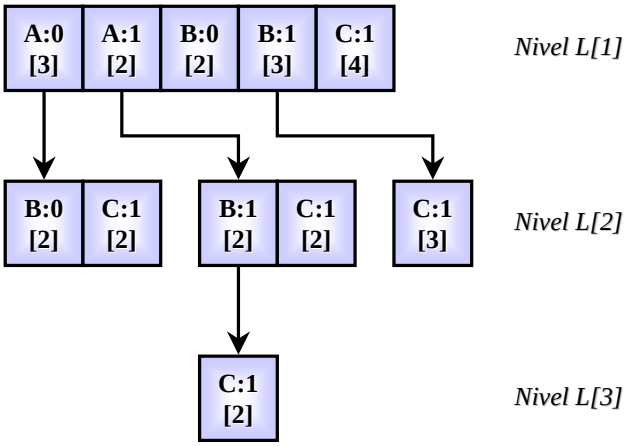

Figura 4.1: Árbol de itemsets correspondiente a los itemsets mostrados en la tabla 4.2

el tamaño de los conjuntos de candidatos que se generan en cada iteración del algoritmo. Además de la tabla hash empleada para acceder eficientemente a los datos de un nodo del árbol, TBAR utiliza una versión generalizada de DHP durante la etapa de generación de itemsets. En particular, la implementación de TBAR incluye la posibilidad de utilizar localmente una tabla DHP en cada nodo del árbol (a diferencia de la tabla DHP global de la propuesta original de Park, Chen y Yu).

La fase de generación de candidatos del algoritmo TBAR es bastante simple si tenemos en cuenta cómo se almacenan los itemsets. Para generar el conjunto de candidatos C_{k+1} sólo hay que crear un nodo hijo para cada item $a : v$ que se encuentre en el nivel $L[k]$ y añadirle al nodo recién creado todos los items que aparecen a la derecha del item $a : v$ en el nodo del nivel $L[k]$. Los items que tengan el mismo atributo a que el item $a : v$ pueden descartarse directamente cuando se trabaja con bases de datos relacionales (recuérdese la Primera Forma Normal).

Una vez generado el conjunto de candidatos C_{k+1}, se obtiene el soporte de cada itemset candidato de tamaño $(k + 1)$ (esto es, el valor que aparece acompañando a los items del nivel $L[k + 1]$ del árbol de itemsets). Todos los itemsets de tamaño $k + 1$ que no sean relevantes (p.ej. los que no alcancen el umbral mínimo de soporte) se eliminan y el árbol podado queda con el conjunto de itemsets relevantes L_{k+1} en su nivel $L[k + 1]$.

En los siguientes apartados se analizará con mayor detalle la implementación de las distintas primitivas que conforman el TDA (Tipo de Dato Abstracto) Árbol de Itemsets. Gracias al árbol de itemsets, TBAR requiere menos recursos computacionales que Apriori para extraer los itemsets relevantes presentes en una base de datos (tanto en tiempo de CPU como en espacio de almacenamiento, como se verá en la sección 4.2.3).

4.2.2.1. Inicialización del árbol de itemsets

La primitiva `Init` del TDA Árbol de Itemsets crea la estructura de datos, establece sus parámetros (v.g. el umbral de soporte mínimo *MinSupp*) e incluye en el nodo raíz del árbol todos los items presentes en la base de datos (es decir, el conjunto de candidatos C_1). Además, se recopila la información

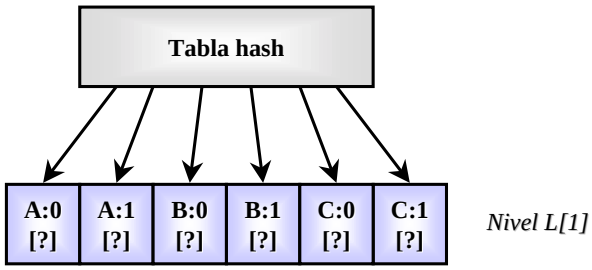

Figura 4.2: Árbol de itemsets tras su inicialización. Una tabla hash asegura el acceso eficiente a los items del nodo raíz.

que sea necesaria para asegurar un acceso eficiente a los itemsets del árbol mediante tablas hash y, opcionalmente, crea la infraestructura necesaria para poder aplicar DHP.

La figura 4.2 muestra cómo queda el árbol de itemsets tras su inicialización a partir de los datos de la tabla 4.1.

4.2.2.2. Obtención de los itemsets relevantes

Una llamada al método `Relevants(k)` permite obtener el conjunto de itemsets relevantes L_k a partir del conjunto de k-itemsets que se encuentre ya en el árbol (esto es, el conjunto de k-itemsets candidatos C_k).

Para obtener los k-itemsets relevantes a partir de los k-itemsets potencialmente relevantes sólo hay que recorrer secuencialmente la base de datos para contar el número de ocurrencias de cada k-itemset candidato. De esta manera se obtiene el soporte de todos los itemsets de C_k y, mediante una sencilla poda, se consigue el conjunto L_k eliminando todos aquellos itemsets que no sean relevantes.

Cuando se utiliza un umbral mínimo de soporte *MinSupp*, los itemsets relevantes son aquéllos k-itemsets candidatos cuyo soporte (obtenido recorriendo secuencialmente los datos) sea mayor o igual que el umbral establecido por el usuario.

Las tablas hash empleadas internamente en cada nodo del árbol permiten

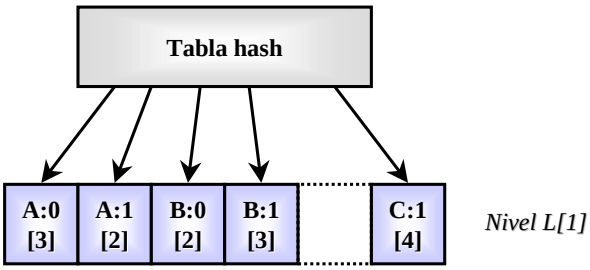

Figura 4.3: Árbol de itemsets tras eliminar los items no relevantes del conjunto de candidatos C[1].

contabilizar el soporte de los k-itemsets candidatos de una forma efectiva durante el recorrido secuencial de los datos, de forma que se minimice el número de comparaciones necesario para recorrer el árbol de itemsets desde su raíz hasta el nodo que contiene el k-ésimo item del itemset candidato.

El paso final de poda es trivial: todo lo que queda por hacer es eliminar los items correspondientes a k-itemsets no relevantes de los nodos situados en el nivel $L[k]$ del árbol de itemsets.

4.2.2.3. Generación de candidatos

La primitiva `Candidates(k)` del TDA Árbol de Itemsets es la encargada de generar el conjunto de k-itemsets candidatos. Tal como se mencionó anteriormente, el proceso de generación de candidatos consiste en copiar todos los items a la derecha de un item en el nodo actual del árbol e incluirlos en un nodo hijo que cuelgue del nodo actual asociado al item seleccionado.

Este mecanismo de copia puede restringirse para que genere únicamente itemsets potencialmente relevantes. Si corresponde a una tabla de una base de datos relacional, por ejemplo, el itemset nunca puede incluir dos items para una misma columna de la tabla. TBAR, además, permite reducir el tamaño del conjunto de candidatos utilizando DHP.

Es digna de mención la omisión en TBAR del paso de poda incluido en la fase de generación de candidatos de Apriori. Experimentalmente se ha com-

probado que el ahorro de espacio obtenido al reducir el tamaño del conjunto de candidatos no compensa al coste que conllevan las comprobaciones adicionales que conlleva esta etapa del algoritmo Apriori. De hecho, la estructura de datos empleada por TBAR es adecuada para acceder y actualizar eficientemente el soporte de los itemsets candidatos independientemente de su número, pues se utilizan tablas hash en cada nodo del árbol que agilizan el recorrido secuencial de la base de datos. El paso de poda realizado por Apriori no supone ninguna mejora computacional para TBAR y, por tanto, se elimina de este último.

Además, la poda del conjunto de candidatos realizada por Apriori sólo es útil al generar los conjuntos C_k cuando $k \geq 3$ porque el mecanismo de construcción de los candidatos a partir de la reunión natural $L_{k-1} \bowtie L_{k-1}$ sobre los primeros $k-2$ items de L_{k-1} realiza implícitamente las comprobaciones necesarias para $k = 2$.

Como el cuello de botella del algoritmo suele encontrarse en sus primeras iteraciones (al obtener C_2), resulta más productivo utilizar técnicas como DHP y omitir comprobaciones adicionales asociadas a la poda de Apriori que apenas reducen del conjunto de candidatos.

En iteraciones posteriores, incluso cuando el número de itemsets relevantes sigue siendo elevado, resulta más útil emplear DHP localmente en cada nodo en vez de la poda de Apriori. En situaciones como ésta, es importante recordar que cuantos más itemsets frecuentes haya, mayor será el tamaño del conjunto de candidatos y mayor será el impacto de DHP sobre éste.

Las figuras 4.4 a 4.6 muestran la evolución del árbol de itemsets asociado a los datos de la tabla 4.1 hasta llegar a su configuración final, la mostrada anteriormente en la tabla 4.1.

4.2.2.4. Derivación de reglas de asociación

La última primitiva que nos queda por ver del TDA Árbol de Itemsets es la que permite derivar reglas de asociación a partir de la información almacenada en el árbol.

Para obtener las reglas de asociación que se derivan de los itemsets almacenados en el árbol se necesitan dos iteradores que nos permitan recorrer el

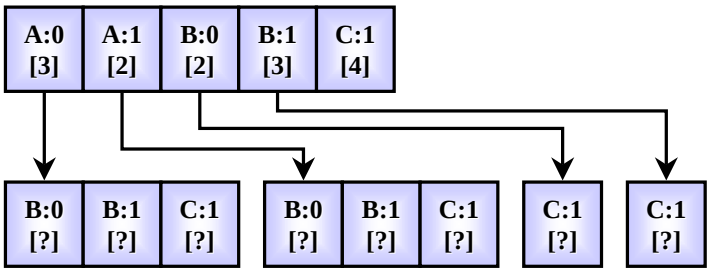

Figura 4.4: Itemsets candidatos de tamaño 2

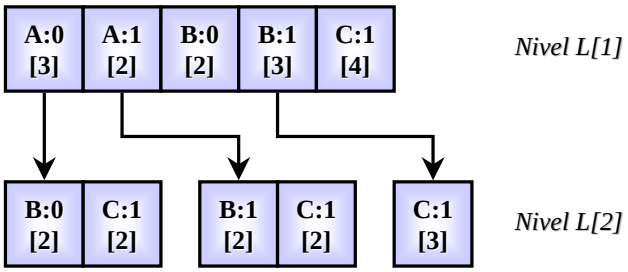

Figura 4.5: Itemsets relevantes de tamaño 2

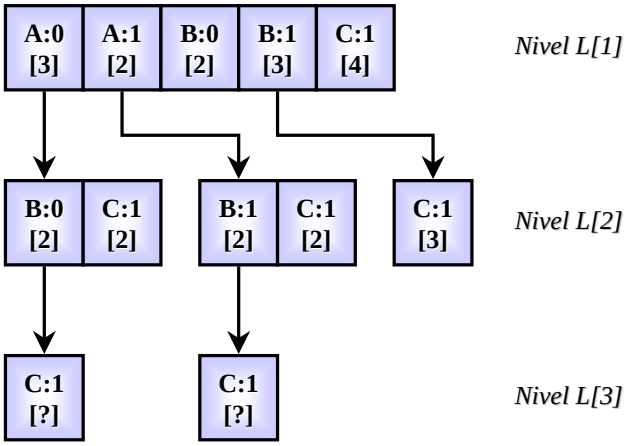

Figura 4.6: Itemsets candidatos de tamaño 3

árbol adecuadamente:

- El primer iterador obtiene una enumeración de todos los k-itemsets relevantes incluidos en el árbol de itemsets con $k \geq 2$. Es decir, nos devuelve aquellos itemsets a partir de los cuales se puede derivar una regla de asociación (cualquiera que tenga al menos un item para el antecedente y otro para el consecuente de la regla).

- El segundo iterador nos ofrece la posibilidad de enumerar todos los subconjuntos propios de un itemset dado. A partir de un subconjunto propio l_i del itemset l_k ($l_i \subset l_k$) se puede derivar la regla de asociación $l_i \Rightarrow (l_k - l_i)$ cuyo soporte es el soporte de l_k y cuya confianza se obtiene dividiendo el soporte de l_k por el de l_i.

A continuación se muestra el pseudocódigo del algoritmo que nos permite obtener las reglas de asociación derivadas del árbol de itemsets utilizando los dos iteradores anteriores:

```
// Aplicando el primer iterador

Para cada k-itemset relevante l_k del árbol (k ≥ 2)

     // Generar las reglas derivadas de l_k
     // aplicando el segundo iterador

     Para cada itemset l_i ⊂ l_k

          // Seleccionar las reglas cuya confianza
          // esté por encima del umbral MinConf

          Si soporte(l_k) ≥ MinConf*soporte(l_i)

             Devolver la regla l_i ⇒ (l_k − l_i)
```

El pseudocódigo anterior devuelve todas las reglas cuya confianza esté por encima del umbral *MinConf* establecido por el usuario, si bien el algoritmo se

puede modificar fácilmente para emplear otras medidas de consistencia (sección 4.4.3) o forzar que las reglas verifiquen ciertas restricciones [8] [147].

Veamos ahora algunos detalles sobre la implementación de los dos iteradores empleados para recorrer el árbol de itemsets y obtener un conjunto de reglas de asociación:

- El primer iterador es el que enumera todos los itemsets presentes en el árbol de itemsets con tamaño mayor o igual a dos. El recorrido del árbol comienza con el itemset vacío (itemset de tamaño 0 que no incluye ningún item). Dado un itemset particular, el iterador intenta, en primer lugar, expandir el k-itemset actual buscando en los nodos hijo del nodo actual para formar itemsets de tamaño $k+1$. Cuando ya no se puede expandir más el nodo actual (porque no existen en el árbol nodos por los que se pueda descender desde el nodo actual), entonces se buscan k-itemsets alternativos en el nodo actual del árbol. Todos estos k-itemsets tienen en común los primeros $k-1$ items y sólo difieren en su k-ésimo item. Finalmente, cuando no se pueden encontrar más k-itemsets alternativos, el iterador da marcha atrás ascendiendo por el árbol de itemsets para encontrar m-itemsets ($m < k$) que compartan los primeros $m-1$ items del k-itemset actual y difieran de éste en su m-ésimo item. En definitiva, el iterador explora el árbol de itemsets como un algoritmo típico de vuelta atrás o *backtracking* [22].

- El segundo iterador tiene como misión obtener todos los subconjuntos propios de un itemset dado, para lo cual realiza un recorrido similar del árbol de itemsets. Comenzando nuevamente con el itemset vacío, este iterador busca extensiones del itemset actual realizando una exploración en profundidad del árbol de itemsets hasta ir agotando todas las posibilidades, momento en el cual realiza una vuelta atrás para explorar otras alternativas.

El algoritmo descrito para extraer todas las reglas de asociación que se puedan derivar del conjunto de itemsets almacenado en el árbol de itemsets es especialmente eficiente gracias, de nuevo, a las tablas hash locales que propor-

ciona el TDA Árbol de Itemsets para acceder de una forma rápida a los items almacenados en cada nodo del árbol.

Hay que destacar, además, que el algoritmo propuesto aquí no genera reglas duplicadas, uno de los mayores inconvenientes del algoritmo de generación de reglas planteado en [7].

4.2.3. Resultados experimentales

Apriori y TBAR se han implementado como aplicaciones escritas en el lenguaje de programación Java. Estas implementaciones utilizan JDBC, *Java DataBase Connectivity*. Básicamente, se escogió el interfaz estándar de acceso a bases de datos relacionales de Java por su portabilidad, ya que existen controladores JDBC que funcionan prácticamente para todas las bases de datos comerciales existentes en la actualidad (y muchas de las gratuitas).

Tanto Apriori como TBAR se implementaron también en C++ utilizando C++Builder y BDE (*Borland Database Engine*). Empleando la implementación en C++ de ambos algoritmos se obtuvieron resultados similares a los reflejados en este apartado de la memoria, por lo cual nos centraremos exclusivamente en los resultados de los experimentos realizados con las implementaciones de ambos algoritmos en Java.

En el trabajo de Sarawagi et al. [137] se puede encontrar una discusión detallada acerca de las alternativas de implementación disponibles para todo aquél que desee implementar algoritmos de extracción de conocimiento. Se pueden utilizar distintos grados de acoplamiento entre la implementación del algoritmo y el sistema gestor de bases de datos para llegar a un compromiso entre la portabilidad del código y su eficiencia. La descripción de una implementación de Apriori fuertemente acoplada al sistema gestor de bases de datos DB2 de IBM se puede encontrar en [6].

En este trabajo hemos optado por una implementación débilmente acoplada al sistema gestor de bases de datos. Aunque su rendimiento pueda ser peor en principio, la implementación es de esta forma independiente de la base de datos utilizada y permite explotar al máximo la flexibilidad de un lenguaje de programación de alto nivel como Java, el cual ofrece además independencia de la plataforma sobre la que se trabaje.

Por otro lado, en sistemas reales en los cuales el servidor que alberga la base de datos ha de ofrecer servicio a muchas aplicaciones cliente de forma remota, resulta más adecuado descargar al servidor del costoso proceso de extracción de reglas de asociación (en vez de hacer que el servidor se encargue de extraer las reglas de asociación mediante una implementación fuertemente acoplada que se ejecutaría en el espacio de memoria del propio gestor de bases de datos).

La mayor portabilidad de la implementación débilmente acoplada y su mayor adecuación a entornos cliente/servidor compensa cualquier pequeña pérdida de rendimiento que se pueda producir respecto a una implementación fuertemente acoplada. Aún mejor, una implementación independiente del sistema gestor de bases de datos e, incluso, del sistema operativo permite encapsular el algoritmo de extracción de reglas de asociación en un componente fácilmente reutilizable en aplicaciones de muy diversa índole (véase la arquitectura propuesta en el capítulo 6).

Los algoritmos TBAR y Apriori se han aplicado a distintos conjuntos de datos clásicos, algunos de los cuales se pueden conseguir del Machine Learning Database Repository de la Universidad de California en Irvine:

`http://www.ics.uci.edu/~mlearn/MLRepository.html`

En concreto, para los experimentos reflejados en esta memoria se han empleado los siguientes conjuntos de datos:

- GOLF: Éste es el sencillo conjunto de datos utilizado por J.R. Quinlan en su artículo clásico sobre ID3 [129]. El conjunto de datos, de sólo 14 tuplas, se emplea como ejemplo para construir un árbol de clasificación que decide si jugar al golf o no en función de las condiciones meteorológicas (estado general, temperatura, humedad y viento).

- VOTE: Los registros de este conjunto de datos del repositorio de la UCI incluye los votos de los 435 congresistas de Estados Unidos correspondientes a 16 votaciones identificadas como clave en el *Congressional Quarterly Almanac* de 1984. Las votaciones de cada miembro del Con-

greso van acompañadas de su partido político, que establece la clase en este problema de clasificación.

- SOYBEAN: También del repositorio de la UCI, es un conjunto de datos preparado para construir modelos que permitan diagnosticar la 'enfermedad de la soja'. Contiene 683 tuplas para 19 clases diferentes con 35 atributos predictores para cada uno de los ejemplos del conjunto de datos.

- MUSHROOM: Conjunto de datos obtenido del repositorio de la UCI que incluye la descripción de 8.124 muestras hipotéticas correspondientes a 23 especies de setas. Cada especie aparece identificada como comestible o venenosa (clase que incluye a las setas definitivamente venenosas y a aquéllas cuya comestibilidad se desconoce). Las muestras incluyen 22 atributos y 2.480 valores desconocidos.

- CENSUS: Recoge unas 300.000 tuplas que recopilan datos extraídos de la base de datos de la Oficina del Censo de EE.UU.. Esta base de datos se utiliza para determinar el nivel de ingresos para la persona representada por cada registro (superior o inferior a $50K USD). La base de datos original del censo ha sido reemplazada en el repositorio de la UCI por el conjunto de datos ADULT, de unas 50.000 tuplas.

Los resultados obtenidos para los conjuntos de datos arriba descritos aparecen descritos en la tabla 4.3 y fueron realizados utilizando la versión 1.2.2 del kit de desarrollo de Sun Microsystems utilizando tres configuraciones diferentes:

- **Configuración 1 (C1): Ordenador personal estándar** con procesador Intel Pentium 166MHz, 32MB de EDO RAM, Windows NT 4.0 Workstation y Personal Oracle Lite 3.0.

- **Configuración 2 (C2): Ordenador personal accediendo a un servidor.** La aplicación de *Data Mining* se ejecuta en un ordenador personal Pentium 90MHz (con Windows NT 4.0 Workstation y 32MB de memoria principal) que accede a la base de datos en un Pentium II MMX dual

Datos	Tamaño (en items)	Itemsets relevantes	Algoritmo	Tiempo (en segundos) C1	C2	C3
GOLF	0.07K	104	Apriori	0.9	-	-
			TBAR	0.6	-	-
VOTE	7.4K	6734	Apriori	102	36	7.5
			TBAR	259	69	5.0
SOYBEAN	24.6K	70047	Apriori	998	272	65
			TBAR	259	69	15
MUSHROOM	186.9K	29807	Apriori	1743	583	151
			TBAR	688	188	38
CENSUS	3.7M	101456	Apriori	-	-	8975
			TBAR	-	-	3414

Tabla 4.3: Resultados experimentales obtenidos por TBAR y Apriori para las distintas configuraciones (sin emplear DHP en ninguno de los dos algoritmos).

a 333MHz con 128MB de SDRAM en el que está instalado el DBMS Oracle 8i sobre Windows NT 4.0 Server. Cliente y servidor se conectan mediante TCP/IP a través de una red local Ethernet a 10Mbps.

- **Configuración 3 (C3): Biprocesador Pentium II.** Tanto la aplicación cliente como el servidor Oracle 8i se ejecutan sobre el Pentium II dual descrito para la configuración anterior.

Los resultados recogidos en la tabla 4.3 corresponden a las versiones básicas de Apriori y TBAR. Los tiempos medidos muestran cómo TBAR es siempre más rápido que Apriori, incluso sin utilizar DHP. Dicha técnica puede emplearse para mejorar el rendimiento de cualquier algoritmo derivado de Apriori. En el caso de TBAR, en vez de utilizar una tabla DHP global para cada iteración del algoritmo, se emplean tablas DHP locales para cada nodo del árbol de itemsets con el objetivo de obtener aún mejores resultados. La tabla 4.4 muestra cómo TBAR sigue siendo bastante mejor que Apriori cuando se emplea DHP para reducir el tamaño del conjunto de candidatos.

TBAR reduce el tiempo requerido por Apriori incluso para pequeños conjuntos de datos. Es más, conforme aumenta el número y el tamaño de los itemsets descubiertos, la diferencia entre ambos algoritmos se acrecienta.

Datos	Apriori + DHP	TBAR + DHP	Mejora relativa	Sin DHP
VOTE	9.3 segundos	3.0 segundos	$\times$ 3.1	$\times$ 1.5
SOYBEAN	26.9 segundos	8.0 segundos	$\times$ 3.4	$\times$ 3.9
MUSHROOM	76.9 segundos	31.3 segundos	$\times$ 2.5	$\times$ 2.5

Tabla 4.4: Efecto de DHP sobre Apriori y TBAR. Nótese que los tiempos reflejados en esta tabla no son directamente comparables con los mostrados en la tabla 4.3 por haber sido obtenidos utilizando una configuración diferente (Pentium II 350Mhz con 64MB de RAM e Interbase como sistema gestor de bases de datos).

TBAR no es sólo más rápido que Apriori, sino que también requiere menos espacio en memoria (lo que retrasa la necesidad de intercambiar páginas de memoria cuando ésta escasea). Este hecho es de especial relevancia en configuraciones donde la memoria disponible es escasa, como por ejemplo el PC utilizado para ejecutar la aplicación cliente en las configuraciones C1 y C2. TBAR, además, produce una menor fragmentación de memoria, lo que facilita el trabajo del recolector de basura de Java y permite la ejecución continuada de la aplicación sin necesidad de reiniciar la máquina virtual Java. Este aspecto es esencial si deseamos ofrecer nuestro componente de extracción de reglas de asociación como un servicio web al que se pueda acceder mediante SOAP o a través de servlets.

Uno de los factores que dominan el tiempo total de ejecución del algoritmo es, obviamente, el tiempo empleado en recorrer secuencialmente el conjunto de datos completo en cada iteración. Este recorrido consume la mayor parte del tiempo de ejecución correspondiente a la base de datos del censo y justifica la menor mejora relativa obtenida por TBAR para este conjunto de datos.

En cualquier caso, ha de tenerse muy en cuenta que la mejora relativa obtenida por TBAR respecto a Apriori aumenta corforme avanza el número de iteraciones realizadas. La evolución del tiempo de ejecución de TBAR frente al requerido por Apriori se puede observar en las figuras 4.7, 4.8 y 4.9.

El tiempo de ejecución de TBAR, como el de cualquier otro algoritmo de la familia de Apriori, aumenta linealmente con el tamaño del conjunto de datos de entrada. TBAR mejora a Apriori cuando hay miles de itemsets relevantes y

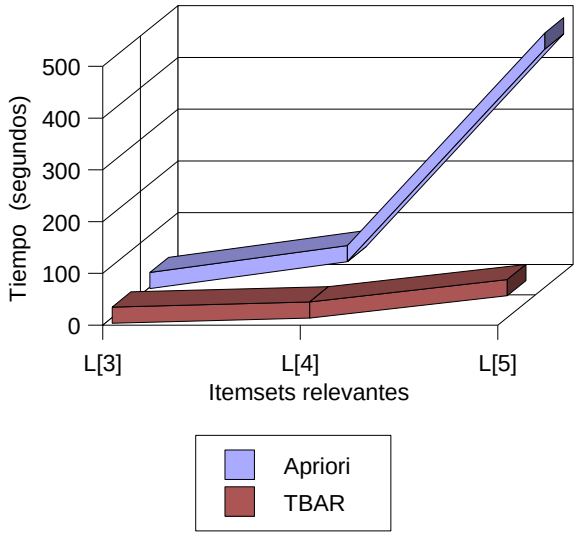

Figura 4.7: TBAR vs. Apriori (SOYBEAN)

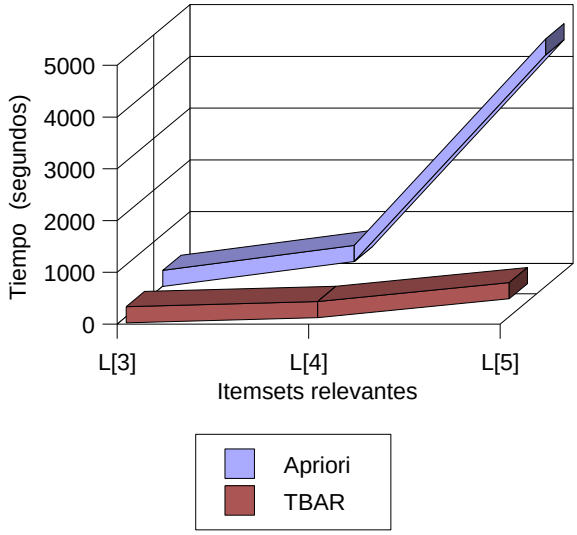

Figura 4.8: TBAR vs. Apriori (MUSHROOM)

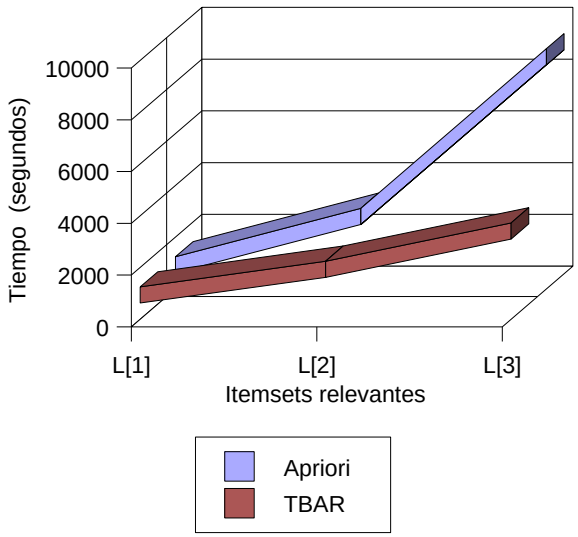

Figura 4.9: TBAR vs. Apriori (CENSUS)

también cuando el tamaño de los itemsets aumenta. Estas características hacen que TBAR resulte especialmente adecuado para realizar tareas de extracción de conocimiento en bases de datos y, como caso particular, resulte de especial utilidad en el modelo de clasificación ART descrito en el capítulo anterior.

Por ejemplo, baste mencionar que el tiempo de ejecución de Apriori cuadruplica el tiempo empleado por TBAR en el conjunto de datos SOYBEAN. Es digno de mención que, en este conjunto de datos, TBAR consigue obtener todos los itemsets relevantes de tamaño 5 en el tiempo requerido por Apriori para descubrir los itemsets de tamaño 4.

Trabajando con el conjunto de datos MUSHROOM, de forma análoga a lo sucedido con SOYBEAN, TBAR encuentra todos los itemsets relevantes de tamaño 4 en el tiempo que Apriori necesita para realizar sus tres primeras iteraciones hasta llegar a obtener L_3. Además, TBAR encuentra todos los itemsets de tamaño 5 antes de que Apriori llegue a obtener los itemsets de tamaño 4. En posteriores iteraciones la estructura de datos empleada por TBAR le permite a éste ejecutarse exclusivamente en memoria principal mientras que Apriori ha de recurrir a un dispositivo de almacenamiento secundario cuando se queda sin

espacio en memoria.

Aparte de los conjuntos de datos anteriores, que son relativamente pequeños, también se han realizado pruebas con un conjunto de datos de mayor tamaño: la base de datos de la Oficina del Censo de Estados Unidos. Este conjunto de datos incluye varios atributos de tipo numérico que se trataron como atributos categóricos una vez discretizados utilizando técnicas de agrupamiento como el conocido algoritmo de las K medias. Por ejemplo, el atributo numérico WKSWORK (que nos indica cuántas semanas del año ha trabajado un individuo) se trató como 0, [1,40], [41,52] y el atributo MARSUPWT se dividió en cinco intervalos equiprobables [146]. Basten como botón de muestra dos de las reglas más sencillas que se extrajeron de este conjunto de datos:

```
if asex='Female' then income<50000
   with confidence = 97.4\% and support = 50.6\%

if asex='Male' then income<50000
   with confidence = 89.8\% and support = 43.1\%
```

4.2.4. Observaciones finales sobre TBAR

En esta sección se ha presentado y evaluado el algoritmo TBAR, cuyo acrónimo proviene de *Tree-Based Association Rule mining*. TBAR es un algoritmo eficiente de extracción de reglas de asociación que mejora al algoritmo Apriori [7]. TBAR utiliza una estructura de datos en forma de árbol de enumeración de subconjuntos especialmente adaptada para el problema de extraer itemsets de un conjunto de datos e incluye distintas técnicas basadas en tablas hash (como DHP) para mejorar la fase de obtención de itemsets frecuentes en el proceso de extracción de reglas de asociación.

TBAR asume que, si un itemset es relevante, también lo han de ser todos sus subconjuntos. Esta propiedad de monotonía es esencial en cualquier algoritmo derivado de Apriori para podar el conjunto de itemsets candidatos que se genera en cada iteración del algoritmo. Aquí se ha utilizado TBAR para extraer los itemsets frecuentes presentes en una base de datos relacional, aunque se podría utilizar cualquier otro criterio de relevancia siempre y cuando se verificase la propiedad de monotonía de los itemsets relevantes (véase la sección

4.4.2).

TBAR se ideó intentando mejorar el rendimiento de Apriori en bases de datos relacionales y, como cualquier otro algoritmo de extracción de reglas de asociación, puede adaptarse para resolver otro tipo de problemas. En el capítulo anterior se vio cómo un algoritmo de extracción de reglas de asociación se puede utilizar para construir el modelo de clasificación ART y en la sección 2.3.2 se citaron otros enfoques alternativos. La escalabilidad de los algoritmos como TBAR permite su uso eficiente en la resolución de problemas de *Data Mining* y, también, en otros problemas aparentemente no relacionados con la extracción de reglas de asociación [152].

En paralelo con nuestro trabajo inicial sobre extracción de reglas de asociación, Bayardo propuso una estructura de datos similar para extraer itemsets de bases de datos especialmente indicada para itemsets de tamaño elevado porque no necesita obtener todos los subconjuntos de un itemset dado para llegar a descubrirlo [14], una propiedad de especial interés en el análisis de secuencias de ADN, por ejemplo. Su algoritmo, denominado MaxMiner, se parece a TBAR aunque ambos fueron diseñados con objetivos diferentes. Aunque son similares, TBAR funciona mejor para resolver problemas de extracción de reglas de asociación mientras que MaxMiner está optimizado para encontrar eficientemente itemsets de un tamaño elevado sin necesidad de tener que encontrar antes todos sus subconjuntos (2^k para un itemset de tamaño k).

Aparte del mencionado trabajo de Bayardo, Han y sus colaboradores también utilizan una estructura de datos en forma de bosque similar a la de TBAR en su algoritmo FP-Growth de extracción de reglas de asociación [74]. Su árbol, al que denominan árbol de patrones frecuentes, almacena el contenido de la base de datos de una forma compacta a partir de la cual pueden extraerse fácilmente todos los itemsets frecuentes presentes en una base de datos. Aunque sólo necesitan recorrer dos veces el conjunto de datos para construir el árbol completo, por desgracia, el tamaño de éste árbol puede llegar a hacerlo inmanejable si se utilizan valores pequeños para el umbral de soporte mínimo.

Como cierre de esta sección, baste mencionar que existen técnicas generales que pueden aplicarse a cualquier algoritmo de extracción de reglas de asociación y, por ende, son aplicables también a TBAR:

- Existen técnicas de actualización incremental que permiten mantener un conjunto de reglas de asociación conforme la base de datos de la que se derivaron se va modificando [31] [32] [60]. Dichas técnicas pueden aplicarse sobre el árbol de itemsets de TBAR de la misma forma que sobre cualquier otro algoritmo de extracción de reglas de asociacíón basado en la filosofía de Apriori.

- Aunque no se ha implementado una versión paralela de TBAR, una versión paralela de nuestro algoritmo puede derivarse con relativa facilidad de las versiones paralelas existentes para Apriori [5] y de otros algoritmos paralelos de extracción de reglas de asociación como los que aparecen en la tabla 2.6.

4.3. $\overline{T}$ en ART: Reglas de asociación con restricciones

Para lograr que el proceso de generación de hipótesis candidatas en ART sea lo más eficiente posible, se puede adaptar fácilmente un algoritmo general de extracción de reglas de asociación como el descrito en la sección anterior de esta memoria.

En esta sección se describe cómo realizar una serie de ajustes sobre el algoritmo TBAR con el objetivo de acelerar, en la medida que sea posible, el proceso de construcción de clasificadores ART:

4.3.1. Extracción de itemsets

La primera tarea que ha de llevar a cabo TBAR para generar las hipótesis candidatas que serán evaluadas por ART para construir un árbol de clasificación consiste en encontrar todos los itemsets relevantes que permanecen ocultos en el conjunto de entrenamiento. En el marco tradicional de extracción de reglas de asociación, el objetivo es encontrar todos los itemsets cuyo soporte iguale, al menos, al umbral mínimo de soporte establecido por el algoritmo ART, que puede variar conforme avanzamos en la construcción del árbol de decisión.

En realidad, sólo estamos interesados en aquellos patrones o itemsets que nos sirvan para derivar reglas de la forma $A \Rightarrow C$, donde A es un itemset de tamaño máximo dado por el parámetro *MaxSize* del algoritmo ART y C es un item de la forma $clase = c_k$ que nos indica la clase correspondiente para los casos de entrenamiento que verifican el antecedente A.

El algoritmo TBAR ha de ir generando todos los k-itemsets de tamaño k hasta llegar a los itemsets de tamaño $MaxSize + 1$ (cuando el tamaño del antecedente es igual a $MaxSize$) para poder derivar a partir de ellos todas las reglas de asociación existentes que tengan el formato $A \Rightarrow C$. Como se comentó en el apartado 4.2.2.4, el algoritmo encargado de extraer las reglas de asociación del árbol de decisión puede modificarse fácilmente para extraer sólo el subconjunto de reglas que verifican el formato requerido por ART.

Además, nos podríamos preguntar si es realmente necesario generar todos los itemsets presentes en la base de datos si sólo nos interesan las reglas que tiene como consecuente un valor para el atributo que determina la clase en nuestro problema de clasificación.

Por desgracia, tanto para generar los itemsets $A \cup C$ como para poder evaluar las reglas $A \Rightarrow C$ es necesario que el árbol de itemsets incluya el patrón dado por el antecedente A. De hecho, las distintas medidas existentes que permiten estimar la validez de una regla dada suelen considerar el soporte del antecedente A para evaluar la regla (véase la sección 4.4.3 de este mismo capítulo), por lo que no podemos omitir la generación del itemset A cuando construimos el árbol de itemsets, aun cuando pudiésemos idear un algoritmo que permitiese obtener todos los patrones $A \cup C$ sin haber descubierto antes el itemset A.

No obstante, el proceso de generación de itemsets puede mejorarse algo si consideramos que sólo se generan itemsets de tamaño $MaxSize + 1$ como máximo. Como no vamos a utilizar ninguna regla que tenga más de $MaxSize$ atributos en su antecedente, al llegar a la etapa de generación de los itemsets de tamaño $MaxSize + 1$, podemos olvidarnos automáticamente de generar como candidatos los itemsets de ese tamaño que no incluyan al atributo de la clase. De esta forma se puede conseguir un ahorro considerable de tiempo de ejecución si establecemos un valor bajo para el parámetro $MaxSize$, ya que

el tamaño del conjunto de candidatos generado en la última iteración se puede reducir ostensiblemente.

4.3.2. Generación de reglas

Una vez que se han obtenido los itemsets relevantes, se pueden extraer todas las reglas de asociación que de ellos se derivan recorriendo directamente el árbol de itemsets. Por tanto, en la implementación ART puede omitirse la fase de generación de reglas del algoritmo de extracción de reglas de asociación, ya que una exploración adecuada del árbol de itemsets es suficiente para llevar a cabo las etapas de extracción y selección de reglas necesarias para construir el clasificador ART. Al no generar explícitamente las reglas, aunque sí implícitamente, la implementación de ART puede ahorrarse el tiempo que sería necesario para construir los objetos que representan a cada una de las reglas, así como el espacio de memoria que ocuparían tales objetos en caso de crearse.

Dado que las únicas reglas en las que estamos interesados han de tener el formato $A \Rightarrow C$, se puede sustituir el primer iterador utilizado por el algoritmo de extracción de reglas de TBAR. Recordemos que este iterador va devolviendo todos los itemsets de tamaño mayor o igual a 2 que estén presentes en el árbol de itemsets, a partir de los cuales se derivarán después las reglas de asociación correspondientes. Sin embargo, a la hora de generar hipótesis candidatas en ART, sólo nos interesan aquellos itemsets en los que esté involucrada la clase en nuestro problema de clasificación (para obtener reglas con la clase en el consecuente). Se puede modificar fácilmente el primer iterador utilizado en la sección 4.2.2.4 de forma que sólo se vayan considerando los itemsets que incluyen atributo de la clase para nuestro problema de clasificación.

Así mismo, el segundo iterador también se puede modificar para agilizar el proceso de extracción de reglas de asociación. Como sólo estamos interesados en reglas del tipo $A \Rightarrow C$, a partir de un itemset $A \cup C$ en el que interviene la clase, generaremos el único subconjunto propio $A \in A \cup C$ que nos sirve para obtener la regla $A \Rightarrow C$.

4.4. Evaluación de las reglas obtenidas

A pesar de haber presentado en esta memoria un algoritmo de construcción de modelos de clasificación que funciona adecuadamente utilizando el modelo clásico de extracción de reglas de asociación, tal como se vio en el capítulo anterior, no carece de interés el estudio de medidas alternativas que permitan evaluar la calidad de las hipótesis generadas durante el proceso de construcción del árbol de decisión ART, aunque solamente sea para que el usuario pueda añadir alguna restricción adicional al tipo de reglas que dan lugar a las hojas del árbol de clasificación construido por ART.

En esta sección se estudian distintas medidas que se pueden utilizar como criterios para evaluar la calidad de las reglas obtenidas en el proceso de extracción de reglas de asociación realizado por algoritmos como TBAR. Estas medidas son de interés, no sólo a la hora de construir clasificadores ART, sino en cualquier proceso de extracción de conocimiento.

El proceso de extracción de conocimiento en bases de datos (KDD) pretende obtener, a partir de una gran cantidad de datos, modelos descriptivos y/o predictivos potencialmente útiles y, a ser posible, desconocidos hasta el momento. En este contexto, el énfasis recae sobre la obtención de conocimiento *potencialmente útil* e indica la importancia que tiene la perspectiva del usuario en KDD. En la práctica, el usuario debe disponer de mecanismos que le permitan guiar el proceso de adquisición de conocimiento en función de sus objetivos. Por desgracia, el papel desempeñado por los objetivos del usuario muchas veces se desdeña por ser excesivamente específico, poco universal y "no científico" (un mero alarde de ingeniería [93]). No obstante, el estudio de medidas alternativas para evaluar la calidad de las reglas obtenidas puede servir para adaptar mejor el proceso de extracción de conocimiento a los objetivos y necesidades del usuario.

En los siguientes apartados se repasan distintas medidas que se pueden utilizar para evaluar la validez de las reglas descubiertas (4.4.3), así como la relevancia de los patrones o itemsets de las que se derivan (4.4.2), haciendo especial hincapié en sus ventajas e inconvenientes de cara a la construcción de clasificadores ART.

4.4.1. Propiedades deseables de las reglas

Antes de pasar a estudiar las medidas concretas conviene revisar lo que entendemos intuitivamente por una regla válida. En principio, cuando evaluamos una regla del tipo $A \Rightarrow C$, sólo la consideraremos útil si estimamos la validez de la implicación $A \rightarrow C$:

- C debería ocurrir más a menudo cuando A se verifica.
- $\neg C$ debería suceder con menos frecuencia cuando se cumple A.

Opcionalmente, puede que nos interese también fijarnos en el grado de cumplimiento de la implicación $\neg C \rightarrow \neg A$, la contraposición de $A \rightarrow C$. $A \rightarrow C$ y $\neg C \rightarrow \neg A$ son completamente equivalentes desde el punto de vista lógico, por lo que podemos considerar que las reglas potencialmente útiles también han de cumplir las siguientes dos propiedades:

- $\neg A$ debería ocurrir más a menudo cuando se verifica $\neg C$.
- A debería ocurrir menos a menudo cuando no se cumple C.

En ocasiones resulta interesante comprobar si se verifican estas dos últimas condiciones, tal como sucede en alguna de las medidas que se estudiarán a continuación [136]. A la hora de construir modelos de clasificación, cuyo objetivo suele ser de tipo predictivo, estas propiedades no tienen relación directa con la precisión del clasificador, al menos en principio, pues lo que más suele importar en esos casos es ser capaces de determinar la clase c_k cuando se verifica el antecedente A de la regla. No obstante, puede que resulte aconsejable tener las propiedades anteriores en cuenta si para nosotros es de interés la interpretabilidad de los modelos de clasificación obtenidos.

Pasemos sin más dilación al análisis de las distintas medidas que nos permiten estimar la calidad de las hipótesis formuladas durante la construcción de un modelo de clasificación como ART. En la sección 4.4.3 se pueden encontrar múltiples medidas de evaluación de reglas, algunas de las cuales tienen en cuenta todas las propiedades deseables que se han mencionado en los párrafos anteriores.

4.4.2. Medidas de relevancia de un itemset

Antes de centrar nuestra atención en la relación existente entre el antecedente y el consecuente de una regla, discutiremos brevemente las formas alternativas de evaluar el interés que para nosotros tendrá un itemset presente en la base de datos.

Supongamos que nuestro patrón (denominado itemset según la terminología usual en extracción de reglas de asociación) es igual a la unión de los patrones que figuran en el antecedente y el consecuente de nuestra regla de asociación $A \rightarrow C$. Es decir, el itemset I será igual a $A \cup C$, donde A es el itemset que aparece en el antecedente de la regla de asociación y C será el item $C = c_k$ que determina la clase c_k en nuestro problema de clasificación.

Se han propuesto distintos criterios para evaluar la relevancia de un patrón o itemset I:

4.4.2.1. Soporte

$$soporte(I) = P(I) = P(A \cap C)$$

Esta es la medida utilizada con mayor asiduidad para evaluar la importancia de I, antes incluso de utilizar cualquier otro criterio de evaluación. Como se comentó al discutir el algoritmo TBAR de extracción de reglas de asociación, el soporte de un itemset verifica una propiedad de monotonía muy importante para permitir un proceso eficiente de extracción de reglas de asociación:

$$X \subset Y \Rightarrow soporte(Y) \leq soporte(X) \tag{4.1}$$

El principal inconveniente de utilizar el umbral de soporte estriba en que ha de fijarse un valor elevado para dicho umbral si queremos evitar una explosión combinatoria durante la generación de reglas de asociación, algo que puede impedir la obtención de reglas potencialmente interesantes que no son muy frecuentes en los datos. Este hecho causó la aparición de la siguiente medida de relevancia de un itemset, si bien no es crítico en la construcción del árbol ART, ya que podemos utilizar un valor alto para el umbral de soporte mínimo de las reglas sin que se degrade el rendimiento del clasificador si empleamos

un valor relativo para dicho umbral (tal como se hizo en los experimentos recogidos en el capítulo anterior de esta memoria).

4.4.2.2. Fuerza colectiva

La fuerza colectiva de un itemset I se define de la siguiente manera [2]:

$$C(I) = \frac{1 - v(I)}{1 - E[v(I)]} \cdot \frac{E[v(I)]}{v(I)}$$

donde $v(I)$ es el índice de 'violación' del itemset I, igual al número de tuplas que contienen algún ítem de I pero no todos, y $E[v(I)]$ es su valor esperado, el cual se calcula suponiendo que existe independencia estadística entre los distintos items del itemset. Si tenemos $I = \{i_1, i_2, .., i_k\}$, entonces $E(v(I)) = 1 - \prod P(i_j) - \prod(1 - P(i_j))$

La fuerza colectiva de un itemset es un número real mayor o igual que cero. Un valor igual a 0 indica una correlación negativa perfecta (cuando la presencia de un ítem excluye la presencia de los demás items del itemset), mientras que un valor igual a 1 se obtiene cuando la presencia del itemset es la que cabría si la aparición de los distintos items individuales se considera independiente.

Esta medida se propuso para sustituir los itemsets frecuentes por itemsets "fuertemente colectivos" que permitan un proceso de obtención de reglas de asociación más eficiente y productivo. El modelo empleado trata de eliminar la generación de reglas espúreas (aquéllas que no aportan nada nuevo), algo común si se utiliza el umbral de soporte mínimo *MinSupp* en la extracción de reglas de asociación. De esta forma, el uso de la medida de fuerza colectiva evita tener que resolver un problema de *Data Mining* de segundo orden a partir del conjunto obtenido de reglas de asociación, cuyo tamaño podría superar con facilidad al del conjunto de datos de entrada.

En cuanto a las propiedades de la fuerza colectiva, Aggarwal y Yu conjeturan [2] que la fuerza colectiva de un itemset es mayor o igual que k_0 si todos los subconjuntos del itemset tienen una fuerza colectiva mayor o igual que k_0. Por tanto, este criterio no verifica la propiedad de monotonía que cumplía el soporte de un itemset y permitía obtener eficientemente el conjunto de itemsets frecuentes usando TBAR o cualquier otro algoritmo derivado de Apriori.

Por otro lado, la utilización del criterio basado en la fuerza colectiva de los itemsets para generar hipótesis candidatas requeriría que el usuario estableciese un nuevo parámetro cuyo valor adecuado y significado no siempre son fáciles de evaluar: un umbral mínimo de fuerza colectiva (mayor que 1 para que tenga sentido).

En resumen, Aggarwal y Yu utilizan su medida de fuerza colectiva como criterio alternativo (o,incluso, ortogonal y complementario) a la medida de soporte para reducir el número de itemsets considerados relevantes y, en consecuencia, la cantidad de reglas de asociación que de ellos se derivan.

A pesar de lo anterior, el uso de la fuerza colectiva como medida de relevancia de los itemsets no parece resultar de especial interés en el modelo de clasificación ART, ya que su implementación con TBAR permite tratar eficientemente el elevado número de reglas candidatas que se pueda derivar de la utilización de un umbral de soporte mínimo.

4.4.3. Medidas de cumplimiento de una regla

En esta sección se estudian distintas medidas que nos permitirán evaluar la validez de una regla derivada de los patrones relevantes presentes en la base de datos teniendo en cuenta la relación entre su antecedente y su consecuente. Usualmente, para evaluar de alguna manera la calidad de una regla $A \Rightarrow C$ se emplean medidas estadísticas que intentan cuantificar la relación existente entre su antecedente A y su consecuente C. A continuación se comentan algunas de ellas, analizando sus propiedades más interesantes de cara a la construcción de un clasificador ART:

4.4.3.1. Confianza

$$confianza(A \Rightarrow C) = P(C|A) = \frac{P(A \cap C)}{P(A)}$$

La confianza es la medida típica utilizada para evaluar la validez de una regla. Su significado es muy intuitivo, pues la confianza de una regla $A \Rightarrow C$ nos indica la proporción de tuplas que, verificando el antecedente, cumplen la regla.

La confianza nos permite medir el grado de asociación existente entre el antecedente y el consecuente de la regla: conforme la confianza de una regla $A \Rightarrow C$ aumenta, la confianza de la regla $A \Rightarrow \neg C$ disminuye al ser $P(C|A) + P(\neg C|A) = 1$.

Sin embargo, la confianza de una regla no permite establecer una relación de causalidad entre el antecedente y el consecuente de la regla, ni puede utilizarse para realizar inferencias, porque no considera el contrapuesto de la regla. En el caso de que deseemos considerar $\neg C \Rightarrow \neg A$ (el contrapuesto de la regla $A \Rightarrow C$) se obtiene una versión causal de la confianza de las reglas [93]:

4.4.3.2. Confianza causal

$$confianza_{causal}(A \Rightarrow C) = \frac{1}{2}(P(C|A) + P(\neg A|\neg C))$$

En la medida de confianza causal se suma la confianza debida a la implicación directa $A \rightarrow C$ y la confianza debida a su contraposición $\neg C \rightarrow \neg A$. El promedio sirve únicamente para que la medida quede normalizada en el intervalo $[0, 1]$.

Esta medida puede devolvernos un valor demasiado alto incluso cuando la implicación directa $A \rightarrow C$ no tiene un soporte significativo pero sí lo tiene su contrarrecíproca $\neg C \rightarrow \neg A$. Por este motivo, la confianza causal no resulta adecuada para resolver problemas de clasificación.

4.4.3.3. Soporte causal

De forma análoga a como se obtiene la medida anterior, también se puede derivar una versión causal de la medida de soporte vista en el apartado anterior de esta sección:

$$soporte_{causal}(A \Rightarrow C) = P(A \cap C) + P(\neg A \cap \neg C)$$

Igual que sucedía con la confianza causal, aun cuando $P(A \cap C)$ sea muy pequeña, puede que el valor de $P(\neg A \cap \neg C)$ sea elevado, por lo que el soporte causal de la regla $A \rightarrow C$ será muy alto (a pesar de no ser la regla excesivamente útil para clasificar un dato cuando se verifique su antecedente). Por tanto, la utilización del soporte causal tampoco resulta interesante en ART.

4.4.3.4. Confirmación

$$confirmación(A \Rightarrow C) = P(A \cap C) - P(A \cap \neg C)$$

Este criterio evalúa una regla teniendo en cuenta cuándo se verifica el antecedente pero no el consecuente de la regla, lo que puede servirnos para resaltar una regla en función de lo común que resulte su antecedente.

Si observamos que $P(A \cap \neg C) = P(A) - P(A \cap C)$, obtenemos que el grado de confirmación de la regla es $P(A \cap C) - P(A \cap \neg C) = 2P(A \cap C) - P(A)$. Como el soporte de la regla viene dado por $P(A \cap C)$ y el soporte del antecedente es $P(A)$, se puede obtener una definición alternativa de la confianza en función del soporte:

$$confirmación(A \Rightarrow C) = soporte(A \Rightarrow C) - soporte(A)$$

de donde se deriva directamente

$$confirmación(A \Rightarrow C) \leq soporte(A \Rightarrow C) \tag{4.2}$$

La medida de confianza se limita a penalizar el soporte de la regla en función del soporte de su antecedente. Dadas dos reglas igual de comunes en el conjunto de datos de entrada, la confirmación selecciona como regla más interesante aquélla cuyo antecedente es menos común en los datos. Esta propiedad hace que la confianza pueda resultar potencialmente útil en la extracción de nuevo conocimiento de una base de datos (al menos, de cara al usuario). Sin embargo, la confianza es de dudosa utilidad en problemas de clasificación al no tener en cuenta la relación entre el antecedente y el consecuente de la regla.

Si bien esta medida no resulta útil de un modo directo en la construcción de clasificadores, la idea en que se basa puede emplearse para obtener tres nuevos criterios:

- La **confirmación causal**, cuando incluimos la regla contrapuesta en nuestra evaluación de $A \Rightarrow C$:

$$confirmación_{causal}(A \Rightarrow C) =$$
$$P(A \cap C) + P(\neg A \cap \neg C) - 2P(A \cap \neg C)$$

que, obviamente, verifica

$$confirmación_{causal}(A \Rightarrow C) \leq soporte_{causal}(A \Rightarrow C) \qquad (4.3)$$

En concreto, la confirmación causal se puede definir en función de medidas ya analizadas de la siguiente manera:

$$\begin{aligned} confirmación_{causal}(A \Rightarrow C) &= soporte_{causal}(A \Rightarrow C) \\ &- soporte(A) \\ &+ soporte(C) \end{aligned}$$

Igual que la confirmación, la confirmación causal penaliza la evaluación de una regla en función del soporte de su antecedente. Por otro lado, cuanto más común sea la clase, más importancia le dará a la regla, lo cual dificulta la construcción de modelos de clasificación cuando existen clases muy poco frecuentes. Además, ya se vio con anterioridad que el soporte causal no resulta adecuado en la resolución de problemas de clasificación, por lo que no le prestaremos más atención a esta medida.

- La **confianza confirmada** se obtiene si modificamos la medida básica de confianza para tener en cuenta la calidad de la regla $A \Rightarrow \neg C$ (es decir, consideramos el caso de que se cumpla el antecedente pero no el consecuente):

$$confianza_{confirmada}(A \Rightarrow C) = P(C|A) - P(\neg C|A)$$

Esta medida, obviamente, cumple la siguiente propiedad

$$confianza_{confirmada}(A \Rightarrow C) \leq confianza(A \Rightarrow C) \qquad (4.4)$$

Si tenemos en cuenta que $P(\neg C|A) = 1 - P(C|A)$, entonces $P(C|A) - P(\neg C|A) = 2P(C|A) - 1$, lo que permite expresar la confianza confirmada como

$$confianza_{confirmada}(A \Rightarrow C) = 2 \cdot confianza(A \Rightarrow C) - 1$$

De la igualdad anterior se deduce que, a la hora de establecer un orden entre las distintas reglas utilizadas como hipótesis candidatas en la construcción del clasificador ART, la confianza confirmada es completamente equivalente a la confianza. De hecho, la confianza confirmada sólo difiere de la confianza en su interpretación semántica, al estar definida sobre el intervalo $[-1, 1]$ en vez de estarlo sobre el intervalo $[0, 1]$.

- Finalmente, la **confianza causal confirmada** se deriva de las definiciones de confianza causal y confianza confirmada:

$$confianza_{causal-confirmada}(A \Rightarrow C) = \frac{1}{2}(P(C|A) + P(\neg A|\neg C) - P(\neg C \cap A))$$

En función de la confianza confirmada, la expresión anterior se puede formular de la siguiente manera:

$$confianza_{causal-confirmada}(A \Rightarrow C) = \frac{1}{2}(confianza_{confirmada}(A \Rightarrow C) + P(\neg A|\neg C))$$

Como se ha comprobado arriba, la confianza confirmada es equivalente a la confianza a la hora de construir modelos de clasificación con ART, por lo que las conclusiones relativas a la confianza causal son aplicables a la confianza causal confirmada, de modo que ésta tampoco resulta adecuada para resolver problemas de clasificación.

4.4.3.5. Convicción

$$convicción(A \Rightarrow C) = \frac{P(A)P(\neg C)}{P(A \cap \neg C)}$$

La convicción es una medida introducida por Brin et al. [24] como alternativa a la confianza utilizada habitualmente para extraer reglas de asociación en bases de datos relacionales. En este contexto, cuando las reglas de asociación vienen caracterizadas por su convicción y no por su confianza se les denomina 'reglas de implicación'.

La convicción, similar en cierta medida a la confirmación, se centra en la relación $A \Rightarrow \neg C$:

$$convicción(A \Rightarrow C) = \frac{soporte(\neg C)}{confianza(A \Rightarrow \neg C)} \tag{4.5}$$

Cuanto menor sea la probabilidad $P(A \cap \neg C)$ mayor será el valor de la convicción de la regla, por lo que podríamos utilizar la convicción para encontrar reglas correspondientes a clases poco comunes.

Sin embargo, el dominio de esta medida no está acotado y no resulta fácil comparar valores de convicción porque las diferencias entre ellas no tienen significado, lo que nos impide idear una heurística de selección automática del umbral de convicción en ART.

Un poco más adelante se verá una medida alternativa, los grados de certeza de Shortliffe y Buchanan, que sí está acotada y es equivalente a la convicción en los casos que nos interesan a la hora de construir clasificadores ART. En concreto, cuando hay al menos dos clases en nuestro problema de clasificación ($soporte(C) < 1$) y la regla es útil para mejorar la precisión del clasificador, lo que sucede cuando su factor de certeza es posititvo ($CF(A \Rightarrow C) > 0$), se puede definir el factor de certeza de la regla como

$$CF(A \Rightarrow C) = 1 - \frac{1}{convicción(A \Rightarrow C)}$$

Utilizando la expresión anterior se evita tener que trabajar con una medida cuyo dominio no esté acotado, sustituyéndola por otra cuyas propiedades se analizarán con detalle un poco más adelante en el apartado 4.4.3.11.

4.4.3.6. Interés

$$interés(A \Rightarrow C) = \frac{P(A \cap C)}{P(A)P(C)} = \frac{P(C|A)}{P(C)}$$

Esta medida [145] hace que, cuanto más comunes sean A y C, menor interés tenga la regla, algo habitual cuando se trata de guiar el proceso de extracción de conocimiento en grandes bases de datos. Entre las propiedades de esta medida destaca su simetría: el interés de $A \Rightarrow C$ equivale al de $C \Rightarrow A$.

Como sucedía con la anterior medida, su dominio no está acotado, lo que puede dificultar su interpretación al utilizarse en el modelo de clasificación ART. En cierto modo, se puede considerar complementaria a la convicción si tenemos en cuenta la siguiente igualdad y la comparamos con la ecuación 4.5, aunque en este caso sí nos centramos en la asociación $A \Rightarrow C$:

$$interés(A \Rightarrow C) = \frac{confianza(A \Rightarrow C)}{soporte(C)} \tag{4.6}$$

4.4.3.7. Dependencia

$$dependencia(A \Rightarrow C) = |P(C|A) - P(C)|$$

Se considera el equivalente discreto a la correlación en dominios continuos. Sin embargo, no resulta adecuada para resolver problemas de clasificación porque su valor es elevado para clases comunes incluso aunque la confianza de la regla $A \Rightarrow C$ sea mínima:

$$dependencia(A \Rightarrow C) = |confianza(A \Rightarrow C) - soporte(C)| \tag{4.7}$$

4.4.3.8. Dependencia causal

$$\begin{aligned} &dependencia_{causal}(A \Rightarrow C) = \\ &\frac{1}{2}((P(C|A) - P(C)) + (P(\neg A|\neg C) - P(\neg A)) \\ &-(P(\neg C|A) - P(\neg C)) + (P(A|\neg C) - P(A))) \end{aligned}$$

Es una variación de la medida de dependencia anterior orientada al análisis de series temporales (cuando A ocurre antes que C). Como sucedía con la anterior, tampoco resulta adecuada para resolver problemas de construcción de modelos de clasificación [93].

4.4.3.9. Medida de Bhandari

$$Bhandari(A \Rightarrow C) = |P(A \cap C) - P(A)P(C)|$$

La medida de Bhandari es otra medida derivada de la medida de dependencia, como demuestra la igualdad siguiente:

$$Bhandari(A \Rightarrow C) = soporte(A) \cdot dependencia(A \Rightarrow C) \tag{4.8}$$

De la expresión anterior se desprende que la medida de Bhandari tampoco será de interés a la hora de construir modelos de clasificación con ART, al no serlo la dependencia.

4.4.3.10. Divergencia Hellinger

La divergencia Hellinger se ideó para evaluar la cantidad de información que aporta una regla [98] y se puede interpretar como una medida de distancia entre las probabilidades a priori y a posteriori de las clases del problema:

$$Hellinger(A \Rightarrow C) = \sqrt{P(A)} \cdot$$
$$[(\sqrt{P(A \cap C)} - \sqrt{P(C)})^2 - (\sqrt{1 - P(A \cap C)} - \sqrt{1 - P(C)})^2]$$

Esta medida ha sido empleada con anterioridad en la construcción de clasificadores y será evaluada con ART en el apartado 4.4.4 de esta memoria.

4.4.3.11. Factores de certeza

Los factores de certeza fueron ideados por Shortliffe y Buchanan para representar la incertidumbre en las reglas del sistema experto de diagnóstico médico MYCIN. Desde entonces, han sido reconocidos como uno de los mejores modelos existentes para el desarrollo de sistemas expertos basados en reglas. De hecho, en [136] ya se ha propuesto su utilización en el ámbito de la extracción de reglas de asociación.

El factor de certeza de una regla de asociación $A \Rightarrow C$ se define como

$$CF(A \Rightarrow C) = \frac{confianza(A \Rightarrow C) - soporte(C)}{1 - soporte(C)}$$

cuando $confianza(A \Rightarrow C) > soporte(C)$,

$$CF(A \Rightarrow C) = \frac{confianza(A \Rightarrow C) - soporte(C)}{soporte(C)}$$

cuando $confianza(A \Rightarrow C) < soporte(C)$, y

$$CF(A \Rightarrow C) = 0$$

en otro caso.

Esta medida de validez de una regla se puede interpretar como el grado de variación de la probabilidad de que C aparezca en una tupla cuando se consideran únicamente las tuplas que contienen A. En otras palabras, cuanto mayor sea un factor de certeza positivo, mayor será la disminución de la probabilidad de que C no esté en una tupla de las que contienen A.

En situaciones extremas, la confianza de una regla determina su factor de certeza:

$$confianza(A \Rightarrow C) = 1 \Rightarrow CF(A \Rightarrow C) = 1$$

$$confianza(A \Rightarrow C) = 0 \Rightarrow CF(A \Rightarrow C) = -1$$

No obstante, los factores de certeza tienen en cuenta la probabilidad de la clase C además de la confianza de la regla y pueden utilizarse en sustitución de la confianza para evaluar las reglas de asociación descubiertas por un algoritmo de extracción de reglas de asociación como TBAR (sección 4.2).

Además, los factores de certeza verifican una propiedad interesante cuando son positivos [136], que es el caso en el que son verdaderamente útiles para resolver problemas de clasificación:

$$CF(A \Rightarrow C) = CF(\neg C \Rightarrow \neg A) \tag{4.9}$$

En la resolución de problemas de clasificación nos podemos encontrar frente a diversas situaciones, para las cuales resulta conveniente analizar el

comportamiento de los factores de certeza como mecanismo de evaluación de las reglas candidatas a formar parte del modelo de clasificación ART:

1. Si existen en nuestro problema clases más comunes que otras y se evalúan dos reglas candidatas con el mismo valor de confianza pero correspondientes a clases de distinta frecuencia, al utilizar factores de certeza se escoge primero la regla correspondiente a la clase menos común, la que permite mejorar más el comportamiento del clasificador.

   Demostración: Supongamos que tenemos

   $$x = confianza(A \Rightarrow C) = confianza(B \Rightarrow D) \leq 1$$

   y

   $$c = soporte(C) > soporte(D) = d$$

   Al ser $(x-1) \leq 0$:

   $$c(x-1) \leq d(x-1)$$
   $$cx - c \leq dx - d$$
   $$-c - dx \leq -d - cx$$

   Sumando $x + cd$ a ambos lados obtenemos

   $$x - c - dx + cd \leq x - d - cx + cd$$

   que se puede expresar como

   $$(x-c)(1-d) \leq (x-d)(1-c)$$

   para llegar a

   $$\frac{x-c}{1-c} \leq \frac{x-d}{1-d}$$

   Esta inecuación se puede expresar como

   $$\frac{confianza(A \Rightarrow C) - soporte(C)}{1 - soporte(C)} \leq \frac{confianza(B \Rightarrow D) - soporte(D)}{1 - soporte(D)}$$

   o, lo que es lo mismo:

   $$CF(A \Rightarrow C) \leq CF(B \Rightarrow D)$$

   Por tanto, partiendo de reglas con la misma confianza se obtiene el resultado más deseable para el clasificador: seleccionar aquella regla que corresponde a la clase menos común (D). QED.

2. En determinadas situaciones, la utilización de factores de certeza en ART es completamente equivalente al uso de la confianza de las reglas.

   Supongamos que tenemos dos reglas con factor de certeza positivo tales que:

   $$CF(A \Rightarrow C) > CF(B \Rightarrow C)$$

   lo que equivale a escribir

   $$\frac{confianza(A \Rightarrow C) - soporte(C)}{1 - soporte(C)} > \frac{confianza(B \Rightarrow C) - soporte(C)}{1 - soporte(C)}$$

   La expresión anterior se puede reducir a la siguiente

   $$confianza(A \Rightarrow C) > confianza(B \Rightarrow C)$$

   en dos situaciones

   - Cuando ambas reglas se refieren a una misma clase c_k.
   - Cuando las reglas corresponden a distintas clases que tienen la misma probabilidad: $soporte(c_1) = soporte(c_2)$.

   Por tanto, ART seleccionará siempre las reglas en el mismo orden independientemente de si se utiliza la confianza o los factores de certeza para evaluar las reglas cuando (a) las reglas se refieren a una misma clase o (b) las clases a las que se refieren las reglas son equiprobables.

3. Existen ocasiones, no obstante, en las que el comportamiento de ART varía si empleamos los factores de certeza de las reglas en sustitución de su confianza (esto es, cuando un factor de certeza mayor no implica necesariamente una confianza mayor).

   Demostración: Comprobemos qué sucede cuando el factor de certeza varía de una regla a otra y, además, la probabilidad de las clases que aparecen en los consecuentes de las reglas es diferente.

   Dadas dos reglas cuyos factores de certeza son positivos

   $$CF(A \Rightarrow C) > CF(B \Rightarrow D)$$

Supongamos que

$$c = soporte(C) \neq soporte(D) = d$$

$$x = confianza(A \Rightarrow C) \qquad confianza(B \Rightarrow D) = y$$

Se obtiene la siguiente desigualdad:

$$\frac{x-c}{1-c} > \frac{y-d}{1-d}$$

expresión que, simplificada, se queda en

$$x(1-d) - c > y(1-c) - d$$

Para completar la demostración distinguimos dos casos diferentes:

- Cuando la clase más común corresponde a la regla con mayor factor de certeza ($c > d$)

  $$x(1-d) - d > x(1-d) - c > y(1-c) - d$$

  $$x(1-d) > y(1-c)$$
  $$x > \frac{1-c}{1-d} y$$

  Es decir,

  $$confianza(A \Rightarrow C) > K \cdot confianza(B \Rightarrow D)$$

  $$K = \frac{soporte(\neg C)}{soporte(\neg D)} < 1$$

- Cuando la regla con mayor factor de certeza corresponde a la clase menos común ($c < d$)

  $$x(1-c) - c > x(1-d) - c > y(1-c) - d$$

  $$x(1-c) - c > y(1-c) - d$$
  $$x > y - \frac{d-c}{1-c}$$

  Esto es,

  $$confianza(A \Rightarrow C) > confianza(B \Rightarrow D) - \Delta$$

  $$\Delta = \frac{soporte(D) - soporte(C)}{soporte(\neg C)} > 0$$

En resumen, aunque el factor de certeza de una regla está íntimamente relacionado con su confianza, un factor de certeza mayor no implica necesariamente una confianza mayor:

$$CF(A \Rightarrow C) > CF(B \Rightarrow D)$$

$$\not\Rightarrow$$

$$confianza(A \Rightarrow C) > confianza(B \Rightarrow D)$$

La frecuencia relativa de cada clase determina el orden en que ART selecciona las reglas si utilizamos como medida de validez su factor de certeza. Es decir, el comportamiento de ART variará si empleamos el factor de certeza de una regla a la hora de evaluar hipótesis alternativas. De hecho, puede suceder que ART elija para ramificar el árbol de decisión reglas que, intuitivamente, no resultan adecuadas para construir un clasificador preciso.

Por ejemplo, supongamos que se nos presentan dos reglas:

- $A \Rightarrow c_1$ con soporte igual al 2 % y confianza 50 %.
- $B \Rightarrow c_2$ con soporte igual al 50 % y confianza 95 %.

El factor de certeza para la primera de ellas sería igual a 3/8 mientras que para la segunda regla vale 3/4 ($15/20 = 6/8 > 3/8$) si tenemos una clase c_1 con probabilidad 0.2 y c_2 con probabilidad 0.8. Si las probabilidades de las clases difiriesen, pasando a valer 0.1 y 0.9, el factor de certeza de la primera regla sería igual a 4/9 (40/90) mientras que para la segunda valdría 1/2 (5/10), que sigue siendo mayor que para la primera regla. Sin embargo, si la clase c_1 tuviese probabilidad 0.94 y la probabilidad de c_2 fuese 0.06, entonces el factor de certeza de la primera regla pasaría a ser 46/94 y el de la segunda sería ahora igual a 1/6 (¡menor que el factor de certeza de la primera regla!).

Por tanto, debemos tener muy en cuenta las características de los factores de certeza al utilizarlos para resolver problemas de clasificación. En ocasiones pueden resultar útiles (p.ej. a la hora de incluir en nuestro modelo de clasificación reglas relativas a clases muy poco frecuentes), aunque también pueden acabar seleccionando reglas de una forma poco intuitiva (como en el caso descrito en el párrafo anterior, donde una pequeña variación en las proporciones de

las clases conduce a la obtención de un modelo de clasificación radicalmente distinto y, probablemente, mucho más complejo).

4.4.3.12. Cuestión de utilidad

Las propiedades de los factores de certeza descritas en el apartado anterior, no obstante, sugieren una posible mejora al criterio básico empleado por ART para evaluar las reglas obtenidas en la etapa de extracción de reglas.

Como se comentó antes, a la hora de construir clasificadores, sólo son verdaderamente interesantes aquellas reglas cuyo factor de certeza es positivo, pues son las que acrecientan nuestro conocimiento y permiten mejorar el porcentaje de clasificación. Por definición, un factor de certeza positivo se obtiene para una regla $A \Rightarrow C$ si

$$confianza(A \Rightarrow C) > soporte(C)$$

Cuando construimos un clasificador, esta condición nos indica que añadir la regla al modelo de clasificación construido es mejor que emplear una clase por defecto, al menos respecto a la clase C en el conjunto de entrenamiento.

Lo anterior nos sugiere añadir una restricción adicional a las reglas que consideran potencialmente interesantes. Si tenemos en cuenta las definiciones del soporte y la confianza vistas con anterioridad, esta restricción adicional, que se corresponde con la condición de arriba, se puede escribir como

$$P(A \cap C) > P(A) \cap P(C)$$

Podemos establecer, pues, un 'criterio de utilidad' que restrinja el conjunto de reglas consideradas si requerimos que las reglas evaluadas por ART verifiquen siempre la condición anterior. Este criterio, además, evita que tengamos que modificar la medida de evaluación empleada habitualmente en el proceso de extracción de reglas de asociación (esto es, la confianza de las reglas).

Los resultados que se han obtenido al utilizar este y otros criterios para evaluar la calidad de las hipótesis candidatas en la construcción de clasificadores ART se muestran en la sección siguiente.

4.4.4. Resultados experimentales

El algoritmo ART no aboga por ninguna medida específica para evaluar las reglas a partir de las cuales se construye el clasificador ART, por lo que se puede incorporar cualquiera de las medidas descritas en este capítulo.

Se ha realizado un estudio experimental para estimar la idoneidad de las distintas medidas propuestas en la sección anterior a la hora de construir clasificadores. Estos experimentos se realizaron empleando los mismos conjuntos de datos utilizados para evaluar ART en el capítulo anterior (tabla 3.5), utilizando en todas las pruebas validación cruzada (10-CV), un umbral de soporte mínimo relativo igual al 5 % de las tuplas y la heurística de selección automática del umbral descrita en la sección 3.1.3 (aplicada, en esta ocasión, a medidas de evaluación distintas a la confianza). Los resultados obtenidos empíricamente se resumen en la tabla 4.5 y aparecen representados gráficamente en las figuras de las páginas siguientes.

Las conclusiones que se pueden derivar de los resultados obtenidos se pueden resumir en los siguientes puntos:

- El criterio de utilidad propuesto al final de la sección anterior de esta memoria (apartado 4.4.3.12) consigue mejorar la precisión del clasificador ART de una forma consistente, tal como podíamos esperar (figura 4.10). Dicho criterio construye árboles con más hojas que la versión básica de ART (figura 4.12), lo que se traduce en un tiempo de entrenamiento algo mayor (figura 4.13) al tener que recorrer más veces el conjunto de entrenamiento para construir el clasificador (figura 4.14).

- El uso de los factores de certeza, por su parte, no logra mejorar el rendimiento de ART. Esto probablemente se deba a algunas propiedades de los factores de certeza que resultan poco intuitivas a la hora de construir clasificadores (apartado 4.4.3.11).

	Confianza	Utilidad	CF	Convicción	Interés	Hellinger
Precisión (10-CV)	79.22 %	83.10 %	77.67 %	77.79 %	53.71 %	48.05 %
Tiempo de entrenamiento	18.5s	22.4s	12.7s	10.7s	2.9s	1.8s
Topología del árbol						
- Hojas del árbol	36.9	50.0	28.5	30.0	4.2	1
- Nodos internos	18.3	25.2	16.2	17.3	2.6	0
- Profundidad media	7.41	8.71	7.39	7.33	2.12	1.00
Operaciones de E/S						
- Registros	36400	50100	33100	26700	20300	7000
- Recorridos	63	89	55	59	11	3

Tabla 4.5: Resumen de los experimentos realizados con distintas medidas de evaluación de las reglas.

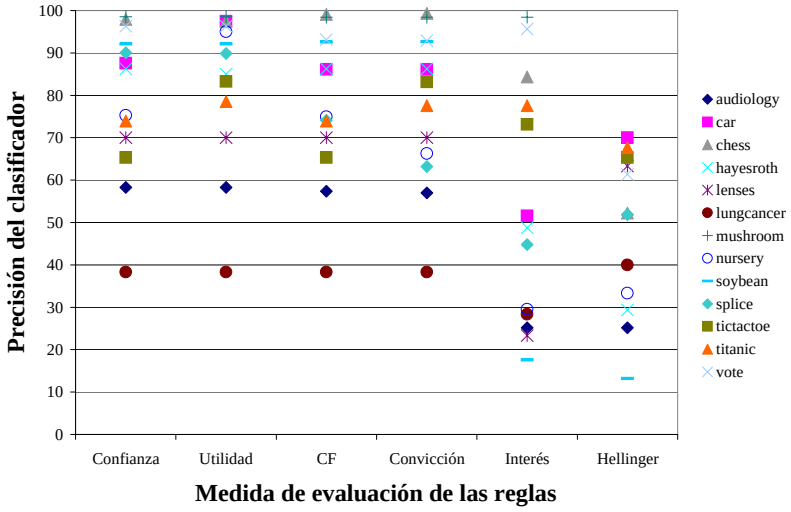

Figura 4.10: Precisión del clasificador ART para distintas medidas de evaluación de las reglas.

- Como cabría esperar, la convicción consigue resultados similares a los factores de certeza. Este hecho se debe a que, desde el punto de vista de ART, la convicción y los factores de certeza son equivalentes en los casos en que resulta adecuado añadir una regla al modelo de clasificación ART (esto es, cuando el factor de certeza es positivo), tal como se mencionó en el apartado 4.4.3.5.

- Al utilizar el interés (apartado 4.4.3.6) como medida de evaluación de las reglas, también se obtienen los resultados que intuitivamente se podían esperar: como el dominio de esta medida no está acotado, resulta inútil la heurística de selección automática del umbral utilizada en la construcción del clasificador ART. La introducción de un margen de tolerancia no resulta adecuada para esta medida por su definición. En este caso, puede que resultase más conveniente emplear un factor de tolerancia en vez de un margen de tolerancia.

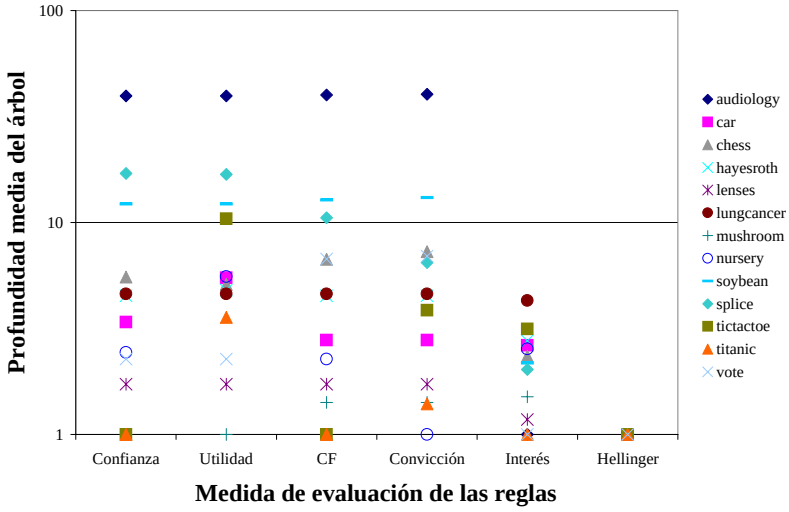

Figura 4.11: Profundidad del árbol ART para distintas medidas de evaluación de las reglas.

Sin embargo, la propia definición de la medida de interés dificulta que podamos establecer, en general, un valor inicial deseable para esta medida de evaluación de las reglas. El valor adecuado para este parámetro dependerá del problema, por lo que el interés no parece ser una alternativa viable para la evaluación de las reglas en ART. Podemos concluir, por tanto, que será preferible la utilización de medidas acotadas como la confianza (y su variante, el criterio de utilidad) o los factores de certeza.

- La misma discusión realizada para la medida de interés es aplicable a la divergencia Hellinger (apartado 4.4.3.10). Aunque el dominio de esta medida sí está acotado, la interpretación y comparación de valores de divergencia Hellinger resulta difícil, por lo que tampoco se recomienda su utilización. De hecho, en la experimentación no se construyeron buenos clasificadores ART con esta medida por no establecer adecuadamente un valor inicial deseable de divergencia Hellinger (necesario para la heurística de selección automática del umbral empleada por ART).

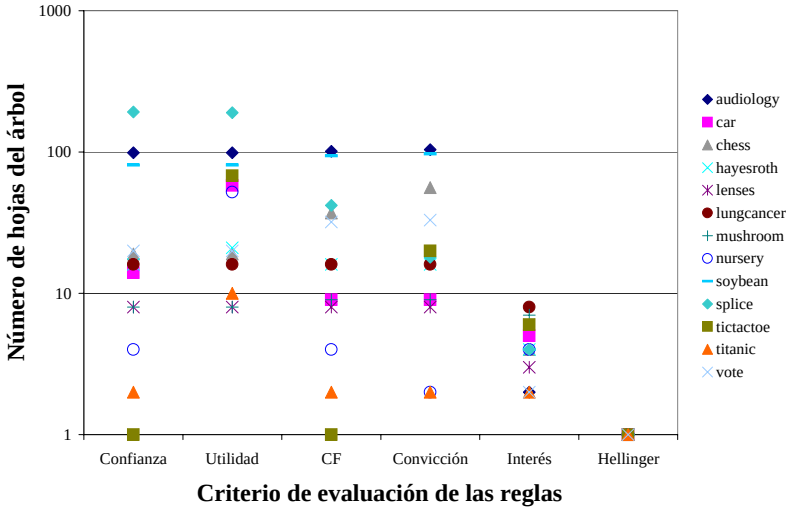

Figura 4.12: Número de hojas del árbol ART para distintas medidas de evaluación de las reglas.

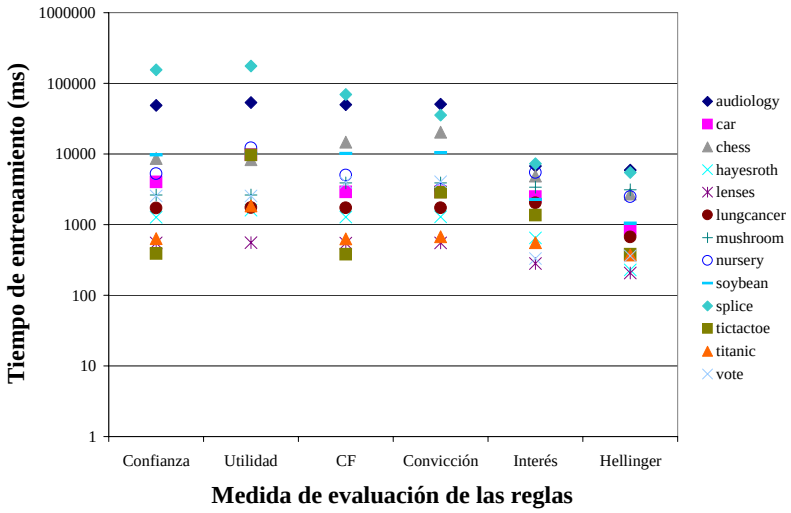

Figura 4.13: Tiempo de entrenamiento para distintas medidas de evaluación de las reglas.

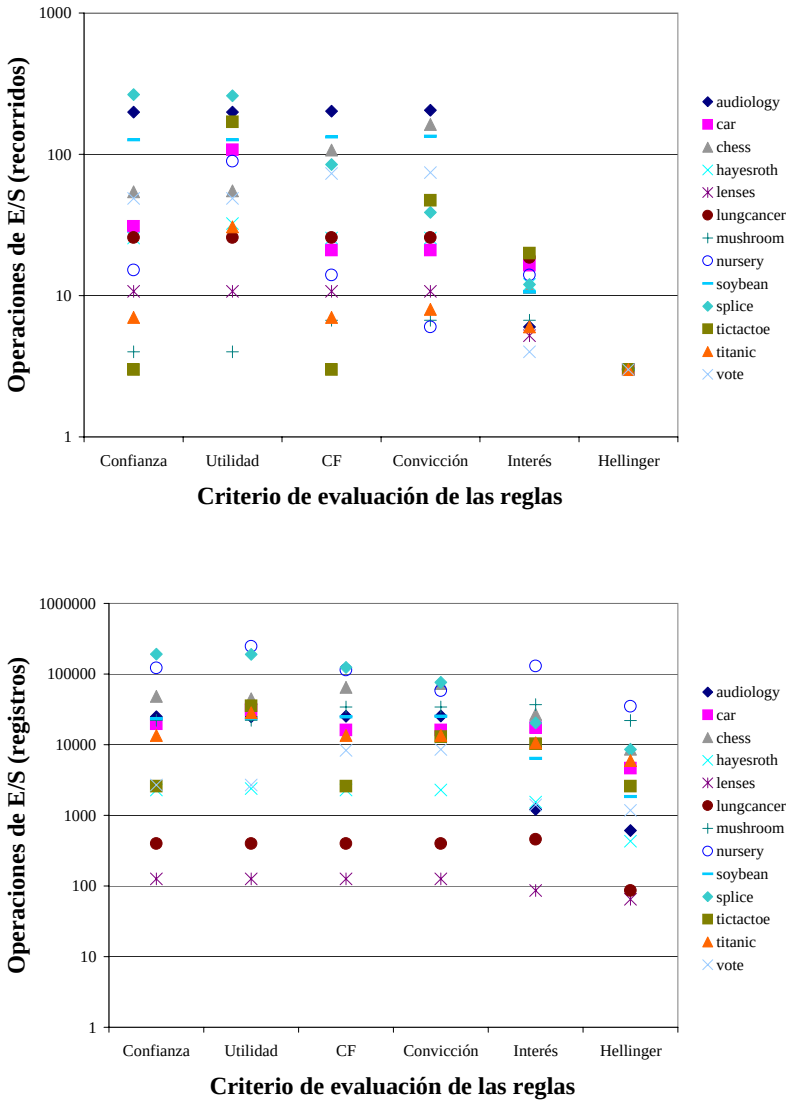

Figura 4.14: Número de operaciones de E/S para distintas medidas de evaluación de las reglas.

En resumen, de las medidas discutidas, sólo resultan adecuadas, como alternativa general a la confianza, el criterio de utilidad y los factores de certeza. La convicción logra buenos resultados pero, al ser equivalente a los factores de certeza en las situaciones que nos interesa y no estar acotado su dominio, se prefiere el empleo de factores de certeza. Un dominio no acotado también es el principal obstáculo con el que nos encontramos al emplear la medida de interés. Finalmente, la divergencia Hellinger resulta poco adecuada por ser difícil de interpretar para el usuario.

A pesar de lo anterior, hay que destacar que todos los criterios expuestos pueden resultar adecuados en situaciones concretas cuando el usuario desea obtener un modelo de clasificación cuyas reglas componentes verifiquen propiedades concretas, por lo que siempre es deseable poner a su disposición un amplio abanico de posibilidades, con el objetivo de permitirle guiar el proceso de extracción de conocimiento.

Además, gracias a la utilización de un algoritmo general de extracción de reglas de asociación como TBAR, empleado aquí como parte de ART, el uso de una o otra medida no influye en el coste computacional que supone la construcción del clasificador. Este coste es, sencillamente, proporcional a la complejidad del modelo construido, tal como se puede apreciar en las figuras de las páginas anteriores.

Capítulo 5

Manejo de atributos continuos

No todo lo que puede contarse cuenta, y no todo lo que cuenta puede contarse

ALBERT EINSTEIN

En conjuntos de datos reales, no todos los datos con los que nos encontramos son de tipo nominal o categórico, sino que también suele haber datos de tipo numérico. Costes, precios de venta, tamaños o pesos son algunos ejemplos de cantidades que pueden aparecer como atributos numéricos en cualquier base de datos. Es más, en el ámbito de las bases multidimensionales y de las aplicaciones OLAP, todas las medidas con las que se trabaja son cantidades numéricas que resumen la información de una empresa respecto a una serie de dimensiones de tipo categórico, las cuales se pueden tratar a distintos niveles de granularidad definiendo jerarquías de conceptos.

Resulta necesario, por tanto, disponer de métodos que nos permitan trabajar con atributos continuos a la hora de construir un modelo de clasificación como el propuesto en el capítulo 3. Las técnicas de discretización comentadas en la sección 5.1 nos permitirán ampliar el ámbito de aplicación de ART y de otros muchos algoritmos de aprendizaje. La sección 5.2 presenta un método alternativo de discretización que puede ser aplicado de una forma eficiente para construir árboles de decisión n-arios con atributos continuos. El método

propuesto se añade así al repertorio de técnicas de discretización empleadas usualmente para construir árboles de decisión, que son objeto de estudio en la sección 5.3. Finalmente, en el apartado 5.4, se describen algunos resultados obtenidos experimentalmente al emplear distintas técnicas de discretización para construir árboles de clasificación con un algoritmo clásico como C4.5 y con el modelo de clasificación ART.

5.1. La discretización de espacios continuos

Cuando trabajamos con modelos simbólicos como ART y nos encontramos con patrones definidos sobre un espacio continuo (esto es, ejemplos de entrenamiento cuyos atributos toman valores de tipo numérico), no nos queda más remedio que agrupar los datos de los que disponemos de la forma que resulte más útil para su procesamiento posterior.

5.1.1. El caso general: Métodos de agrupamiento

El *agrupamiento* de un conjunto de patrones mediante la utilización de alguna métrica de similitud que resulte adecuada es un problema bastante estudiado en Inteligencia Artificial, también conocido como aprendizaje no supervisado en contraste con el aprendizaje supervisado realizado al construir modelos de clasificación a partir de ejemplos previamente etiquetados.

En su caso general, los métodos de agrupamiento intentan, dado un conjunto de patrones definidos sobre un espacio multidimensional, agrupar dichos patrones en un pequeño número de agrupamientos, de forma que cada agrupamiento contenga únicamente aquellos patrones que sean similares en cierta medida. Este proceso puede verse también como una clasificación no supervisada de los patrones del conjunto de entrenamiento: el proceso de generar automáticamente clases que no están predefinidas. Tanto en el capítulo 8 del libro de Han y Kamber [75] como en los apuntes de clase de Ullman [153] se pueden encontrar excelentes introducciones a este tema.

Igual que los algoritmos TDIDT objeto de esta memoria, los métodos de agrupamiento son métodos heurísticos, si bien los primeros suelen emplear

medidas de pureza en sus reglas de división (sección 2.1.1) mientras los segundos utilizan medidas de similitud como las descritas en el anexo 5.5 para decidir cómo se agrupan los datos.

En su incansable búsqueda de mejores métodos de agrupamiento, los investigadores en el tema han propuesto numerosos algoritmos. Muchos de estos algoritmos son iterativos y su grado de complejidad varía desde la sencillez de métodos de agrupamiento como el algoritmo de las K Medias hasta algoritmos altamente parametrizados como ISODATA, acrónimo de *Iterative Self-Organizing Data Analysis Techniques* con la A final añadida para hacer pronunciable su nombre. Algoritmos iterativos de este tipo aparecen descritos con detalle en cualquier libro de texto de Reconocimiento de Formas [54] [151] y suelen caracterizarse porque los agrupamientos obtenidos dependen del orden de presentación de los patrones del conjunto de entrenamiento. Esta dependencia se puede disminuir si se emplean estrategias modernas de búsqueda como GRASP [45], acrónimo de *Greedy Randomized Adaptive Search Procedure*. También existen métodos de agrupamiento basados en modelos teóricos, como las técnicas basadas grafos, si bien suelen resultar poco prácticos para resolver problemas reales por motivos de eficiencia computacional (cualquier técnica de complejidad superior a $O(n \log n)$ es de poca utilidad en la resolución de problemas de *Data Mining*).

El gran número de métodos heurísticos de agrupamiento existentes puede clasificarse, a grandes rasgos, en dos grandes grupos: los métodos basados en centroides y los métodos jerárquicos de agrupamiento.

- Los métodos de agrupamiento basados en centroides utilizan prototipos (puntos centrales o centroides) para caracterizar los agrupamientos y asignan cada patrón al agrupamiento cuyo centroide está más cercano. El conocido algoritmo de las K Medias pertenece a esta categoría y es uno de los más sencillos.

- Los métodos de agrupamiento jerárquico son incrementales y pueden ser aglomerativos o divisivos. Aunque en español quizá sería más correcto denominarlos aglomerantes y divisores, aquí se emplea la terminología usual en Reconocimiento de Formas:

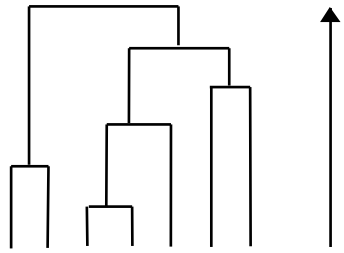

Figura 5.1: Dendrograma que ilustra el funcionamiento de un método de agrupamiento jerárquico.

- Los métodos jerárquicos aglomerativos son métodos de tipo ascendente. Partiendo de un agrupamiento para cada patrón del conjunto de entrenamiento, van combinando los agrupamientos más cercanos hasta que se cumpla algún criterio de parada. El proceso de agrupamiento, cuando se realiza en una única dimensión, se puede dibujar mediante un dendrograma como el de la figura 5.1, donde la flecha indica la evolución del método de agrupamiento aglomerativo.

- A diferencia de los métodos aglomerativos, los métodos jerárquicos divisivos son de tipo descendente. Comenzando con un único agrupamiento que contiene todos los patrones del conjunto de entrenamiento, se va dividiendo este agrupamiento hasta que se verifique el criterio de parada del algoritmo. En este caso, el dendrograma de la figura 5.1 se construiría de arriba hacia abajo.

Los métodos de agrupamiento jerárquico se han utilizando ampliamente en distintos ámbitos. En [133], por ejemplo, puede encontrarse una descripción detallada del uso de este tipo de algoritmos en la resolución de problemas de recuperación de información.

5.1.2. El caso unidimensional: Métodos de discretización

Dividir los valores de un atributo continuo en un conjunto de intervalos adyacentes corresponde al caso unidimensional de los métodos de agrupamiento. Este caso, conocido como **discretización**, es de especial importancia en Inteligencia Artificial, pues permite que muchos algoritmos de aprendizaje ideados para funcionar con atributos nominales o categóricos puedan también utilizarse con conjuntos de datos que incluyen valores numéricos [85], algo esencial en la resolución de problemas reales.

La discretización de los valores no sólo permite construir modelos de clasificación más compactos y sencillos, que resultan más fáciles de comprender, comparar, utilizar y explicar, sino que permite mejorar la precisión del clasificador y hace más rápido el aprendizaje [51]. En otras palabras, "la discretización amplía las fronteras de muchos algoritmos de aprendizaje" [85].

En problemas de clasificación, la discretización suele realizarse para cada atributo continuo por separado por cuestiones de eficiencia. Al considerar cada atributo numérico por separado, sólo hay que agrupar los patrones unidimensionales definidos por los valores del atributo continuo. Cualquier método de agrupamiento de los mencionados en el apartado 5.1.1 puede emplearse para resolver este problema, incluso los métodos de agrupamiento basados en grafos. En el caso unidimensional, la implementación de estos métodos es bastante sencilla y además resulta óptima desde un punto de vista teórico, ya que se maximiza la distancia entre agrupamientos (v.g. construyendo un árbol generador minimal [20]). Sin embargo, estos modelos clásicos de aprendizaje no supervisado no tienen en cuenta el contexto en el que se aplican (el problema de clasificación), algo que sí hacen otros métodos existentes, como veremos a continuación.

5.1.2.1. Clasificación

Los métodos de discretización empleados como herramienta auxiliar para resolver problemas de aprendizaje supervisado pueden clasificarse atendiendo a varios criterios:

- **Métodos de discretización supervisada vs. Métodos de discretización no supervisada** (dependiendo de si se utiliza o no información sobre las clases para agrupar los datos).

  Los métodos supervisados emplean información sobre la clase a la que pertenece cada caso del conjunto de entrenamiento a la hora de evaluar y escoger los puntos de corte que definen los intervalos en que quedan agrupados los valores numéricos; los métodos no supervisados no tienen en cuenta esta información.

  Los métodos de discretización no supervisada utilizan la distribución de valores de un atributo continuo como única fuente de información, algo que no parece ser adecuado a la hora de agrupar valores numéricos en intervalos adyacentes si tenemos más información disponible, como sucede al intentar construir un modelo de clasificación. En cambio, las técnicas de discretización supervisada, como el algoritmo propuesto en la sección 5.2, sí tienen en cuenta la información adicional de la que disponemos acerca de las clases cuando intentamos resolver un problema de clasificación.

- **Métodos de discretización local (dinámicos) vs. Métodos de discretización global (estáticos)**.

  En un método global, la discretización se realiza a priori. Los métodos locales, en cambio, discretizan los atributos continuos en una región localizada del espacio (p.ej. la correspondiente a un nodo de un árbol de decisión) y, por lo tanto, la discretización que realizan de un atributo dado puede no ser única para todo el espacio definido por los patrones del conjunto de entrenamiento.

  Hay que tener en cuenta, no obstante, que la distinción entre métodos locales y métodos globales de discretización es una cuestión de uso independiente del algoritmo de discretización concreto que se utilice.

5.1.2.2. Algoritmos de discretización

Los métodos de discretización más simples crean un número preestablecido de intervalos. *Equiwidth* y *Equidepth* [75, p.110] son ejemplos no supervisados de este tipo: *Equiwidth* crea intervalos de la misma amplitud, mientras que *Equidepth* define intervalos de la misma frecuencia en el conjunto de entrenamiento. El discretizador 1R de Holte [83] es una variante supervisada de este tipo de métodos, que ajusta los límites de los intervalos en función de la clase a la que pertenezcan los casos del conjunto de entrenamiento. Existen incluso otros métodos, como el no supervisado basado en la regla 3-4-5 [75, p.135], que intentan conseguir intervalos que nos resulten más fáciles de leer e interpretar (como el intervalo $[1000, 2000]$), ya que los intervalos generados por otros métodos suelen resultar poco intuitivos ($[978, 2163]$, por ejemplo).

La Teoría de la Información también ha originado la aparición de algunas técnicas de discretización, como el uso de la ganancia de información o del criterio de proporción de ganancia en la construcción de árboles de decisión binarios con atributos continuos (sección 5.3.1.1). El método de discretización supervisada de Fayyad e Irani [58], basado en el principio MDL de Rissanen, es un ejemplo destacado de este tipo de métodos. La distancia de Màntaras [105] puede emplearse como alternativa a la propuesta de Fayyad e Irani, al igual que la medida de contraste presentada por Van de Merckt [154].

Zeta [82] pertenece a otro tipo de técnicas de discretización. Zeta mide el grado de asociación entre los valores nominales de dos atributos categóricos: el atributo continuo discretizado y la clase en el problema de clasificación. Zeta selecciona los intervalos que consiguen la máxima precisión del clasificador que resulta de intentar predecir la clase en función única y exclusivamente del atributo discretizado. El estadístico χ^2 también puede utilizarse en métodos de discretización alternativos (como ChiMerge, Chi2 o ConMerge).

En el informe [85] puede encontrarse un amplio repaso de los métodos citados así como una taxonomía que permite categorizarlos. La referencia [51] también puede ser un buen punto de partida para todo aquél que esté interesado en el tema.

5.2. Discretización contextual: Un enfoque alternativo

En la sección anterior se presentaron algunas nociones de aprendizaje no supervisado y se comentaron varios métodos clásicos de discretización. En ésta se introduce una técnica que nos permite discretizar los valores de un atributo continuo atendiendo a la similitud existente entre las distribuciones de las clases para cada valor o conjunto de valores adyacentes del atributo continuo que se desea discretizar. En secciones posteriores se verá cómo puede emplearse este método de discretización para construir árboles de clasificación n-arios con atributos continuos.

El método de discretización propuesto en esta sección es un método de agrupamiento jerárquico y un método de discretización supervisada atendiendo a los criterios de clasificación expuestos en la sección anterior de esta memoria.

5.2.1. La discretización contextual como método de discretización supervisada

Al construir un árbol de decisión, un método no supervisado no tiene en cuenta si la partición generada del conjunto de entrenamiento conduce a un árbol de decisión mejor o no. Es decir, no considera el contexto en el que se ha de realizar la discretización. Por lo tanto, desde un punto de vista conceptual, el uso exclusivo de la distribución de valores de un atributo continuo a la hora de discretizarlo no resulta del todo adecuado para resolver problemas de clasificación. Sin embargo, éste es el enfoque utilizado por muchos de los métodos de discretización: los métodos no supervisados (Equidepth, Equiwidth, 3-4-5, k-Means...).

A pesar de que los métodos de discretización existentes se suelen basar en la utilización de alguna medida de pureza asociada a cada intervalo resultante del proceso de discretización, el método de discretización contextual mide la similitud entre valores adyacentes para seleccionar los puntos de corte que determinan los intervalos en que se discretiza un atributo continuo. La primera estrategia es la usual en la construcción de árboles de decisión con algoritmos TDIDT, mientras que la opción escogida es la empleada por la mayor parte de métodos de agrupamiento estándar (apartado 5.1.1).

A la hora de realizar la discretización, obviamente, sólo tienen significado los intervalos resultantes de agrupar valores adyacentes en el sentido tradicional, aquellos valores más cercanos entre sí utilizando cualquier métrica de distancia (ya sea la euclídea, la de Manhattan o la de Mahalanobis), pues en el caso unidimensional se mantiene la relación de adyacencia entre los distintos valores numéricos independientemente de la métrica empleada. En realidad, no será necesario calcular ninguna distancia si ordenamos los valores del atributo continuo que aparecen en el conjunto de entrenamiento de forma que los valores adyacentes numéricamente estén en posiciones consecutivas.

Si tenemos en cuenta lo anterior, nos basta medir la similitud entre pares de valores adyacentes para decidir cómo han de agruparse los valores de un atributo continuo. La similitud existente entre pares de valores adyacentes se puede medir utilizando cualquiera de los modelos descritos en el apartado 5.5, teniendo en cuenta que la discretización contextual emplea la distribución de las clases para caracterizar cada valor del atributo (como método de discretización supervisada que es) en vez de utilizar directamente los valores del atributo, como harían los métodos de agrupamiento tradicionales (no supervisados).

Combinando las ideas expuestas en los dos párrafos anteriores se consigue un algoritmo de discretización dependiente del contexto (supervisado) que utiliza las técnicas habituales de los métodos de agrupamiento (no supervisados).

De hecho, el método de discretización contextual puede verse como un algoritmo de agrupamiento estándar si se redefine la noción de patrón empleada usualmente por estos métodos. En vez de considerar el patrón constituido por el vector característico de un ejemplo de entrenamiento (esto es, el vector dado por los valores del ejemplo para cada uno de los atributos de nuestro problema), se utiliza la distribución de clases para un valor de un atributo continuo como patrón que caracteriza el valor del atributo.

Con el fin de que los resultados obtenidos sean interpretables y tengan sentido, se establece un orden entre los valores del atributo como restricción adicional: dos valores x e y de un atributo, con $x < y$, estarán en un mismo agrupamiento (intervalo) sólo si todos los valores z presentes en el conjunto de entrenamiento tales que $x < z < y$ también pertenecen a dicho agrupamiento.

5.2.2. La discretización contextual como método de discretización jerárquica

El método general de discretización contextual, que primero ordena los valores de un atributo y después emplea las diferencias existentes entre las distribuciones de clases para los distintos valores del atributo, se ha de plantear como un método de agrupamiento jerárquico.

Una vez que tenemos el conjunto ordenado de valores de un atributo continuo, podemos utilizar cualquiera de los criterios descritos en la sección 5.5 para decidir qué pares de valores o intervalos adyacentes combinar (si implementamos nuestro proceso de discretización como un algoritmo de agrupamiento jerárquico aglomerativo) o cómo seleccionar el punto de corte por el cual dividir un intervalo dado (si empleamos un enfoque divisivo).

5.2.2.1. Discretización contextual aglomerativa

Los pasos correspondientes a la implementación aglomerativa del método de discretización contextual se especifican en la figura 5.2. Partiendo de un agrupamiento (intervalo en nuestro caso) para cada valor del atributo continuo que queremos discretizar, se selecciona en cada iteración el par de intervalos adyacentes cuyas distribuciones de clases sean más similares según el criterio de similitud seleccionado por el usuario, que puede ser cualquiera de los que se describen en el apartado 5.5. Los intervalos así seleccionados se funden en un único intervalo que los reemplaza.

A la hora de implementar el algoritmo, se ha de establecer de antemano un criterio de parada que se verifique cuando la discretización del atributo continuo alcance las propiedades que el usuario desee. A continuación se describen algunos de los criterios de parada más usuales:

- El criterio de parada más elemental consiste en establecer de antemano el número de agrupamientos o intervalos que deseamos generar (de forma idéntica a como funcionan algoritmos no supervisados como el de las K Medias, Equiwidth o Equidepth, e incluso algunos métodos supervisados como Zeta).

1. Crear un intervalo para cada valor del atributo continuo.
2. Identificar los dos intervalos adyacentes más similares.
3. Combinar los dos intervalos identificados en el punto anterior.
4. Mientras queden más de dos intervalos y no se verifique ningún criterio de parada establecido por el usuario, volver al paso 2.

Figura 5.2: Versión aglomerativa del método de discretización contextual.

- Algunos métodos de discretización establecen algún tipo de umbral que determina hasta dónde ha de seguir el proceso de discretización jerárquica (como los basados en el estadístico χ^2).
- También se puede emplear algún criterio de parada automático como los propuestos por los métodos que utilizan conceptos de Teoría de la Información (vg: el principio MDL de Rissanen).

Independientemente del criterio que se emplee, su idoneidad dependerá de las características del conjunto de datos concreto y de los objetivos que el usuario tenga en mente al realizar la discretización.

5.2.2.2. Discretización contextual divisiva

El algoritmo correspondiente a la implementación divisiva del método de discretización contextual se describe en la figura 5.3. En este caso, se parte de un único intervalo que abarca el dominio completo del atributo continuo en el conjunto de entrenamiento. En cada iteración, se escoge un punto de corte por el cual dividir el intervalo actual en dos subintervalos, de forma que se maximice la disimilitud existente entre las distribuciones de clases de los intervalos resultantes (utilizando cualquier criterio de los que aparecen en la sección 5.5).

Como al dividir un intervalo en dos se modifica la similitud existente entre dichos intervalos y los intervalos adyacentes al intervalo original, se pueden

1. Comenzar con un único intervalo que incluya todos los valores del atributo continuo.
2. Identificar el punto de corte que da lugar a la configuración de intervalos más disimilares entre sí.
3. Utilizar dicho punto de corte para dividir en dos uno de los intervalos de la configuración actual de intervalos.
4. Mientras haya menos intervalos que valores diferentes y no se verifique ningún criterio de parada establecido por el usuario, volver al paso 2.

Figura 5.3: Implementación divisiva del método de discretización contextual.

tener en cuenta estos intervalos al seleccionar el punto de corte óptimo.

Sean $[a, v_{min-1}]$, $[v_{min}, v_{max}]$ y $[v_{max+1}, b]$ tres intervalos consecutivos que se utilizan para discretizar los valores de un atributo continuo cualquiera. Al evaluar un punto de corte x correspondiente al intervalo $[v_{min}, v_{max}]$ para dividir éste en dos subintervalos $[v_{min}, x]$ y $(x, v_{max}]$, se ha de minimizar la siguiente función de energía:

$$\begin{aligned} \Delta Energía &= s([a, v_{min-1}], [v_{min}, x]) \\ &+ s([v_{min}, x], (x, v_{max}]) \\ &+ s([x, v_{max}], [v_{max+1}, b]) \\ &- s([a, v_{min-1}], [v_{min}, v_{max}]) \\ &- s([v_{min}, v_{max}], [v_{max+1}, b]) \end{aligned}$$

donde s representa la medida de similitud utilizada para evaluar la similitud entre intervalos adyacentes, los tres primeros sumandos corresponden a la configuración final en que quedan los intervalos y los dos últimos términos corresponden al estado inicial del conjunto de intervalos.

Podría haberse aplicado un razonamiento similar al caso anterior, cuando los intervalos se construyen empleando un método aglomerativo. Sin embar-

go, en la versión aglomerativa no es tan importante incluir el entorno de los intervalos que se funden. Dada una serie de intervalos consecutivos I_1, I_2, I_3 e I_4, cuando se decide fusionar un intervalo I_2 con su vecino I_3 empleando el método aglomerativo, ya sabemos que ni I_2 se parece demasiado a I_1 ni I_3 se asemeja a I_4 más de lo que lo hacen I_2 e I_3 (pues, en tal caso, se fusionarían esos intervalos en vez de I_2 e I_3).

Al emplear el método divisivo, no obstante, no sabemos qué relación guardan los intervalos recién creados con su entorno, por lo que se emplea la función de energía de la página anterior para no crear parejas de intervalos que difieran entre sí pero sean similares a los intervalos ya existentes a su alrededor.

5.2.2.3. Eficiencia de la discretización contextual

En la discretización contextual, cualquier modificación que se realice sobre un conjunto de intervalos será una modificación local que no afectará globalmente a todos los intervalos, al existir un orden lineal preestablecido para los intervalos en que se discretiza un atributo continuo.

En la versión aglomerativa del método, al combinar dos intervalos adyacentes en uno nuevo, sólo se ve modificada la similitud existente entre los intervalos adyacentes al intervalo recién creado y dicho intervalo. Por tanto, ya que todas las demás medidas de similitud no se ven afectadas por el cambio en la configuración de los intervalos, no es necesario recalcularlas en cada iteración.

De igual forma, en la implementación divisiva del método de discretización contextual, sólo es necesario recalcular la función de similitud para los intervalos modificados en cada iteración del algoritmo, independientemente de si se usa o no una función de energía para guiar el método de agrupamiento.

Esta interesante propiedad del método de discretización contextual implica que su implementación puede realizarse eficientemente. Dado que sólo hay que actualizar localmente el conjunto de intervalos actual, cada iteración del algoritmo puede realizarse en $O(\log N)$ pasos si mantenemos indexados los intervalos de la configuración actual, siendo N el número de valores diferentes del atributo continuo. Obviamente, como mucho se tendrán que efectuar $N-1$

iteraciones. Por tanto, la discretización contextual es de orden $O(NlogN)$, a diferencia de los métodos generales de agrupamiento jerárquico, los cuales suelen ser de orden $O(N^2)$.

5.2.3. Uso de la discretización contextual como método de discretización local

Por su carácter iterativo, cualquier método de discretización jerárquico nos ofrece la posibilidad de ir evaluando las distintas configuraciones de intervalos que se generan en cada iteración. Esta propiedad hace que el método de discretización contextual propuesto en esta memoria resulte especialmente interesante como método de discretización local durante la construcción de árboles de decisión con algoritmos TDIDT, tal como se verá en la sección 5.3.2.

Cuando se construye un árbol de decisión y se emplea el método de discretización aquí propuesto, la topología del árbol de decisión construido dependerá del conjunto de datos particular y no será necesario establecer de antemano su factor de ramificación cuando se emplean atributos continuos. Además, al ser un método de discretización eficiente ($O(NlogN)$), la complejidad algorítmica del proceso de construcción global del árbol de decisión no se verá afectada, lo que nos permite aplicar esta técnica en problemas de *Data Mining* para ampliar la flexibilidad de cualquier algoritmo TDIDT, desde C4.5 [131] hasta RainForest [69].

La construcción de árboles de decisión con atributos continuos será objeto de estudio en la sección 5.3, pero antes veremos un ejemplo de funcionamiento del método de discretización propuesto.

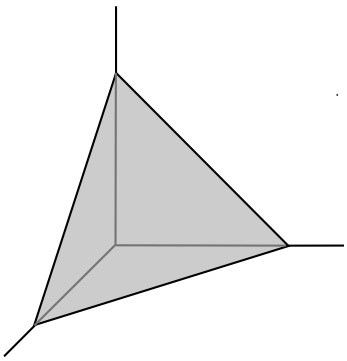

Figura 5.4: Plano que representa los valores posibles para las probabilidades de las distintas clases cuando $J = 3$.

5.2.4. Un pequeño ejemplo

Dado un conjunto de intervalos adyacentesI_1, I_2.. I_n para el atributo continuo A, caracterizamos cada uno de esos intervalos con la distribución de clases presente en los ejemplos de entrenamiento correspondientes a cada intervalo. Si existen J clases diferentes, cada intervalo I tendrá un vector característico asociado $V_I = (p_1, p_2, .., p_J)$, en donde p_j es la probabilidad de la j-ésima clase en el conjunto de ejemplos de entrenamiento cuyo valor de A está incluido en el intervalo I.

Supongamos que nuestro problema de clasificación fuese binario (esto es, $J = 2$). Para un subconjunto dado del conjunto de entrenamiento, la probabilidad de que el ejemplo pertenezca a la clase j se denota p_j. Como la suma de las probabilidades ha de ser igual a 1, en nuestro problema de clasificación binario tenemos que $p_2 = 1 - p_1$. Esto es, los valores posibles para las probabilidades en nuestro problema de decisión binario pueden representarse como una recta en $\mathbb{R}_2$.

Análogamente, los valores posibles para las probabilidades de las clases cuando tenemos tres de ellas ($J = 3$) puede representarse como el plano $p_1 + p_2 + p_3 = 1$ mostrado en la figura 5.4. Esta figura nos permite observar, de un modo gráfico, que la distancia euclídea definida sobre los vectores

característicos $(p_1, p_2, .., p_J)$ puede servir de candidata para medir la similitud entre dos distribuciones de clases distintas. En realidad, la distancia euclídea mide su disimilitud, pues la distancia disminuye cuanto más similares son las distribuciones. En la sección 5.5 se describirán otras medidas alternativas a la distancia euclídea.

Consideremos un problema de clasificación con tres clases. Sea A un atributo continuo con cuatro valores distintos presentes en el conjunto de entrenamiento (v_1, v_2, v_3 y v_4). En esta situación, cada valor v_i del atributo continuo A viene caracterizado por un vector tridimensional V_i que contiene, en su j-ésima componente, la frecuencia relativa de la clase j para los ejemplos del conjunto de entrenamiento cuyo valor de A es v_i. De modo alternativo, también se podría emplear una estimación laplaciana de estas probabilidades:

$$V_i(j) = \frac{n_j + 1}{N + J}$$

donde n_j es el número de ejemplos de la clase j correspondientes a v_i, N es el tamaño del conjunto de entrenamiento y J el número de clases en nuestro problema.

Al utilizar la versión aglomerativa del método contextual, se parte inicialmente de un intervalo para cada valor del atributo A. Supongamos que nuestros vectores característicos quedan de la siguiente manera:

$$V_1 = (0{,}3, 0{,}4, 0{,}3)$$

$$V_2 = (0{,}2, 0{,}6, 0{,}2)$$

$$V_3 = (0{,}8, 0{,}1, 0{,}1)$$

$$V_4 = (0{,}6, 0{,}4, 0{,}0)$$

Si estuviésemos empleando el método de discretización como herramienta auxiliar durante la construcción de un árbol de decisión (sección 5.3.2), evaluaríamos el árbol cuaternario resultante de utilizar cada uno de los cuatro valores para construir un subárbol.

A continuación, se combinan los dos intervalos adyacentes cuyas distribuciones de clases son más similares. Si decidimos emplear la distancia euclídea

$d_2^2(V_i, V_j)$ para medir la disimilitud existente entre vectores característicos, obtenemos lo siguiente:

$$d_2^2(V_1, V_2) = 0{,}06$$

$$d_2^2(V_2, V_3) = 0{,}62$$

$$d_2^2(V_3, V_4) = 0{,}14$$

Al ser la distancia euclídea una medida de disimilitud, (V_1, V_2) es el par de vectores más similar. Por tanto, combinamos los valores v_1 y v_2 para obtener $V_{12} = (0{,}25, 0{,}5, 0{,}25)$ suponiendo que tanto v_1 como v_2 representan al mismo número de ejemplos de entrenamiento.

Entonces podríamos evaluar la calidad de la partición actual del conjunto de valores continuos del atributo A, lo cual nos conduciría, en el contexto de un algoritmo TDIDT, a la construcción de un árbol de decisión ternario: el resultante de emplear $\{v_1, v_2\}$, $\{v_3\}$ y $\{v_4\}$ como conjuntos discretos de valores.

De nuevo, volvemos a calcular la distancia entre los intervalos adyacentes para obtener:

$$d_2^2(V_{12}, V_3) = 0{,}3475$$

$$d_2^2(V_3, V_4) = 0{,}14$$

Nótese que la segunda distancia $d_2^2(V_3, V_4)$ sigue siendo la misma que antes, por lo que no sería necesario calcularla nuevamente.

Ahora decidimos combinar $\{v_3\}$ y $\{v_4\}$ pues son los intervalos adyacentes más similares. Nos quedan, de este modo, dos agrupamientos: $\{v_1, v_2\}$ y $\{v_3, v_4\}$.

Si estuviésemos construyendo un árbol de decisión, evaluaríamos el árbol binario correspondiente y finalizaríamos la ejecución del algoritmo ya que no tiene sentido continuar combinando intervalos adyacentes.

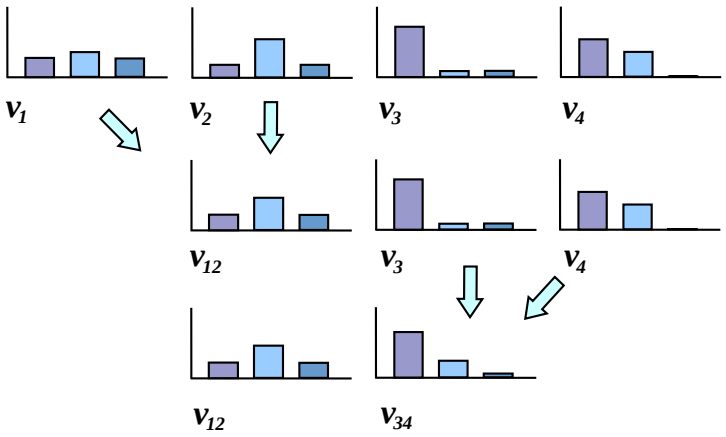

Figura 5.5: Agrupamiento contextual de intervalos adyacentes.

El proceso completo que hemos seguido para discretizar los valores del atributo continuo A aparece dibujado esquemáticamente en la figura 5.5.

Como ya se comentó en la sección anterior, hay que resaltar que sólo hay que volver a calcular dos medidas de similitud/disimilitud en cada iteración de este algoritmo. Cuando v_i se combina con v_{i+1}, hay que obtener la distribución de clases resultante $V_{i,i+1}$. Una vez que disponemos de ella, sólo tenemos que evaluar la similitud entre $V_{i,i+1}$ y sus nuevos vecinos (los intervalos adyacentes que corresponden a V_{i-1} y V_{i+2}). Las demás distribuciones de clases y medidas de similitud permanecen inalterables, por lo que no es necesario calcularlas nuevamente.

Aunque en este ejemplo se ha empleado la distancia euclídea, también se podría haber utilizado otra medida para evaluar la similitud entre las distribuciones de clases. La tabla 5.1 recoge algunas de las medidas de similitud que pueden emplearse en la discretización contextual (y, por extensión, en cualquier método de agrupamiento jerárquico tradicional). En el anexo 5.5 se puede encontrar una descripción detallada de las distintas medidas que figuran el la tabla 5.1.

Modelos basados en medidas de distancia

Distancia de Minkowski	$d_r(x,y) = \left(\sum_{j=1}^{J} \lvert x_j - y_j \rvert^r\right)^{\frac{1}{r}}, \quad r \geq 1$
- Distancia euclídea	$d_2(x,y) = \sqrt{\sum_{j=1}^{J} (x_j - y_j)^2}$
- Manhattan	$d_1(x,y) = \sum_{j=1}^{J} \lvert x_j - y_j \rvert$
- Dominio	$d_\infty(x,y) = \text{máx}_{j=1..J} \lvert x_j - y_j \rvert$
Bhattacharyya	$R(x,y) = \sqrt{1 - \sum_{j=1}^{J} \sqrt{x_j, y_j}}$

Modelos basados en medidas de correlación

Producto escalar	$S.(x,y) = x \cdot y = \sum_{j=1}^{J} x_j y_j$
Índice de correlación	$\rho(x,y) = 1 - \left(\frac{4}{N_x + N_y}\right) d_2^2$
	donde $N_v = \sum_{j=1}^{J} (2v_j - 1)^2$

Modelos basados en Teoría de Conjuntos

Modelo de Tversky	$s(a,b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A)$,
	donde $\theta, \alpha, \beta \geq 0$
- Restle	$-S_{Restle}(A,B) = \lvert A \square B \rvert$
	$-S_\square(A,B) = \sup_x \mu_{A \square B}(x)$
- Intersección	$S_{MinSum}(A,B) = \lvert A \cap B \rvert$
	$-S_{Enta}(A,B) = 1 - \sup_x \mu_{A \cap B}(x)$
Modelo proporcional	$s(a,b) = \dfrac{f(A \cap B)}{f(A \cap B) + \alpha f(A - B) + \beta f(B - A)}$
	donde $\alpha, \beta \geq 0$
- Gregson	$S_{Gregson}(A,B) = \dfrac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$

Notas:

$\lvert A \rvert = \sum_x \mu_A(x)$
$\mu_{A \cap B}(x) = \text{mín}\{\mu_A(x), \mu_B(x)\}$
$\mu_{A \cup B}(x) = \text{máx}\{\mu_A(x), \mu_B(x)\}$
$\mu_{A \square B}(x) = \text{máx}\{\text{mín}\{\mu_A(x), 1 - \mu_B(x)\}, \text{mín}\{1 - \mu_A(x), \mu_B(x)\}\}$

Tabla 5.1: Algunas medidas de similitud

5.3. Atributos continuos en árboles de decisión

En esta sección se comentan las alternativas disponibles a la hora de construir árboles de decisión con atributos continuos (sección 5.3.1), tras lo cual se analizan las aplicaciones de los métodos de discretización presentados en las secciones anteriores de este capítulo en la construcción de clasificadores que utilizan árboles de decisión como modelo de representación del conocimiento obtenido.

Posteriormente, se propone entrelazar un método jerárquico de discretización con el proceso de evaluación de hipótesis alternativas realizado para ramificar el árbol de decisión (sección 5.3.2). Este método nos permitirá construir árboles n-arios arbitrarios sin degradar la precisión del clasificador obtenido. Además, gracias a la eficiencia de métodos de discretización como el propuesto en la sección 5.2, se consigue un mayor grado de libertad sin que la complejidad computacional del proceso de construcción del árbol se vea afectada.

5.3.1. Árboles binarios vs. árboles n-arios

Como se vio en el capítulo 2, algunos algoritmos de inducción de árboles de decisión, como CART, construyen árboles binarios, mientras que los algoritmos pertenecientes a la familia de ID3 prefieren construir árboles n-arios. Algoritmos del tipo de C4.5, que construyen árboles de decisión n-arios para atributos discretos, añaden una rama al árbol para cada uno de los valores posibles del atributo. En ocasiones se permiten tests más complejos en los cuales se agrupan valores para reducir el factor de ramificación del árbol, algo similar en cierto sentido a las ramas ‘else’ del árbol de decisión construido por ART. CART, por su parte, lleva esta situación al extremo: todos los valores del atributo se agrupan en dos conjuntos para construir un árbol de decisión binario, incluso aunque el atributo sea de tipo categórico.

5.3.1.1. Árboles binarios con atributos continuos

Los algoritmos TDIDT, tanto los pertenecientes a la familia de ID3 como CART, suelen construir árboles binarios cuando trabajan con atributos continuos. Cuando el conjunto de entrenamiento incluye atributos de tipo numérico, el árbol de decisión se ramifica usualmente empleando tests binarios de la forma $atributo \leq umbral$, donde el umbral se escoge de forma que la partición del conjunto de entrenamiento generada por el test minimice la medida de impureza utilizada como regla de división al construir el árbol.

El formato de los tests utilizados se restringe para poder examinar todos los tests candidatos de una forma eficiente. Tales tests sólo involucran a un único atributo de tipo numérico para que los árboles resultantes sean más fáciles de comprender y se evite la explosión combinatoria que resultaría si se permitiese la aparición de múltiples atributos en un único test [131], como sucede en el caso de las combinaciones lineales en CART [23].

Para escoger el umbral utilizado en el árbol de decisión, se comprueban todos los valores posibles de dicho umbral teniendo en cuenta que, si utilizamos una medida de impureza como criterio de división, basta con evaluar aquellos puntos de corte que se encuentran en el límite entre dos clases [58]. El proceso de evaluación de los puntos de corte candidatos se puede efectuar de una manera eficiente si ordenamos el conjunto de entrenamiento utilizando los valores del atributo. Dado que un atributo sólo puede tomar un conjunto finito de valores $\{v_1, v_2..v_n\}$ en el conjunto de entrenamiento (ya que éste es también finito), cualquier umbral t_i existente entre v_i and v_{i+1} tendrá el mismo efecto al dividir el conjunto de entrenamiento. Por tanto, sólo hay que comprobar un máximo de $n - 1$ posibles umbrales para cada atributo numérico presente en el conjunto de entrenamiento. Aunque pueda parecer un proceso computacionalmente costoso, la evaluación de los distintos umbrales posibles puede realizarse con un único recorrido secuencial de los datos una vez que éstos han sido ordenados. Además, el rendimiento del algoritmo de inducción puede mejorarse si se emplean conjuntos AVC (RainForest [69]) o se implementan técnicas escalables como el middleware descrito en [30].

Una vez que se ha establecido que el mejor umbral posible debe encontrar-

se entre v_i y v_{i+1}, se ha de seleccionar un valor concreto para el umbral que aparecerá en el árbol de decisión. La mayor parte de los algoritmos escogen el punto medio del intervalo $[v_i, v_{i+1}]$:

$$t_i = \frac{v_i + v_{i+1}}{2}$$

C4.5, no obstante, prefiere escoger el mayor valor de A presente en el conjunto de entrenamiento que no excede del punto medio del intervalo $[v_i, v_{i+1}]$, es decir:

$$t_i = max \left\{ v \mid v \leq \frac{v_i + v_{i+1}}{2} \right\}$$

De esta forma, C4.5 se asegura de que cualquier valor utilizado como umbral en el árbol de decisión tiene sentido, pues aparece en el conjunto de entrenamiento, si bien es cierto que el umbral seleccionado puede resultar engañoso si la muestra del conjunto de valores del atributo presente en el conjunto de entrenamiento no es representativa.

Para evitar problemas de este tipo, aquí se propone utilizar un enfoque ligeramente distinto consistente en seleccionar el umbral entre v_i y v_{i+1} en función del número de casos de entrenamiento que queden por encima y por debajo del umbral. Si tenemos L ejemplos de entrenamiento con valor $v \leq v_i$ y R casos con $v \geq v_{i+1}$, el umbral t_i se define de la siguiente forma:

$$t_i = \frac{R * v_i + L * v_{i+1}}{L + R}$$

El número R de ejemplos cuyo valor de A es mayor que el umbral multiplica a la cota inferior v_i, mientras que la cota superior v_{i+1} se multiplica por el número de ejemplos que quedan por debajo del umbral (L). De esta forma, el umbral estará más cerca de v_i que de v_{i+1} cuando haya menos casos de entrenamiento con $v \leq v_i$. Igualmente, el umbral estará más cerca de v_{i+1} cuando la mayor parte de los ejemplos del conjunto de entrenamiento tengan un valor de A menor o igual a v_i.

Aunque esta ligera modificación del algoritmo usual no supondrá, por lo general, ninguna mejora sobre la precisión final del clasificador, los árboles de decisión resultantes parecen ser más adecuados. Esto es así especialmente

cuando los árboles no están balanceados, una situación bastante común cuando hay atributos continuos en el conjunto de entrenamiento. De hecho, a pesar de que normalmente no se mencione, C4.5 incluye un parámetro que, por defecto, establece que ninguna de las dos ramas resultantes al ramificar un árbol utilizando un atributo continuo puede tener menos del 20 % de los casos de entrenamiento [131].

5.3.1.2. Árboles n-arios con atributos continuos

La utilización de técnicas que permitan construir árboles n-arios puede, en principio, conducir a una reducción en la complejidad del modelo construido sin deteriorar la precisión del clasificador. Además, los árboles n-arios tienden a ser menos profundos y suelen resultar más fáciles de interpretar para los humanos.

Cuando sólo empleamos atributos de tipo categórico o atributos continuos previamente discretizados, algoritmos como C4.5 son capaces de construir árboles n-arios directamente. Sin embargo, cuando los atributos son continuos, los árboles de decisión resultantes son exclusivamente binarios, puesto que los nodos internos del árbol de decisión construido sólo incluirán tests binarios de la forma $atributo \leq valor$. La utilización de esos tests binarios implica que un atributo continuo puede aparecer varias veces en un camino desde la raíz hasta una hoja del árbol. Si bien es verdad que tales repeticiones de un atributo podrían simplificarse al convertir el árbol de decisión en un conjunto de reglas, no es menos cierto que el árbol construido tendrá más hojas, será innecesariamente más profundo y resultará más difícil de interpretar que si empleásemos algún mecanismo que nos permitiese construir árboles n-arios con atributos continuos. Tal como se comenta en [132], “divisiones no binarias sobre atributos continuos hacen que los árboles sean más fáciles de comprender y también parece conducir a árboles más precisos en algunos dominios”. Este hecho sugiere la posibilidad de utilizar métodos de discretización como los comentados en la sección 5.1 al construir un árbol de decisión cuando existen atributos continuos en el conjunto de entrenamiento.

Si utilizásemos tests n-arios, en vez de tests exclusivamente binarios, y pudiésemos determinar con una certeza suficiente que la división realizada del

conjunto de entrenamiento nos permite discriminar bien las distintas clases de nuestro problema de clasificación, entonces podríamos descartar el atributo numérico tras su utilización. De esta forma no podrían aparecer atributos repetidos en un camino del árbol desde la raíz hasta una hoja, de modo que el árbol resultante sería potencialmente más pequeño y más fácil de comprender para nosotros. Además, su construcción sería más rápida y su precisión no se vería afectada significativamente.

En la práctica, se pueden construir árboles n-arios directamente si agrupamos los valores numéricos de un atributo continuo en una serie de intervalos antes de construir el árbol y consideramos los intervalos seleccionados como si fuesen valores de un atributo categórico. Esta solución corresponde a la utilización de cualquier método de *discretización global*.

También existe la posibilidad de agrupar los valores numéricos de un atributo continuo dado en función de la situación particular en cada nodo del árbol. De esta forma, los intervalos utilizados para ramificar el árbol de decisión podrán variar ajustándose mejor a los datos del conjunto de entrenamiento que correspondan a cada nodo del árbol. Esta estrategia se conoce con el nombre de *discretización local*.

En el apartado siguiente se propone un método que permite utilizar discretización local de tal forma que no es necesario establecer de antemano el factor de ramificación del árbol de decisión.

5.3.2. Discretización local jerárquica en árboles n-arios

Aplicado localmente al construir cada nodo del árbol, cualquier método de discretización jerárquica, como el método de discretización contextual presentado en la sección 5.2, permite la construcción de árboles de decisión n-arios sin establecer a priori los intervalos en los que se han de agrupar los valores de un atributo continuo.

Si bien se han empleado métodos de discretización, usualmente jerárquica, para obtener un conjunto de intervalos con los cuales ramificar un árbol n-ario (p.ej. [58]), el conjunto de intervalos obtenido por el método de discretización fija de antemano el factor de ramificación del árbol de decisión. Incluso cuando se emplea discretización local, los métodos de discretización jerárquica se

aplican a priori sobre el conjunto de datos de entrenamiento y el conjunto de intervalos que generan sólo se evalúa como alternativa para ramificar el árbol tras finalizar del proceso de discretización (cuando el algoritmo de discretización termina su ejecución al verificarse el criterio de parada que utilice, que puede ser cualquiera de los mencionados en el apartado 5.2.2).

Por lo general, no se ha tenido en cuenta la estructura iterativa de los métodos jerárquicos de discretización para dotar de mayor flexibilidad al proceso de construcción del árbol de decisión. Si en vez de esperar la terminación del proceso de discretización, evaluamos en cada iteración del algoritmo jerárquico el conjunto de intervalos actual, podemos conseguir un proceso más flexible en el que el factor de ramificación del árbol no lo determina de antemano ninguna regla de parada del método de discretización, sino que lo escoge la propia regla de división que se utiliza en los algoritmos TDIDT de construcción de árboles de decisión.

En consecuencia, la técnica de discretización utilizada es completamente ortogonal a la regla de división empleada para construir el árbol de decisión, ya sea ésta el criterio de proporción de ganancia de C4.5, el índice de diversidad de Gini o cualquier otra medida de impureza. De este modo, se obtiene un método de discretización que no impone ninguna restricción sobre los demás parámetros del algoritmo TDIDT (algo que sí hacen algoritmos como el discretizador MDLP de Fayyad e Irani [58]).

5.3.2.1. Versión aglomerativa

Si empleamos como técnica de discretización un método jerárquico aglomerativo, disponemos inicialmente de un intervalo para cada valor de un atributo continuo y, en cada iteración, se combinan los dos intervalos adyacentes más similares para reducir el número de intervalos en que se discretiza el atributo continuo. Este proceso se puede repetir hasta que sólo queden dos intervalos, que son los que servirían para construir un árbol de decisión binario si se seleccionasen como mejor partición posible del conjunto de entrenamiento. Ahora bien, cada vez que combinamos dos intervalos adyacentes, podemos comprobar la medida de impureza asociada al árbol de decisión que construiríamos si empleásemos el conjunto actual de intervalos para ramificar

1. Crear un intervalo para cada valor del atributo continuo.
2. Identificar los dos intervalos adyacentes más similares.
3. Combinar los dos intervalos identificados en el punto anterior.
4. Evaluar el árbol de decisión que resultaría de utilizar el conjunto actual de intervalos para ramificar el árbol de decisión.
5. Si quedan más de dos intervalos, volver al paso 2.

Figura 5.6: Versión aglomerativa de la discretización jerárquica entrelazada con el proceso de evaluación de hipótesis candidatas que realiza cualquier algoritmo TDIDT.

el árbol. Conforme vamos evaluando alternativas, se registra la mejor partición obtenida hasta el momento, de forma que al final se utilizará el conjunto de intervalos que mejor se adapte a nuestro problema independientemente de su cardinalidad, algo que no permiten las propuestas previas de aplicación de métodos de discretización a la construcción de árboles de decisión.

El algoritmo resultante de entrelazar el método jerárquico aglomerativo de discretización con el proceso de construcción del árbol de decisión puede expresarse como aparece en la figura 5.6. Como se puede apreciar en dicha figura, el entrelazamiento entre el método de discretización y el proceso de construcción de árboles de decisión es lo que nos permite emplear tests n-arios sobre un atributo continuo sin tener que establecer de antemano el número de intervalos.

Por otra parte, cuando se emplea un método aglomerativo, no tiene sentido evaluar las particiones generadas durante las primeras iteraciones (cuando cada valor del atributo continuo en el conjunto de entrenamiento constituye un intervalo por sí mismo) y puede ser útil emplear otro método de discretización para generar la configuración inicial de los intervalos (por ejemplo, un método sencillo como *Equiwidth* o, especialmente, *Equidepth*). De esta forma se puede reducir la influencia que pueda tener sobre el proceso de discretización la

1. Discretizar los valores del atributo utilizando un método sencillo (por ejemplo, *Equidepth*).
2. Evaluar el árbol de decisión que resultaría de utilizar el conjunto actual de intervalos para ramificar el árbol de decisión.
3. Identificar los dos intervalos adyacentes más similares.
4. Combinar los dos intervalos identificados en el punto anterior.
5. Evaluar el árbol de decisión que resultaría de utilizar el conjunto actual de intervalos para ramificar el árbol de decisión.
6. Si quedan más de dos intervalos, volver al paso 3.

Figura 5.7: Versión aglomerativa con pre-discretización del método propuesto.

presencia de ruido en el conjunto de entrenamiento (y se disminuye el número de veces que se ha de medir la similitud entre intervalos adyacentes cuando empleamos el discretizador contextual de la sección 5.2). El método propuesto quedaría entonces como se muestra en la figura 5.7

5.3.2.2. Variante divisiva

De forma análoga a como se obtiene el algoritmo de la figura 5.6, se puede formular la versión divisiva del método jerárquico propuesto. Si deseamos emplear un método jerárquico divisivo para ramificar un árbol de decisión n-ario podemos emplear el algoritmo de la figura 5.8.

En este caso, resulta aconsejable establecer un factor de ramificación máximo para reducir el número de árboles alternativos que se evalúan. Este valor máximo puede determinarse automáticamente utilizando las reglas de prepoda: cuando los subconjuntos resultantes del conjunto de entrenamiento son demasiado pequeños, no es necesario seguir ampliando el conjunto de intervalos actual.

Si empleamos un método jerárquico divisivo para discretizar los valores

1. Comenzar con un único intervalo que incluya todos los valores del atributo continuo.
2. Identificar el punto de corte que da lugar a la configuración de intervalos más disimilares entre sí.
3. Utilizar dicho punto de corte para dividir en dos uno de los intervalos de la configuración actual de intervalos.
4. Evaluar el árbol de decisión que resultaría de utilizar el conjunto actual de intervalos para ramificar el árbol de decisión.
5. Mientras el número de intervalos sea inferior al factor de ramificación máximo deseado para el árbol de decisión, volver al paso 2.

Figura 5.8: Implementación divisiva del método propuesto.

de un atributo continuo, la presencia de ruido en el conjunto de entrenamiento afectará menos a los intervalos resultantes de la discretización. El algoritmo divisivo es menos sensible a la presencia de datos erróneos pues comienza con un único intervalo que cubre al conjunto de entrenamiento completo y lo va dividiendo de una forma más global que un método aglomerativo, que parte de las distribuciones de clases particulares para cada valor del atributo continuo.

5.3.2.3. Eficiencia

La utilización, en general, de cualquier método de discretización jerárquico nos permite, pues, construir árboles n-arios arbitrarios. Además, un algoritmo de discretización como el propuesto en la sección 5.2, cuya complejidad computacional es de orden $O(N \log N)$, se puede utilizar para generar una serie de conjuntos de intervalos sin afectar a la complejidad computacional del proceso de inducción del árbol de decisión. C4.5, por ejemplo, requiere realizar una ordenación de los valores de los atributos continuos en cada nodo, por lo que también es de orden $O(N \log N)$.

En algoritmos como el discretizador contextual de la sección 5.2, al existir una relación de adyacencia preestablecida entre los intervalos empleados en la discretización, sólo hay que recalcular dos medidas de similitud en cada iteración cuando se emplea un método aglometativo y tres cuando se utiliza un método divisivo. Como se puede emplear una estructura de datos similar a un árbol parcialmente ordenado, cada iteración se puede realizar en $O(\log N)$ operaciones. Como se realizan un máximo de $O(N)$ iteraciones, la complejidad resultante del algoritmo es $O(N \log N)$. Además, tanto la versión aglomerativa con pre-discretización como la divisiva acotan el número de iteraciones necesarias, con lo cual la complejidad del proceso se reduce a la complejidad de la etapa de preprocesamiento, $O(N)$ por lo general.

Por tanto, el método propuesto se puede considerar adecuado para realizar tareas de *Data Mining*, en las cuales los conjuntos de datos utilizados suelen tener un volumen considerable y es esencial disponer de algoritmos eficientes.

En la siguiente sección se presentan los resultados experimentales que se han conseguido utilizando el método de discretización descrito en la sección 5.2 para construir árboles de decisión n-arios con atributos continuos, tal como se describe en este apartado, y se comparan estos resultados con los obtenidos por las versiones básicas de C4.5 y ART, así como con la utilización de otras técnicas de discretización propuestas en la literatura.

5.4. Resultados experimentales

Se han realizado una serie de experimentos con los dieciséis conjuntos de datos de tamaño pequeño y mediano que aparecen descritos en la tabla 5.2. La mayor parte de esos conjuntos de datos se pueden obtener gratuitamente del *Machine Learning Repository* de la Universidad de California en Irvine, al cual se puede acceder a través de la siguiente dirección web:

http://www.ics.uci.edu/∼mlearn/MLRepository.html

Como en capítulos anteriores de esta memoria, todos los resultados comentados en esta sección se obtuvieron realizando validación cruzada con 10 particiones de cada conjunto de datos (10-CV).

Conjunto de datos	#Ejemplos	#Atributos	#Clases
ADULT	48842	14	2
AUSTRALIAN	690	14	2
BREAST	699	9	2
BUPA	245	7	2
CAR	1728	6	4
GLASS	214	10	6
HAYESROTH	160	4	3
HEART	270	13	2
IONOSPHERE	351	34	2
IRIS	150	4	3
PIMA	768	8	2
SPAMBASE	4601	57	2
THYROID	2800	29	2
WAVEFORM	5000	21	3
WINE	178	13	3
YEAST	1484	8	10

Tabla 5.2: Conjuntos de datos utilizados en los experimentos.

En los siguientes apartados de esta sección se comparan los resultados obtenidos por las versiones estándar de los algoritmos C4.5 y ART con los logrados al utilizar métodos de discretización en la construcción de los árboles de decisión. En concreto, se han implementado nueve métodos de discretización:

- Las tres variantes del discretizador contextual propuesto en la sección 5.2 de la presente memoria: su versión aglomerativa (Contextual A, figura 5.2 de la página 179), una variante de ésta con pre-discretización que utiliza *Equidepth* (Contextual B) y la versión divisiva del discretizador contextual (Contextual C, figura 5.3, página 180). Al emplear estos métodos de discretización de forma local en la construcción del árbol de decisión se utiliza la técnica entrelazada descrita en la sección 5.3.2 (figuras 5.6 a 5.8, páginas 194 a 196).

- Tres métodos de discretización supervisada ya existentes: el discretizador 1R (*One Rule*) de Holte [83], el método de Fayyad e Irani (MDLP) basado en el principio MDL de Rissanen [58] y Zeta, el método ideado por Ho y Scott [82].

- Otros tres métodos no supervisados estándar: el omnipresente algoritmo de la K Medias y dos sencillos métodos de discretización muy utilizados en KDD (*Equiwidth* y *Equidepth*).

Los resultados obtenidos por los métodos de discretización citados se comparan con la versión habitual de C4.5, que realiza tests binarios sobre los atributos continuos, en el apartado 5.4.1 y con la versión básica de ART, que considera categóricos todos los atributos, en el apartado 5.4.2.

5.4.1. Discretización en algoritmos TDIDT

En los experimentos se ha implementado el algoritmo C4.5, paradigma de los algoritmos TDIDT, que utiliza el criterio de proporción de ganancia como regla de división y emplea la poda pesimista descrita por Quinlan [131] con $CF = 0{,}25$. Las figuras 5.9 y 5.10 muestran los resultados medidos experimentalmente relativos a la precisión y complejidad de los árboles obtenidos.

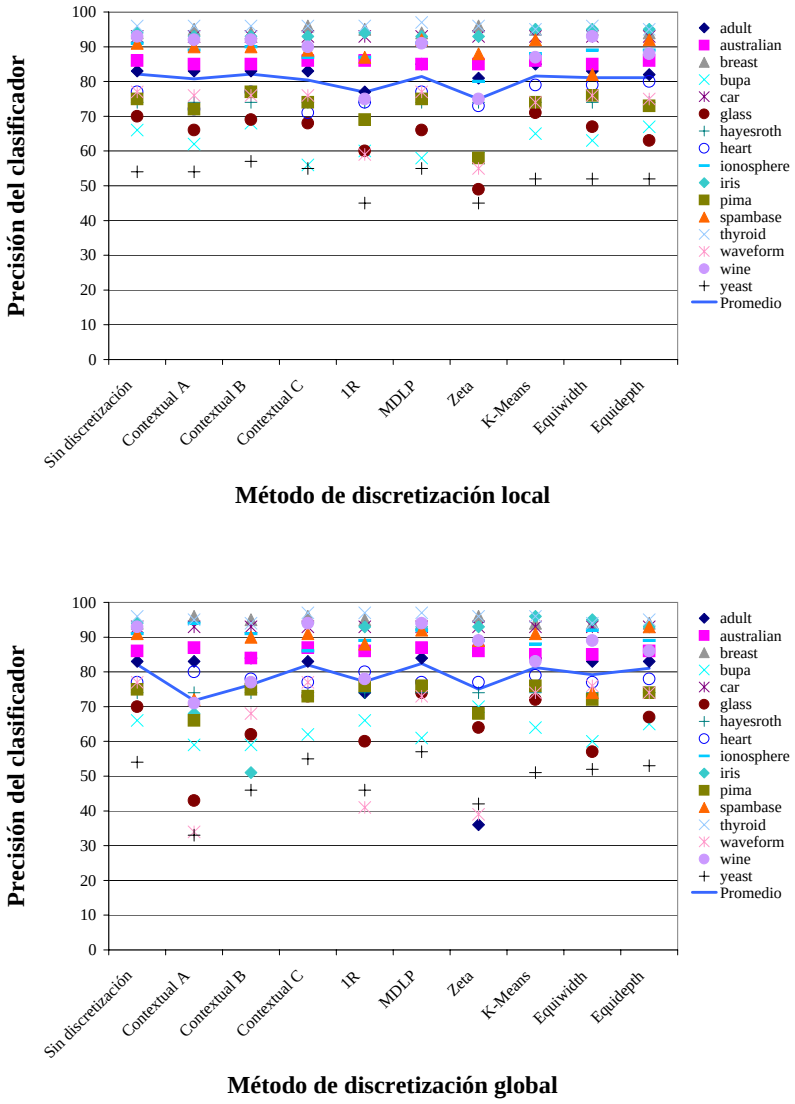

Figura 5.9: Precisión del clasificador TDIDT cuando se utilizan distintas técnicas de discretización.

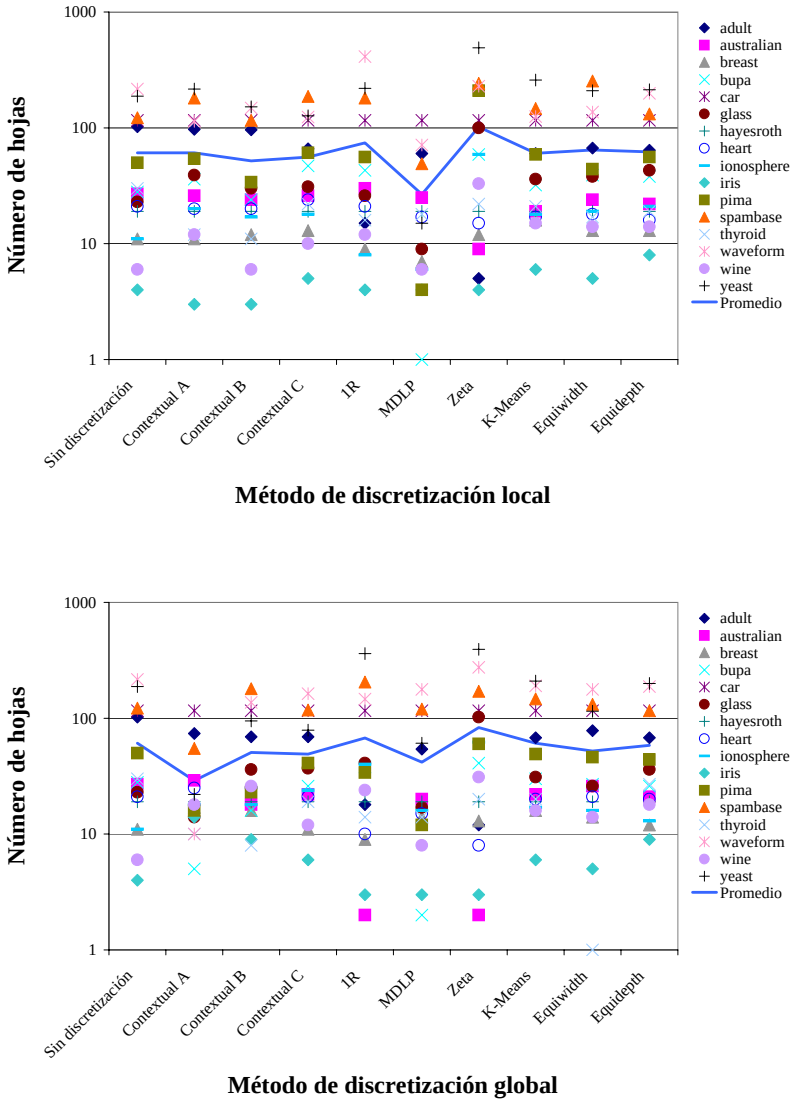

Figura 5.10: Número de hojas del árbol TDIDT cuando se emplean distintas técnicas de discretización.

5.4.1.1. Discretización local

La tabla 5.3 resume los resultados que se han obtenido al construir árboles de decisión utilizando distintos métodos de discretización localmente en cada nodo del árbol.

Con objeto de mejorar la robustez del discretizador contextual en presencia de ruido, los experimentos que figuran en las tablas de este apartado correspondientes al discretizador contextual se han realizado utilizando la variante aglomerativa precedida de un proceso de discretización previo con Equidepth, tal como se describe en la figura 5.7. En concreto, en los experimentos se utilizan 25 intervalos como punto de partida de la versión aglomerativa del método de discretización contextual. Igual que en los demás experimentos de esta sección, el discretizador contextual emplea la distancia euclídea para medir la disimilitud entre las distribuciones de clases de los intervalos adyacentes.

La tabla 5.3 muestra que el método de discretización contextual propuesto en esta memoria es comparable a otros métodos existentes en términos de la precisión del clasificador obtenido y, además, tiende a construir árboles de decisión pequeños.

Es digno de mención destacar que el método de discretización contextual mejora el error estimado del árbol de clasificación respecto a todas las demás técnicas de discretización. Dicha estimación del error es la utilizada por la poda pesimista de Quinlan. Sin embargo, no se aprecian diferencias notables en el error real medido al utilizar validación cruzada. Los resultados concretos de precisión del clasificador para cada conjunto de datos y método de discretización utilizando validación cruzada se muestran en la tabla 5.4.

Respecto al tamaño del árbol, el método de discretización contextual planteado en esta memoria obtiene árboles que, en media, contienen sólo el 84 % de los nodos empleados por el árbol binario construido por C4.5 y únicamente se ve mejorado por el método de discretización MDLP de Fayyad e Irani (véase la tabla 5.5).

	C4.5	Contextual	1R	MDLP	Zeta	KMeans	Equiwidth	Equidepth
Precisión (10-CV)	82.00 %	82.04 %	76.90 %	81.44 %	74.87 %	81.53 %	81.06 %	81.01 %
- Error medido	18.00 %	17.96 %	23.10 %	18.56 %	25.13 %	18.47 %	18.94 %	18.99 %
- Error estimado	12.74 %	14.34 %	16.88 %	16.44 %	19.99 %	16.36 %	17.27 %	17.20 %
Tiempo (segundos)	6.93	12.17	7.30	7.92	15.53	20.82	8.61	8.26
Tamaño del árbol	110.9	83.8	96.1	42.5	116.8	84.3	93.5	89.0
- Nodos internos	49.9	32.1	21.8	15.8	15.3	24.0	29.0	27.0
- Nodos hoja	60.9	51.7	74.3	26.7	101.6	60.3	64.5	62.0
Profundidad media	3.91	3.40	2.37	3.02	1.84	2.80	4.43	2.70

Tabla 5.3: Resumen de los experimentos realizados con discretización local.

	C4.5	Contextual	1R	MDLP	Zeta	KMeans	Equiwidth	Equidepth
ADULT	82.60 %	82.58 %	77.10 %	85.27 %	80.56 %	84.73 %	82.92 %	82.42 %
AUSTRALIAN	85.65 %	84.93 %	85.80 %	85.07 %	84.78 %	85.51 %	84.64 %	85.65 %
BREAST	93.99 %	93.84 %	94.85 %	94.28 %	95.57 %	95.28 %	95.28 %	94.27 %
BUPA	66.10 %	67.83 %	59.76 %	57.71 %	57.71 %	64.71 %	62.96 %	66.72 %
CAR	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %
GLASS	70.06 %	68.77 %	59.70 %	66.34 %	49.46 %	70.58 %	67.38 %	63.46 %
HAYESROTH	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %
HEART	77.04 %	76.67 %	74.07 %	76.67 %	73.33 %	78.89 %	78.52 %	79.63 %
IONOSPHERE	90.60 %	90.32 %	87.21 %	91.16 %	79.76 %	88.31 %	89.17 %	88.60 %
IRIS	94.00 %	93.33 %	94.00 %	93.33 %	93.33 %	95.33 %	94.67 %	94.67 %
PIMA	74.85 %	76.81 %	68.74 %	74.72 %	58.19 %	74.20 %	76.03 %	72.76 %
SPAMBASE	90.52 %	90.24 %	87.09 %	92.00 %	87.83 %	91.68 %	82.22 %	92.11 %
THYROID	96.18 %	95.61 %	96.46 %	97.29 %	96.04 %	95.32 %	96.04 %	94.54 %
WAVEFORM	76.80 %	76.50 %	58.86 %	76.80 %	55.36 %	74.38 %	75.74 %	74.58 %
WINE	92.71 %	91.57 %	75.07 %	90.92 %	74.51 %	87.09 %	92.68 %	88.20 %
YEAST	54.25 %	56.94 %	45.15 %	54.79 %	44.81 %	51.76 %	52.09 %	51.89 %
Promedio	82.00 %	82.04 %	76.90 %	81.44 %	74.87 %	81.53 %	81.06 %	81.01 %
Mejor-Peor-Igual (1 %)		3-2-11	0-10-2	3-3-10	1-10-5	5-5-6	3-7-6	2-7-7

Tabla 5.4: Precisión del clasificador obtenido con discretización local.

	C4.5	Contextual	1R	MDLP	Zeta	KMeans	Equiwidth	Equidepth
ADULT	145.8	134.5	18.0	80.2	5.6	79.8	95.2	94.4
AUSTRALIAN	39.9	36.2	40.5	36.9	11.6	28.3	33.9	30.9
BREAST	20.8	17.2	15.5	11.4	19.8	22.2	18.4	18.4
BUPA	55.0	38.5	55.1	1.8	67.3	42.1	53.8	48.1
CAR	162.0	162.0	162.0	162.0	162.0	162.0	162.0	162.0
GLASS	44.8	48.8	31.2	15.0	101.3	47.3	51.2	57.1
HAYESROTH	24.8	24.8	24.8	24.8	24.8	24.8	24.8	24.8
HEART	35.2	30.9	33.8	28.2	19.7	25.6	26.1	24.4
IONOSPHERE	21.0	28.0	11.2	10.4	61.0	24.4	27.8	29.5
IRIS	7.4	5.0	5.8	5.2	5.5	7.0	6.5	10.4
PIMA	99.6	56.2	72.5	6.9	218.5	76.2	58.4	73.6
SPAMBASE	244.0	203.8	229.6	94.9	288.1	225.2	400.1	225.0
THYROID	55.7	16.8	24.5	29.5	32.4	31.7	25.8	28.8
WAVEFORM	431.6	295.4	517.7	134.5	257.9	187.0	204.4	295.4
WINE	11.0	11.3	13.4	11.4	35.2	21.9	20.3	20.0
YEAST	375.2	231.7	281.8	26.2	558.4	343.0	286.8	281.1
Complejidad relativa	100 %	84 %	82 %	55 %	126 %	91 %	95 %	96 %

Tabla 5.5: Tamaño del árbol resultante (en número de nodos) al utilizar distintos métodos de discretización local.

El uso de árboles n-arios con atributos numéricos permite mejorar el porcentaje de clasificación del algoritmo TDIDT estándar en algunos de los experimentos.

En concreto, el método de discretización contextual consiguió mejoras importantes en el porcentaje de clasificación de tres de los conjuntos de datos (BUPA, PIMA y YEAST). De hecho, este método de discretización sólo empeora de modo apreciable los resultados obtenidos por C4.5 en dos de los dieciséis conjuntos de datos (GLASS y WINE).

Otros métodos de discretización consiguen resultados similares, si bien es verdad que el método de discretización contextual propuesto en la sección 5.2 tiende a ser algo mejor que ellos. Aunque es cierto que el discretizador contextual no siempre mejora el porcentaje de clasificación obtenido por otros métodos, también es cierto que en los conjuntos de datos en que peor se comporta la pérdida de precisión es mínima. En cualquier caso, ha de mencionarse que la poda pesimista que se realiza sobre el árbol influye en algunos de los resultados (como en el conjunto de datos BUPA cuando se utiliza el discretizador MDLP de Fayyad e Irani y se poda el árbol de decisión de una forma excesivamente agresiva).

Los resultados comentados relativos a la precisión del clasificador son más relevantes si tenemos en cuenta que la complejidad del árbol de decisión suele disminuir de forma apreciable cuando se utilizan árboles n-arios con atributos continuos. El tamaño del árbol, dado por el número total de nodos del árbol (tanto nodos internos como hojas), suele ser menor al emplear técnicas de discretización local, tal como muestra la tabla 5.5. La profundidad media del árbol de decisión también tiende a disminuir cuando se utilizan métodos de discretización local para los atributos numéricos del conjunto de datos de entrenamiento.

Aunque tradicionalmente se asume que los árboles de decisión binarios tienen menos hojas y son más profundos que los árboles n-arios [108], en los experimentos realizados nos encontramos con que los árboles binarios no tienen menos hojas necesariamente (al menos, tras el proceso de poda). Como muestra la tabla 5.3, el árbol binario podado, que aparece en la columna etiquetada C4.5, puede tener bastantes más hojas que los árboles obtenidos utilizando

métodos de discretización como el propuesto en la sección 5.2 (Contextual) o el ideado por Fayyad e Irani (MDLP). Por tanto, podemos afirmar que se pueden lograr árboles de decisión más pequeños si se emplean tests n-arios para los atributos continuos del conjunto de entrenamiento.

5.4.1.2. Discretización global

También se han realizado pruebas utilizando distintos métodos de discretización global con los mismos conjuntos de datos utilizados en la sección anterior. Se ha evaluado el comportamiento de los mismos métodos de discretización que se emplearon localmente en cada nodo del árbol, aunque esta vez la discretización se realizó de forma global antes de comenzar a construir el árbol de decisión. Una vez discretizados globalmente, los atributos continuos pueden considerarse como si fueran atributos categóricos, de forma que algoritmos TDIDT como C4.5 pueden construir árboles n-arios con ellos sin necesitar ningún mecanismo adicional para su procesamiento.

La tabla 5.6 resume los resultados que se han obtenido utilizando la variante divisiva del discretizador contextual (más robusta ante la presencia de ruido) y los otros seis discretizadores empleados en los experimentos de la sección anterior. Las tablas 5.7 y 5.8 muestran los resultados obtenidos por cada discretizador para cada uno de los conjuntos de datos empleados en los experimentos. La primera muestra los porcentajes de clasificación obtenidos utilizando validación cruzada y en la segunda se recoge el tamaño de los árboles obtenidos.

Tal como se sugiere en estudios anteriores realizados por otros autores [51] [85], el empleo de métodos de discretización global mejora la eficiencia de los algoritmos TDIDT, reduce la complejidad de los árboles de decisión resultantes y mantiene la precisión de los modelos de clasificación construidos.

A pesar de que el método de discretización contextual expuesto en la sección 5.2 se ideó inicialmente como método de discretización local para su empleo en la construcción de árboles n-arios (tal como se describe en el apartado 5.3.2), se ha comprobado experimentalmente que, igual que otros métodos de discretización, se comporta adecuadamente cuando se utiliza globalmente.

	C4.5	Contextual	1R	MDLP	Zeta	KMeans	Equiwidth	Equidepth
Precisión (10-CV)	82.00 %	81.92 %	77.24 %	82.40 %	74.91 %	81.25 %	79.24 %	81.16 %
- Error	18.00 %	18.08 %	22.76 %	17.60 %	25.09 %	18.75 %	20.76 %	18.84 %
- Error estimado	12.74 %	16.60 %	20.53 %	15.94 %	22.12 %	16.96 %	19.47 %	17.24 %
Tiempo (segundos)	6.93	1.62	6.30	1.02	10.54	1.03	1.05	1.02
Tamaño del árbol	110.9	70.7	77.6	62.7	92.1	85.5	75.4	82.4
- Nodos internos	49.9	21.8	10.1	20.8	9.0	24.4	23.2	24.1
- Nodos hoja	60.9	48.9	67.4	41.8	83.0	61.1	52.2	58.2
Profundidad media	3.91	2.76	1.70	3.03	1.55	2.80	4.09	2.60

Tabla 5.6: Resumen de los experimentos realizados con discretización global.

	C4.5	Contextual	1R	MDLP	Zeta	KMeans	Equiwidth	Equidepth
ADULT	82.60 %	82.67 %	74.37 %	84.32 %	36.20 %	84.45 %	83.11 %	82.93 %
AUSTRALIAN	85.65 %	86.67 %	85.51 %	86.67 %	85.51 %	84.78 %	84.93 %	86.23 %
BREAST	93.99 %	95.71 %	95.13 %	94.14 %	95.71 %	93.99 %	94.13 %	94.42 %
BUPA	66.10 %	62.13 %	65.54 %	60.61 %	69.61 %	64.43 %	59.76 %	65.30 %
CAR	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %	92.88 %
GLASS	70.06 %	73.31 %	59.83 %	74.29 %	63.53 %	71.90 %	56.56 %	67.25 %
HAYESROTH	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %	73.75 %
HEART	77.04 %	77.41 %	80.37 %	77.41 %	76.67 %	78.52 %	77.41 %	77.78 %
IONOSPHERE	90.60 %	86.33 %	89.18 %	91.72 %	88.88 %	88.32 %	91.75 %	89.47 %
IRIS	94.00 %	94.00 %	93.33 %	93.33 %	93.33 %	96.00 %	95.33 %	93.33 %
PIMA	74.85 %	73.02 %	75.76 %	75.50 %	67.82 %	75.89 %	72.50 %	74.19 %
SPAMBASE	90.52 %	90.74 %	88.15 %	92.50 %	89.11 %	90.55 %	74.40 %	92.65 %
THYROID	96.18 %	96.64 %	97.04 %	96.96 %	96.46 %	95.82 %	93.89 %	94.89 %
WAVEFORM	76.80 %	76.56 %	41.40 %	73.44 %	38.62 %	74.40 %	76.18 %	73.80 %
WINE	92.71 %	94.38 %	77.97 %	93.79 %	88.73 %	83.10 %	89.31 %	86.50 %
YEAST	54.25 %	54.58 %	45.62 %	57.01 %	41.71 %	51.15 %	51.96 %	53.11 %
Promedio	82.00 %	81.92 %	77.24 %	82.40 %	74.91 %	81.25 %	79.24 %	81.16 %
Mejor-Peor-Igual (1 %)		4-3-9	2-7-7	7-2-7	2-8-6	5-5-6	2-7-7	1-6-9

Tabla 5.7: Precisión del clasificador cuando se emplea discretización global.

	C4.5	Contextual	1R	MDLP	Zeta	KMeans	Equiwidth	Equidepth
ADULT	145.8	93.7	21.3	66.9	12.6	90.5	108.7	94.5
AUSTRALIAN	39.9	32.4	3.0	31.2	3.0	32.6	37.5	31.4
BREAST	20.8	15.0	17.6	18.4	21.0	19.8	18.5	16.0
BUPA	55.0	35.5	40.8	3.0	44.4	38.7	37.5	32.6
CAR	162.0	162.0	162.0	162.0	162.0	162.0	162.0	162.0
GLASS	44.8	51.3	47.3	29.1	103.5	43.1	36.3	50.0
HAYESROTH	24.8	24.8	24.8	24.8	24.8	24.8	24.8	24.8
HEART	35.2	30.0	14.2	24.7	11.2	29.7	30.0	28.0
IONOSPHERE	21.0	33.5	42.0	27.3	63.5	23.3	22.8	17.2
IRIS	7.4	8.1	4.4	4.0	4.0	8.1	6.3	11.7
PIMA	99.6	54.4	35.8	20.2	60.5	64.6	64.9	56.7
SPAMBASE	244.0	170.3	232.8	216.2	196.0	222.9	193.4	206.2
THYROID	55.7	28.2	21.4	23.5	31.4	28.8	1.0	37.9
WAVEFORM	431.6	263.2	147.8	238.7	276.1	267.6	274.8	258.5
WINE	11.0	14.5	26.4	13.8	31.9	20.9	18.2	21.8
YEAST	375.2	114.8	399.8	98.8	427.2	290.3	169.5	269.0
Complejidad relativa	100 %	84 %	84 %	68 %	105 %	90 %	95 %	91 %

Tabla 5.8: Complejidad del clasificador obtenido cuando se emplea discretización global.

5.4.2. ART con discretización

Del mismo modo que se ha evaluado la obtención de árboles de decisión n-arios con algoritmos TDIDT (esto es, C4.5 con discretización), en este apartado se presenta un estudio empírico del uso de los distintos métodos de discretización en el modelo de clasificación ART.

En el caso de ART, la discretización es fundamental para permitir la utilización de este modelo en la construcción de clasificadores a partir de conjuntos de datos que incluyan atributos de tipo numérico.

5.4.2.1. Precisión

En cuanto a los resultados relativos a la precisión del clasificador ART, resumidos en la tabla 5.9, se puede apreciar que los métodos de discretización utilizados globalmente suelen conseguir resultados mejores que localmente. Aunque este resultado pueda parecer poco intuitivo, se consiguen mejores resultados si discretizamos los atributos continuos antes de construir el clasificador ART en vez de discretizarlos localmente en cada nivel del árbol.

Respecto a los resultados obtenidos con las distintas variantes del discretizador contextual de la sección 5.2, se puede apreciar que la versión divisiva del mismo consigue mejores resultados que las variantes aglomerativas, tal como cabría esperar, porque es menos sensible a la existencia de ruido en los datos del conjunto de entrenamiento.

Los resultados concretos de los que se extrajeron los porcentajes medios que figuran en la tabla 5.9 aparecen en la figura 5.11 de la página 214.

5.4.2.2. Complejidad

Por otra parte, los resultados experimentales relativos a la complejidad del árbol ART se resumen en la tabla 5.10 y aparecen reflejados en la figura 5.12 de la página 215. Igual que antes, los clasificadores globales tienden a conseguir mejores resultados al lograr árboles más pequeños de menor profundidad (figura 5.13, página 216) y, por tanto, de construcción más eficiente (figura 5.14, página 217)

Método de discretización	Precisión
Global - Contextual C (divisivo)	78.03 %
Global - MDLP	77.70 %
Global - K Means	77.27 %
Global - Zeta	76.47 %
Global - Equidepth	75.93 %
Local - Equidepth	75.42 %
Local - K Means	74.54 %
Local - MDLP	74.16 %
Local - Contextual B (con Equidepth)	74.04 %
Local - Contextual C (divisivo)	74.03 %
Global - 1R (Holte)	73.92 %
Local - Contextual A (aglomerativo)	73.21 %
Global - Equiwidth	73.15 %
Local - 1R (Holte)	72.02 %
Local - Equiwidth	71.68 %
Global - Contextual B (con Equidepth)	71.44 %
Global - Contextual A (aglomerativo)	69.75 %
Local - Zeta	69.33 %
Sin discretización	55.50 %

Tabla 5.9: Porcentajes de clasificación obtenidos con ART empleando distintos métodos de discretización (ordenados de mejor a peor).

Método de discretización	Hojas
Local - MDLP	11.0
Global - MDLP	15.8
Global - Contextual A (aglomerativo)	15.9
Sin discretización	20.3
Global - Equiwidth	24.3
Global - K Means	31.5
Global - Contextual C (divisivo)	33.2
Global - Contextual B (con Equidepth)	33.7
Local - Equiwidth	35.2
Local - Contextual A (aglomerativo)	36.9
Global - Equidepth	38.6
Local - K Means	39.9
Local - Equidepth	40.9
Local - Contextual B (con Equidepth)	71.9
Local - Contextual C (divisivo)	88.4
Local - 1R (Holte)	93.9
Global - 1R (Holte)	104.7
Local - Zeta	109.1
Global - Zeta	111.8

Tabla 5.10: Número medio de hojas del árbol construido por ART al utilizar distintos métodos de discretización (ordenados de mejor a peor).

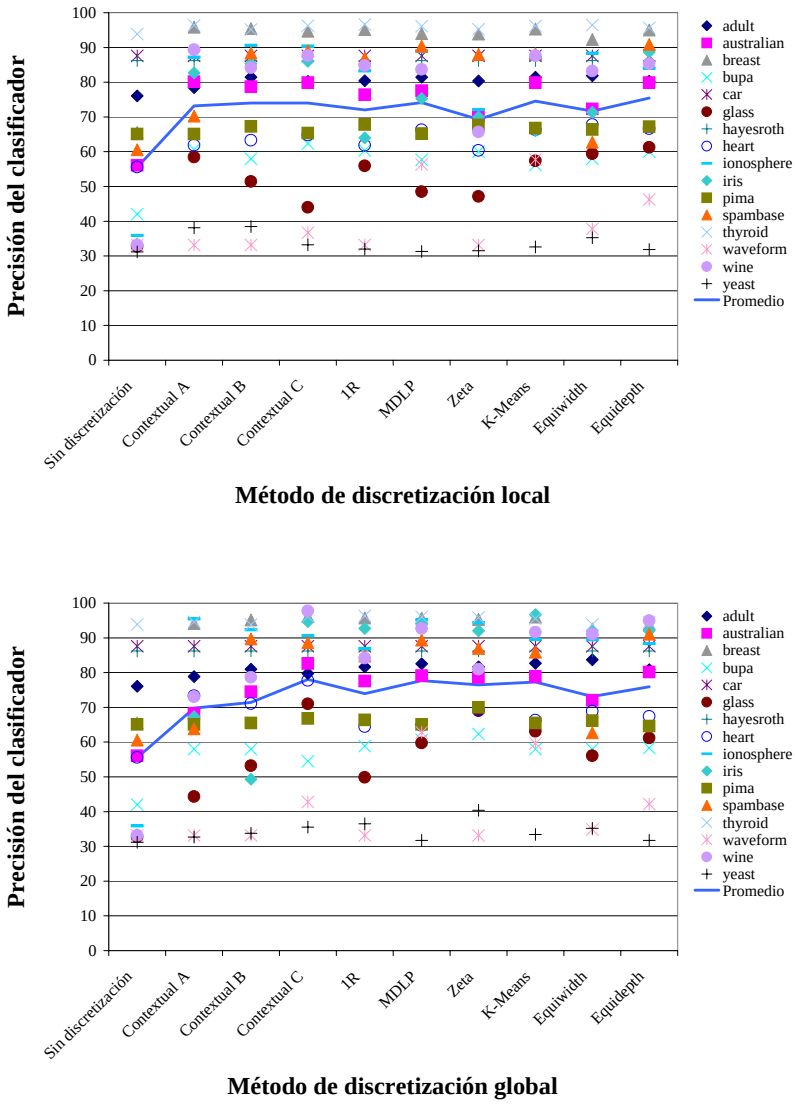

Figura 5.11: Precisión del clasificador ART al utilizar distintos discretizadores.

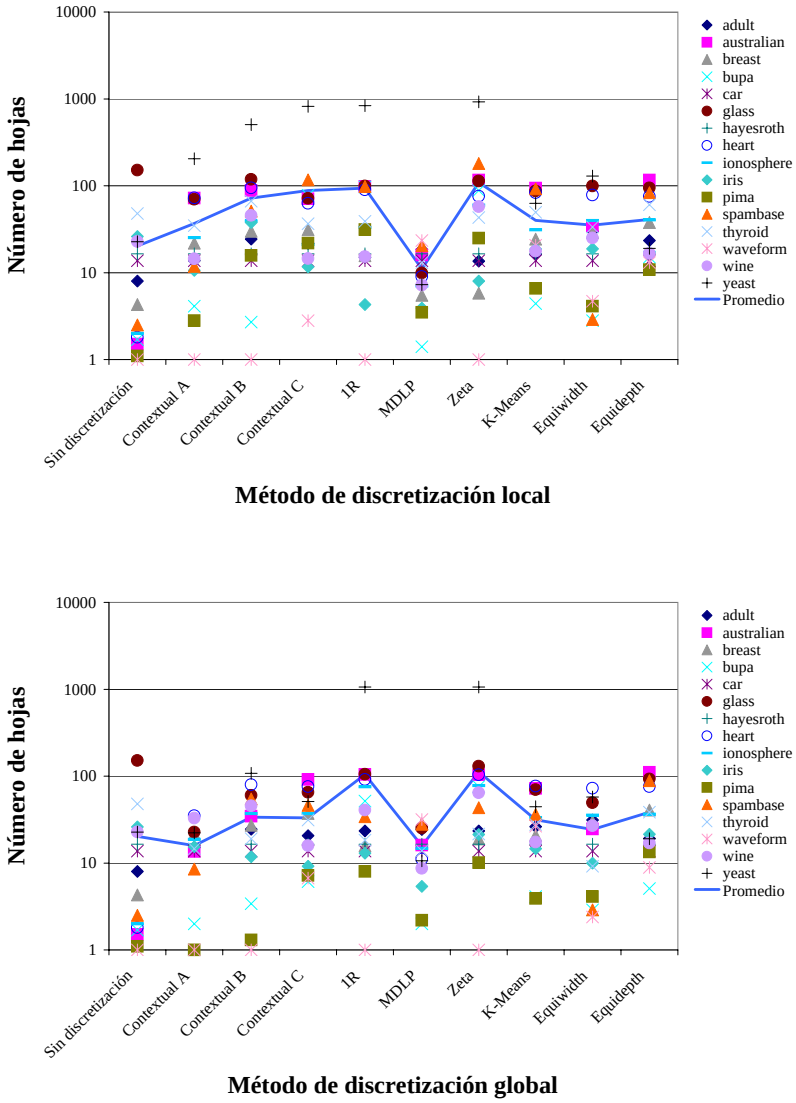

Figura 5.12: Número de hojas del clasificador ART cuando se utilizan distintas técnicas de discretización.

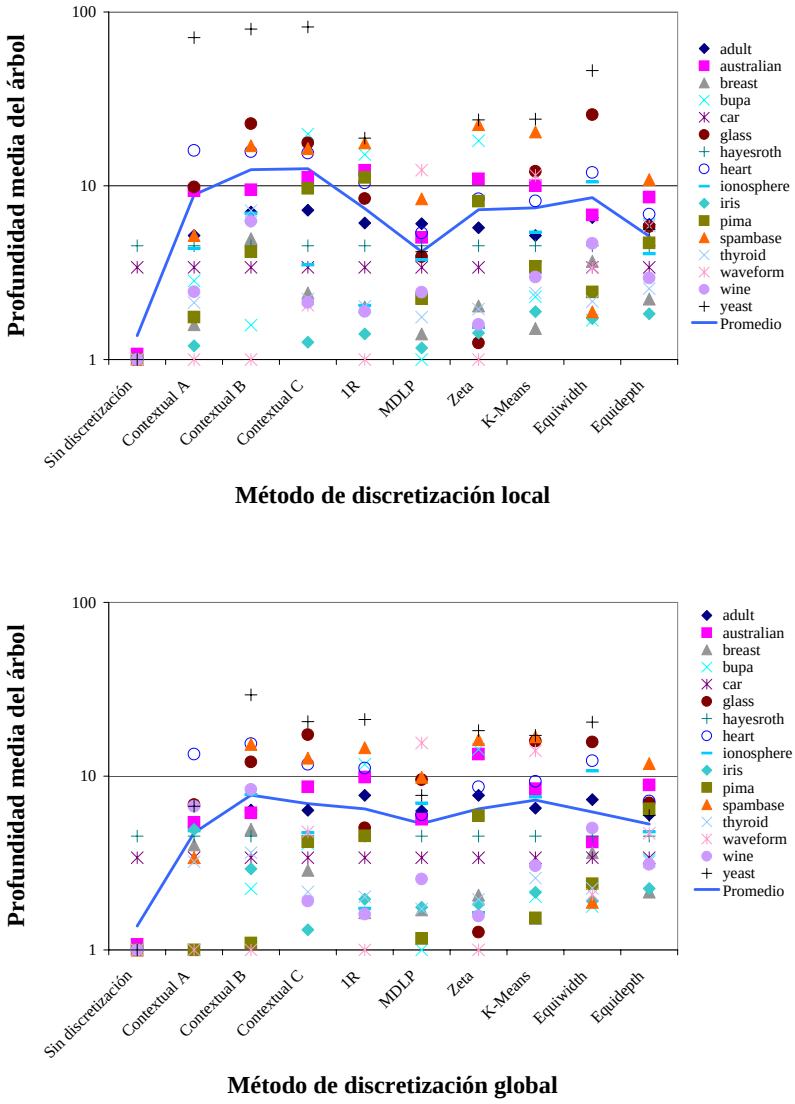

Figura 5.13: Profundidad media del árbol ART cuando se utilizan distintas técnicas de discretización.

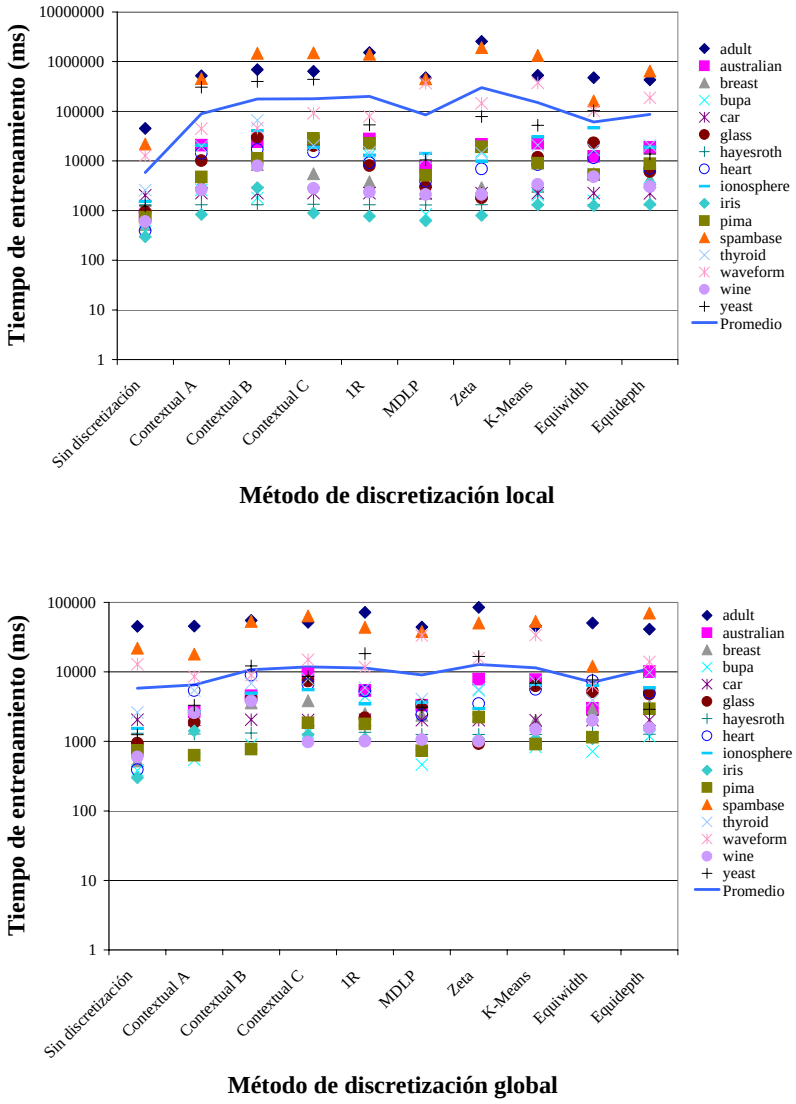

Figura 5.14: Tiempo de entrenamiento necesario para construir el clasificador ART cuando se utilizan distintas técnicas de discretización.

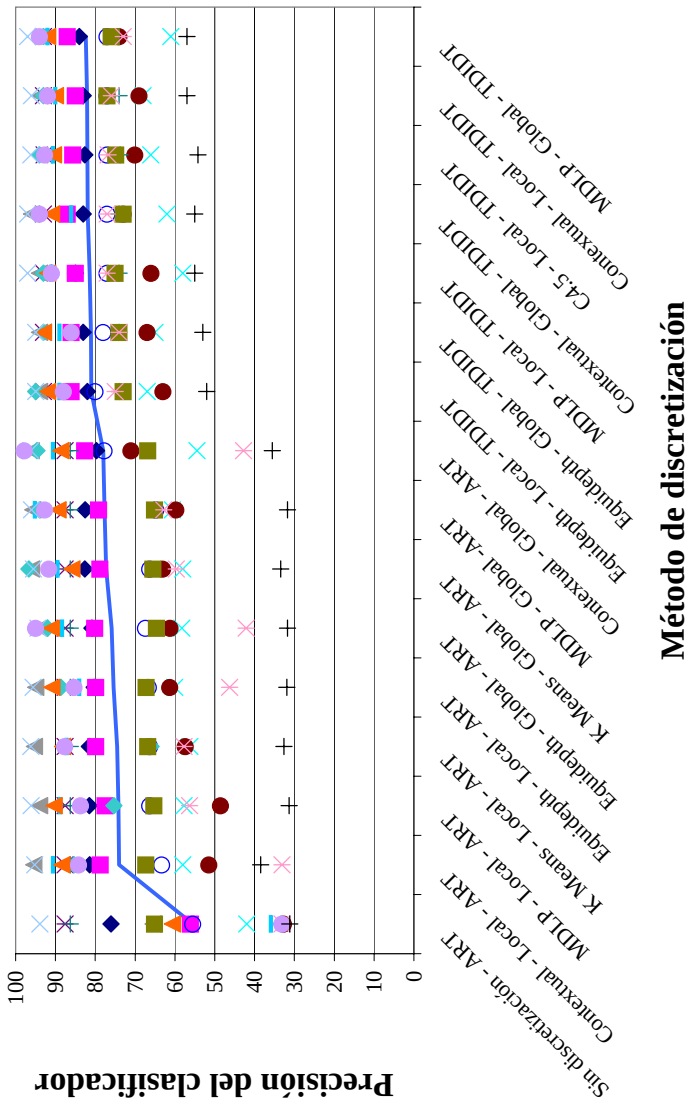

Figura 5.15: Efectos de la discretización en la precisión de los clasificadores.

5.4.3. Observaciones finales

Teniendo en cuenta los resultados anteriores, se recomienda el uso de ART con métodos de discretización global porque con ellos se suelen obtener clasificadores más precisos y compactos en menos tiempo.

Hay que tener en cuenta que los resultados obtenidos por ART cuando se trabaja con atributos continuos son ligeramente peores que los que se logran al emplear un algoritmo TDIDT tradicional como C4.5, tal como se aprecia en la figura 5.15, al menos en lo que respecta al porcentaje de clasificación obtenido.

Sin embargo, si incluimos en nuestra comparación el tamaño de los árboles obtenidos, se observa que ART consigue árboles de decisión más pequeños y, por tanto, más fáciles de interpretar cuando se utiliza el clasificador contextual propuesto en la sección 5.2 o el discretizador MDLP de Fayyad e Irani [58]. El compromiso entre precisión del clasificador y complejidad del modelo obtenido al que se llega se puede apreciar mejor en las figuras 5.16 y 5.17 que aparecen en las páginas siguientes. En dichas figuras se ordenan de mejor a peor algunos de los métodos de discretización utilizados, tanto local como globalmente, para construir clasificadores ART y TDIDT. En el caso de ART, la discretización es esencial para que se puedan conseguir buenos clasificadores.

En cuanto al método de discretización propuesto en este capítulo, hay que destacar que, utilizado como método de discretización global consigue resultados comparables a los obtenidos por los mejores métodos existentes (MDLP). Además, utilizando el enfoque propuesto en el apartado 5.3.2 se dota de mayor flexibilidad al proceso de construcción del árbol de decisión.

En cualquier caso, gracias a su eficiencia, el método de discretización contextual se añade a las técnicas de discretización que se pueden emplear en aplicaciones *Data Mining*, con la peculiaridad de que utiliza la estructura tradicional de los algoritmos de agrupamiento en la resolución de problemas de discretización. A diferencia de otros métodos de discretización supervisada (como MDLP, 1R o Zeta), los cuales emplean medidas de pureza para evaluar la calidad de un intervalo sin tener en cuenta su entorno, el discretizador propuesto utiliza medidas de similitud para evaluar conjuntos de intervalos adyacentes.

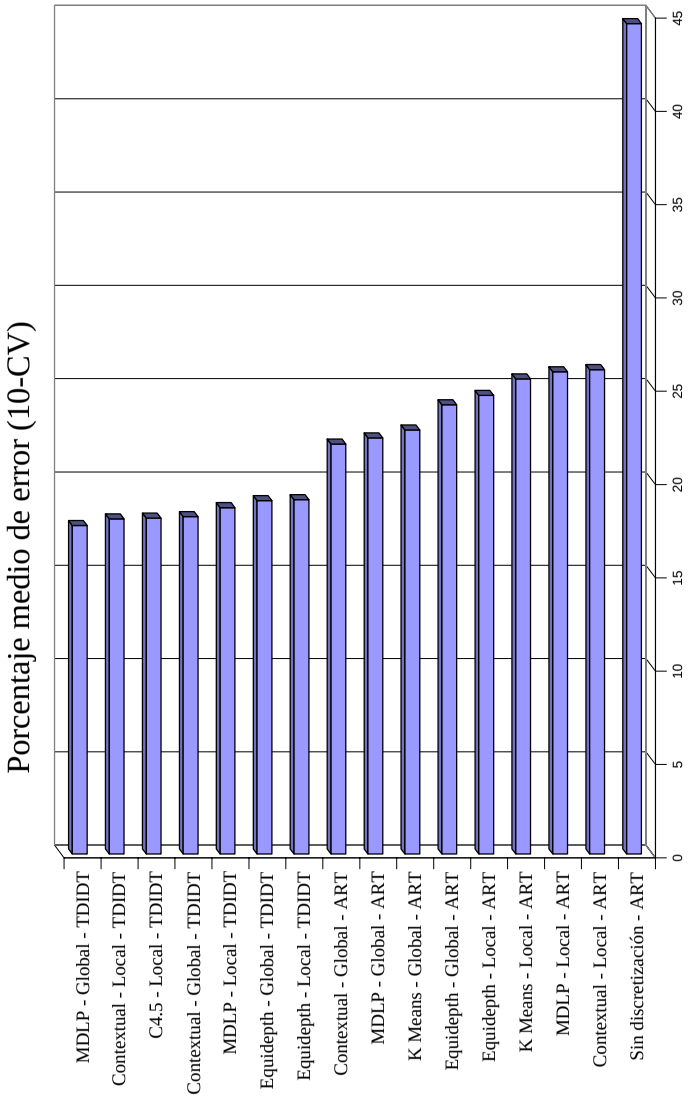

Figura 5.16: Efectos de la discretización en el error cometido por los clasificadores.

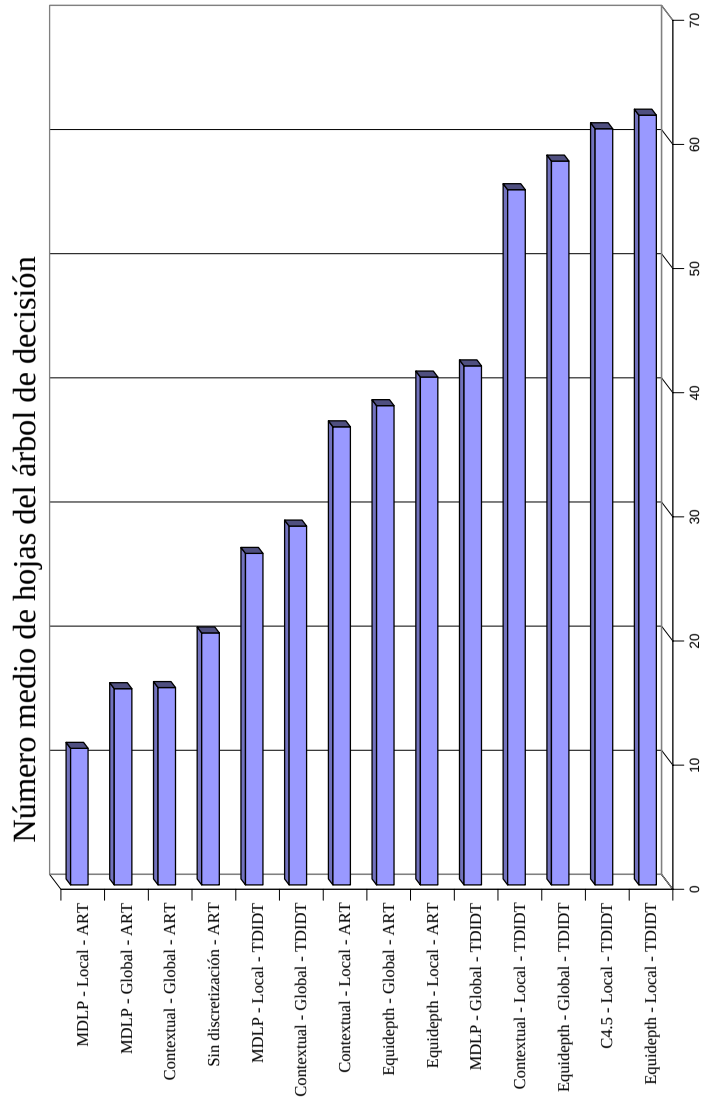

Figura 5.17: Efectos de la discretización en la complejidad de los clasificadores.

5.5. Anexo: Medidas de similitud

En esta sección se comentan algunas de las medidas que nos permiten comparar objetos y evaluar su similitud. Estas medidas son de gran utilidad en los métodos de agrupamiento y se pueden emplear con el método de discretización contextual propuesto en la sección 5.2.

Hablamos de medidas de similitud cuando el valor de la medida utilizada aumenta conforme son más similares los objetos que se comparan (distribuciones de clases en el método de discretización contextual). De forma complementaria, denominamos medidas de disimilitud a aquellas medidas cuyo valor disminuye cuanto más aumenta el parecido entre los objetos que se comparan, como es el caso de la distancia euclídea.

En el caso que nos ocupa, cualquier medida de similitud debe fomentar la combinación de intervalos adyacentes que compartan su clase más común, si bien este criterio es insuficiente en problemas donde las poblaciones de las distintas clases están muy desequilibradas. Si un 95 % de los ejemplos de entrenamiento pertenecen a una misma clase, es probable que la clase más común sea siempre la misma para todos los valores del atributo, por lo que fijarnos sólo en la clase más común no resulta práctico a la hora de discretizar un atributo continuo con un método supervisado como la discretización contextual. Cuanta más información aporte, más útil será la medida de similitud a la hora de establecer el orden en que se van combinando o dividiendo los intervalos. Por tanto, hay que utilizar siempre medidas de similitud que tengan en cuenta la probabilidad de todas las clases del problema, no sólo la más común, en clara oposición a lo que sucede con las reglas de división utilizadas al construir árboles de decisión (página 23).

En los apartados siguientes se describen brevemente los distintos modelos en que se basan las medidas de similitud que figuran en la tabla 5.1 de la página 187, las cuales pueden emplearse en métodos de agrupamiento jerárquico como la discretización contextual. En [164] se puede encontrar un análisis más riguroso de dichas medidas desde un punto de vista psicológico.

5.5.1. Modelos basados en medidas de distancia

Las medidas de similitud empleadas habitualmente por los métodos de aprendizaje no supervisado (esto es, los métodos de agrupamiento o *clustering*) suelen formularse como medidas de disimilitud utilizando métricas de distancia.

Cualquier función d que mida distancias debe verificar las siguientes tres propiedades:

- Propiedad reflexiva:

$$d(x, x) = 0$$

- Propiedad simétrica:

$$d(x, y) = d(y, x)$$

- Desigualdad triangular:

$$d(x, y) \leq d(x, z) + d(z, y)$$

Existe una clase particular de medidas de distancia que se ha estudiado exhaustivamente. Es la métrica r de Minkowski:

$$d_r(x, y) = \left(\sum_{j=1}^{J} |x_j - y_j|^r \right)^{\frac{1}{r}}, \qquad r \geq 1$$

Como una medida de distancia es, en realidad, una medida de disimilitud, podemos formular la distancia de Minkowsky como una medida de similitud de la siguiente forma:

$$S_{d_r}(x, y) = 1 - d_r(x, y)$$

Existen tres casos particulares de la clase de funciones definida por la distancia de Minkowsky que se utilizan habitualmente en la resolución de problemas de agrupamiento:

- La distancia euclídea ($r = 2$):

$$d_2(x, y) = \sqrt{\sum_{j=1}^{J} (x_j - y_j)^2}$$

- La distancia de Manhattan ($r = 1$):

$$d_1(x, y) = \sum_{j=1}^{J} |x_j - y_j|$$

- La "métrica de dominio" (conforme r tiende a ∞, la métrica de Minkowski viene determinada por la diferencia entre las coordenadas correspondientes a la dimensión para la que $|x_j - y_j|$ es mayor):

$$d_\infty(x, y) = \max_{j=1..J} |x_j - y_j|$$

En el método de discretización contextual, la desigualdad triangular no es estrictamente necesaria porque no utilizamos la medida de similitud para establecer la adyacencia entre patrones (ésta viene dada por el valor numérico o intervalo al que representa el vector característico). Por tanto, se podrían utilizar medidas de similitud que no verificasen la desigualdad triangular. De hecho, existen otros tipos de medidas de similitud:

5.5.2. Modelos basados en medidas de correlación

El producto escalar de dos vectores es la medida más simple de este tipo. Puede emplearse para medir la similitud existente entre vectores que representan distribuciones de clases igual que se emplea en muchos sistemas de recuperación de información para emparejar los términos de una consulta con los términos que aparecen en un documento. Matemáticamente, la medida de similitud dada por el producto escalar se define como:

$$S.(x, y) = x \cdot y = \sum_{j=1}^{J} x_j y_j$$

También se puede medir de forma alternativa la correlación entre dos vectores para reflejar la similitud entre dos conjuntos difusos a partir de la distancia euclídea d_2 al cuadrado:

$$\rho(x,y) = 1 - \left(\frac{4}{N_x + N_y}\right) d_2^2$$

donde

$$N_v = \sum_{j=1}^{J} (2v_j - 1)^2$$

Esta distancia estandarizada nos sirve de enlace con el tercer grupo de modelos que nos permiten medir la similitud entre dos objetos:

5.5.3. Modelos basados en Teoría de Conjuntos

Desde un punto de vista psicológico, Tversky describió la similitud como un proceso de emparejamiento de características. La similitud entre objetos puede expresarse como una combinación lineal de sus características comunes y de las características exclusivas de cada uno de ellos:

$$s(a,b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A), \quad \theta, \alpha, \beta \geq 0$$

donde a y b son los objetos que se comparan mientras que A y B representan los conjuntos de características asociados a los objetos a y b respectivamente.

Cuando $\alpha = \beta = 1$ y $\theta = 0$, entonces $-s(a,b) = f(A - B) + f(B - A)$. Éste es el modelo propuesto por Restle para medir la disimilitud entre objetos fijándose únicamente en las características exclusivas de cada uno de ellos.

También podemos establecer $\alpha = \beta = 0$ y $\theta = 1$ para obtener el modelo más simple: $s(a,b) = f(A \cap B)$. En este caso, nos fijamos exclusivamente en las características que a y b tienen en común.

Otro modelo de interés es el modelo proporcional en el cual se normaliza la medida de similitud para que quede entre 0 y 1:

$$s(a,b) = \frac{f(A \cap B)}{f(A \cap B) + \alpha f(A - B) + \beta f(B - A)}, \quad \alpha, \beta \geq 0$$

Cuando la función f es aditiva (esto es, $f(A \cup B) = f(A) + f(B)$ cuando $A \cap B = \varnothing$), el modelo proporcional generaliza las propuestas de otros autores:

	T-normas ($\cap$)	T-conormas ($\cup$)
F	$min\{x, y\}$	$max\{x, y\}$
T_1	xy	$x + y - xy$

Tabla 5.11: Algunos pares de operadores T

- Gregson ($\alpha = \beta = 1$):

$$s(a, b) = \frac{f(A \cap B)}{f(A \cup B)}$$

- Eisler & Ekman ($\alpha = \beta = \frac{1}{2}$):

$$s(a, b) = \frac{2f(A \cap B)}{f(A) + f(B)}$$

- Bush & Mosteller ($\alpha = 1$, $\beta = 0$):

$$s(a, b) = \frac{2f(A \cap B)}{f(A)}$$

En todas las funciones de similitud presentadas en este apartado, la función f suele ser la cardinalidad del conjunto que recibe como argumento, si bien no se descartan otras posibilidades.

La Teoría de los Conjuntos Difusos nos permite utilizar todos estos modelos en conjuntos de objetos en los cuales la función de pertenencia puede variar entre 0 y 1. Así mismo, también define nuevos operadores para realizar las operaciones clásicas sobre conjuntos, como la unión, la intersección o la negación. Por ejemplo, los operadores T (T-normas y T-conormas) pueden emplearse para calcular intersecciones y uniones de conjuntos difusos, como también podrían utilizarse otros operadores, como los operadores promedio utilizados por algunos modelos difusos de recuperación de información [97].

Aunque también existen distintos cardinales difusos para medir la cardinalidad de un conjunto, aquí emplearemos la cardinalidad escalar, que se define como

$$|A| = \sum_x \mu_A(x)$$

La propuesta que Restle definió para conjuntos clásicos puede generalizarse de la siguiente forma haciendo uso de los operadores de la Teoría de Conjuntos Difusos:

$$-S_{Restle}(A,B) = |A \square B|$$

donde $A \square B$ es el conjunto difuso de los elementos que pertenecen a A pero no a B y viceversa. Utilizando el par F de operadores T de la tabla 5.11, dicho conjunto puede definirse como

$$\mu_{A\square B}(x) = \text{máx}\left\{\text{mín}\{\mu_A(x), 1-\mu_B(x)\}, \text{mín}\{1-\mu_A(x), \mu_B(x)\}\right\}$$

Si empleamos una función f distinta, obtenemos otra variación del modelo de Restle:

$$-S_{\square}(A,B) = \sup_x \mu_{A\square B}(x)$$

Cuando, a partir del modelo general de Tversky, fijamos $\alpha = \beta = 0$ y $\theta = 1$ y volvemos a utilizar los operadores mínimo y máximo, se obtiene otra medida de similitud alternativa:

$$S_{MinSum}(A,B) = |A \cap B| = \sum_x min\left\{\mu_A(x), \mu_B(x)\right\}$$

Nótese que, si hubiésemos utilizado la T-norma dada por T_1 en la tabla 5.11 en vez del mínimo, habríamos obtenido una medida de similitud equivalente al producto escalar que vimos en el apartado 5.5.2 de esta memoria.

De forma análoga a como sucede con el modelo de Restle, podemos obtener una variación de la medida de similitud data por S_{MinSum}. Dicha variación ha sido formulada por Enta como un índice de inconsistencia o "grado de separación":

$$-S_{Enta}(A,B) = 1 - \sup_x \mu_{A\cap B}(x)$$

La siguiente medida normaliza la medida de similitud dada por S_{MinSum} y es análoga a la propuesta de Gregson para conjuntos clásicos:

$$S_{Gregson}(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\sum_x min\left\{\mu_A(x), \mu_B(x)\right\}}{\sum_x max\left\{\mu_A(x), \mu_B(x)\right\}}$$

A pesar de que todas las medidas de similitud definidas en los párrafos anteriores se definen sobre conjuntos difusos, también pueden aplicarse a los vectores característicos empleados por el método de discretización contextual propuesto en esta sección. Dichos vectores contienen probabilidades a partir de las cuales puede definirse un conjunto difuso C con función de pertenencia

$$\mu_C(j) = p_j$$

Es obligatorio mencionar que, aunque hemos empleado probabilidades (o, mejor dicho, estimaciones de probabilidades) para definir la función de pertenencia de un conjunto difuso, los modelos difusos y probabilísticos representan distintos tipos de información: los grados de pertenencia a un conjunto difuso nos permiten cuantificar la similitud entre objetos con propiedades definidas de un modo impreciso, mientras que las probabilidades nos dan información acerca de lo que podríamos esperar si realizásemos un número elevado de experimentos (si interpretamos las probabilidades desde un punto de vista estadístico).

No obstante, se puede interpretar la función de pertenencia de un conjunto difuso desde un punto de vista diferente. Podemos verla como una función de posibilidad que nos permite interpretar las conectivas lógicas (T-normas y T-conormas) en el marco de la Teoría de la Probabilidad [52]. De hecho, ésta es la base que nos ofrece la posibilidad de construir funciones de pertenencia experimentalmente. Dada una población de individuos y un concepto difuso C, a cada individuo se le pregunta si un elemento dado j puede considerarse C o no. La función de posibilidad $P(C|j)$ representa la proporción de individuos que respondieron afirmativamente a la pregunta. De esta forma resulta natural permitir que $\mu_C(j)$ sea igual a $P(C|j)$, p_j en nuestro problema de discretización.

Este razonamiento también puede aplicarse para emplear en nuestro problema la distancia de Bhattacharyya, también conocida como distancia de Jeffreis-Matusita. En términos de conjuntos difusos, puede expresarse como:

$$R(A,B) = \sqrt{1 - \sum_x \sqrt{\mu_A^*(x)\mu_B^*(x)}}$$

donde las funciones de pertenencia se normalizan:

$$\mu_A^*(x) = \frac{\mu_A^*(x)}{|A|}$$

Tanto el modelo propuesto por Eisler y Ekman como el de Bush y Mosteller pueden reducirse a la función de similitud S_{MinSum} si asumimos que la función $f(X)$ representa la cardinalidad de X y observamos que su valor se mantiene constante en nuestro problema para cualquier conjunto X, al ser $\sum \mu_C(j) = \sum p_j = 1$.

Con esto se da por cerrado el estudio de las medidas de similitud que pueden emplearse en nuestro método de discretización contextual (y en cualquier otro método de agrupamiento).

5.5.4. Resultados experimentales

Los experimentos realizados han servido para comprobar que el método de discretización propuesto en la sección 5.2, aplicado a la construcción de árboles de decisión tal como se describe en el apartado 5.3.2, nos permite obtener resultados interesantes, independientemente de la medida de similitud empleada.

Las tablas 5.12 a 5.14 resumen los resultados obtenidos para algunos de los conjuntos de datos empleados con anterioridad en este capítulo para evaluar el comportamiento de distintos métodos de discretización. Los experimentos que recogen las tablas se realizaron utilizando validación cruzada (10-CV), el criterio de proporción de ganancia de Quinlan y poda pesimista (con $CF = 0{,}25$), tal como se describe en [131].

Las tablas comparan los resultados obtenidos por el algoritmo TDIDT clásico (que construye árboles de decisión binarios cuando aparecen atributos continuos) con el método propuesto en el apartado 5.3.2, para el cual se han utilizado las distintas medidas de similitud que aparecen en la tabla 5.1.

	Algoritmo	Medidas de similitud basadas en distancias			
	clásico	Euclídea	Manhattan	Dominio	Bhattacharyya
ADULT	**84.59 %**	84.36 %	84.36 %	84.36 %	84.33 %
BUPA	64.36 %	**67.03 %**	**67.03 %**	**66.10 %**	**65.54 %**
GLASS	68.70 %	63.14 %	64.96 %	61.62 %	61.21 %
HEART	75.93 %	**77.41 %**	**77.41 %**	**77.04 %**	**78.15 %**
IONOSPHERE	**91.45 %**	88.62 %	87.76 %	88.33 %	88.04 %
PIMA	72.38 %	**74.59 %**	**74.59 %**	**74.46 %**	**72.88 %**
WAVEFORM	76.94 %	**77.32 %**	**77.84 %**	**77.58 %**	**77.38 %**
WINE	88.63 %	**94.97 %**	**94.97 %**	**94.97 %**	**94.38 %**

	Otras medidas de similitud					
	P.Escalar	Correlación	Restle	MinSum	Enta	Gregson
ADULT	84.54 %	84.20 %	84.22 %	84.37 %	84.30 %	84.37 %
BUPA	58.87 %	**68.16 %**	**65.25 %**	**65.87 %**	**67.03 %**	**65.87 %**
GLASS	**70.09 %**	66.86 %	66.86 %	67.31 %	**70.15 %**	64.98 %
HEART	**76.67 %**	**77.04 %**	**76.30 %**	**78.15 %**	**77.78 %**	**78.15 %**
IONOSPHERE	90.34 %	87.48 %	89.45 %	86.61 %	86.61 %	86.61 %
PIMA	**72.50 %**	**74.46 %**	**75.64 %**	**74.59 %**	**76.15 %**	**74.59 %**
WAVEFORM	76.08 %	**77.48 %**	75.94 %	76.68 %	**77.12 %**	76.32 %
WINE	**91.07 %**	**94.97 %**	**94.97 %**	**92.75 %**	**92.19 %**	**92.19 %**

Tabla 5.12: Precisión del clasificador TDIDT al utilizar distintas medidas de similitud para construir árboles n-arios con atributos continuos.

	Algoritmo	Medidas de similitud basadas en distancias			
	clásico	Euclídea	Manhattan	Dominio	Bhattacharyya
ADULT	3.42	**3.18**	**3.18**	**3.18**	**3.20**
BUPA	6.56	**2.75**	**2.75**	**2.87**	**3.38**
GLASS	5.08	**4.01**	**3.64**	**4.05**	**4.42**
HEART	3.19	**2.61**	**2.61**	**2.61**	**2.48**
IONOSPHERE	4.91	**4.81**	**4.88**	**4.81**	**4.70**
PIMA	6.81	**2.97**	**2.97**	**2.93**	**3.36**
WAVEFORM	9.20	**6.53**	**6.59**	**6.54**	**6.57**
WINE	2.41	2.41	2.41	2.41	2.44

	Otras medidas de similitud					
	P.Escalar	Correlación	Restle	MinSum	Enta	Gregson
ADULT	4.10	**3.26**	**3.23**	3.99	**3.21**	3.99
BUPA	**4.54**	**3.49**	**4.59**	**4.76**	**4.39**	**4.76**
GLASS	**3.94**	**3.84**	**3.32**	**4.07**	**3.67**	**3.93**
HEART	**3.07**	**2.50**	**2.59**	**2.75**	**2.59**	**2.75**
IONOSPHERE	**3.29**	**4.62**	6.51	**4.70**	**4.73**	**4.71**
PIMA	**4.68**	**3.69**	**4.01**	**5.04**	**4.65**	**5.04**
WAVEFORM	**7.15**	**6.63**	**6.99**	**7.10**	**6.75**	**7.03**
WINE	2.65	2.41	2.41	**2.35**	**2.35**	**2.35**

Tabla 5.13: Profundidad media de los árboles obtenidos.

	Algoritmo	Medidas de similitud basadas en distancias			
	clásico	Euclídea	Manhattan	Dominio	Bhattacharyya
ADULT	88.8	**78.6**	**78.6**	**78.7**	**77.5**
BUPA	32.2	**8.6**	**8.6**	**9.5**	**16.5**
GLASS	22.7	**22.4**	**18.7**	23.3	23.4
HEART	21.5	**20.1**	**20.1**	**20.2**	**19.1**
IONOSPHERE	14.7	15.6	15.1	15.5	18.5
PIMA	58.9	**14.6**	**14.6**	**13.7**	**20.4**
WAVEFORM	291.9	**108.7**	**112.8**	**110.3**	**117.8**
WINE	5.3	**5.2**	**5.2**	**5.2**	5.3

	Otras medidas de similitud					
	P.Escalar	Correlación	Restle	MinSum	Enta	Gregson
ADULT	92.1	**79.4**	**81.7**	90.0	**79.9**	90.0
BUPA	44.8	**10.0**	**15.8**	**19.7**	**18.7**	**19.7**
GLASS	34.3	**20.7**	**13.0**	**22.5**	**20.3**	**22.2**
HEART	22.6	**16.6**	**16.8**	**19.4**	**18.1**	**19.4**
IONOSPHERE	19.0	**11.7**	**12.2**	**12.7**	**13.9**	**12.7**
PIMA	**56.1**	**18.8**	**19.1**	**32.1**	**36.1**	**32.1**
WAVEFORM	**137.1**	**102.5**	**105.6**	**109.6**	**102.5**	**114.5**
WINE	8.1	**5.2**	**5.2**	5.3	5.3	5.3

Tabla 5.14: Número medio de hojas de los árboles obtenidos.

Recapitulación

En las tablas anteriores, los resultados que mejoran el rendimiento del algoritmo C4.5 tradicional aparecen en negrita.

En líneas generales, las medidas de similitud basadas en medidas de distancia tienden a obtener resultados uniformes, mientras que se observa una mayor variabilidad en los experimentos realizados con criterios de similitud basados en medidas de correlación. De las medidas basadas en conjuntos, el modelo de Enta destaca ligeramente. No obstante, en la práctica no se aprecian diferencias notables entre las distintas medidas evaluadas, independientemente del modelo de similitud en que tengan su origen.

Como resumen de nuestros experimentos, la tabla 5.15 muestra los resultados relativos que se obtienen al utilizar la distancia euclídea como medida de similitud frente al algoritmo C4.5 clásico. A pesar de los recursos computacionales adicionales que el método del apartado 5.3.2 requiere, el método jerárquico aglomerativo propuesto para construir árboles n-arios arbitrarios consigue buenos resultados al mantener o incluso mejorar el porcentaje de clasificación obtenido a la vez que reduce notablemente la complejidad del árbol de decisión construido.

	Precisión	Profundidad	Hojas	Tiempo
ADULT	-0.23 %	-7.02 %	-11.49 %	+16.8 %
BUPA	+2.67 %	-58.08 %	-73.29 %	+90.8 %
GLASS	-3.23 %	-21.06 %	-1.32 %	+114.6 %
HEART	+1.48 %	-18.18 %	-6.51 %	+58.7 %
IONOSPHERE	-2.83 %	-2.04 %	+6.12 %	+129.4 %
PIMA	+2.21 %	-56.39 %	-75.21 %	+93.3 %
WAVEFORM	+0.38 %	-29.13 %	-62.76 %	+60.8 %
WINE	+6.34 %	0.00 %	-1.89 %	+39.3 %
Promedio	+0.85 %	-23.99 %	-28.29 %	+75.5 %

Tabla 5.15: Mejoras obtenidas al utilizar la distancia euclídea para construir árboles de decisión n-arios con atributos continuos.

Capítulo 6

Cuestión de infraestructura

Este problema de cerrar la brecha para llegar a cierta parte que funcione es ingeniería. Requiere un estudio serio de problemas de diseño... hay un largo camino por recorrer entre los principios básicos y un diseño práctico y económico.

RICHARD FEYNMAN
Física. Volumen II: Electromagnetismo y Materia

En los capítulos anteriores de esta memoria se han descrito algunas de las técnicas de clasificación que se utilizan para modelar grandes conjuntos de datos (capítulo 2), se ha propuesto un modelo de clasificación a medio camino entre los árboles y las listas de decisión (el modelo ART, capítulo 3), se ha analizado cómo pueden formularse hipótesis al construir el clasificador utilizando técnicas de extracción de reglas de asociación (el algoritmo TBAR, capítulo 4) y se ha estudiado el procesamiento de información cuantitativa para construir modelos simbólicos discretos (capítulo 5). Todas las técnicas examinadas son, sin duda, de gran utilidad para resolver problemas reales en situaciones concretas. Sin embargo, no se puede aprovechar todo su potencial utilizándolas de forma aislada: es imprescindible interpretarlas como herramientas particulares utilizables en un marco más general que las englobe. El estudio de un sistema que proporcione la infraestructura necesaria para tal fin es el objeto del presente capítulo.

Aún hoy, los sistemas informáticos son mucho mejores recopilando datos que utilizándolos de una forma razonable [156]. Por este motivo, durante los últimos años se han desarrollado multitud de técnicas de *Data Mining*, entre las que se encuentra el modelo de clasificación ART propuesto esta memoria. Las grandes posibilidades que ofrecen estas técnicas en aplicaciones de todo tipo han atraído la atención de las grandes multinacionales de la Informática y han hecho posible la creación de un floreciente mercado en el que numerosas empresas desarrollan su actividad [78] [124].

El objetivo de este capítulo es plantear una arquitectura general que facilite la implementación y utilización de técnicas de *Data Mining*, como el modelo de clasificación ART propuesto en esta memoria.

Los sistemas de información, entendidos como sistemas informáticos orientados a la resolución de problemas de un determinado tipo, se dividen básicamente en sistemas de procesamiento de transacciones (OLTP: *On-Line Transaction Processing*) y sistemas de ayuda a la decisión (DSS: *Decision-Support Systems*), entre los que se hallan las aplicaciones OLAP (*On-Line Analytical Processing*), las cuales suelen emplear técnicas de *Data Mining*.

Los sistemas de ayuda a la decisión son sistemas cuyo objetivo es facilitar al usuario la toma de decisiones en situaciones no estructuradas [139]. Dichos sistemas tienen necesidades específicas que no pueden resolverse utilizando sistemas de información convencionales [29] [37] [161]. Los sistemas OLTP tradicionales suelen trabajar con fragmentos relativamente pequeños de información (transacciones) mientras que las aplicaciones de ayuda a la decisión requieren el análisis eficiente de cantidades enormes de datos para satisfacer las expectativas de los denominados 'trabajadores del conocimiento' (ejecutivos y analistas).

Para hacer factible el procesamiento de las ingentes cantidades de datos que organizaciones de todo tipo recopilan durante su funcionamiento cotidiano, es necesario el desarrollo de técnicas de *Data Mining* que, además, sean escalables [128]; es decir, técnicas cuyo rendimiento no se degrade en exceso cuando crece el volumen de datos sobre el que han de trabajar. Las investigaciones en este tema se centran principalmente en la búsqueda de algoritmos más eficientes que reduzcan el espacio de búsqueda de posibles soluciones,

mejoren la calidad de las heurísticas empleadas u optimicen el uso de los recursos computacionales disponibles (por ejemplo, integrando los sistemas de extracción de conocimiento con los gestores de bases de datos que almacenan los datos). También pueden resultar de utilidad el uso de técnicas de muestreo (que permiten reducir el tamaño de los conjuntos de datos), el diseño de técnicas incrementales (que permiten actualizar modelos existentes sin tener que volver a reconstruirlos por completo) y, sobre todo, la implementación de técnicas de procesamiento distribuido y paralelo.

En este capítulo se describe un modelo general de sistema de cómputo adecuado para la resolución de problemas de *Data Mining* y, en general, de cualquier tipo de problemas que requieran gran capacidad de cómputo (como pueden ser, por ejemplo, las aplicaciones científicas tradicionalmente asociadas al uso de supercomputadores). El modelo conceptual de un sistema de este tipo se presenta en la sección 6.1. La implementación real de tal sistema se podría llevar a cabo en el futuro construyendo un sistema distribuido basado en componentes. En cierto modo, los requerimientos de este sistema serían similares a los de un sistema multiagente [21] [144]. De hecho, un sistema de agentes inteligentes, tal como se suele definir en Inteligencia Artificial, no es más que un sistema distribuido basado en componentes desde el punto de vista de la Ingeniería del Software, con la peculiaridad de permitir la migración de los agentes entre distintos nodos del sistema (esto es, no sólo se transfieren datos, sino que también se transfiere el código que ha de ejecutarse).

Los aspectos más relevantes del sistema propuesto se discuten en las siguientes secciones. En el apartado 6.2 se comenta la evolución histórica de los sistemas distribuidos y se especifican las necesidades de infraestructura que un sistema distribuido de cómputo ha de satisfacer. La sección siguiente, 6.3, se centra en el estudio de la arquitectura de un sistema basado en componentes. Los sistemas de este tipo resultan especialmente interesantes para el desarrollo de aplicaciones de *Data Mining* porque proporcionan la infraestructura necesaria para que podamos centrar nuestros esfuerzos en el diseño, implementación y evaluación de algoritmos de extracción de conocimiento. El capítulo se cierra con una discusión sobre algunas decisiones particulares de diseño e implementación que se pueden tomar para crear un sistema como el propuesto a

nivel teórico (sección 6.4) y una declaración de intenciones sobre el papel que las técnicas descritas en este capítulo pueden desempeñar en el futuro (sección 6.5).

6.1. Modelo conceptual del sistema

En Ingeniería del Software es importante que la descripción de un sistema se realice de forma totalmente independiente a cómo se vaya a implementar. La construcción de un modelo independiente de cualquier tecnología particular dota de mayor libertad al diseñador y permite el desarrollo de tecnologías compatibles con tantas técnicas de implementación y estándares como se desee. Esta cualidad está ligada al enfoque MDA (*Model-Driven Architecture*), el cual se basa en la idea de que la estructura y el comportamiento esencial de un sistema ha de capturarse de forma abstracta de forma que pueda adaptarse a diferentes tecnologías en su implementación [11]. Este enfoque no sólo permite que el diseñador pueda escoger la tecnología más apropiada para implementar el sistema, sino que también dota al sistema de la capacidad de sobrevivir a la inevitable evolución tecnológica.

Siguiendo el enfoque MDA, en este capítulo se propone un sistema abierto que permita el desarrollo de algoritmos de extracción de conocimiento en bases de datos. Este sistema ha de incluir la posibilidad de que se puedan realizar tareas computacionalmente costosas para analizar enormes conjuntos de datos, agrupar datos, construir modelos de clasificación y extraer patrones y reglas de asociación. Dicho sistema, de hecho, ha de ser utilizable en la realización de cualquier aplicación de cálculo intensivo (p.ej. simulaciones científicas) y debe proveer los mecanismos necesarios para acceder a los datos de una forma eficiente. En general, el modelo conceptual de un sistema de este tipo se ajusta al de la figura 6.1.

El usuario de un sistema de este tipo ha de trabajar con grandes conjuntos de datos, para lo cual el sistema ha de emplear técnicas y herramientas de *Data Mining*. Los datos, que pueden provenir de fuentes heterogéneas, y los algoritmos de extracción de conocimiento disponibles se emplean para construir modelos que pueden tener funciones descriptivas (esto es, facilitan el análisis

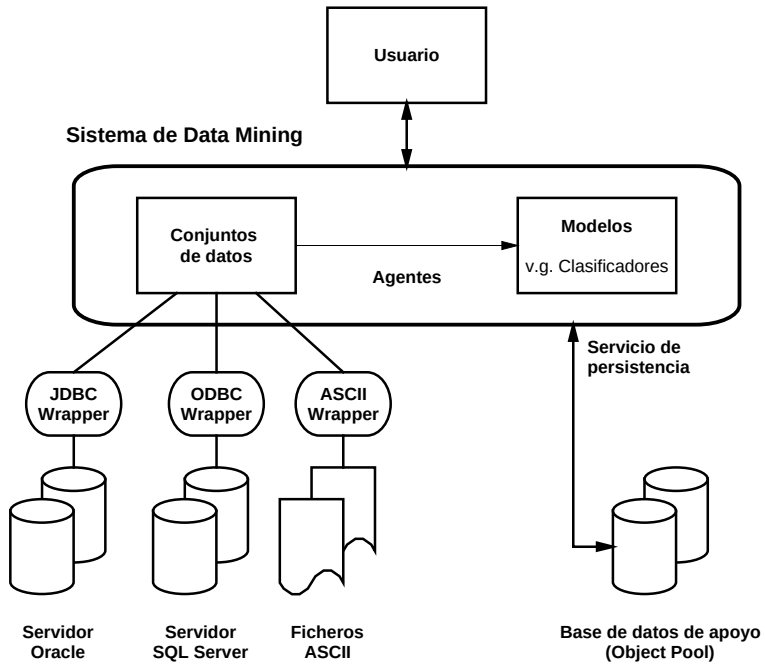

Figura 6.1: Modelo conceptual del sistema

de los datos existentes) o predictivas (cuando son de ayuda en situaciones novedosas). Los modelos construidos pueden también emplearse como entrada de algoritmos de extracción de conocimiento de segundo orden si el usuario desea analizar los modelos ya obtenidos y pueden, también, complementar a los datos de entrada cuando éstos han de ser analizados. Tanto los modelos obtenidos como los metadatos que caracterizan a los conjuntos de datos empleados para obtener dichos modelos han de almacenarse de forma permanente en una base de datos que permita su utilización posterior.

Cualquier sistema de información puede describirse mediante una estructura, una serie de mecanismos y una política según el modelo SMP (*Structure-Mechanism-Policy*) de Perry y Kaiser [123]. La estructura viene dada por los objetos y agregaciones de objetos sobre los cuales operan los mecanismos, que son las herramientas que nos permiten resolver problemas, mientras que la política consiste en una serie de reglas y estrategias impuestas por el entorno.

En nuestro sistema, se ha de proporcionar la infraestructura que permita almacenar los objetos sobre los que se actúa (servicio de persistencia) y mantenga a disposición del usuario los mecanismos que le permiten operar sobre dichos objetos (denominados agentes en el campo de los sistemas inteligentes, o procesos en el ámbito de la Ingeniería del Software). Tanto los objetos como los agentes forman parte del sistema y son ciudadanos de primera clase en el mismo.

- Los **objetos** sobre los que se actúa pueden ser conjuntos de datos almacenados en distintos formatos, ontologías [28] que nos permiten definir jerarquías de conceptos mediante las cuales caracterizar los datos, modelos de clasificación, etcétera.

  Los objetos son entidades permanentes y han de almacenarse de forma persistente para evitar su destrucción (salvo, obviamente, que el usuario expresamente desee eliminarlos). El almacenamiento de tales objetos ha de realizarse de forma transparente al usuario, sin que éste necesite saber ni cómo ni donde se guardan. No obstante, sí es necesario un sistema de recuperación de información que permita al usuario acceder a los objetos existentes en el sistema.

- Los **agentes** proporcionan los mecanismos que se emplean para trabajar con los objetos anteriores: algoritmos y procesos mediante los cuales se resuelven problemas de extracción de conocimiento en bases de datos, incluyendo tareas de preprocesamiento (muestreo, discretización, selección de características...) y de post-procesamiento (exploración de reglas, simplificación y poda, resúmenes...).

  Las instancias particulares de los agentes, a diferencia de los objetos, sólo existen durante la ejecución de la tarea que tengan encomendada, por lo que no es necesario almacenarlos de forma permanente una vez que hayan concluido su misión. Sin embargo, sí resulta necesario almacenar el estado de los agentes para conseguir un sistema fiable. Idealmente, se ha de conseguir una *persistencia ortogonal*, entendida ésta como la posibilidad de que un agente, tras ser interrumpida su ejecución por motivos internos o externos al sistema, tenga la posibilidad de continuar su ejecución por donde se encontrase. En la práctica, no obstante, tendremos que conformarnos con permitir que el agente repita parte del trabajo que hubiese realizado antes de detener su ejecución.

En cuanto al tercer componente del modelo SMP, la existencia de reglas y estrategias que permitan el funcionamiento del sistema, de cara al usuario el sistema es una caja negra respecto a la cual puede desempeñar distintos roles.

El sistema de información de la figura 6.1 puede considerarse un sistema de gestión de modelos (MMS: *Model Management System*). En este tipo de sistemas de ayuda a la decisión hay que diferenciar distintos tipos de usuarios: los usuarios de los modelos existentes se limitan a explorar e interactuar con modelos previamente desarrollados, mientras que los constructores de modelos son los usuarios que crean modelos a partir de los datos y modelos ya existentes. Ambos tipos de usuarios pueden ser científicos, ejecutivos o analistas sin conocimientos informáticos más allá del uso básico de un ordenador personal, por lo cual el sistema ha de ser fácil de usar y su complejidad interna debe quedar oculta de cara al usuario.

En un sistema clásico de ayuda a la decisión, un tercer conjunto de usuarios se encarga de implementar los modelos definidos para que puedan ser uti-

lizados por los usuarios anteriores (en nuestro caso, los agentes que permiten obtener nuevos modelos), mientras que un cuarto y último grupo de usuarios es el responsable de ajustar parámetros, depurar, evaluar, validar y mantener los modelos del sistema. Estas actividades requieren conocimientos que no suelen poseer los usuarios finales del sistema y, además, suelen ser automatizables en parte. Las tareas automatizables pueden diseñarse para que el usuario final se encargue de gestionarlas y goce así de mayor autonomía. El objetivo final es que un usuario no experto pueda sacar el máximo partido de un sistema avanzado de *Data Mining*.

En definitiva, el diseño del sistema debe ayudar a que los usuarios puedan descubrir información útil a partir de sus datos sin tener que ser expertos en técnicas de *Data Mining* [122] y ser fácil de usar [89], pero tampoco debe olvidar que se trata de un entorno dinámico que ha de permitir el uso de nuevas técnicas en cuanto estén disponibles. Para mantener un tiempo de respuesta aceptable de cara al usuario, el sistema se puede implementar en un entorno distribuido (sección 6.2). Por otro lado, dado que el sistema ha de estar preparado para adaptar su configuración a situaciones cambiantes y para evolucionar incorporando nuevas técnicas y algoritmos, es recomendable diseñarlo como un sistema basado en componentes en el cual se puedan añadir y eliminar agentes de forma dinámica (sección 6.3).

6.2. Sistemas distribuidos

Los sistemas distribuidos pueden ofrecer servicios valiosos compartiendo los recursos computacionales de un gran número de usuarios, tanto ciclos no utilizados de CPU como espacio libre de almacenamiento en disco. Tales sistemas permiten formar redes virtuales cuyos nodos de procesamiento se pueden encontrar distribuidos geográficamente, a diferencia de los grandes y costosos sistemas centralizados utilizados mayoritariamente en la actualidad, y pueden disponer de una potencia de cómputo similar a la de los supercomputadores más potentes.

6.2.1. Evolución y tendencias

El modelo tradicional cliente/servidor en el cual un "pequeño" cliente solicita y recibe información de un "gran" servidor está comenzando a dejar paso a otro tipo de arquitectura: un modelo en el cual todos los participantes comparten responsabilidades y son aproximadamente iguales. Este modelo es conocido por el acrónimo P2P, el cual proviene de la expresión inglesa *Peer-to-peer*, y muchos ven en él una de las tecnologías que marcarán el futuro de Internet [72] [160].

6.2.1.1. Sistemas P2P

Los sistemas P2P se caracterizan por ser sistemas distribuidos que no dependen inherentemente de puntos centralizados de control, aunque también es cierto que no excluyen la posibilidad de utilizarlos cuando sea conveniente. Los nodos de una red P2P se relacionan con sus vecinos (desde el punto de vista lógico) y se caracterizan por su capacidad para descubrir otros nodos y conectarse con ellos sin que haya una relación preestablecida entre ellos, tal como sucede en los sistemas cliente/servidor.

Los sistemas P2P tienen el potencial de aprovechar tres recursos de Internet que, hoy en día, el modelo cliente/servidor utiliza por debajo de sus posibilidades: la información disponible, el ancho de banda de las redes que componen Internet y, especialmente, la capacidad de cómputo y de almacenamiento de las máquinas conectadas a la Red. Además de la evidente infrautilización de recursos, el modelo actual ha propiciado la aparición de varios problemas:

- Independientemente del ancho de banda disponible, determinadas zonas de Internet se siguen saturando cuando todo el mundo accede a portales como Yahoo!, desea leer las últimas noticias de la CNN o intenta leer su correo electrónico de alguno de los grandes proveedores de acceso a Internet. Al mismo tiempo, el tráfico en otras zonas de la Red se mantiene muy por debajo de su capacidad.

- A pesar de las mejoras tecnológicas, las necesidades de cómputo y almacenamiento se han acumulado en torno a centros de procesamiento de

datos cuyo crecimiento y necesidades energéticas ha dado lugar incluso a problemas de suministro eléctrico.

- La existencia de sistemas centralizados facilita la censura y el espionaje, comprometiendo la privacidad de la información que se transmite a través de la red.

Frente a la infrautilización de los recursos disponibles y los problemas ocasionados por el modelo cliente/servidor, las tecnologías P2P pueden emplearse en distintos tipos de aplicaciones:

- **Sistemas de publicación de información**

  El intercambio de ficheros de sonido en formato MP3 y de vídeo en formato DivX ha sido, sin duda, la aplicación que ha conseguido alcanzar la masa crítica necesaria para el desarrollo de las tecnologías P2P. Además de compartir ficheros, los usuarios de estos sistemas comparten responsabilidades legales. No obstante, al ser relativamente fácil mantener el anonimato en estos sistemas, resulta difícil (si no imposible) anular su expansión. Algunos sistemas pertenecientes a esta categoría son muy conocidos:

  – Napster, un sistema híbrido C/S-P2P que puso en pie de guerra a la industria discográfica,

  – Gnutella, que sí es un auténtico sistema P2P, aunque mal diseñado porque su algoritmo de enrutamiento por inundación genera muchísimo tráfico innecesario,

  – Kazaa, tristemente conocido por ser un medio eficaz para difusión de virus informáticos,

  – MojoNation, que incluye mecanismos de recompensa a los usuarios en función de los servicios que prestan; y,

  – eDonkey, que transfiere fragmentos de archivos en paralelo y de forma independiente para mejorar el rendimiento del sistema.

También existen otros sistemas de este tipo con aspiraciones más nobles, como Publius (un sistema resistente a cualquier tipo de censura) o Freenet (un almacén de información virtual que permite a cualquier persona publicar o acceder a cualquier tipo de información, asegurando la privacidad del lector/editor así como la seguridad, la integridad y la disponibilidad de la información publicada [36]).

- **Sistemas de comunicación interpersonal**

  Desde los grupos de noticias de UseNet o el intercambio de mensajes entre BBSs con FidoNet hasta llegar a los programas de chat (ICQ o mIRC) y los sistemas de mensajería instantánea (como MSN Messenger), los sistemas distribuidos de comunicación siempre han gozado de gran popularidad entre los internautas. Dichos sistemas, distribuidos y descentralizados, se pueden considerar ejemplos clásicos de sistemas P2P. Estos sistemas existían antes de que se acuñase el acrónimo P2P, si bien utilizan básicamente las mismas técnicas que los sistemas P2P actuales. Algunos de los sistemas más recientes, como Jabber, aprovechan el desarrollo actual de las tecnologías P2P para mejorar el rendimiento de los sistemas clásicos.

  Por otro lado, la evolución de los productos software orientados a facilitar el trabajo en grupo (*groupware*) también ha dado lugar a sistemas P2P como los comercializados por Groove Networks para la gestión de proyectos distribuidos geográficamente en los que pueden colaborar miembros de distintas organizaciones.

- **Sistemas distribuidos de cómputo**

  Un tercer grupo de sistemas que ya han sido implementados con éxito lo constituyen las aplicaciones de cálculo intensivo. La arquitectura de estos sistemas es de especial interés para cualquier investigador interesado en el desarrollo de sistemas de *Data Mining*, los cuales no sólo necesitan una gran potencia de cálculo, sino que requieren una distribución eficiente de los datos sobre los que se trabaja.

  El proyecto SETI@Home fue pionero en este ámbito. Mediante una apli-

cación específica que se ejecuta como protector de pantalla, se aprovecha el tiempo de CPU no utilizado por miles de usuarios que tienen su ordenador conectado a Internet para buscar posibles señales de inteligencia extraterrestre analizando la información recibida del espacio por radiotelescopios. Otros proyectos actuales de este tipo son *eOn* (que simula procesos químicos a escala atómica), *Genome@Home* o *Folding@Home* (ambos dedicados a profundizar en los secretos de la vida). Además de estos proyectos sin ánimo de lucro, también existen empresas que suministran servicios de cómputo distribuido como Parabon Computation, Entropia, United Devices o Avaki.

En la actualidad, existen numerosos proyectos de investigación y desarrollo relativos a la construcción de supercomputadores distribuidos autoconfigurables, proyectos a los que se hace referencia con el anglicismo *grid computing*. El proyecto *Globus* (`http://www.globus.org`) es actualmente el estándar de facto en este campo, si bien la Unión Europea subvenciona un proyecto paralelo denominado *DataGrid*.

Aparte de las aplicaciones descritas en los párrafos anteriores, los sistemas P2P pueden llegar a utilizarse para construir sistemas distribuidos de recuperación de información que eliminen las limitaciones de los sistemas actuales (ya que ningún motor de búsqueda puede mantener actualizado un catálogo completo del contenido de Internet) o directorios distribuidos (que, en su día, podrían sustituir el sistema de nombres actual, DNS, cuya disponibilidad depende de un conjunto de servidores centrales repartidos por medio mundo). También existen soluciones comerciales que hacen uso de técnicas P2P para distribuir contenidos por Internet (*streaming* de audio y vídeo) o realizar copias de seguridad de forma automática.

Dado el auge de estos sistemas distribuidos semi-autónomos, las grandes multinacionales informáticas han mostrado su interés en ellos. Aunque la viabilidad comercial de los sistemas P2P está aún por demostrar, nadie quiere perder la posibilidad de obtener su cuota de mercado en este sector. Puede que el modelo P2P no resulte adecuado para la mayor parte de las aplicaciones informáticas, pero sin duda puede resultar útil para aplicaciones específicas de

carácter científico y técnico.

La popularidad del acrónimo P2P ha propiciado que muchos sistemas sean anunciados como sistemas P2P de forma más que discutible (igual que sucedió en su momento con las bases de datos relacionales o los sistemas orientados a objetos). Por ejemplo, el controvertido proyecto *Hailstorm* de Microsoft (rebautizado como plataforma *.NET My Services*) incorpora un sistema de autentificación global (*.NET Passport*) para facilitar el comercio electrónico y un sistema de notificación (*.NET Alerts*) que pretende ayudar a las empresas a mejorar su eficacia y las relaciones con sus clientes. La plataforma, destinada a "construir aplicaciones centradas en el usuario", se ofrece como un conjunto de servicios web y, si bien difícilmente puede considerarse un sistema P2P, descentraliza algunas funciones que estarían centralizadas en un sistema cliente/servidor clásico.

A pesar de que la plataforma promovida por Microsoft no sea completamente distribuida y abierta, sí marca un giro en las tendencias actuales: Los sistemas distribuidos tienden a utilizar protocolos abiertos de comunicación y XML para intercambiar datos [40]. Proyectos alternativos, como el proyecto JXTA (de 'yuxtaposición') iniciado por Sun Microsystems [162], pretenden llegar más allá y generalizar el uso de protocolos P2P abiertos que permitan comunicarse a cualquier dispositivo (desde teléfonos móviles a ordenadores personales y PDAs) y colaborar con otros dispositivos de la red sin estar sujetos a ningún estándar propietario. Los servicios web [41] y el lenguaje XML [16] se perfilan como la base sobre la que se construirán los sistemas distribuidos del futuro.

Anderson y Kubiatowicz van más allá de los sistemas P2P proponiendo la creación de un sistema operativo global que permitiese conectar la capacidad de procesamiento y de almacenamiento de millones de ordenadores independientes [10]. Su supercomputador virtual, gestionado por ISOS (*Internet-Scale Operating System*), permitiría alquilar tiempo y espacio en ordenadores personales, haciendo en gran medida innecesarios los costosos centros de procesamiento de datos conocidos popularmente como "granjas de servidores". No obstante, no todo el mundo está de acuerdo en que sistemas P2P de tal magnitud lleguen a existir algún día [61].

6.2.1.2. La taxonomía IFCS

Si empleamos la taxonomía IFCS (*Individual-Family-City-State*) que Perry y Kaiser utilizaron para caracterizar la evolución de los entornos de desarrollo de software [123], podemos observar cómo los sistemas de cómputo intensivo necesarios para determinado tipo de aplicaciones (como es el caso de los sistemas utilizados para resolver problemas de *Data Mining*) parecen desarrollarse de un modo similar al delineado por dicha taxonomía:

- **Individuo (I)**: Multiprocesadores y supercomputadores de propósito específico se emplean para resolver problemas de cálculo intensivo.

- **Familia (F)**: Conjuntos de ordenadores conectados a una red local de alta velocidad coordinan su funcionamiento para formar multicomputadores.

- **Ciudad (C)**: Múltiples sistemas, puede que geográficamente distribuidos, cooperan en la resolución de problemas específicos, como es el caso de los sistemas P2P actuales.

- **Estado (S)**: Un sistema de coordinación, como podría ser un futuro ISOS [10] a nivel global, proporciona la infraestructura necesaria para que la cooperación entre ordenadores independientes no se limite a casos puntuales.

6.2.2. Requisitos del sistema

El desarrollo de un sistema distribuido flexible que permita resolver problemas de cálculo reservados a los supercomputadores más potentes y pueda ser empleado eficazmente en aplicaciones OLAP requiere tener en cuenta distintos aspectos:

6.2.2.1. Comunicación entre nodos de procesamiento

La existencia de un mecanismo de comunicación común es imprescindible para que conjuntos de agentes (desarrollados potencialmente de forma independiente) puedan coordinar su trabajo de una forma útil. Un lenguaje estándar de comunicación entre agentes garantiza que los mensajes enviados sean correctos sintácticamente y puedan ser interpretados de un modo consistente desde un punto de vista semántico.

De hecho, cualquier sistema distribuido necesita algún mecanismo de comunicación que sirva de enlace entre los distintos nodos del sistema. Un sencillo protocolo como SOAP [41] es suficiente para que sistemas autónomos puedan comunicarse entre sí aprovechando los protocolos estándar de Internet, pues SOAP puede funcionar sobre HTTP (el protocolo utilizado para transmitir páginas web en Internet) o SMTP (utilizado para enviar mensajes de correo electrónico).

Más importante si cabe que el mecanismo concreto de comunicación es la topología de la red de comunicación. Básicamente, existen cuatro arquitecturas generales que se utilizan para construir topologías más complejas:

- En un sistema centralizado múltiples clientes acceden a un servidor central. La figura 6.2 muestra la topología de los sistemas cliente/servidor típica de las redes en estrella. En estos sistemas, si el servidor central falla, todo el sistema se viene abajo. Además, las escalabilidad del sistema viene limitada por la capacidad del servidor central (su capacidad de cómputo y el ancho de banda al que pueda dar servicio).

- Un sistema conectado en anillo como el de la figura 6.3 permite un mayor grado de tolerancia ante la presencia de fallos en máquinas concretas del anillo y es más escalable.

- Un sistema jerárquico, figura 6.4, tiene topología en forma de árbol. Los servidores de nombres en Internet que proporcionan el servicio DNS se organizan de esta forma. Los sistemas jerárquicos sobrellevan mejor las limitaciones de los sistemas centralizados y han demostrado sobradamente su escalabilidad (de hecho, el servicio DNS original sobrevive aún a la explosión de Internet).

- Una arquitectura completamente descentralizada, como la que aparece en la figura 6.5, es la empleada por algunos de los sistemas P2P actuales (p.ej. Gnutella). Estos sistemas son difíciles de gestionar y pueden presentar graves problemas de seguridad. Además, su escalabilidad es difícil de evaluar por la carga adicional que supone mantener la coherencia y la consistencia de un sistema no estructurado.

Los resultados más interesantes se consiguen, no obstante, cuando se combinan distintas topologías para conseguir arquitecturas híbridas que aprovechan las ventajas y disminuyen los inconvenientes de las topologías que las forman:

- Un sistema centralizado en el cual se sustituye el servidor por un anillo, figura 6.6, resulta especialmente útil para mantener la sencillez de un sistema centralizado y la redundancia que ofrece un anillo. Esta arquitectura es útil para mejorar la escalabilidad de un sistema centralizado.

- De forma alternativa, el anillo puede combinarse con un modelo centralizado para facilitar la implementación de aplicaciones de propósito específico [55], tal como muestra la figura 6.7.

- También se puede construir un sistema jerárquico introduciendo conexiones redundantes entre los nodos del árbol para mejorar la tolerancia del sistema ante posibles fallos. El sistema FreeNet se basa en una topología de este tipo [36].

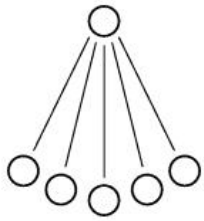

Figura 6.2: Sistema centralizado cliente/servidor

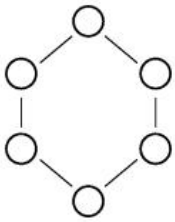

Figura 6.3: Red en anillo

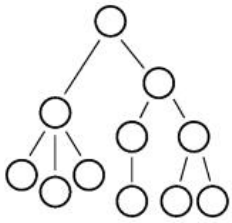

Figura 6.4: Sistema jerárquico

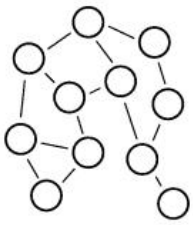

Figura 6.5: Sistema descentralizado

- Un sistema descentralizado híbrido en el que partes del sistema se centralizan, como el de la figura 6.8, goza de mayor flexibilidad y es más fácil de gestionar que un sistema descentralizado puro. Este modelo es especialmente útil cuando el sistema rebasa los límites de una organización concreta y se convierte en un servicio compartido por distintas entidades autónomas. El correo electrónico en Internet puede tomarse como modelo clásico de este tipo de sistemas: un conjunto de servidores de correo se encargan de intercambiar mensajes mientras que los usuarios finales del sistema siempre acceden a servidores concretos.

Como parece lógico, esta última arquitectura es la que consigue un compromiso mejor entre la sencillez de una topología centralizada y la flexibilidad extrema de un sistema descentralizado. Cada nodo central de la red estará conectado a otros nodos centrales y tendrá, además, conexiones con nodos satélite que pueden aportar recursos computacionales sin tener acceso completo al resto de la red.

6.2.2.2. Acceso a los datos

Otro aspecto importante relativo a la comunicación entre los distintos componentes de un sistema distribuido cuando pretendemos construir un sistema OLAP es decidir qué estrategia seguir para acceder a los datos que se utilizan como fuente de información a la hora de construir modelos.

Tradicionalmente, los algoritmos de aprendizaje se han implementado de forma que el usuario accede a ellos a través, posiblemente, de una interfaz gráfica. Si bien dichos algoritmos suelen acceder a datos que se hallan almacenados en ficheros locales, existe la posibilidad de que los datos estén almacenados en una base de datos a la cual se accede utilizando un lenguaje de consulta estándar (como es el caso de SQL para las bases de datos relacionales). En esta situación, además, se han de tener en cuenta las distintas posibilidades de acoplamiento que pueden existir entre la implementación del algoritmo de *Data Mining* y el sistema gestor de bases de datos, tal como se estudian en [6]. En el artículo de Maniatty y Zaky en SIGKDD Explorations [106] se comentan distintas técnicas que se pueden utilizar a bajo nivel para

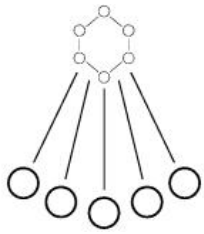

Figura 6.6: Sistema centralizado con anillo.

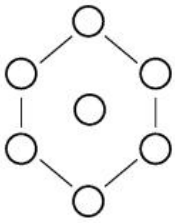

Figura 6.7: Sistema en anillo con coordinación centralizada.

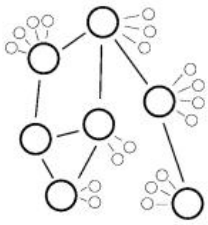

Figura 6.8: Sistema híbrido centralizado-descentralizado

permitir un acceso eficiente a los datos, los cuales pueden provenir de fuentes de datos heterogéneas.

6.2.2.3. Dinámica del sistema

Además de utilizar un mecanismo básico de comunicación entre nodos de procesamiento y de poseer una topología específica, un sistema distribuido dinámico necesita disponer de servicios adicionales que le permitan funcionar correctamente. Estos servicios serán los responsables de que el sistema se adapte a situaciones cambiantes de forma autónoma.

Los estándares en que se basan los servicios web [41] pueden sernos de gran utilidad en este sentido: el lenguaje WSDL (*Web Services Description Language*) permite dar a conocer los servicios que cada nodo del sistema ofrece, así como la forma de acceder a ellos, mientras que un directorio UDDI (*Universal Description, Discovery, and Integration*) facilita a los usuarios del sistema el acceso a los distintos proveedores de servicios a modo de 'guía telefónica'. Es decir, WSDL puede emplearse para divulgar los servicios que ofrece cada nodo del sistema distribuido y UDDI puede utilizarse para que nuevos nodos sean capaces de descubrir sistemas y servicios, así como averiguar cómo sumarse al sistema distribuido.

Aparte de los servicios descritos en el párrafo anterior, que son comunes a todas las aplicaciones distribuidas que han de funcionar en un entorno dinámico, un sistema distribuido de cómputo debe proveer los mecanismos necesarios para poder delegar tareas en otros nodos, transfiriendo el código que ha de ejecutarse y suministrando los datos que sean necesarios. Si consideramos que las tareas de cómputo son realizadas en el sistema por agentes semi-autónomos, podemos afirmar que es imprescindible que tales agentes sean capaces de migrar de un nodo a otro. La movilidad de los componentes del sistema permite distribuir de una forma razonable la carga del sistema en su conjunto [117] pero también puede verse forzada por razones externas al sistema en sí (por ejemplo, cuando se pierden conexiones con una región determinada de la red sobre la que se opera el sistema distribuido). Un mecanismo transparente de migración debería, en principio, ser capaz de continuar la ejecución del agente desde el punto en el que se encontrase antes de migrar, sin perder los cálculos

que ya hubiese efectuado e, incluso, sin que el agente sea consciente de que ha sido trasladado.

Además de los mecanismos básicos que permiten que el sistema vaya evolucionando de una forma dinámica, sería en principio deseable que el sistema de comunicación no fuese tan rígido como los estrictos protocolos utilizados habitualmente para implementar software en sistemas distribuidos. Mediante la especificación explícita de políticas de funcionamiento no implícitas en la implementación del sistema, se pueden conseguir sistemas reconfigurables en los cuales los nodos gozan de cierta capacidad de decisión. Algunos trabajos realizados en el campo de los agentes inteligentes [21] pueden ser muy útiles en este sentido.

Por ejemplo, se pueden utilizar autómatas finitos para hacer que la comunicación entre nodos se realice mediante una conversación más flexible que una secuencia estructurada de mensajes [21]. Si A realiza una oferta para hacer uso de un servicio ofrecido por B, B puede aceptar la oferta de A, puede declinarla, puede no responder siquiera o podría, incluso, pedir una clarificación sobre la oferta realizada u ofrecer una estimación del tiempo que tardará en realizar la tarea encomendada (para saber si la oferta seguirá en pie o expirará si B decide aceptarla más adelante).

6.2.2.4. Seguridad y fiabilidad

Un sistema distribuido dinámico como el descrito en párrafos anteriores ha de incorporar los mecanismos de seguridad necesarios para garantizar la integridad de los datos que se transmiten entre los distintos nodos de procesamiento, así como evitar la pérdida de datos cuando parte del sistema sufre una avería.

La seguridad es esencial para evitar la inspección y/o manipulación tanto de los datos que se manejan como del código que se ha de ejecutar en su tránsito de un nodo a otro. Si utilizamos una arquitectura híbrida como la de la figura 6.8, se ha de establecer un perímetro de seguridad en torno a los nodos centrales de la red. Estos nodos son los que poseen información sobre la topología de la red y sirven de intermediarios cuando se transmiten datos de un punto a otro. La seguridad en los nodos periféricos no ha de ser tan

estricta, aunque siempre es conveniente que se utilicen técnicas criptográficas de protección de datos a la hora de transmitir y almacenar información. Dichas técnicas permiten un intercambio seguro de información confidencial a través de un medio compartido con usuarios y sistemas [26].

Por otro lado, para asegurar la fiabilidad del sistema se ha de introducir cierta redundancia que permita mejorar su disponibilidad y su tolerancia frente a posibles fallos. La transparencia que otorga la movilidad de código y datos descrita en el apartado anterior permite, además, que el sistema pueda replicar agentes y ejecutarlos en máquinas independientes por motivos de seguridad o fiabilidad, así como hacer copias de seguridad de la información que sea vital para el funcionamiento del sistema en su conjunto.

El uso de recursos en el sistema debe estar monitorizado y controlado en cada nodo, para mantener una contabilidad del consumo de cada agente y detectar comportamientos anómalos. La existencia de mecanismos de monitorización es esencial para detectar intentos de acceso no autorizados y mantener el uso autorizado de recursos dentro de unos límites razonables (de forma que no interfiera con otras tareas paralelas).

6.2.2.5. Planificación y asignación de recursos

Además de los mecanismos descritos en las secciones anteriores que permiten que los distintos nodos del sistema se comuniquen entre sí de una forma segura, un sistema distribuido necesita disponer de una estrategia de planificación y asignación de recursos.

En un sistema descentralizado como el de la figura 6.8, en el que un nodo particular sólo posee información parcial acerca del estado de la red en su conjunto, se han de diseñar heurísticas que permitan a los nodos del sistema negociar de forma autónoma con sus vecinos el reparto de tareas entre ellos. Este reparto ha de realizarse en función de la capacidad de cómputo, espacio de almacenamiento y ancho de banda disponibles en los distintos nodos de procesamiento, si bien también pueden influir otros factores.

Cuando los distintos nodos del sistema distribuido no están bajo el control de una misma organización, además, es necesario establecer un mecanismo de compensación que remunere a los distintos nodos del sistema en función de

los servicios prestados (tiempo de CPU o espacio de almacenamiento). Esta recompensa puede corresponderse con un pago real, aunque también puede emplearse para formar un sistema económico virtual en el cual se ofrezcan incentivos que motiven la cooperación de los integrantes del sistema. Un sistema de este tipo es análogo a un banco en el que cada cliente posee una cuenta en la que cobra por los servicios prestados a terceros y mediante la que paga por lo que consume. Por tanto, aun siendo un sistema básicamente descentralizado, es necesaria una entidad central que se encargue de contabilizar el uso de recursos que realiza cada agente del sistema. Obviamente, este sistema económico es innecesario si toda la infraestructura que da soporte al sistema distribuido de cómputo se encuentra bajo el control de una única organización (como puede ser el caso de la Intranet de una empresa).

6.3. Sistemas basados en componentes

Durante los últimos años [57] [95], hemos presenciado el vertiginoso desarrollo de los sistemas basados en componentes, hoy en día utilizados en los sistemas informáticos de casi todas las empresas. Dichos sistemas pretenden ayudar a los desarrolladores en la construcción de sistemas cada vez más complejos. El objetivo final de tales sistemas es mejorar la productividad del proceso de desarrollo de software promoviendo la reutilización de componentes y de patrones de diseño, un sueño perseguido por la Ingeniería del Software desde sus orígenes.

El origen de los sistemas actuales basados en componentes se puede encontrar en métodos de análisis y diseño como ROOM (*Real-Time Object-Oriented Modeling*), tal como se comenta en [11]. A diferencia de otros métodos de análisis que se centran en un modelado estructural utilizando básicamente diagramas de clases y diagramas de modelización de datos (E/R o CASE*Method), ROOM interpreta el software como un conjunto de procesos que interactúan: se sustituyen los tipos de datos abstractos (clases de objetos definidos por una especificación independiente de su representación) por actores que encapsulan procesos además de datos. Estos actores (agentes, si utilizamos la terminología actual) disponen de una serie de puertos que definen los men-

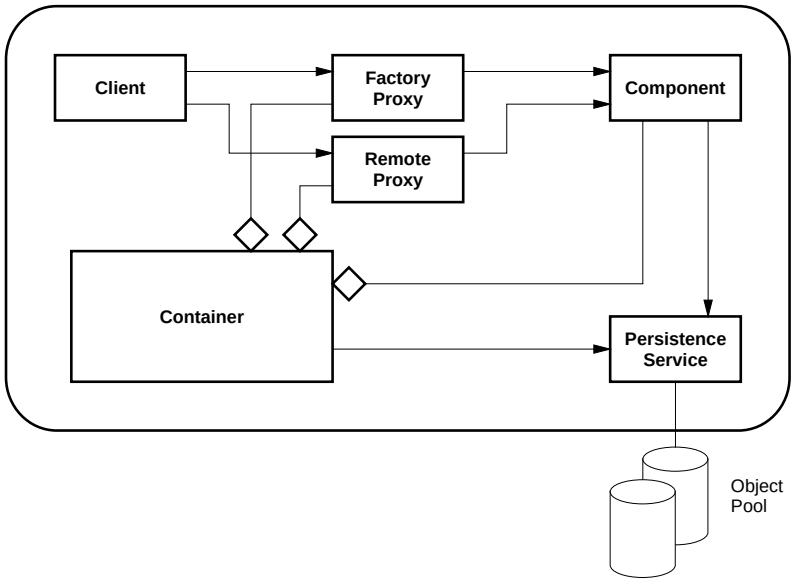

Figura 6.9: Patrón de diseño de los sistemas basados en componentes

sajes que pueden recibir o enviar; es decir, definen la interfaz del componente. Aunque se ideó antes de la aparición de la moda que han marcado tecnologías basadas en componentes como CORBA, COM/COM+ o Java Beans, ROOM incluye las características básicas de un sistema basado en componentes.

6.3.1. Patrón de diseño

Los sistemas basados en componentes utilizados en la actualidad, como los Enterprise JavaBeans de Java 2 Enterprise Edition [99] o la plataforma .NET de Microsoft [113], se basan todos en un patrón arquitectónico común [91]. La figura 6.9 muestra una representación simplificada de este patrón de diseño [65]. El patrón, que se puede modelar como una colaboración parametrizada en UML [116], incluye seis roles que aparecen como rectángulos en la figura 6.9:

- **Cliente (*client*):** Cualquier entidad que solicita un servicio de un componente del sistema. En los sistemas actuales (OLTP), tal solicitud se realiza invocando un método de alguna de las interfaces de los componentes. En un sistema OLAP, la solicitud se podría realizar utilizando un lenguaje de consulta específico para tareas de *Data Mining*, como DMQL u *OLE DB for Data Mining* (de Microsoft).

  En vez de acceder al componente directamente, el cliente utiliza internamente un par de intermediarios que transmiten las llamadas del cliente al componente. Este nivel de indirección, oculto desde la perspectiva del cliente, hace posible que el cliente no tenga que ser consciente de la localización real del componente al que accede y permite además que se puedan interceptar los mensajes que se transmiten con el fin de depurar, controlar y monitorizar el funcionamiento del sistema.

- **Intermediario constructor (*factory proxy*):** A través de él se facilita el acceso a los métodos estáticos de las clases que definen el interfaz de los componentes. Es el encargado de permitir operaciones genéricas como la creación de componentes o la búsqueda de un componente concreto.

- **Intermediario remoto (*remote proxy*):** Se encarga de facilitar el acceso a los métodos que gobiernan el estado y funcionamiento de las instancias particulares de los distintos componentes del sistema. Maneja operaciones específicas para cada tipo de componente, como inspeccionar su estado, fijar sus parámetros, etcétera.

- **Componente (*component*):** Son los bloques lógicos que forman el sistema. En el caso particular que nos ocupa, tanto los conjuntos de datos como los modelos construidos a partir de ellos son componentes del sistema. Los agentes que implementan los algoritmos de extracción de conocimiento también pueden considerarse componentes del sistema, si bien usualmente sólo se emplean a través del intermediario constructor para invocar la construcción de modelos de conocimiento (aunque tampoco se descarta la posibilidad de que se pueda acceder a ellos mientras dure su ejecución).

- **Contenedor (*container*):** Representa el entorno del sistema en tiempo de ejecución y es la parte del sistema encargada de proporcionar la infraestructura necesaria para su funcionamiento. El contenedor contiene tanto a los componentes como a los intermediarios que permiten a un cliente acceder a los componentes, tal como muestran las agregaciones en el diagrama de la figura 6.9.

  El contenedor ha de ofrecer los servicios necesarios para un sistema de cómputo distribuido, incluyendo dispositivos de seguridad que protejan los datos de accesos no autorizados, mecanismos de comunicación entre procesos/agentes, un servicio de persistencia que permita almacenar de forma permanente las instancias concretas de los distintos tipos de componentes del sistema y la capacidad de agregar dinámicamente al sistema nuevos agentes y componentes sin tener que detener su ejecución, algo conocido en los sistemas actuales por la expresión 'despliegue en caliente' (*hot deployment*).

- **Servicio de persistencia (*persistence service*):** Este servicio permite el almacenamiento permanente de los componentes del sistema y su funcionamiento ha de ser gestionado y coordinado por el contenedor de forma autónoma. En la base de datos que da soporte a este servicio (*object pool*) han de almacenarse tanto los metadatos que se empleen para caracterizar los conjuntos de datos utilizados como los modelos obtenidos a partir de ellos y toda la información relativa a las sesiones que los usuarios tengan abiertas en el sistema (esto es, los agentes que estén en ejecución). La información relativa a los objetos (conjuntos de datos y modelos) se almacena para permitir su uso futuro en otras sesiones, mientras que la relativa a los agentes se guarda por cuestiones de fiabilidad. Con el objetivo de permitir que el sistema sea tolerante a fallos, se preserva el estado del sistema en ejecución para que éste se pueda recuperar tras sufrir algún tipo de avería o corte de suministro eléctrico, por ejemplo.

Los sistemas de este tipo existentes en la actualidad están orientados al desarrollo de aplicaciones OLTP, por lo que únicamente suelen suministrar servicios básicos de procesamiento de información. Aquí se propone confeccionar un sistema de este tipo adaptado a las necesidades de las aplicaciones OLAP y de los problemas de extracción de conocimiento en bases de datos. Por extensión, un sistema como el planteado en este capítulo también resulta adecuado para un gran abanico de aplicaciones científicas y técnicas que requieren una elevada capacidad de cómputo sólo obtenible con supercomputadores o sistemas distribuidos como los descritos en la sección anterior de este capítulo.

Los sistemas basados en componentes actuales, como los Enterprise JavaBeans o los componentes COM+, suelen incluir entre sus servicios la posibilidad de realizar transacciones distribuidas (conjuntos de operaciones que han de efectuarse de forma atómica, de modo que, si algunas de ellas no se completa con éxito, los efectos que hayan producido las demás han de anularse). En un sistema multiagente como el propuesto aquí este servicio es innecesario. Sin embargo, se necesitan otra serie de servicios que usualmente no ofrecen los sistemas existentes: son imprescindibles mecanismos de planificación y asignación de recursos que permitan afinar el rendimiento del sistema distribuido y también son necesarios mecanismos de monitorización y notificación que permitan gestionar la ejecución de las tareas que realizan los agentes del sistema. Así mismo, también se necesitan mecanismos que doten al sistema de mayor flexibilidad, de forma que sea capaz de reconfigurarse sobre la marcha y evolucionar [9].

Los servidores de aplicaciones existentes en el mercado desempeñan las funciones del contenedor en el modelo de la figura 6.9. Aunque sin duda son adecuados para proporcionar soluciones de comercio electrónico, como demuestra su popularidad, tanto en sistemas B2B (*business-to-business*) como aplicaciones B2C (*business-to-customer*), los sistemas actuales carecen de los mecanismos de control adicionales que requieren las aplicaciones de cálculo intensivo como las de extracción de conocimiento en bases de datos. Los sistemas basados en componentes diseñados para este tipo de aplicaciones deben incluir un núcleo que proporcione la capacidad de monitorizar el uso de recur-

sos en el sistema (tanto tiempo de CPU como ocupación de memoria, espacio de almacenamiento disponible o ancho de banda consumido por las operaciones de entrada/salida), gestionar la evolución dinámica del sistema y, a la vez, ofrecer a los programadores un entorno lo suficientemente flexible en el que diseñar sus algoritmos y desarrollar nuevas técnicas de computación distribuida. En el siguiente apartado se comentan brevemente algunas de las características que debe poseer el núcleo de un sistema basado en componentes apto para el tipo de aplicaciones que nos interesa (esto es, aplicaciones OLAP de ayuda a la decisión).

6.3.2. El kernel del sistema

El núcleo o kernel de un sistema basado en componentes como el descrito en el apartado anterior debe ejecutarse en cada una de las máquinas que forman el sistema distribuido. Dicho núcleo, al que de aquí en adelante denominaremos ETREK*, es responsable de la realización efectiva de las siguientes tareas:

- Respecto a los clientes del sistema (que pueden ser usuarios accediendo al mismo a través de una interfaz web u otros programas que contratan los servicios del sistema), ETREK se encarga de
  - gestionar las sesiones abiertas (requiriendo la autentificación de los usuarios cada vez que se conectan al sistema),
  - procesar las solicitudes de los usuarios (por ejemplo, construir un modelo mediante la realización de una tarea de *Data Mining* por parte de un agente),
  - recopilar metainformación acerca de los conjuntos de datos que se utilizan como punto de partida en el proceso de extracción de conocimiento, y

*ETREK es un acrónimo que proviene de *EnTerprise Run-time Environment Kernel*, el núcleo de un sistema basado en componentes, aún en fase de diseño, que tiene su origen en el desarrollo de *TMiner*, un sistema de *Data Mining* que incluye algoritmos de extracción de reglas de asociación, construcción de clasificadores y algunos métodos de agrupamiento.

  - dar acceso a los modelos de conocimiento que hayan sido obtenidos con anterioridad por el usuario o por otros usuarios del sistema (teniendo en cuenta posibles restricciones de acceso).

- En relación con los agentes del sistema (procesos encargados de realizar las tareas de cálculo necesarias para la obtención de modelos de conocimiento), ETREK se encarga de la planificación y asignación de recursos, así como de la monitorización del funcionamiento del sistema. También es responsable de la notificación al usuario de la terminación de una tarea cuando éste así lo solicite. Dicha notificación puede realizarse via correo electrónico, por ejemplo.

- Finalmente, la instancia de ETREK que se ejecuta en un nodo particular de un sistema distribuido es responsable de otras tareas relativas al funcionamiento interno del propio sistema:

  - El kernel tiene la misión de gestionar la base de datos que da soporte al servicio de persistencia del sistema: el almacén de información en el que se guarda la metainformación acerca de los datos, los modelos de conocimiento obtenidos a partir de ellos y toda la información que sea necesaria para que el sistema sea fiable y tolerante a fallos, como las sesiones de usuario abiertas, los agentes en ejecución, etc..
  - El kernel del sistema, además, debe proveer los mecanismos necesarios para poder añadir nuevos componentes de forma dinámica sin paralizar su funcionamiento (despliegue “en caliente” o *hot deployment*).
  - Por otro lado, el kernel del sistema local ha de coordinar su trabajo con otras instancias de ETREK que se estén ejecutando en otros nodos del sistema distribuido. En primer lugar, debe mantener información relativa a otros nodos para ser capaz de transmitir datos siguiendo la ruta óptima (capacidad de enrutamiento) y de comprobar la disponibilidad y descubrir, en su caso, la presencia de nuevos recursos en la red (capacidad de descubrimiento), como

pueden ser nuevos nodos que se añaden dinámicamente a la red o nodos ya existentes que ofertan nuevos servicios. Así mismo, una instancia concreta de ETREK también ha de tener constancia de la carga que soportan sus nodos vecinos en el sistema para poder tomar las decisiones adecuadas que permitan balancear la carga del sistema.

ETREK debe realizar todas las funciones descritas en los párrafos anteriores de la forma más transparente posible de cara al usuario del sistema, con el objetivo de facilitarle su uso en la medida que sea posible. Transparencia y usabilidad son las dos cualidades básicas que el sistema ha de tener, y ambas determinan muchas de las decisiones concretas de diseño que han de tomarse para implementar un sistema que posea las características expuestas en esta sección. A continuación, en el siguiente apartado de esta memoria, se analizan algunas de esas decisiones de diseño y se comentan algunas de las tecnologías existentes en la actualidad que permiten la implementación de un sistema como el propuesto.

6.4. Diseño e implementación

Los grandes sistemas de software se encuentran entre los sistemas más complejos creados nunca por el hombre.

FREDERICK P. BROOKS
The Mythical Man-Month

En este apartado se discutirán distintos aspectos particulares relativos al diseño arquitectónico de un sistema que cumpla con los requisitos expuestos en las secciones anteriores y algunas de las tecnologías disponibles en la actualidad que permiten la implementación real del sistema.

En primer lugar, se comentarán brevemente los principios de diseño a los que resulta aconsejable dar prioridad para poder llevar a cabo con éxito proyectos de desarrollo de software de la envergadura del sistema propuesto. Es esencial en proyectos de cierto tamaño que todos las personas involucradas

compartan una visión común que les permita dirigir todos sus esfuerzos en la misma dirección [70], independientemente de las técnicas de Ingeniería del Software que se empleen para planificar y gestionar el proyecto de desarrollo [109] [110].

En el apartado 6.4.2 se incluye un ejemplo de la aplicación de los principios comentados. En concreto, se ha escogido el diseño de una parte fundamental del sistema: el subsistema que permite acceder a datos de fuentes potencialmente heterogéneas. El uso de conocidos patrones de diseño permite modelizar de una forma elegante los conjuntos de datos a los que se ha de acceder y permite aislar al resto del sistema de las peculiaridades que puedan presentar las distintas fuentes de datos.

La sección siguiente, el apartado 6.4.3, se centra en otro de los componentes esenciales del sistema propuesto: el servicio de persistencia. La utilización de modelos de datos [77] flexibles permite obtener un diseño sencillo y eficaz de la base de datos que se utiliza para almacenar de forma persistente toda la información del sistema.

Una vez estudiado el diseño de los mecanismos que facilitan el acceso a los datos y del servicio que permite almacenar datos de forma persistente, en el apartado 6.4.4 se comentará brevemente una de las tecnologías posibles que se podrían utilizar para implementar el sistema: la plataforma Java y todos los estándares que la rodean. Esta alternativa resulta especialmente adecuada para un sistema distribuido que ha de funcionar en un entorno heterogéneo en el que coexisten distintos sistemas operativos.

Esta sección, en la que se delinea una estrategia para la implementación de un sistema distribuido basado en componentes, se cierra con el apartado 6.4.5, en el que se muestra la configuración del sistema que podría resultar de la aplicación de las directrices que analizamos a continuación.

6.4.1. Principios de diseño

El esfuerzo de diseño requerido para el desarrollo del sistema propuesto en este capítulo debe estar siempre orientado hacia la consecución de dos cualidades: transparencia (6.4.1.1) y usabilidad (6.4.1.2). El grado de transparencia y la facilidad de uso del sistema determinan, sin duda alguna, las posibilidades de éxito de un proyecto de este tipo. Las soluciones aplicadas en proyectos previos, modeladas mediante patrones de diseño (6.4.1.3), ofrecen una inestimable ayuda en el diseño de nuevos sistema como en el caso que nos ocupa.

6.4.1.1. Transparencia

El sistema diseñado ha de ser lo más transparente posible, tanto para los usuarios del sistema como para los programadores que le añaden funcionalidad implementando nuevos componentes.

De cara a los programadores, el sistema ha de ofrecer una serie de interfaces bien definidos que les permitan desarrollar nuevos componentes con relativa facilidad. Estas interfaces han de ocultar la complejidad interna del sistema, ofreciendo una especificación de la funcionalidad del sistema independiente de la representación que se haya escogido en su implementación. Ocultar detalles de implementación mediante su encapsulación es un objetivo esencial en el desarrollo de cualquier sistema de cierta envergadura, para lo cual ha de aplicarse una estrategia “divide y vencerás”. Esta estrategia nos permite resolver problemas complejos descomponiéndolos en otros más simples. La descomposición realizada determina la estructura del sistema y, gracias a nuestra capacidad de abstracción, se eliminan detalles que dificultarían el diseño del sistema. La eliminación de los detalles adecuados es, obviamente, la que conduce a diseños de mayor calidad. Este proceso, no automatizable, es uno de los mayores atractivos del desarrollo de software.

Por otro lado, respecto a los usuarios finales del sistema, los cuales no tienen por qué ser conscientes de la complejidad del sistema al que acceden, también es importante que el sistema oculte su complejidad interna. Sólo así se podrá lograr el cumplimiento de nuestro segundo objetivo: el relacionado con la usabilidad del sistema.

6.4.1.2. Usabilidad

La facilidad de uso del sistema, su usabilidad [89], es esencial si queremos que tenga éxito un sistema distribuido basado en componentes orientado a la resolución de problemas de extracción de conocimiento. Como cualquier otro sistema software, un sistema así es utilizado por personas, de modo que su facilidad de uso determina el grado de aceptación del que llegan a gozar. Este factor es más crítico aún si tenemos en cuenta que los 'trabajadores del conocimiento' (los analistas y ejecutivos a los que se orienta el sistema propuesto) no suelen ser expertos en Informática.

Entre las funciones del sistema se ha de incluir la capacidad de almacenar cualquier tipo de información para que los usuarios puedan volver a utilizarla en el futuro, algo de lo que se encargará el servicio de persistencia descrito en el apartado 6.4.3.

Dado que el objetivo final del sistema es la obtención de modelos descriptivos y predictivos, es esencial que el conocimiento extraíble de ellos se represente y comunique de la forma adecuada. Es imprescindible que el sistema facilite a los usuarios mecanismos mediante los que puedan compartir la información que hayan obtenido en sus sesiones de trabajo. Por tanto, sería aconsejable incluir herramientas que permitan la colaboración entre grupos de usuarios (conocidas por el término inglés *groupware*). La implementación de estas herramientas ha de ser especialmente cautelosa respecto a cuestiones de seguridad: se ha de garantizar la privacidad de la información que cada usuario obtiene de sus datos. Una política de seguridad adecuada en esta situación es hacer que, por defecto, se niegue siempre el acceso de un usuario a cualquier documento o modelo creado por otro usuario distinto, salvo que este último le haya concedido privilegios de lectura o modificación.

6.4.1.3. Patrones de diseño

Por suerte, el diseño de un sistema de las características expuestas a lo largo de este capítulo es factible hoy en día gracias a la experiencia acumulada a lo largo de los últimos años en el desarrollo de complejos sistemas software. Distintos autores se han encargado de recopilar esta experiencia en forma

de patrones de diseño [62] [65] [77], los cuales recogen soluciones que han demostrado ser de utilidad en el desarrollo de software.

El conocimiento de patrones de diseño complementa al enfoque educativo tradicional que se centra en el estudio de metodologías y técnicas de resolución de problemas para el análisis y diseño de software, como, por ejemplo, las descritas en las excelentes colecciones de ensayos de Bentley [15] o Plauger [126].

Uno de los patrones de diseño más conocidos es el modelo MVC (Modelo-Vista-Controlador) en el cual se separan físicamente los módulos o clases que modelizan información (modelos) de aquéllos que permiten acceder a ellos de distintas formas (vistas), los cuales se enlazan a sus respectivos modelos utilizando mecanismos auxiliares que también se implementan de forma independiente (controladores). La arquitectura resultante de la aplicación de este modelo es excelente porque produce una buena modularización del código (alta cohesión y acoplamiento débil), lo que facilita el mantenimiento del sistema.

Como se mencionará en el apartado 6.4.4, una implementación en Java del modelo MVC como Struts [44] puede facilitar la infraestructura necesaria para desarrollar la interfaz de usuario del sistema planteado en este capítulo. No obstante, antes de profundizar en posibles vías de implementación, analizaremos en los apartados siguientes el diseño de dos partes fundamentales del sistema que nos ocupa: las que proporcionan acceso a los datos (6.4.2) y el servicio de persistencia (6.4.3).

6.4.2. Modelización de conjuntos de datos

Consideremos, en primer lugar, el problema de acceder a los conjuntos de datos que sirven de entrada a los algoritmos ejecutados por los agentes del sistema para construir modelos, ya sean éstos predictivos o descriptivos.

Muchas herramientas de extracción de conocimiento, especialmente aquéllas que implementan técnicas de aprendizaje automático, trabajan con conjuntos de datos en forma de tablas (en el sentido del término empleado por las bases de datos relacionales), si bien es cierto que algunos sistemas existentes funcionan directamente sobre bases de datos multidimensionales [75]. Cada tabla contiene un conjunto de tuplas de longitud fija que se puede obte-

ner de una base de datos relacional utilizando interfaces estándares de acceso a bases de datos como ODBC o JDBC. Los datos también pueden provenir de otras fuentes, como pueden ser servidores DSTP [Data Space Transfer Protocol, National Center for Data Mining, University of Illinois at Chicago, 2000], documentos en formato XML [eXtensible Markup Language] o simples ficheros ASCII, por ejemplo.

Todos los conjuntos de datos tabulares incluyen un conjunto de columnas a las cuales se les puede denominar atributos o campos. A las distintas columnas de los conjuntos de datos se les suele asociar un identificador único para poder hacer referencia a ellas y un tipo que nos indica el dominio de sus valores (cadenas, números, fechas, etc.). Además, se debe permitir la definición de relaciones de orden entre los valores de los distintos atributos y se ha de facilitar la posibilidad de agrupar dichos valores para formar jerarquías de conceptos.

Por otro lado, hemos de tener en cuenta que los conjuntos de datos pueden provenir de fuentes de distinta naturaleza, a pesar de lo cual sigue siendo imprescindible interpretarlos de una forma uniforme que facilite su utilización y nos permita implementar algoritmos independientes del formato particular que tengan los datos. Esto es, el sistema ha de encargarse de gestionar de una forma transparente para los agentes los datos que provienen de fuentes heterogéneas.

El subsistema de acceso a los datos debería, por tanto, ser capaz de efectuar consultas heterogéneas que accedan a distintas bases de datos y fuentes de información. Los conjuntos de datos a los que se accede de forma independiente, aun proveniendo de fuentes heterogéneas, han de poder combinarse unos con otros con el objetivo de estandarizar la representación de conceptos (integración), eliminar redundancias y errores (limpieza), agregar información (resumen) o, simplemente, eliminar la parte de los datos que no sea de nuestro interés (filtrado).

Todas las operaciones mencionadas arriba podrían efectuarse utilizando modelos formales y lenguajes de consulta. No obstante, los usuarios típicos del sistema puede que no gocen de la preparación necesaria para dominar tales lenguajes de consulta y poder definir sin ayuda externa los conjuntos de datos que necesiten. Dichos usuarios, probablemente descartarían directamente el uso de un sistema que les exija conocer cualquier tipo de lenguaje de

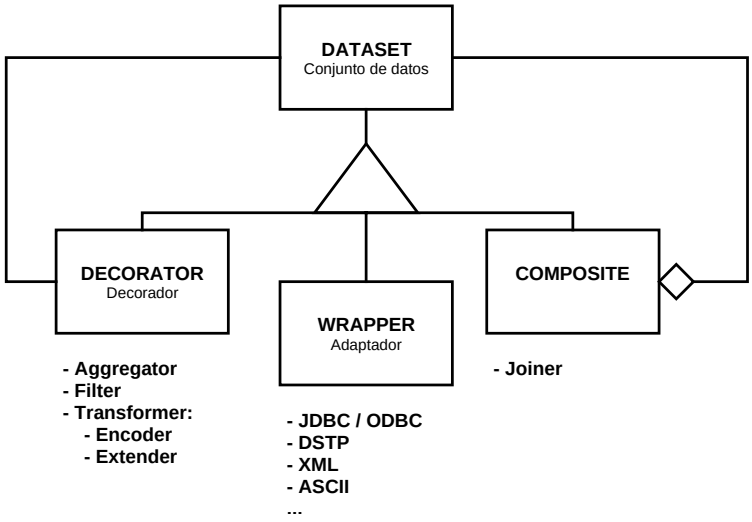

Figura 6.10: Diagrama de clases que ilustra el patrón de diseño utilizado en la modelización de los conjuntos de datos.

especificación de consultas.

Con el objetivo de facilitar la aceptación de un sistema en el que no se limite la capacidad de formular consultas complejas y, a la vez, resulte fácil de utilizar por usuarios con distinta preparación, en esta sección se propone utilizar algunos de los patrones de diseño estructurales más conocidos [65]. En concreto, se pueden emplear los patrones de diseño *wrapper* (adaptador), *decorator* (decorador) y *composite* (compuesto) para modelar cualquier conjunto de datos tal como se muestra en el diagrama de clases de la figura 6.10.

En vez de tener que emplear un lenguaje de consulta más o menos complejo, el usuario sólo tendrá que ir construyendo de forma ascendente los conjuntos de datos que desee utilizar. Para ello, el usuario empleará una familia de componentes de construcción de conjuntos de datos que le proporcionarán los mecanismos necesarios para construir sus propios conjuntos de datos personalizados a partir de las fuentes de datos disponibles (p.ej., bases de datos o ficheros en un formato determinado).

Los componentes que le permiten al usuario construir sus propios conjun-

tos de datos se describen a continuación[**]:

- Los adaptadores (***wrappers***) son responsables de facilitar un interfaz de acceso uniforme a fuentes de datos de distintos tipos. Cuando los datos se almacenan en bases de datos relacionales, los conjuntos de datos pueden expresarse como el resultado de realizar consultas SQL a través de interfaces estándares como ODBC, JDBC o BDE. Tales interfaces son independientes de la base de datos particular y permiten acceder a la mayor parte de los sistemas gestores de bases de datos existentes en la actualidad (Oracle, IBM DB2, Microsoft SQL Server, InterBase...). Cuando los datos se almacenan en otros formatos (localmente en ficheros ASCII o XML, o de forma remota en servidores DSTP, por ejemplo), se requieren adaptadores específicos para cada formato. Como es lógico, cada tipo de fuente de información necesita su propio adaptador específico.

- Los integradores (***joiners***) se emplean para reunir múltiples conjuntos de datos independientes, interpretando esta reunión en el sentido del álgebra relacional. Estos componentes, provenientes del uso del patrón de diseño *composite* (compuesto), permiten combinar la información que proviene de diferentes fuentes. Con ellos se puede añadir información a los registros de un conjunto de datos particular (como cuando su utiliza un *data warehouse* con un esquema en estrella) y también permiten establecer relaciones maestro/detalle entre dos conjuntos de datos.

- Los agregadores (***aggregators***) nos sirven para resumir los conjuntos de datos disponibles para poder obtener una visión más general de ellos. Las agregaciones son especialmente útiles en muchas aplicaciones de análisis OLAP, en las cuales las tendencias generales presentes en los datos son mucho más interesante que los detalles particulares. Las funciones de agregación más comunes incluyen contar el número de datos existentes (COUNT), sumarlos (SUM), calcular su media aritmética

[**]En este apartado, como sucede en otras partes de este capítulo, aparecen determinados anglicismos por ser de uso habitual y se mantienen en el texto para no desorientar al lector que esté familiarizado con ellos al utilizar traducciones que puedan llevar a confusión.

(AVG) y obtener la varianza (VAR), así como recuperar el valor mínimo (MIN), el valor máximo (MAX), los valores más altos (TOP) y los valores más bajos (BOTTOM).

- Los filtros (***filters***) se emplean para seleccionar parte de un conjunto de datos y quedarnos con un subconjunto del conjunto de datos original. Desde el punto de vista del álgebra relacional, los filtros nos permiten realizar proyecciones (escoger columnas determinadas de un conjunto de datos) y selecciones (quedarnos con algunas de las tuplas del conjunto de datos). En aplicaciones de *Data Mining*, los filtros pueden usarse para muestrear datos, formar conjuntos de entrenamiento y prueba en un proceso de validación cruzada o, simplemente, para elegir una parte de los datos cuyo procesamiento posterior pueda resultar de interés para el usuario.

- Los transformadores (***transformers***) también son necesarios para que el usuario pueda modificar las columnas de un conjunto de datos. Dentro de ellos, se pueden identificar dos tipos principales:

  - Los codificadores (***encoders***) se emplean para codificar conjuntos de datos. Por ejemplo, se pueden usar para establecer un formato de codificación uniforme en la representación de información que, aun teniendo un significado único, se puede presentar de distintas formas según cuál sea su procedencia. De hecho, es frecuente que una entidad determinada pueda tener distintas representaciones en un mismo conjunto de datos. Por tanto, los codificadores resultan esenciales para realizar tareas de limpieza e integración de datos.
  - Los extensores (***extenders***), por su parte, permiten añadir nuevas columnas a un conjunto de datos. Esos atributos adicionales, a los que se les suele denominar campos calculados, son útiles para convertir unidades de medida y, sobre todo, para gestionar fechas (p.ej. se puede obtener el día de la semana, la semana del año, el mes, el trimestre o la temporada correspondiente a una fecha concreta). El valor de un campo calculado ha de venir completamente

determinado por los valores de los demás campos de un registro o tupla (en caso contrario, tendríamos que utilizar un agregador o un integrador en vez de un extensor). Normalmente, el cómputo de un campo calculado se especifica utilizando una expresión aritmética (con operadores como +, -, * o /), si bien también se suelen permitir funciones predefinidas y expresiones más complejas que involucren sentencias condicionales (del tipo *if-then-else*, por ejemplo).

La familia descrita de componentes permite que el usuario construya sus propios conjuntos de datos agregando unos componentes con otros. Al combinar los distintos tipos de componentes, se puede construir una estructura en forma de árbol análoga a la que construiría el planificador de consultas de un sistema de bases de datos. Dada esta analogía, la estructura en árbol creada por el usuario es apta para la aplicación de las técnicas estándar de optimización de consultas [118], con lo que se puede mejorar notablemente el rendimiento del sistema a la hora de acceder a los datos.

La consecuencia más importante de esta forma de acceder a los datos, que ofrece la misma flexibilidad que cualquier lenguaje de consulta por complejo que éste sea, es que el usuario puede enlazar fácilmente los componentes descritos para modelar cualquier conjunto de datos que desee utilizar.

6.4.3. Servicio de persistencia

El servicio de persistencia ha de encargarse de almacenar de una forma fiable toda la información utilizada por el sistema, tanto la metainformación relativa a los conjuntos de datos como los modelos obtenidos por los distintos usuarios y el estado actual del sistema. Para ello puede emplear una base de datos de apoyo en la que se almacenen datos sobre los usuarios y grupos de usuarios del sistema, los permisos de acceso de los que goza cada uno de ellos, los modelos y los conjuntos de datos utilizados por cada usuario, las sesiones activas en el sistema, las tareas pendientes (esto es, los agentes que se encuentran en ejecución) y cualquier otra información referente a la configuración del sistema.

Siguiendo la filosofía de los patrones de diseño, en vez de diseñar una base de datos de la forma tradicional (creando una tabla para cada tipo de entidad que pueda existir en el sistema), se puede diseñar una base de datos que permita la evolución de los componentes existentes en el sistema y facilite su mantenimiento. En este tipo de diseños se suele diferenciar un nivel operacional, que incluye los datos propiamente dichos, de un nivel de conocimiento en el que se guarda metainformación acerca de los datos almacenados en la base de datos. De esta forma, si el esquema lógico de la base de datos se ve modificado, sólo es necesario modificar el contenido de las tablas del nivel de conocimiento, sin tener que alterar el esquema físico de la base de datos. En cierta medida, este tipo de diseños utiliza las mismas ideas en que se basa la implementación del catálogo en los sistemas modernos de gestión de bases de datos.

El diagrama entidad/relación de la figura 6.11 muestra un diseño flexible que no requiere modificación alguna sean cuales sean los cambios que se produzcan en el sistema.

En esta base de datos es imprescindible almacenar información acerca de los usuarios por motivos de seguridad. Una clave de acceso, que siempre ha de almacenarse encriptada utilizando algoritmos como MD5 o SHA, es la que permite al usuario autentificarse y acceder a los recursos del sistema. También resulta aconsejable mantener información acerca del uso que el usuario hace del sistema, para poder monitorizar su funcionamiento y ser capaz de detectar situaciones anómalas que puedan provenir de accesos no autorizados.

En el diseño propuesto, el usuario creará sesiones de trabajo en las cuales podrá almacenar todo tipo de información, como veremos más adelante. Las sesiones creadas por él serán de su propiedad y solamente él tendrá la capacidad de dar acceso a la información de la sesión a otros usuarios del sistema.

Los usuarios del sistema se agrupan en comunidades o grupos de usuarios que pueden compartir sesiones, de forma que el usuario propietario de una sesión puede permitir el acceso a esa sesión únicamente a las grupos de usuarios que él decida. A través del acceso compartido a sesiones, el usuario propietario de una sesión concede, de forma indirecta, permisos de acceso a toda la información relativa a su sesión. De esta forma, los conjuntos de datos y modelos obtenidos a partir de ellos pueden compartirse entre grupos de usuarios.

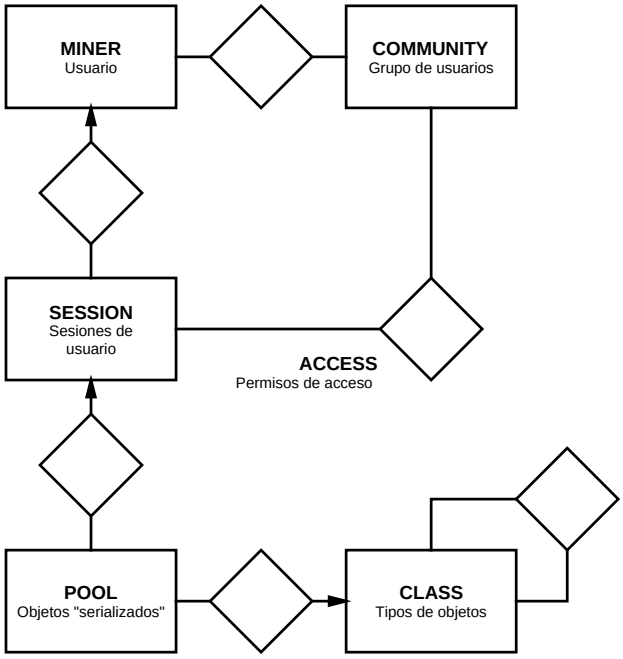

Figura 6.11: Diagrama entidad/relación de la base de datos que da soporte al servicio de persistencia y permite definir permisos de acceso para grupos de usuarios del sistema.

La información almacenada referente a cada sesión de usuario se guarda dividida en dos partes: un almacén de objetos "serializados" (nivel operacional) y una jerarquía de tipos asociados a dichos objetos que nos permite navegar por el contenido de la base de datos de cada sesión (nivel de conocimiento). Al almacenar información relativa a los tipos de los objetos presentes en la base de datos, este sencillo diseño proporciona capacidades de introspección al sistema. Además, el sistema resulta extremadamente flexible, pues no es necesario modificar nunca el esquema de la base de datos aunque cambien los tipos de componentes existentes en el sistema.

En esta base de datos se almacena toda la información de la que dispone el usuario acerca de los conjuntos de datos que emplea, así como los distintos modelos que haya ido obteniendo con anterioridad, de forma que puede reutilizarlos en el futuro si así lo desea. Por ejemplo, el usuario podría almacenar modelos de clasificación para ser capaz de clasificar nuevos datos en el futuro o podría utilizar los resultados producidos por un método de agrupamiento para seleccionar el fragmento de los datos correspondiente a uno de los agrupamientos y analizarlo utilizando cualquiera de las técnicas de las que disponga en el sistema.

Aparte de la información con la que conscientemente trabaja el usuario, la base de datos que da apoyo al servicio de persistencia también ha de mantener las sesiones activas y las tareas pendientes en cada momento con el objetivo de que el sistema sea tolerante a fallos. De esta forma, se puede reducir el impacto que podría tener en el sistema un corte de suministro eléctrico, por ejemplo.

6.4.4. Implementación del sistema en Java

En las secciones anteriores se han analizado algunas cuestiones relativas al diseño de la infraestructura necesaria para que el sistema funcione. En esta sección se comentará brevemente un conjunto de tecnologías existentes que permite la implementación de un sistema como el descrito en este capítulo.

La plataforma Java, que incluye el lenguaje de programación Java, su amplia gama de bibliotecas estándar y un entorno de ejecución portable (la máquina virtual Java), resulta especialmente adecuada para desarrollar aplicaciones que hayan de funcionar en sistemas heterogéneos porque existen máquinas

virtuales Java para casi todos los sistemas operativos existentes en el mercado (como Windows, Linux y las distintas variantes de UNIX). Este hecho permite que el código escrito en un entorno de desarrollo particular pueda funcionar en multitud de sistemas operativos sin requerir modificaciones de ningún tipo. Dado que el sistema distribuido propuesto en este capítulo debería funcionar en sistemas heterogéneos, Java parece ser una buena opción a la hora de implementarlo.

Java resulta especialmente adecuado para el desarrollo de aplicaciones distribuidas en Internet o en intranets por el amplio apoyo que su gama de bibliotecas estándar ofrece al programador de aplicaciones, lo que permite utilizar distintos modelos de cómputo distribuido y paralelo [121] [155]. Mediante el uso de RMI [*Remote Method Invocation*], Java proporciona la posibilidad de realizar llamadas a métodos remotos y construir sistemas distribuidos. La propia arquitectura de Java, con su cargador de clases dinámico, su capacidad de introspección y su modelo de componentes (JavaBeans), facilita el desarrollo de sistemas dinámicos basados en componentes. Además, tecnologías relacionadas como Jini [33] [149] permiten crear con facilidad la infraestructura necesaria para la implementación de un sistema como el propuesto en este capítulo. En la plataforma Java se puede encontrar la misma funcionalidad que ofrecen los servicios web citados como estándares actuales en el desarrollo de sistemas distribuidos (sección 6.2). A grandes rasgos, RMI, JavaBeans y Jini se corresponden con SOAP, WSDL y UDDI, respectivamente.

Gracias a su independencia de la plataforma sobre la que se ejecuta, un sofisticado modelo de seguridad, RMI y la capacidad de "serializar" objetos (esto es, la capacidad de empaquetar los objetos para su almacenamiento o transmisión y posterior restauración en la misma o en otra máquina virtual), el lenguaje Java ha sido escogido como estándar de facto para el desarrollo de sistemas multiagente [25]. Aglets (de IBM), Grasshopper (IKV++), MOLE (Universidad de Stuttgart), Paradigma (Universidad de Southampton), RONIN (Universidad de Maryland), Voyager (ObjectSpace), Concordia (Mitsubishi Electric ITA Horizon Labs), Odyssey (General Magic) o Jumping Beans (Ad Astra) son sólo algunos de los sistemas multiagente que ya se han implementado utilizando Java.

Respecto a la ejecución de los agentes en un sistema como el analizado en este capítulo, hay que tener en cuenta varios aspectos si finalmente se decide utilizar Java:

- **Movilidad:** La movilidad de los agentes desarrollados en Java está limitada, porque no se puede almacenar el estado de un agente en tiempo de ejecución para que después restaure su estado y prosiga su ejecución por donde fuese. En Java se puede serializar el estado de un objeto, pero no su entorno de ejecución (la pila de la hebra correspondiente al agente). No es, pues, posible la implementación de un mecanismo de *persistencia ortogonal* en Java (aquél que permitiese mover un agente de un nodo a otro del sistema distribuido y reanudar su ejecución de forma transparente). Es más, la migración de un agente de un nodo a otro no es transparente a no ser que se utilicen versiones especializadas de la máquina virtual Java (p.ej. Aroma).

- **Seguridad interna (control de acceso):** Java, con su modelo de seguridad basado en permisos, sí resulta especialmente adecuado para construir sistemas distribuidos reconfigurables, con políticas de seguridad flexibles, fácilmente adaptables y extensibles que permitan controlar localmente la ejecución del software.

- **Seguridad externa (transmisión de datos):** Las bibliotecas estándar de Java incluyen implementaciones de todo tipo de técnicas criptográficas de protección de datos que permiten un intercambio seguro de información confidencial a través de un medio compartido con otros usuarios y sistemas. En el caso de Internet, la seguridad se suele conseguir empleando SSL [*Secure Sockets Layer*], que utiliza técnicas criptográficas de clave pública.

- **Monitorización del uso de recursos: tiempo de CPU, memoria y ancho de banda.** Es quizá uno de los puntos más débiles de la plataforma Java, que ha dado lugar a la propuesta de extensiones como JRes [42] [43]. El recolector de basura de Java, por ejemplo, ofrece poco control sobre el uso real de memoria.

Característica	Tecnologías y técnicas
Portabilidad	Java
	HTML dinámico estándar
Interfaz	Struts [44]
	Servlets & Pushlets
	JavaMail
Interoperabilidad	JDBC [Java DataBase Connectivity]
	Familia de protocolos TCP/IP: HTTP, SMTP...
Concurrencia	Hebras
Distribución	RMI [Remote Method Invocation]
	Jini
	Arquitectura multicapa (proxies)
Movilidad	Serialización
Seguridad	Claves encriptadas (SHA o MD5)
	HTTPS (SSL [Secure Socket Layer])
Persistencia	Base de datos de apoyo (figura 6.11)
Soporte multilingüe	Ficheros de recursos

Tabla 6.1: Algunas de las características deseables del sistema y las técnicas que facilitan su implementación.

- **Flexibilidad:** El cargador de clases dinámico de la máquina virtual Java y la capacidad de introspección de los objetos son, sin duda, las mayores bazas de Java frente a otros lenguajes monolíticos como C o C++.

En [142] se puede leer un interesante artículo que describe algunas de las mejores y de las peores características de Java con las que uno se topa cuando se quiere implementar un sistema de procesamiento de consultas cuyos requerimientos son similares en cierta medida a los del sistema propuesto en este capítulo. La tabla 6.1 resume algunas de las características deseables del sistema, así como las tecnologías y técnicas que permiten que la implementación del sistema sea factible.

6.4.5. Despliegue del sistema

Dado que un sistema como el descrito en este capítulo no puede implementarse utilizando servidores de aplicaciones tradicionales, orientados al desarrollo de aplicaciones OLTP, ha sido necesario diseñar un sistema distribuido de propósito especial, que no de utilidad limitada. El principal rasgo diferencial del sistema propuesto es su capacidad para controlar mejor los recursos computacionales de los se dispone en un entorno distribuido.

La figura 6.12 muestra la configuración que un sistema así podría tener si se siguen las directrices expuestas en los apartados anteriores. En el sistema de la figura, un servidor web Apache equipado con un contenedor de servlets sirve de interfaz del sistema con el usuario a través de web, mientras que la base de datos empleada por el servicio de persistencia puede residir en cualquier sistema gestor de bases de datos relacionales.

6.5. Una mirada hacia el futuro

La predicción es difícil, especialmente cuando se trata del futuro
NIELS BÖHR

En este capítulo se han abordado distintas cuestiones que han de resolverse para poder implementar un sistema distribuido basado en componentes que resulte adecuado para tareas de extracción de conocimiento en bases de datos y, en general, para cualquier aplicación de cálculo intensivo.

Los servidores de aplicaciones comerciales, máximos exponentes hoy en día de los sistemas distribuidos basados en componentes, se restringen al campo de las aplicaciones OLTP, las cuales realizan un procesamiento de datos a relativamente bajo nivel. Este cápítulo, utilizando el mismo modelo arquitectónico que rige a estos sistemas, propone la implementación de sistemas más capaces, sistemas capaces de realizar tareas computacionalmente más complejas que tradicionalmente se han asociado a otras arquitecturas de propósito específico y a sistemas OLAP en los que se implementan aplicaciones de ayuda a la decisión.

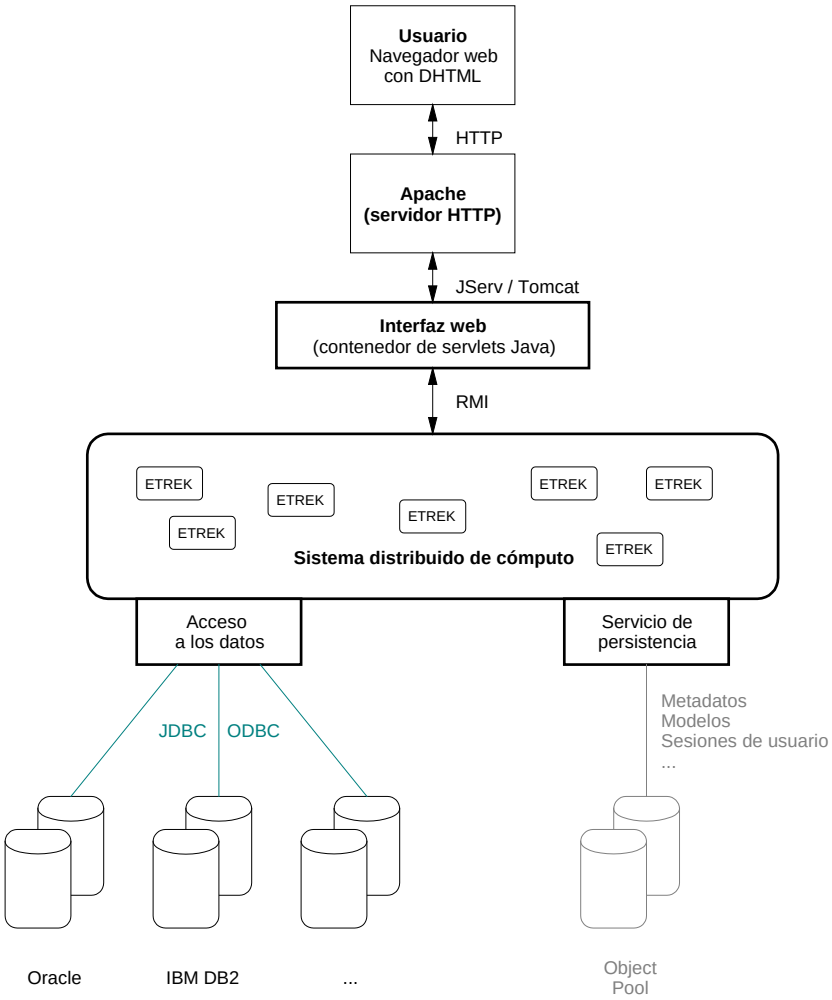

Figura 6.12: Diagrama de despliegue que muestra una posible configuración del sistema.

El modelo de cómputo utilizado en la actualidad infrautiliza los recursos de los que disponemos conectados a redes locales, intranets corporativas o la red de redes, Internet, por lo que no resulta del todo descabellado centrar nuestros esfuerzos en el desarrollo de nuevos sistemas que aprovechen al máximo la capacidad de cálculo de los ordenadores personales autónomos que se hallan interconectados por modernas redes de comunicación y transmisión de datos. El modelo expuesto en este capítulo puede ser de utilidad como base para construir un supercomputador virtual, flexible y potente. Flexible porque se utiliza un modelo basado en componentes y potente porque puede aprovechar los recursos no utilizados de los dispositivos conectados a los sistema distribuidos (tiempo de CPU y espacio de almacenamiento, principalmente).

Quién sabe si algún día sistemas como el propuesto en este capítulo harán de la capacidad de cómputo y almacenamiento de datos un servicio público como puede ser el suministro eléctrico en el mundo desarrollado actual. Solamente el tiempo decidirá si algo así puede llegar a suceder.

Capítulo 7

Conclusiones

Somos la suma de nuestras decisiones

WOODY ALLEN
Delitos y Faltas (1989)

En este trabajo se ha presentado una nueva estrategia para construir árboles de decisión que ha conseguido resultados prometedores experimentalmente: el modelo de clasificación ART. En realidad, este modelo de clasificación, que se presenta en el capítulo 3 de la presente memoria, se puede considerar un híbrido entre los algoritmos de construcción de árboles de decisión y los algoritmos de inducción de listas de decisión, técnicas de aprendizaje supervisado que se describen en las secciones 2.1 y 2.2 respectivamente.

Como algoritmo de construcción de árboles de decisión, ART se caracteriza por construir árboles de decisión n-arios y ser capaz de utilizar simultáneamente varios atributos para ramificar el árbol de decisión, lo que le permite construir modelos de clasificación más compactos.

En cuanto a la interpretación del método ART como algoritmo de inducción de listas de decisión, cabe destacar que ART mejora la eficiencia de propuestas anteriores al extraer en paralelo conjuntos de reglas que sirven para ramificar cada nivel del árbol de decisión, mientras que las técnicas habituales de inducción de listas de decisión se limitan a ir descubriendo de una en una

las reglas que forman parte del modelo de clasificación construido por ellas.

En definitiva, como árbol de decisión, ART ofrece un modelo más flexible que CART o C4.5 al construir árboles n-arios con ramas 'else', mientras que como algoritmo de inducción de listas de decisión obtiene un modelo de menor profundidad, con lo que resulta más fácil de interpretar, y además lo hace de una forma más eficiente gracias a las técnicas empleadas en su implementación.

La implementación de ART se basa en la utilización de una técnica eficiente de extracción de reglas de asociación. Esta técnica, habitual en aplicaciones de *Data Mining*, es la que da nombre al modelo de clasificación propuesto en esta memoria, ya que ART es acrónimo de *Association Rule Tree*.

Gracias al uso de eficientes algoritmos de extracción de reglas de asociación, la complejidad del proceso de construcción del clasificador ART es comparable a la de los populares algoritmos TDIDT de construcción de árboles de decisión y puede llegar a ser varios órdenes de magnitud más eficiente que algoritmos de inducción de reglas como CN2 o RIPPER, especialmente cuando aumenta el tamaño de los conjuntos de datos (véanse, por ejemplo, las figuras de la página 103).

Al estar basado en algoritmos de extracción de reglas de asociación, ART no sólo es eficiente, sino que también es escalable. Ésta es una característica esencial en la resolución de problemas de *Data Mining*, pues permite la utilización del método propuesto en esta memoria para extraer información de enormes conjuntos de datos.

El algoritmo de extracción de reglas de asociación utilizado por ART también ofrece un mecanismo simple y efectivo para tratar una amplia variedad de situaciones sin necesidad de recurrir a otras técnicas más específicas, complejas y artificiales. En concreto, el proceso de extracción de reglas de asociación es responsable de que ART se comporte correctamente ante la presencia de información con ruido o la existencia de claves primarias en el conjunto de entrenamiento, ya que ART se basa en la extracción de itemsets frecuentes para construir el modelo de clasificación.

Por otro lado, la topología particular del árbol construido por ART facilita tratar de una forma elegante la presencia de valores desconocidos para los

atributos involucrados en el test de un nodo interno del árbol: cuando se le presenta un caso de prueba con valores desconocidos y no es aplicable el test que dio lugar a la ramificación del árbol, ART simplemente envía el caso de prueba a la rama 'else' del nodo.

Debido en parte a la topología del árbol, ART es capaz de construir clasificadores que destacan por su simplicidad e inteligibilidad, además de por su robustez ante la presencia de ruido. ART se caracteriza por construir modelos de clasificación de menor complejidad que los obtenidos mediante la utilización de algoritmos TDIDT y de complejidad similar a las listas de decisión que se obtienen empleando algoritmos más ineficientes. Es más, la simplicidad de los modelos de clasificación ART se consigue sin sacrificar la precisión del clasificador, lo que hace de ART una alternativa interesante en situaciones en las que el tamaño de los árboles de decisión TDIDT los hace prácticamente inmanejables.

Como caso particular, ART logra resultados interesantes cuando se utiliza para clasificar uniones de genes en secuencias de ADN, tal como se muestra en la sección 3.3. En este problema, ART descubre y aprovecha las simetrías existentes en las secuencias de nucleótidos alrededor de una unión para construir un modelo de clasificación mucho más sencillo que el que se obtiene utilizando otros métodos de construcción de árboles de decisión.

En cuanto a la utilización de ART por parte de usuarios no expertos, es destacable el hecho de que ART requiere un conjunto limitado de parámetros que usualmente no hay que ajustar. Básicamente, los parámetros empleados por ART son los habituales en cualquier proceso de extracción de reglas de asociación y sirven para acotar el espacio de búsqueda explorado al construir el clasificador.

Uno de los parámetros habituales en el proceso de extracción de reglas de asociación (y, por tanto, en ART) es el umbral mínimo de confianza que se le exige a las reglas de asociación para ser consideradas como hipótesis candidatas en la construcción del árbol de decisión. Este parámetro puede utilizarse, por ejemplo, para establecer una restricción a priori sobre la precisión de las reglas seleccionadas para formar parte del clasificador. Esta posibilidad es muy interesante en la resolución de algunos problemas y necesaria cuando no

se permiten errores, como en el caso de la identificación de setas comestibles comentado en la sección 3.1.3 de esta memoria.

En la misma sección del presente trabajo, también se ha expuesto el uso de técnicas heurísticas que permiten seleccionar automáticamente el mejor conjunto de reglas descubiertas, evitando de este modo que el usuario tenga que establecer manualmente los parámetros empleados en la construcción del clasificador ART. En particular, se propone la utilización de un margen de tolerancia en la selección de reglas candidatas a formar parte del árbol ART. Esta heurística concreta consigue resultados excepcionales sin que el usuario tenga que estimar los valores adecuados para los parámetros utilizados en la construcción de clasificadores ART.

En la sección 4.4 de esta memoria, se presentan alternativas al uso de la confianza como criterio de estimación de la calidad de las reglas obtenidas. Si bien las medidas analizadas no consiguen resultados netamente superiores a los obtenidos al utilizar la confianza, medidas como los factores de certeza o el interés de las reglas pueden resultar útiles en situaciones particulares dependiendo de la semántica del problema que se desee resolver.

De hecho, del estudio de medidas alternativas para evaluar las reglas extraídas surgió la idea de imponer restricciones adicionales a las reglas empleadas para construir el clasificador ART. En el apartado 4.4.3.12 se describe un criterio que permite mejorar el porcentaje de clasificación obtenido por ART a cambio de un pequeño aumento en la complejidad del clasificador construido.

En lo que respecta al uso en la práctica del modelo de clasificación ART, también hay que mencionar que se ha comprobado experimentalmente su buen funcionamiento cuando se emplean técnicas de discretización. Estas técnicas permiten construir clasificadores ART con atributos continuos, algo esencial si deseamos aplicar el modelo de clasificación ART en la resolución de problemas reales. Al realizar los experimentos que aparecen recogidos en la sección 5.4.2, se llegó a la conclusión de que ART funciona mejor cuando se emplean técnicas de discretización de forma global (antes de comenzar la construcción del clasificador), pues el uso de técnicas de discretización local (en cada nodo del árbol) se traduce generalmente en clasificadores de construcción más costosa computacionalmente, más complejos en cuanto a su tamaño y algo menos

precisos que los obtenidos realizando una discretización global de los atributos continuos del conjunto de entrenamiento.

Aparte del modelo de clasificación ART, cuyas mejores cualidades se han comentado en los párrafos anteriores, durante el desarrollo del trabajo descrito en esta memoria se han obtenido resultados adicionales que han dado lugar a una serie de 'subproductos' con entidad propia, cuya utilidad va más allá de su uso en ART. Algunos de los más relevantes se describen a continuación.

El algoritmo TBAR de extracción de reglas de asociación

El algoritmo TBAR [19], presentado en la sección 4.2 de esta memoria, es un algoritmo eficiente de extracción de reglas de asociación que resulta especialmente adecuado para su utilización en conjuntos de datos densos, como los que se pueden encontrar en bases de datos relacionales y se emplean habitualmente para resolver problemas de clasificación. El algoritmo TBAR, como técnica general de extracción de reglas de asociación, mejora el rendimiento de otros algoritmos ampliamente utilizados, como es el caso de Apriori [7].

El discretizador contextual

En la sección 5.2 se presentó este método de discretización, jerárquico y supervisado si nos atenemos a las categorías descritas en el capítulo 5. Este método, que obtiene buenos resultados cuando se utiliza para construir clasificadores TDIDT y ART, utiliza la estrategia tradicional de los métodos de agrupamiento clásicos para discretizar los valores de un atributo continuo. El discretizador contextual emplea medidas de similitud entre intervalos adyacentes (cualquiera de las que aparecen en el anexo 5.5) en vez de utilizar medidas de pureza como suelen hacer los demás métodos existentes de discretización supervisada.

Reglas de división alternativas para la construcción de árboles de decisión con algoritmos TDIDT

Las medidas de pureza son las que se suelen utilizar como reglas de división para decidir cómo se ramifica un árbol de decisión, tal como se puede leer

en el apartado 2.1.1. En dicho apartado se mencionaron dos medidas, MAX-DIF y el Índice Generalizado de Gini, que son de formulación más sencilla que los demás criterios existente y obtienen resultados comparables con los conseguidos utilizando reglas de división más complejas [18].

Árboles n-arios arbitrarios utilizando técnicas discretización jerárquica en algoritmos TDIDT

También se ha descrito en esta memoria el empleo de técnicas de discretización jerárquica para construir árboles n-arios arbitrarios con atributos numéricos, árboles en los cuales no se restringe el factor de ramificación del árbol final (sección 5.3.2).

Una arquitectura distribuida de cómputo basada en componentes

Finalmente, en el capítulo 6 se plantea una arquitectura adecuada para la resolución de problemas de cómputo intensivo, como puede ser la construcción de clasificadores ART en aplicaciones de *Data Mining*.

La arquitectura propuesta, distribuida al estilo de los sistemas P2P actuales, está basada en componentes para facilitar el desarrollo de aplicaciones que hagan uso de la infraestructura que ofrece.

Además, esta arquitectura general incluye dos subsistemas cuyo ámbito de aplicación va más allá de los límites del sistema planteado:

- El modelo propuesto para los conjuntos de datos con los que trabaja el sistema (sección 6.4.2) se puede utilizar en cualquier sistema que tenga que manipular conjuntos de datos o acceder a distintas fuentes de datos de una manera uniforme aunque éstas sean heterogéneas.

- El almacén de información propuesto al describir el servicio de persistencia del sistema basado en componentes (sección 6.4.3) también puede ser útil en un amplio rango de aplicaciones, al poder almacenar objetos de cualquier tipo sin tener que modificar la estructura de la base de datos subyacente (funcionando de una forma similar al catálogo de una base de datos relacional).

Trabajo futuro

Una vez que se han descrito los resultados más relevantes que se han obtenido durante la realización de este trabajo, se sugiere una serie de puntos de partida para líneas de investigación futuras:

- Resulta de especial interés estudiar la posibilidad de construir un modelo de clasificación híbrido TDIDT-ART que seleccione la estrategia de ART o de los algoritmos TDIDT en función de la situación que se le presente en cada nodo durante la construcción de un árbol de decisión.

- También es interesante la incorporación de técnicas difusas en el modelo de clasificación ART, para lo cual se necesitan algoritmos de extracción de reglas de asociación difusas.

- Igualmente, puede resultar de provecho investigar formas alternativas de manejar atributos continuos, como puede ser la utilización de técnicas de extracción de reglas de asociación cuantitativas [146] [114] [1] [12].

- Del mismo modo, la extensión de ART para tratar problemas de regresión (esto es, cuando los atributos continuos aparecen en el consecuente de las reglas) es otra línea de trabajo futuro.

- El proceso de extracción de reglas candidatas es otra faceta de ART que merece un estudio adicional para encontrar técnicas que obtengan mejores reglas de una forma más eficiente.

- La introducción de pesos en las reglas puede ser otra estrategia que permita obtener clasificadores ART más precisos.

- El estudio de medidas alternativas de evaluación de las reglas obtenidas en siempre una línea de trabajo abierta que puede conducir a la consecución de buenos resultados.

- También puede ser interesante el análisis de criterios de preferencia alternativos que nos permitan seleccionar las mejores reglas del conjunto de reglas disponible para construir cada nivel del árbol ART.

- Un estudio más en profundidad acerca de la posible utilización de ART como técnica de aprendizaje incremental también es deseable, ya que la existencia de métodos eficientes que permitan actualizar un clasificador dado un conjunto de actualizaciones de una base de datos es muy importante en aplicaciones de *Data Mining*.

- De forma complementaria, puede ser beneficioso el empleo de técnicas de post-procesamiento que permitan mejorar el rendimiento del clasificador ART una vez construido.

No leas para contradecir o refutar, ni para creer o dar por bueno, ni para buscar materia de conversación o de discurso, sino para considerar y ponderar lo que lees.

FRANCIS BACON

Los expertos dicen que dos cosas determinan dónde estará usted dentro de cinco años a partir de ahora: los libros que lee y las personas con las que se asocia.

Bibliografía

[1] Aggarwal, C. C., Sun, Z. & Yu, P. S. (1998). *Online algorithms for finding profile association rules*. Proceedings of the 1998 ACM CIKM 7th International Conference on Information and Knowledge Management. Bethesda, Maryland, USA, November 3-7, 1998, pp. 86-95

Artículo dedicado a la extracción de reglas de asociación con atributos numéricos en el que se propone la utilización de un árbol S para mantener un índice multidimensional.

[2] Aggarwal, C. C. & Yu, P. S. (1998a). *A New Framework for Itemset Generation*. Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, Washington, June 1-3, 1998, pp. 18-24.

En este artículo se realiza una crítica del modelo clásico empleado en la obtención de reglas de asociación y se propone sustituir los itemsets frecuentes por itemsets "fuertemente colectivos" para conseguir un proceso de obtención de reglas más eficiente y productivo. El modelo empleado trata de eliminar la generación de reglas espúreas (aquéllas que no aportan nada nuevo) y evitar la no obtención de reglas potencialmente interesantes debida al valor de *MinSupport*, el cual, en ocasiones, ha de fijarse demasiado alto para evitar una explosión combinatoria en la generación de reglas.

[3] Aggarwal, C. C. & Yu, P. S. (1998b). *Mining Large Itemsets for Association Rules*. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998.

Repaso general sobre los métodos de obtención de reglas de asociación (itemsets, para ser precisos) en el que se propone el uso de la "fuerza colectiva" *[collective strength]* para caracterizar los itemsets involucrados en las reglas de asociación.

[4] Agrawal, R., Imielinski, T. & Swami, A. (1993). *Mining association rules between sets of items in large databases*. Proceedings of the 1993 ACM

SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993, pp. 207-216

Primer artículo donde se presenta el concepto de regla de asociación y se propone el algoritmo AIS.

[5] Agrawal, R. & Shafer, J.C. (1996). *Parallel Mining of Association Rules*. IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 962-969, December 1996.

Trabajo en el que se expone cómo implementar el algoritmo Apriori en un multiprocesador sin memoria compartida realizando la comunicación mediante paso de mensajes (con MPI), lo que es aplicable también a un cluster de estaciones de trabajo conectadas a través de una red de interconexión de alta capacidad. Se proponen tres alternativas: *Count Distribution*, *Data Distribution* y *Candidate Distribution*.

[6] Agrawal, R. & Shim, K. (1996). *Developing Tightly-Coupled Applications on IBM DB2/CS Relational Database System: Methodology and Experience*, Second International Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon, August 1996, pp. 287-290.

En esta ponencia, resumen de un informe técnico de IBM, se presenta la implementación del algoritmo Apriori de extracción de reglas de asociación utilizando capacidades propias del sistema gestor de bases de datos relacionales de IBM.

[7] Agrawal, R. & Srikant, R. (1994). *Fast Algorithms for Mining Association Rules*. VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, pp. 487-499

Uno de los artículos más importantes sobre reglas de asociación. En él se presentan los algoritmos *Apriori* y *AprioriTID*, así como su combinación *AprioriHybrid*, para la obtención de todas las reglas de asociación existentes en una base de datos de transacciones. En el artículo se muestra cómo esta familia de algoritmos mejora los algoritmos anteriores (AIS y SETM).

[8] Ali, K., Manganaris, S. & Srikant, R. (1997). *Partial Classification using Association Rules*. Proceedings of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining, August 14-17, 1997, Newport Beach, California, USA, pp. 115-118

Se expone la posibilidad de utilizar reglas de asociación para construir modelos de clasificación parcial que, si bien no siempre consiguen una clasificación precisa, pueden ser útiles para describir clases individuales. Como medida de la importancia que tiene cada regla de asociación a la hora de clasificar se utiliza el "riesgo relativo".

[9] Andersen, A., Blair, G., Goebel, V., Karlsen, R., Stabell-Kulo, T. & Yu, W. (2001). *Arctic Beans: Configurable and reconfigurable enterprise component architectures*. IEEE Distributed Systems Online, Vol. 2, No. 7.

En este artículo se presenta el proyecto Arctic Beans, que pretende dotar de mayor flexibilidad a los sistemas basados en componentes utilizados en la actualidad (como COM+, EJB o CORBA). Su idea es incluir mecanismos que le permitan al sistema configurarse, reconfigurarse y evolucionar de forma semi-autónoma.

[10] Anderson, D.P. & Kubiatowicz, J. (2002). *The Worldwide Computer*, Scientific American, March 2002.

Interesante artículo de los promotores del proyecto SETI@Home que discute los servicios que serían necesarios para crear la infraestructura necesaria para la construcción de un supercomputador virtual que incluyese la capacidad de todos los sistemas conectados a Internet: ISOS [*Internet-Scale Operating System*].

[11] Atkinton, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqau, R, Muthig, D., Paech, B., Wüst, J. & Zettel, J. (2002). *Component-based product line engineering with UML*. Addison-Wesley Component Software Series. ISBN 0-201-73791-4.

Libro en el que se describe KobrA, un método de desarrollo de software basado en componentes, el cual tiene sus raíces en los métodos dirigidos a objetos, Cleanroom, OPEN y otras técnicas basadas en componentes y orientadas a la creación de líneas de productos.

[12] Aumann, Y. & Lindell, Y. (1999). *A statistical theory for quantitative association rules*. Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, California, USA, pp. 261-270

Trabajo dedicado a la obtención de reglas de asociación con atributos numéricos en el cual se obtiene la distribución de los valores de los atributos numéricos correspondiente a los itemsets formados por atributos categóricos (que han de obtenerse en primer lugar) para encontrar perfiles que indiquen distribuciones representativas.

[13] Bayardo Jr., R. J. (1997). *Brute-Force Mining of High-Confidence Classification Rules*. En KDD-97, Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, August 14-17, 1997, Newport Beach, California, USA, pp. 123-126.

Se presentan una serie de técnicas de poda que permiten optimizar el proceso de extracción de reglas de asociación destinadas a ser utilizadas para construir clasificadores.

[14] Bayardo Jr., R. J. (1998). *Efficiently mining long patterns from databases*. Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA, pp. 85-93.

En este artículo se propone una forma alternativa de enfrentarse al problema de obtener el conjunto de itemsets frecuentes a la que se utiliza en los algoritmos derivados de *Apriori*. Tales algoritmos no son adecuados cuando hay k-itemsets frecuentes con k elevado, ya que todos sus subconjuntos (2^k) son, a su vez, frecuentes y como tales han de ser tratados. El algoritmo propuesto, *Max-Miner*, extrae eficientemente sólo aquéllos itemsets relevantes que no estén incluidos en otros itemsets relevantes.

[15] Bentley, J. (2000). *Programming Pearls*, 2nd edition. ACM Press / Addison-Wesley, ISBN 0-201-65788-0.

Uno de los pocos libros de Informática que puede considerarse una auténtica joya, por el ingenio y la perspicacia con la que Jon Bentley escribe sus ensayos. Originalmente, el contenido recopilado en este libro apareció en la columna homónima de *Communications of the ACM*.

[16] Bergholz, A. (2000). *Extending your markup: An XML tutorial*. IEEE Internet Computing, July / August 2000, pp. 74-79.

Breve y útil guía para todo aquél que desee familiarizarse con la sintaxis del lenguaje XML [*eXtensible Markup Language*] y la multitud de estándares que le rodea: DTD [*Document Type Definition*], XML Schema, XSL [*eXtensible Stylesheet Language*], XSLT [*XSL Transformations*]...

[17] Berry, M.J.A. & Linoff, G. (1997). *Data Mining Techniques: for Marketing, Sales, and Customer Support*. John Wiley and Sons, 1997.

Un libro destinado a ejecutivos y gerentes para que éstos se familiaricen con distintas técnicas de *Data Mining* existentes, su uso y sus limitaciones. El libro abarca desde el análisis de las transacciones comerciales [*basket data analysis*] hasta la utilización de árboles de decisión, métodos de agrupamiento, redes neuronales y algoritmos genéticos.

[18] Berzal, F., Cubero, J. C., Cuenca, F. & Martín-Bautista, M. J. (2001). *On the quest for easy-to-understand splitting rules*, pendiente de publicación en Data & Knowledge Engineering.

Trabajo en el que se proponen dos criterios de división alternativos (MAXDIF y GG) para la construcción de árboles de decisión. Ambos criterios consiguen resultados comparables a los de cualquier otra regla de división, si bien su complejidad es menor, con lo cual se facilita la comprensión del proceso de construcción del árbol de decisión por parte del usuario final de un sistema de extracción de conocimiento.

[19] Berzal, F., Cubero, J. C., Marín, N. & Serrano, J. M. (2001). *TBAR: An efficient method for association rule mining in relational databases*, Data & Knowledge Engineering, 37 (2001), pp. 47-64.

Artículo donde se presenta el algoritmo $\overline{T}$ (*TBAR*), el cual utiliza una estructura de datos basada en un árbol de enumeración de subconjuntos que permite mejorar las prestaciones de *Apriori* en la obtención de itemsets frecuentes.

[20] Bow, S.-T. (1991). *Pattern Recognition and Image Processing*. Marcel Dekker, 1991. ISBN 0-8247-8583-5.

Libro de texto de Reconocimiento de Formas en el que aparece descrito el algoritmo de agrupamiento basado en grafos citado en la sección 5.1.

[21] Bradshaw, J.M., Greaves, M., Holmback, H., Karygiannis, T., Jansen, W., Suri, N. & Wong, A. (1999). *Agents for the masses?*. IEEE Intelligent Systems, March / April 1999, pp. 53-63.

Informe que delinea las distintas líneas de investigación relacionadas con el desarrollo de sistemas distribuidos con agentes inteligentes. Se hace hincapié en los mecanismos de comunicación entre agentes y en la gestión de sistemas multiagente con el objetivo de simplificar y potenciar el desarrollo de tales sistemas.

[22] Brassard, G. & Bratley, P. (1997). *Fundamentos de algoritmia*. Prentice-Hall, ISBN 84-89660-00-X.

Libro de texto de Teoría de Algoritmos que incluye, entre otros, un capítulo dedicado a distintos algoritmos de exploración de grafos, como la "vuelta atrás" utilizada en TBAR para recorrer el árbol de itemsets y generar un conjunto de reglas de asociación.

[23] Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, California, USA, 1984.

El libro de CART, un clásico en la amplia literatura dedicada a la construcción de árboles de decisión.

[24] Brin, S., Motwani, R., Ullman, J. D. & Tsur, S. (1997). *Dynamic Itemset Counting and Implication Rules for Market Basket Data*. Proceedings of the ACM SIGMOD international conference on Management of Data, May 11 - 15, 1997, Tucson, Arizona, USA, pp. 255-264

Artículo en el que se presenta el algoritmo *DIC [Dynamic Itemset Counting]*. Este algoritmo, derivado de *Apriori*, reduce el número de veces que se ha de recorrer la base de datos para obtener sus itemsets frecuentes. Además, sus autores proponen el uso de "reglas de implicación" basadas en una medida de convicción como alternativa al uso de la confianza en las reglas de asociación.

[25] Butte, T. (2002). *Technologies for the development of agent-based distributed applications*. ACM Crossroads, 8:3, Spring 2002, pp. 8-15.

Análisis de los requisitos necesarios para la implementación de sistemas multiagente en entornos distribuidos en el que se incluye una discusión sobre las "facilidades"que ofrece Java para el desarrollo de este tipo de aplicaciones.

[26] Canavan, J.E. (2001). *Fundamentals of Network Security*. Artech House, 2001. ISBN 1-58053-176-8.

En este libro se analizan distintas vulnerabilidades existentes en los sistemas informáticos distribuidos, se describen algunas amenazas que se pueden presentar en forma de ataques y se describen los distintos mecanismos de seguridad que se pueden emplear para prevenirlos y neutralizarlos (p.ej. técnicas criptográficas de protección de datos).

[27] Chan, P.K. (1989). *Inductive learning with BCT*. Proceedings of the 6th International Workshop on Machine Learning, Ithaca, NY, June 130 - July 2.

En este artículo se propone BCT [*Binary Classification Tree*], un híbrido de ID3 [129] y CN2 [35] que construye árboles binarios en los que cada nodo contiene una expresión booleana definida potencialmente sobre varios atributos.

[28] Chandrasekaran, B., Josephson, J.R. & Benjamins, V.R. (1999). *What are Ontologies, and why do we need them?*. IEEE Intelligent Systems, January/February 1999, pp. 20-26.

Buena introducción al tema de las ontologías, una línea de investigación de moda en Inteligencia Artificial: la creación de teorías acerca de los tipos de objetos existentes, sus propiedades y las relaciones existentes entre ellos, así como su evolución temporal. El proyecto CYC de Lenat y la base de datos Wordnet son ejemplos destacables en este ámbito.

[29] Chaudhuri, S. & Dayal, U. (1997). *An Overview of Data Warehousing and OLAP Technology*. ACM SIGMOD Record, March 1997.

Excelente artículo que describe la arquitectura de un sistema OLAP y analiza algunas cuestiones relativas a su diseño.

[30] Chaudhuri, S., Fayyad, U. & Bernhardt, J. (1999). *Scalable Classification over SQL Databases*. IEEE: Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 23-26, 1999.

Ponencia en la que se propone la utilización de una arquitectura multicapa para construir clasificadores a partir de datos almacenados en bases de datos relacionales. En la arquitectura propuesta por los investigadores de Microsoft, existiría una capa intermedia de software entre la base de datos y el algoritmo de construcción de árboles de decisión que se encargaría de enviar a este último información resumida sobre los datos almacenados en la base de datos.

[31] Cheung, D.W., Han, J., Ng, V.T. & Wong, C.Y. (1996). *Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique*, Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, Louisiana, February 1996, pp. 106-114.

Ponencia que trata del mantenimiento de un conjunto de reglas de asociación dada una serie de modificaciones sobre la base de datos de la que se obtuvieron las reglas, lo que se conoce como extracción incremental de reglas de asociación.

[32] Cheung, D.W., Ng, V.T. & Tam, B.W. (1996). *Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules*, KDD'96, Second International Conference on Knowledge Discovery and Data Mining, Oregon, August 1996, pp. 307-310.

Continuación del artículo anterior [31] en el cual se extienden los algoritmos incrementales de extracción de reglas de asociación para que puedan trabajar con reglas de asociación generalizadas, que son reglas de asociación construidas utilizando jerarquías de conceptos.

[33] Clark, D. (1999). *Service with a (smart) smile: networks Jini-style*. IEEE Intelligent Systems, May / June 1999, pp.81-83.

Es este artículo se describen las características básicas de Jini, una tecnología que facilita el desarrollo de sistemas distribuidos dinámicos. También se comentan el origen de las ideas en que se basa Jini (el proyecto Linda de la Universidad de Yale) y se mencionan sus principales competidores (productos de funcionalidad similar ofrecidos por otras empresas rivales de Sun Microsystems).

[34] Clark, P. & Boswell, R. (1991). *Rule induction with CN2: Some Recent Improvements*. In Y. Kodratoff, editor, Machine Learning - EWSL-91, Berlin, 1991, Springer-Verlag, pp. 151-163

Se proponen dos mejoras sobre el algoritmo CN2 básico: el uso de una estimación laplaciana del error como función de evaluación alternativa y la obtención de un conjunto de reglas no ordenado. Los resultados obtenidos por este CN2 mejorado se comparan con C4.5.

[35] Clark, P. & Nibblett, T. (1989). *The CN2 Induction Algorithm*. Machine Learning Journal, Kluwer Academic Publishers, 3(4) pp. 261-183.

Se expone el algoritmo CN2, que fue desarrollado con el objetivo de conseguir un método eficiente de aprendizaje inductivo para la obtención de reglas de producción en presencia de ruido o de información incompleta. En este trabajo, CN2 se compara con ID3 y AQ.

[36] Clarke, I. (1999). *A distributed decentralises information storage and retrieval system*. University of Edinburg, Division of Informatics, 1999.

Informe en el que se describe la posible implementación de un sistema distribuido de almacenamiento y recuperación de información que no necesita un elemento central de control o administración. El sistema propuesto, implementado en FreeNet (http://freenetproject.org/), permite publicar información de forma anónima y evitar cualquier tipo de censura (se mantienen copias redundantes de los documentos para evitar su posible desaparición).

[37] Codd, E.F., Codd, S.B. & Salley, C.T. (1998). *Providing OLAP to User-Analysts: An IT Mandate*, Hyperion Solutions Corporation, 1998.

Informe de Codd y sus asociados en los que se analiza la evolución de los sistemas OLAP como complemento de los sistemas OLTP tradicionales. Aunque resulta interesante su lectura, hay que mantener una perspectiva crítica respecto a lo que en él afirma (no en vano, es un informe pagado por una casa comercial).

[38] Cohen, W. (1995). *Fast effective rule induction*. Proc. 12th International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115-123.

En esta ponencia se presenta el algoritmo RIPPERk, *Repeated Incremental Pruning to Produce Error Reduction*. Este algoritmo constituye una mejora sobre el algoritmo IREP y permite obtener listas de decisión más compactas mejorando la precisión de los clasificadores IREP.

[39] Cortijo Bon, F. (1999). Apuntes de *Reconocimiento de Formas*, E.T.S. Ingeniería Informática, Universidad de Granada.

Esta asignatura, opcional en los estudios de Ingeniería Informática, incluye varios temas relacionados con el contenido de esta memoria. Entre otros temas, en ella se estudian distintos tipos de clasificadores y se analizan varios métodos de agrupamiento.

[40] Coyle, F.P. (2002). *XML, Web Services and the changing face of distributed computing*, Ubiquity, ACM IT Magazine and Forum, Volume 3, Issue 10, April 23-29, 2002.

Artículo en el que se analizan las tendencias actuales relativas a la implementación de sistemas distribuidos, en los cuales se tiende a utilizar protocolos independientes del lenguaje de programación, del sistema operativo y del protocolo de transporte.

[41] Curbera, F., Duftlet, M., Khalaf, R., Nagy, W., Mukhi, N. & Weerawarana, S. (2002). *Unraveling the Web Services Web: An introduction to SOAP, WSDL, and UDDI*. IEEE Internet Computing, March / April 2002, pp. 86-93.

Como su título indica, este artículo repasa las tecnologías en que se basan los servicios web: SOAP como protocolo de comunicación basado en XML, WSDL como lenguaje universal de descripción de servicios y UDDI como especificación de un registro centralizado de servicios a modo de directorio.

[42] Czajkowski, G. & von Eicken, T. (1998). *JRes: a resource accounting interface for Java*. Proceedings of the conference on Object-oriented programming, systems, languages, and applications (OOPSLA'98), Vancouver, British Columbia, Canada, 1998, pp. 21-35.

En este artículo se propone una extensión de la máquina virtual Java estándar, denominada JRes, que permita monitorizar y controlar el uso de memoria, el tiempo de CPU y los recursos de entrada/salida (conexiones de red, ancho de banda...). Para ello se establecen límites sobre el uso de recursos que pueden hacer las distintas hebras de una aplicación multihebra.

[43] Czajkowski, G., Mayr, T., Seshadri, P. & von Eicken, T. (1999). *Resource Control for Java Database Extensions*. 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '99), San Diego, California, USA, May 3–7, 1999.

En esta ponencia se explora el uso de JRes para monitorizar el uso de un sistema de bases de datos, detectando posibles ataques distribuidos, monitorizando el uso de recursos de cada usuario y tomando medidas que puedan aprovecharse en la optimización de consultas.

[44] Davis, M. (2001). *Struts, an open-source MVC implementation*, IBM developerWorks, February 2001.

Breve artículo en el que se describe la arquitectura de Struts, que forma parte del proyecto Jakarta, un proyecto amparado por la fundación Apache. Struts implementa el modelo MVC en Java con servlets y JSP.

[45] Díaz, A. Glover, F., Ghaziri, H.M., González, J.L., Laguna, M., Moscato, P. & Tseng, F.T. (1996). *Optimización heurística y redes neuronales*. Editorial Paraninfo, ISBN 84-283-2264-4.

En este libro se analizan distintas técnicas de búsqueda heurística, entre las que se encuentran el enfriamiento simulado (al que se denomina "recocido simulado" en este libro), los algoritmos genéticos, la búsqueda tabú y GRASP [*Greedy Randomized Adaptive Search Procedure*].

[46] Domingos, P. (1996). *Linear-time rule induction*. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996, pp. 96-101

Artículo en el que se describe el algoritmo CWS, que mejora propuestas anteriores como CN2.

[47] Domingos, P. (1998). *Occam's Two Razors: The Sharp and the Blunt*. Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), August 27-31, 1998, New York City, USA, pp. 37-43

Crítica al uso del Principio de Economía de Occam como garantía de que la simplicidad de un modelo implica su mayor precisión (si bien la simplicidad como objetivo sí es apropiada en muchas ocasiones, para facilitar la comprensibilidad de los modelos construidos).

[48] Domingos, P. (1999). *The Role of Occam's Razor in Knowledge Discovery*. Data Mining and Knowledge Discovery Volume 3, Number 4, December 1999, pp. 409-425

Muchos sistemas de aprendizaje inductivo incorporan una preferencia explícita por modelos simples (la Navaja de Occam), si bien este criterio no siempre se utiliza correctamente. Este artículo expone que su uso continuado puede llegar a impedir la consecución de resultados interesantes en KDD.

[49] Dong, G. & Li, J. (1999). *Efficient mining of emerging patterns: discovering trends and differences*. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, CA USA, pp. 43-52.

En este artículo se propone el uso de EPs *[Emerging Patterns]*, itemsets cuyo soporte se ve incrementado significativamente de un conjunto de datos a otro.

[50] Dong, G., Zhang, X., Wong, L. & Li, J. (1999). *CAEP: Classification by Aggregating Emerging Patterns*. Proceedings of the Second International Conference on Discovery Science, Tokyo, Japan, 1999, pp. 30-42.

En esta ponencia se presenta CAEP, un clasificador que se construye a partir de EPs *[Emerging Patterns]* y consigue excelentes resultados cuando se compara con C4.5 o CBA.

[51] Dougherty, J., Kohavi, R., and Sahami, M. (1995). *Supervised and unsupervised discretization of continuous features*. Proceedings of the 12th International Conference on Machine Learning, Los Altos, CA, Morgan Kaufmann, 1995, pp. 194–202.

Este artículo repasa distintos métodos de discretización utilizados en Inteligencia Artificial. Los autores categorizan distintas propuestas en función de si son métodos supervisados o no supervisados y del uso global o local que se les suele dar.

[52] Dubois, D. and Prade, H. (1993). *Fuzzy sets and probability: misunderstandings, bridges and gaps*. Proceedings of the Second IEEE Conference on Fuzzy Systems, 1993, pp. 1059–1068.

Breve artículo en el que se intenta establecer un enlace entre la Teoría de la Probabilidad y la Teoría de los Conjuntos Difusos.

[53] Dreyfus, H. L. (1994). *What Computers Still Can't Do. A Critique of Artificial Reason*. The MIT Press, fourth printing, 1994.

Libro muy recomendable repleto de comentarios y críticas acerca de los supuestos logros de la Inteligencia Artificial (y de los que investigan en el tema).

[54] Duda, R.O. & Hart, P.E. (1973). *Pattern Classification and Scene Analysis*, John Wiley & Sons, ISBN 0471223611.

El libro de texto clásico de Reconocimiento de Formas. Wiley ha editado recientemente una versión revisada y extendida del libro (Richard O. Duda, Peter E. Hart, David G. Stork: *Pattern Classification (2nd Edition)*, 2000, ISBN 0471056693) en la que se mantiene un capítulo dedicado al aprendizaje no supervisado.

[55] Dykstra, J. (2002). *Software verification and validation with Destiny: A parallel approach to automated theorem proving*. ACM Crossroads, issue 8.3, Spring 2002, pp. 23-27.

En este artículo se describe Destiny, un sistema paralelo de demostración de teoremas con una arquitectura bastante peculiar: centralizada desde el punto de vista del control de la carga del sistema, en anillo desde el punto de vista de los nodos encargados de procesar datos.

[56] Elder IV, J.F. (1995). *Heuristic search for model structure: the benefits of restraining greed*. AI & Statistics - 95, 5th International Workshop on Artificial Intelligence and Statistics, Ft. Lauderdale, Florida, 3-6 January, pp. 199-210.

Ponencia en la que se propone un algoritmo de construcción de árboles de decisión en el que la bondad de una partición no se mide por la pureza de los hijos resultantes, sino por la de los 'nietos'.

[57] Fayad, M. & Schmidt, D.C., eds. (1997). *Object-oriented application frameworks*, Communications of the ACM, October 1997, pp. 32ss.

Sección especial de la revista mensual de la ACM dedicada al desarrollo de sistemas basados en componentes utilizando técnicas de orientación a objetos.

[58] Fayyad, U.M. & Irani, K.B. (1993). *Multi-interval discretization of continuous-valued attributes for classification learning*. Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1022-1027.

Trabajo pionero en la construcción de árboles de decisión n-arios con atributos continuos. Tras demostrar que para obtener el umbral óptimo basta con evaluar los puntos de corte entre casos de distintas clases, propone un método supervisado de discretización basado en el Principio MDL de Rissanen.

[59] Fayyad, U.M., Piatetsky-Shapiro, G. & Smyth, P. (1996). *The KDD process for extracting useful knowledge from volumes of data*, Communications of the ACM, November 1996, pp. 27-34.

Este artículo ofrece una visión general de aquello a lo que nos referimos al hablar de KDD, revisa algunos temas relacionados y concluye con una enumeración de los desafíos a los que hemos de enfrentarnos: gran volumen de datos, información incompleta o redundante, diseño de técnicas de interacción con el usuario, desarrollo de algoritmos incrementales...

[60] Feldman, R., Amir, A., Auman, Y., Zilberstien A. & Hirsh, H. (1997).*Incremental Algorithms for Association Generation*, Proceedings of the First Pacific-Asia Conference on Knowledge Discovery and Data Mining. En "KDD: Techniques and Applications", H. Lu et al. eds., World Scientific, Singapur, pp. 227-240.

Artículo en el que se decriben distintas técnicas incrementales de extracción de reglas de asociación.

[61] Flammia, G. (2001). *Peer-to-peer is not for everyone*. IEEE Intelligent Systems, May / June 2001, pp. 78-79.

Breve artículo en el que se acotan las posibles aplicaciones que el modelo P2P pueda llegar a tener en el desarrollo futuro, defendiendo la vigencia del modelo cliente/servidor. A diferencia de [10], el autor no espera que los sistemas P2P provoquen un cambio radical en el diseño de sistemas distribuidos.

[62] Fowler, M. (1997). *Analysis patterns: Reusable object models*. Addison-Wesley, ISBN 0-201-89542-0.

Un libro sobre patrones de diseño de utilidad en el análisis orientado a objetos de distintas aplicaciones comunes. En cierto modo, el contenido de este libro se complementa con las monografías de Hay [77] y Gamma et al. [65].

[63] Freitas, A. A. (2000). *Understanding the Crucial Differences between Classification and Discovery of Association Rules - A Position Paper*. SIGKDD Explorations, 2:1, July 2000, pp. 65-69.

Trabajo en el que se marcan claramente las diferencias conceptuales existentes entre clasificación y obtención de reglas de asociación.

[64] Fürnkranz, J., and Widmer, F. (1994). *Incremental reduced error pruning*. In Machine Learning: Proceedings of the 11th Annual Conference, New Brunswick, New Jersey, Morgan Kaufmann, 1994.

Ponencia que describe el algoritmo de inducción de listas de decisión IREP, *Incremental Reduced Error Pruning*.

[65] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns*, Addison-Wesley, ISBN: 0-201-633612.

El libro de "la Banda de los Cuatro", basado en la tesis doctoral de Erich Gamma, que marcó el comienzo del interés que hoy suscitan los patrones de diseño (en general, el estudio de buenas soluciones a problemas de desarrollo de software como complemento del tradicional estudio de las técnicas que permiten obtenerlas).

[66] Gause, D.C. & Wweinberg, G.M. (1989). *Exploring Requirements: Quality Before Design*, Dorset House, September 1989, ISBN: 0932633137.

Interesante libro de Don Gause y Jerry Weinberg en el que se tratan distintas técnicas útiles a la hora de realizar el análisis de requisitos de un sistema; esto es, cómo descubrir qué es lo que hay que hacer para avanzar en el árbol de decisión que conduce a la resolución de un problema.

[67] Gehrke, J., Ganti, V., Ramakrishnan, R. & Loh, W.-Y. (1999a). *BOAT - Optimistic Decision Tree Construction*. Proceedings of the 1999 ACM SIGMOD international conference on Management of Data, May 31 - June 3, 1999, Philadelphia, PA USA, pp. 169-180

Artículo en el que se presenta BOAT, un algoritmo de la familia TDIDT que permite la construcción eficiente, escalable, e incluso incremental de árboles de decisión al seleccionar un subconjunto de datos que se utiliza para generar un árbol que posteriormente se refina.

[68] Gehrke, J., Loh, W.-Y. & Ramakrishnan, R. (1999b). *Classification and regression: money can grow on trees*. Tutorial notes for ACM SIGKDD 1999 international conference on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, California, USA, pp. 1-73

Tutorial de KDD'99 en el que se ofrece un repaso de distintos algoritmos de construcción de árboles de decisión.

[69] Gehrke, J, Ramakrishnan, R. & Ganti, V. (2000). *RainForest - A Framework for Fast Decision Tree Construction of Large Datasets*. Data Mining and Knowledge Discovery, Vol. 4, No. 2/3, pp. 127-162.

En este artículo se presenta una familia de algoritmos TDIDT denominada *RainForest*. Se propone un método general de construcción eficiente de árboles de decisión separando sus propiedades de escalabilidad de los demás factores que influyen en la calidad de los árboles construidos.

[70] Gilb, T. (1988). *Principles of Software Engineering Management*. Addison-Wesley, ISBN 0-201-19246-2.

En esta obra se defiende un enfoque incremental en el desarrollo de software, haciendo especial hincapié en el establecimiento de objetivos concretos cuantificables. Aunque su lectura es algo tediosa, este libro está repleto de buenos consejos e ideas prácticas.

[71] Giordana, A. & Neri, F. (1996). *Search-intensive concept induction*. Evolutionary Computation 3(4):375-416, Massachusetts Institute of Technology.

En este artículo se describe REGAL, que emplea un algoritmo genético distribuido para inducir descripciones de conceptos (esto es, reglas de clasificación) utilizando Lógica de Primer Orden.

[72] Gong, L., ed., (2002). *Peer-to-peer networks in action*. IEEE Internet Computing, January / February 2002, pp. 37ss.

Sección especial dedicada a una de las tecnologías más prometedoras de las que han aparecido en los últimos años, que ha dado lugar a multitud de proyectos interesantes, tanto desde el punto de vista teórico como desde el punto de vista práctico. FreeNet, Gnutella y JXTA Search se encuentran entre los sistemas analizados en los artículos que componen esta sección de IEEE Internet Computing.

[73] Glymour, C., Madigan, D., Pregibon, D. & Smyth, P. (1996). *Statistical Inference and Data Mining*. Communications of the ACM, November 1996, pp. 35-41.

Este artículo trata de lo que la Estadística puede aportar a las técnicas de Data Mining: básicamente, la evaluación de las hipótesis generadas y de los resultados obtenidos.

[74] Han, J., Pei, J. & Yin, Y. (2000). *Mining Frequent Patterns without Candidate Generation*. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 15 - 18, 2000, Dallas, TX USA, pp. 1-12

Se presenta el *FP-Tree [Frequent-Pattern Tree]*, que permite representar una base de datos transaccional de una forma compacta. En una primera pasada por el conjunto de datos, se obtienen los items frecuentes y se establece un orden entre ellos (de más a menos frecuente). En un segundo recorrido secuencial de los datos se construye el árbol. A partír de él se pueden obtener todos los itemsets frecuentes de forma recursiva.

[75] Han, J. & Kamber, M. (2001). *Data Mining: Concepts and Techniques*. Academic Press, Morgan Kauffman Publishers, 2001.

Libro de texto muy recomendable que recoge en sus páginas muchas de las técnicas utilizadas para resolver problemas de *Data Mining*, desde el uso de *data warehouses* en aplicaciones OLAP hasta la extracción de reglas de asociación o la construcción de distintos tipos de clasificadores.

[76] Han, J.L. & Plank, A.W. (1996). *Background for Association Rules and Cost Estimate of Selected Mining Algorithms*. CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 12 - 16, 1996, Rockville, Maryland, USA, pp. 73-80

En este artículo se intenta comparar, desde un punto de vista estadístico, el coste computacional de distintos algoritmos para la obtención de reglas de asociación (Apriori, AprioriTid, AprioriHybrid, OCD, SETM y DHP) así como estudiar sus propiedades de escalabilidad.

[77] Hay, D.C. (1995). *Data Model Patterns*. Dorset House Publishing, ISBN 0-932633-29-3.

Libro muy interesante dedicado al modelado de datos en el que se describen distintos patrones que pueden ser de utilidad como punto de partida en el diseño de la base de datos de un sistema de información. Más allá de su interés académico, resulta útil para familiarizarse con muchos conceptos y procesos que aparecen durante el desarrollo de cualquier aplicación de gestión empresarial.

[78] Hedberg, S.R. (1999). *Data Mining takes off at the speed of the Web*. IEEE Intelligent Systems, November / December 1999, pp. 35-37.

En este informe se comenta la importancia económica que han adquirido las técnicas de Data Mining en el competitivo mundo empresarial: un mercado de 800 millones de dólares en 2000, con más de 200 empresas ofreciendo soluciones que incorporan técnicas de Data Mining, según META Group.

[79] Herrera, F. (1999). Apuntes de *Bioinformática*, E.T.S. Ingeniería Informática, Universidad de Granada.

Esta asignatura, opcional en los estudios de Ingeniería Informática, incluye un tema dedicado a la utilización de algoritmos genéticos en la construcción de sistemas clasificadores.

[80] Hidber, C. (1999). *Online Association Rule Mining*. Proceedings of the 1999 ACM SIGMOD international conference on Management of Data, May 31 - June 3, 1999, Philadelphia, PA, USA, pp. 145-156

Ponencia en la que se presenta *CARMA [Continuous Association Rule Mining Algorithm]*, un derivado de *Apriori* y *DIC* que sólo necesita recorrer dos veces la base de datos para obtener todos los itemsets frecuentes.

[81] Hipp, J., Güntzer, U. & Nakhaeizadeh, G. (2000). *Algorithms for Association Rule Mining - A General Survey and Comparison*. SIGKDD Explorations, Volume 2, Issue 1, June 2000, pp. 58-64

Estudio comparativo y análisis de las similitudes existentes entre los distintos algoritmos que se han propuesto para la extracción de reglas de asociación en el que se resaltan los aspectos comunes a todos ellos.

[82] Ho, K.M., and Scott, P.D. (1997). *Zeta: A global method for discretization of continuous variables*. 3rd International Conferemce on Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, AAAI Press, pp. 191-194.

Aquí se presenta un método supervisado de discretización basado en Zeta, una medida del grado de asociación existente entre los valores de dos atributos categóricos.

[83] Holte, R.C. (1993). *Very simple classification rules perform well on most commonly used datasets*. Machine Learning, 11:63-90.

Artículo en el que se presentó el discretizador 1R (*One Rule*), un sencillo método supervisado de discretización.

[84] Houtsma, M. & Swami, A. (1993). *Set-oriented mining for association rules*. IBM Research Report RJ9567, IBM Almaden Research Center, San Jose, California, October 1993.

Informe en el que se presentó el algoritmo SETM, un algoritmo equivalente a AIS [4] orientado hacia su implementación en SQL sobre bases de datos relacionales.

[85] Hussain, F., Liu, H., Tan, C.L., and Dash, M. (1999). *Discretization: An enabling technique*. The National University of Singapore, School of Computing, TRC6/99, June 1999.

Este informe repasa distintos métodos de discretización (todos ellos de tipo jerárquico) y propone una taxonomía que permite clasificarlos.

[86] Imielinski, T. & Mannila, H. (1996). *A Database Perspective on Knowledge Discovery*. Communications of the ACM, November 1996, pp. 58-64.

En este artículo se analizan los métodos empleados en Data Mining desde la perspectiva de las bases de datos. Se ponen de manifiesto las limitaciones de SQL a la hora de construir aplicaciones de Data Mining y se expone la necesidad de idear lenguajes de consulta más potentes. Tal como dijo C.J. Date, "El modelo relacional representa el lenguaje ensamblador de los sistemas modernos (y futuros) de bases de datos".

[87] Inmon, W.H. (1996). *The Data Warehouse and Data Mining*. Communications of the ACM, November 1996, pp. 49-50.

En este breve artículo se hace hincapié en que la calidad de los datos recopilados y de la forma en que se almacenan en un *data warehouse* es esencial para obtener buenos resultados al aplicar técnicas de *Data Mining*.

[88] Joshi, M.V., Agarwal, R.C., and Kumar, V. (2001). *Mining needles in a haystack: Classifying rare classes via two-phase rule induction*. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, California, 2001, pp. 91-101.

Los autores proponen un algoritmo de inducción de listas de decisión adecuado para problemas de decisión binarios en los cuales una de las clases es mucho menos frecuente que la otra, lo cual puede provocar que algoritmos como RIPPER o C4.5 no resulten adecuados.

[89] Juristo, N., Windl, H. & Constantine, L., eds. (2001). *Introducing Usability*, IEEE Software Issue on Usability Engineering, Vol. 18, No. 1, January/February 2001, pp. 20ss.

Sección dedicada a la difusión de técnicas que, utilizadas durante el proceso de desarrollo de software, permitan mejorar la usabilidad de los sistemas de información: facilidad de aprendizaje, eficiencia, prevención de errores y satisfacción del usuario.

[90] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. & Griswold, W.G. (2001). *Getting started with AspectJ*. Communications of the ACM, Vol. 44, No. 10, October 2001, pp. 59-65.

En este artículo se presenta AspectJ, una extensión "orientada a aspectos" del popular lenguaje de programación Java. Básicamente, la "orientación a aspectos" pretende mejorar la cohesión de las distintas facetas de un producto software, facilitando su implementación y posterior mantenimiento. Para lograrlo, se aplican a los lenguajes de programación estándar técnicas similares a las utilizadas por los disparadores en las bases de datos relacionales.

[91] Kobryn, C. (2000). *Modeling components and frameworks with UML*, Communications of the ACM, Volume 43, Issue 10, October 2000, pp. 31-38.

Artículo donde se describe el patrón de diseño utilizado por sistemas basados en componentes como J2EE (Sun Microsystems) o .NET (Microsoft Corporation).

[92] Kodratoff, Y. (1988). *Introduction to Machine Learning*. Pitman Publishing, 1988.

Libro sobre *Machine Learning* en el que destaca su tercer apéndice, *"ML in Context"*, en el que se realiza una curiosa analogía entre la educación y el aprendizaje automático.

[93] Kodratoff, Y. (2001). *Comparing Machine Learning and Knowledge Discovery in Databases*. Lecture Notes in Artificial Intelligence, Springer, LNAI 2049, pp. 1-21.

Interesante capítulo de libro que revisa distintas medidas que se pueden utilizar para caracterizar una regla $A \Rightarrow B$ y estimar su validez.

[94] Langley, P. (1996). *Elements of Machine Learning*. Morgan Kaufmann Publishers, 1996.

Libro de texto que pretende ser algo así como una ”tabla periódica”de métodos de aprendizaje e intenta eliminar las fronteras, artificiales en muchas ocasiones, que delimitan los distintos paradigmas existentes: inducción de reglas, aprendizaje analítico, aprendizaje basado en casos, redes neuronales, algoritmos genéticos...

[95] Larsen, G., ed. (2000). *Component-based enterprise frameworks*, Communications of the ACM, October 2000, pp. 24ss.

Sección especial de la revista mensual de la ACM dedicada a la construcción de sistemas basados en componentes y sus aplicaciones comerciales.

[96] Leavitt, N. (2002). *Industry Trends: Data Mining for the corporate masses?*. Computer, IEEE Computer Society, May 2002, pp. 22-24.

En este artículo se expone la situación actual del mercado mundial del *Data Mining*, en el que se destaca la aparición de nuevos estándares y se citan los principales desafíos a los que han de enfrentarse las empresas del sector, para lo cual se hace necesaria la invención de técnicas más eficientes y escalables, la integración con los sistemas de bases de datos existentes y el diseño de aplicaciones más fáciles de usar.

[97] Lee, J.H., Kim, W.Y., Kim, M.H., and Lee, J.L. (1993). *On the evaluation of boolean operators in the extended boolean retrieval framework*. ACM SIGIR’93, Pittsburgh, pp. 291-297

Interesante ponencia en la que se comentan las propiedades de los operadores booleanos que se emplean en distintos modelos (booleanos, obviamente) de Recuperación de Información.

[98] Lee, C.-H. & Shim, D.-G. (1999). *A multistrategy approach to classification learning in databases*. Data & Knowledge Engineering 31, 1999, pp. 67-93

Trabajo en el que se propone un algoritmo híbrido de aprendizaje que combina inducción de reglas para construir un modelo de clasificación parcial con el algoritmo k-NN para clasificar aquellos casos no cubiertos por las reglas obtenidas. También se propone en este artículo la utilización de la divergencia Hellinger para caracterizar las reglas obtenidas como medida de disimilitud que expresa el impacto del antecedente en la distribución del consecuente.

[99] Liu, A. (2001). *J2EE in 2001*, Component Development Strategies, Cutter Information Corp., September 2001.

En este informe se describe la plataforma Java 2 Enterprise Edition de Sun Microsystems y se analizan algunas de sus características más destacadas, incluyendo discusiones acerca de cómo afectan al desarrollo de aplicaciones de gestión en el ámbito empresarial.

[100] Liu, B., Hsu, W. & Ma, Y. (1998). *Integrating Classification and Association Rule Mining*. Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), August 27-31, 1998, New York City, USA, pp. 80-86.

Se expone cómo emplear reglas de asociación para construir un clasificador, *CBA [Classification Based on Associations]*, el cual, utilizando una clase por defecto, puede utilizarse como modelo de clasificación completo.

[101] Liu, B., Hu, M. & Hsu, W. (2000a) *Intuitive Representation of Decision Trees Using General Rules and Exceptions*. Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), July 30 - August 3, 2000, Austin, Texas.

Se construye un clasificador a partir de reglas de asociación utilizando una representación jerárquica que consiste en reglas generales y excepciones a esas reglas (en vez del modelo tradicional en el cual la existencia de demasiadas reglas dificulta la comprensibilidad del modelo construido).

[102] Liu, B., Hu, M. & Hsu, W. (2000b) *Multi-Level Organization and Summarization of the Discovered Rule*. Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 20 - 23, 2000, Boston, MA USA, pp. 208-217

Usando la misma aproximación que en [101], se puede obtener un resumen de la información contenida en un conjunto arbitrario de reglas.

[103] Liu, B., Ma, Y. & Wong, C.K. (2000c). *Improving an Association Rule Based Classifier*. Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-2000), September 13-16, 2000, Lyon, France.

Propone mejorar CBA [100] permitiendo la existencia de múltiples umbrales de soporte mínimo para las diferentes clases del problema y recurriendo a algoritmos TDIDT tradicionales cuando no se encuentran reglas lo suficientemente precisas.

[104] Loh, W.-Y. & Shih, Y.-S. (1997). *Split Selection Methods for Classification Trees*. Statistica Sinica, Vol.7, 1997, pp. 815-840

Trabajo en el que se presenta el algoritmo QUEST de construcción de árboles de decisión, que utiliza un criterio de selección no basado en medidas de impureza y no tiende a seleccionar atributos con un grado de ramificación elevado (como sucede en ID3, por ejemplo).

[105] Lopez de Mantaras, R. (1991). *A Distance-Based Attribute Selection Measure for Decision Tree Induction*. Machine Learning, 6, pp. 81-92.

Artículo en el que se propone una normalización de la ganancia de información alternativa al criterio de proporción de ganancia de C4.5. En este caso, la normalización se basa en una métrica de distancia.

[106] Maniatty, W.A. & Zaki, M.J. (2000). *Systems support for scalable Data Mining*. SIGKDD Explorations, December 2000, Volume 2, Number 2, pp. 56-65.

Artículo publicado en un boletín de uno de los grupos de interés especial de la ACM en el cual se comentan los requerimientos de un sistema paralelo de KDD, centrándose especialmente en su subsistema de entrada/salida.

[107] Mannila, H., Toivonen, H. & Verkamo, A.I. (1993): *Improved methods for finding association rules*. University of Helsinky, Department of Computer Science, C-1993-65, December 1993.

Partiendo del algoritmo AIS de Agrawal, Imielinski y Swami, se proponen algunas mejoras. Por ejemplo, calcular $C[k+1]$ a partir de $C[k] \times C[k]$ en vez de usar $L[k] \times L[k]$ para reducir el número de veces que se ha de recorrer la base de datos. Además, se propone podar a priori el conjunto de candidatos $C[k+1]$ generado a partir de $L[k]$, la base del algoritmo *Apriori* desarrollado independientemente en IBM.

[108] Martin, J. K. (1997). *An Exact Probability Metric for Decision Tree Splitting and Stopping*. Machine Learning, 28, pp. 257-291.

Artículo en el que se puede encontrar un estudio exhaustivo sobre distintas reglas de división propuestas para la construcción de árboles de decisión, la correlación entre ellas y resultados experimentales con árboles de decisión binarios.

[109] McConnell, S. (1996). *Rapid Development: Taming wild software schedules*. Microsoft Press, ISBN 1-55615-900-5.

Este libro, que en su día recibió un *Jolt award* (los oscars en el mundo del software), recopila un extenso conjunto de técnicas que pueden ser útiles en el desarrollo de software. Se discuten las virtudes y defectos de cada una de las técnicas y se incluye gran cantidad de datos obtenidos empíricamente relativos al proceso de desarrollo de software. Es memorable el capítulo dedicado a los errores clásicos que se cometen una y otra vez en proyectos de desarrollo de software.

[110] McConnell, S. (1998). *Software Project Survival Guide*. Microsoft Press, ISBN 1-57231-621-7.

Otra obra del editor jefe de IEEE Software en la que se describe un conjunto de técnicas útiles para garantizar, en la medida en que esto es posible, la finalización con éxito de un proyecto de desarrollo de software. Son dignas de mención las listas de comprobación que el autor incluye en cada capítulo para poder evaluar la progresión adecuada de un proyecto.

[111] Mehta, M., Agrawal, R. & Rissanen, J. (1996). *SLIQ: A Fast Scalable Classifier for Data Mining*. Advances in Database Technology - Proceedings of the Fifth International Conference on Extending Database Technology (EDBT'96), Avignon, France, March 25-29, 1996, pp. 18-32

En este trabajo se presenta el algoritmo SLIQ, un algoritmo de construcción de árboles de decisión diseñado específicamente para aplicaciones de Data Mining dentro del proyecto Quest de IBM en el Almaden Research Center de San Jose, California.

[112] Meretakis, D. & Wüthrich, B. (1999). *Extending naïve Bayes classifiers using long itemsets*. Proceedings of the fifth ACM SIGKDD international conference on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, CA USA, pp. 165-174

Se propone el algoritmo *LB [Large Bayes]*, que utiliza itemsets frecuentes para mejorar los resultados que se pueden conseguir con el algoritmo Naïve Bayes.

[113] Meyer, B. (2001). *.NET is coming*. IEEE Computer, August 2001, pp. 92-97.

En este artículo, escrito por el creador del lenguaje de programación Eiffel, se analiza la plataforma .NET de Microsoft. A diferencia de la plataforma Java, que es independiente del sistema operativo pero está ligada al lenguaje Java, la plataforma .NET es independiente del lenguaje de programación pero requiere utilizar sistemas operativos de Microsoft (al menos por ahora).

[114] Miller, R. J. & Yang, Y. (1997). *Association rules over interval data*. Proceedings of the ACM SIGMOD Conference on Management of Data, May 11-15, 1997, Tucson, AZ USA, pp. 452-461

Trabajo sobre reglas de asociación con atributos numéricos en el que se encuentra poco adecuada la utilización del soporte y la confianza para caracterizar las reglas obtenidas y se propone el uso de un algoritmo de clustering (BIRCH) para obtener intervalos potencialmente interesantes y una medida del grado de asociación entre los items intervalares obtenidos.

[115] Mitchell, T.M. (1997). *Machine Learning*. McGraw-Hill College Div, ISBN: 0070428077.

Este libro de texto, dedicado enteramente al estudio de distintas técnicas de aprendizaje automático, incluye un capítulo destinado al análisis de distintos algoritmos de aprendizaje de conjuntos de reglas, como, por ejemplo, FOIL.

[116] OMG - The Object Management Group (2000). *OMG Unified Modeling Language Specification*, Version 1.3, First Edition, March 2000, http://www.omg.org/technology/uml/

La especificación completa del Lenguaje Unificado de Modelado, un estándar de facto utilizado en el diseño y análisis de sistemas orientados a objetos.

[117] Othman, O., O'Ryan C. & Schmidt, D. C. (2001). *Strategies for CORBA Middleware-Based Load Balancing*. IEEE Distributed Systems Online, Vol. 2, No. 3, March 2001.

En este artículo se presentan las deficiencias de algunas técnicas comunes de balanceado de carga y se propone una nueva, que los autores han diseñado utilizando características estándar de CORBA y TAO, un ORB concreto.

[118] Özsu, M.T. & Valduriez, P. (1991). *Principles of distributed database systems*. Prentice-Hall International, ISBN 0-13-715681-2.

Posiblemente uno de los mejores libros de texto de bases de datos, al menos a juicio del autor de esta memoria. Incluye un excelente resumen de los conceptos en que se fundamentan las bases de datos relacionales y una gran cantidad de información referente a la implementación de sistemas gestores de bases de datos.

[119] Park, J.S., Chen, M.S. & Yu, P.S. (1995). *An effective hash-based algorithm for mining association rules*. Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995, pp. 175-186

En esta ponencia se propone el algoritmo para disminuir el número de itemsets candidatos generados en las primeras iteraciones por algoritmos derivados de Apriori. Además, el tamaño de la base de datos que se debe recorrer se va reduciendo paulatinamente en cada iteración del algoritmo (las transacciones en las que ya sabemos que no se encuentra ningún itemset frecuente no hay por qué comprobarlas en las siguientes iteraciones).

[120] Park, J.S., Chen, M.S. & Yu, P.S. (1997). *Using a Hash-Based Method with Transaction Trimmming for Mining Association Rules*. IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 5, 1997, pp. 813-825.

Artículo básicamente idéntico a la ponencia de los autores en ACM SIGMOD'95. Como 'novedad', se propone reducir el número de pasadas realizadas por la base de datos generando los candidatos $C[k+1]$ a partir de $C[k] \times C[k]$ en vez de utilizar $L[k] \times L[k]$. Esto es válido si el conjunto de k-itemsets candidatos es lo suficientemente pequeño como para caber en memoria principal.

[121] Patil, A.N. (2001). *Build your own Java-based supercomputer*. IBM developerWorks, April 2001.

Propone la utilización de hebras pseudo-remotas para simplificar el desarrollo de programas paralelos. La planificación de tareas se realiza en una única máquina local, aunque el código correspondiente a las tareas se puede ejecutar en máquinas remotas.

[122] Pazzani, M.J. (2000). *Knowledge discovery from data?*, IEEE Intelligent Systems, March / April 2000, pp. 10-13.

Michael Pazzani realiza una dura crítica de la escasa consideración que solemos tener los que diseñamos sistemas de Data Mining con respecto al usuario final de estos sistemas. Pazzani propone añadir la Psicología Cognitiva a las tres áreas de conocimiento que usualmente se asocian con Data Mining (esto es, Estadística, Bases de Datos e Inteligencia Artificial) con el fin de mejorar la utilidad real de los sistemas de KDD y facilitar la obtención de información útil a partir de grandes conjuntos de datos.

[123] Perry, D.E. & Kaiser, G.E. (1991). *Models of software development environments*. IEEE Transactions on Software Engineering, March 1991, Vol. 17, No. 3, pp. 283-295.

Artículo en el que se propone la caracterización de un sistema de desarrollo de software mediante un modelo general con tres componentes (modelo SMP: estructuras, mecanismos y políticas). A partir de ese modelo básico, se delinean cuatro clases de sistemas en función de su escala utilizando una metáfora sociológica (taxonomía IFCS: individuo, familia, ciudad y estado).

[124] Piatetsky-Shapiro, G. (1999): *The Data-Mining Industry coming of age*, IEEE Intelligent Systems, November / December 1999, pp. 32-34.

Artículo de opinión en el que analizan las aplicaciones y tendencias que pueden marcar la evolución de las técnicas de Data Mining en productos comerciales: creación de estándares, paquetes software como Clementine (SPSS) o Intelligent Miner (IBM) y soluciones verticales para aplicaciones específicas (banca, telecomunicaciones, biotecnología, CRM, comercio electrónico, etc.).

[125] Piatetsky-Shapiro, G., editor (2001): *KDnuggets News*, Vol. 1, No. 11, Item 2, May 29, 2001

Resultados de una encuesta efectuada por *KDnuggets* acerca de las aplicaciones más populares de las técnicas de extracción de conocimiento en bases de datos y *Data Mining*: la banca (17 %) y comercio electrónico (15 %); las telecomunicaciones (11 %); la biología y, en particular, la genética (8 %); la detección de fraude (8 %) y la investigación científica en general (8 %).

[126] Plauger, P.J. (1993). *Programming on Purpose: Essays on Software Design*. PTR Prentice Hall, ISBN 0-13-721374-3.

Excelente libro que recopila algunas de las mejores columnas publicadas por Plauger en la revista *Computer Language*, que hoy sigue publicándose mensualmente bajo el título *Software Development* y mantiene a algunas de las mejores firmas del momento entre sus colaboradores.

[127] Pressman, R.S. (1993). *Ingeniería del Software: Un enfoque práctico*. McGraw-Hill, 3ª edición.

La biblia de la Ingeniería del Software, donde se puede encontrar información sobre casi todo lo relacionado con el proceso de desarrollo de software y la cita con la que se abre esta memoria.

[128] Provost, F. & Kolluri, V. (1999). *A survey of methods for scaling up inductive algorithms*, Data Mining and Knowledge Discovery, volume 3, number 2, pp. 131-169.

Estudio en profundidad de tres estrategias que se pueden seguir para poder aplicar algoritmos de aprendizaje a grandes conjuntos de datos: diseñar un algoritmo rápido (restringiendo el espacio de búsqueda, afinando heurísticas de búsqueda de soluciones, optimizando algoritmos existentes o utilizando paralelismo), dividir el conjunto de datos (uso de muestreo, técnicas incrementales o aprendizaje cooperativo) y utilizar una representación relacional de los datos (p.ej. ontologías: lógica proposicional y jerarquías de conceptos).

[129] Quinlan, J.R. (1986a). *Induction on Decision Trees*. Machine Learning, 1, 1986, pp. 81-106

Uno de los trabajos clásicos en *Machine Learning*. En él se describe el algoritmo ID3, que se enmarca dentro de la familia TDIDT de sistemas de aprendizaje inductivo.

[130] Quinlan, J.R. (1986b). *Learning Decision Tree Classifiers*. ACM Computing Surveys, 28:1, March 1996, pp. 71-72

Breve artículo que ofrece una visión general de la construcción de árboles de decisión para resolver problemas de clasificación.

[131] Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. ISBN 1-55860-238-0.

El libro del C4.5, uno de los muchos derivados de ID3. No obstante, el algoritmo C4.5 puede considerarse un híbrido entre CART y C4.

[132] Quinlan, J.R. (1996). *Improved use of continuous attributes in C4.5*. Journal of Artificial Intelligence Research, 4:77-90.

Artículo en el que Quinlan propone algunas mejoras sobre el algoritmo C4.5 cuando intervienen atributos de tipo numérico. Se propone un ajuste de la ganancia obtenida en función del número de valores distintos de un atributo (C4.5 Rel.8) y se comenta la posibilidad de utilizar métodos de discretización local al construir el árbol de decisión.

[133] Rasmussen, E. (1992). *Clustering algorithms*. En W.B. Frakes y R. Baeza-Yates (eds.): *Information Retrieval: Data Structures and Algorithms*, Chapter 16.

Capítulo dedicado a los métodos de agrupamiento en uno de los monográficos clásicos dedicados a problemas de recuperación de información.

[134] Rastogi, R. & Shim, K. (2000). *PUBLIC: A Decision Tree Classifier that integrates building and pruning*. Data Mining and Knowledge Discovery, Vol. 4, No. 4, pp. 315-344.

Artículo en el que se propone PUBLIC, un algoritmo de construcción de árboles de decisión que integra la poda del árbol en su construcción: un nodo no se llega a expandir cuando se está construyendo el árbol cuando se determina que se podaría posteriormente durante la post-poda (con la consiguiente mejora computacional que supone ahorrarse la construcción de nodos que después se eliminarían).

[135] Rivest, R.L. (1987). *Learning decision lists*. Machine Learning Journal, vol. 2, no. 3, pp. 229-246, 1987.

El trabajo en el que presenta formalmente el concepto de lista de decisión como método de representación de funciones booleanas.

[136] Sánchez, D. (1999). *Adquisición de relaciones entre atributos en bases de datos relacionales*. Tesis doctoral, Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, diciembre de 1999.

Tesis doctoral cuya principal aportación consiste en proponer el uso de factores de certeza en vez de utilizar el omnipresente valor de confianza para caracterizar las reglas de asociación extraídas de una base de datos relacional.

[137] Sarawagi, S., Thomas, S. & Agrawal, R. (1998). *Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications*, SIGMOD 1998, Proceedings of the ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA, pp. 343-354.

En esta ponencia, resumen de un informe realizado en el seno del proyecto Quest de IBM, se discuten las distintas alternativas existentes a la hora de implementar algoritmos de extracción de reglas de asociación en bases de datos relacionales. El abanico de posibilidades abarca desde una implementación débilmente acoplada al DBMS en la cual se utilizan lenguajes de propósito general e interfaces estándar de acceso a bases de datos hasta una implementación no portable fuertemente acoplada al sistema gestor de bases de datos particular que haga uso de interfaces no estándar con el objetivo de obtener un rendimiento máximo.

[138] Segal, R. & Etzioni, O. (1994). *Learning decision lists using homogeneous rules*. AAAI-94, American Association for Artificial Intelligence.

Segal y Etzioni proponen el algoritmo BruteDL, que emplea la fuerza bruta para construir un clasificador basado en reglas. Dado que la precisión de una lista de decisión no es una función directa de la precisión de las reglas que la componen, BruteDL realiza una búsqueda en profundidad para obtener un conjunto de reglas en el cual la interpretación de cada regla no depende de su posición.

[139] Sen, A. (1998). *From DSS to DSP: A taxonomic retrospection*. Communications of the ACM Virtual Extension, Mayo 1998, volumen 41, número 5.

Artículo retrospectivo en el que se analiza la evolución de los sistemas de ayuda a la decisión desde sus orígenes hasta la actualidad, evolución que está muy ligada al progreso de las técnicas de Inteligencia Artificial y de los sistemas de bases de datos.

[140] Sestito, S. & Dillon, T.S. (1994). *Automated Knowledge acquisition*. Sydney, Australia: Prentice Hall, 1994.

Libro sobre técnicas de aprendizaje automático que incluye, entre otros, capítulos dedicados por completo a la construcción de árboles de decisión y a la inducción de reglas utilizando la metodología STAR de Michalski.

[141] Shafer, J.C., Agrawal, R. & Mehta, M. (1996). *SPRINT: A Scalable Parallel Classifier for Data Mining*. VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India, pp. 544-555

Artículo sobre SPRINT, un algoritmo eficiente y escalable de construcción de árboles de decisión diseñado para ser paralelizado, lo cual lo hace idóneo para trabajar con grandes bases de datos. Derivado de SLIQ [111], también fue desarrolado dentro del proyecto Quest de IBM.

[142] Shah, M.A., Madden, S., Franklin, M.J. & Hellerstein, J.M. (2001). *Java support for data-intensive systems: Experiences building the Telegraph Dataflow System*. ACM SIGMOD Record, Volume 30, Number 4, December 2001, pp. 103-114.

En este artículo se destacan algunos de los placeres de la programación en Java de aplicaciones de cálculo intensivo, así como algunas de sus limitaciones.

[143] Shih, Y.-S. (1999). *Families of splitting criteria for classification trees*. Statistics and Computing, vol.9, no.4; Oct. 1999; pp. 309-315.

Estudio acerca de las reglas de división utilizadas para construir árboles de decisión binarios.

[144] Shohan, Y. (1999) *What we talk about when we talk about software agents*. IEEE Intelligent Systems, March / April 1999, pp. 28-31.

En esta columna de opinión se critica la propaganda exagerada que existe en torno a la idea de los agentes software, si bien también se resalta la existencia de trabajos de interés que utilizan el término agente en tres ámbitos diferentes: nuevos sistemas expertos (con aplicaciones, por ejemplo, en recuperación de información), sistemas distribuidos (como el descrito en el capítulo 6) y diseño antropomórfico (con el objetivo de obtener máquinas con un comportamiento 'casi humano').

[145] Silverstein, C., Brin, S. & Motwani, R. (1998). *Beyond market baskets: Generalizing association rules to dependence rules*, Data Mining and Knowledge Discovery, 2:39-68.

Artículo en el que se propone la utilización de una medida de interés para caracterizar las reglas de asociación, en vez de utilizar la confianza de la regla.

[146] Srikant, R. & Agrawal, R. (1996). *Mining Quantitative Association Rules in Large Relational Tables*. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996, pp. 1-12

En este artículo se plantea el problema de la extracción de reglas de asociación en bases de datos que contengan atributos numéricos y se propone la utilización del método de partición equitativa *[equi-depth partitioning]*.

Para obtener las reglas de asociación se expone una variante del algoritmo Apriori que hace uso de una medida de interés para podar los conjuntos de candidatos.

[147] Skirant, R., Vu, Q. & Agrawal, R. (1997). *Mining Association Rules with Item Constraints*. KDD'97 Proceedings of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining, Newport Beach, California, August 1997, pp. 67-73.

Este trabajo se centra en la imposición de restricciones al formato de las reglas de asociación y cómo esas restricciones permiten guiar el proceso de extracción de reglas de asociación.

[148] Spell, B. (2000). *Professional Java Programming*, Wrox Press Ltd., December 2000.

Este manual de programación avanzada en Java incluye un capítulo dedicado a la implementación de sistemas distribuidos, comparando el uso de TCP (sockets), CORBA y RMI. Obviamente, RMI es la mejor opción cuando todo el sistema está escrito en Java.

[149] Stang, M. & Whinston, S. (2001). *Enterprise computing with Jini technology*. IT Pro, IEEE Computer Society, January / February 2001, pp. 33-38.

En este artículo se describen las ventajas que ofrece Jini a la hora de construir sistemas distribuidos autoconfigurables. Entre ellas, por ejemplo, se encuentra la posibilidad de distribuir código ejecutable (esto es, se puede dotar de movilidad a los agentes de un sistema multiagente con facilidad).

[150] Taylor, P. C. & Silverman, B. W. (1993). *Block diagrams and splitting criteria for classification trees*. Statistics and Computing, vol. 3, no. 4, pp. 147-161.

Artículo en el que se propone la regla MPI [Mean Posterior Improvement] como alternativa al índice de Gini para construir árboles de decisión binarios.

[151] Tou, J. T., and Gonzalez, R. C. (1974). *Pattern Recognition Principles*. Addison-Wesley, 1974. ISBN 0-201-07587-3.

Libro de texto de Reconocimiento de Formas en el que aparece descrito, con un enrevesado diagrama de flujo de varias páginas, el algoritmo ISODATA citado en la sección 5.1.

[152] Touzet, D., Menaud, J.M., Weis, F., Couderc, P. & Banâtre, M. (2001). *SIDE Surfer: Enriching casual meetings with spontaneous information gathering*. ACM SIGARCH Computer Architecture News, Vol. 29, No.5,

pp. 76-83, December 2001. Ubiquitous Computing and Communication Workshop, PACT (Parallel Architecture and Compilation Techniques) Conference, Barcelona, Septiembre 2001.

En esta ponencia se utiliza una estructura de datos como la de TBAR (capítulo 4) para construir perfiles de usuario que permitan un intercambio de información espontáneo entre dispositivos inalámbricos próximos (tipo PDA).

[153] Ullman, J. (2000). *Data Mining Lecture Notes*. Stanford University CS345 Course on Data Mining, Spring 2000. http://www-db.stanford.edu/∼ullman/mining/mining.html

Apuntes de un curso de Data Mining impartido en la Universidad de Stanford por Jeffrey Ullman, una de las máximas autoridades en el campo de las bases de datos.

[154] Van de Merckt, T. (1993). *Decision trees in numerical attribute spaces*. Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1016-1021.

Artículo en el que se propone la utilización de una medida de contraste para discretizar atributos continuos al construir árboles de decisión. Se basa en la idea de buscar agrupamientos tan distantes como sea posible en el espacio, independientemente de si contienen o no elementos de distintas clases. También se propone una variante supervisada que tiene en cuenta la entropía de la partición resultante.

[155] Vanhelsuwé, L. (1997). *Create your own supercomputer with Java*. JavaWorld, January 1997

Artículo en el que se propone la utilización de Java para realizar tareas de cómputo distribuido utilizando UDP para transferir información de una máquina a otra.

[156] Waltz, D. & Hong S.J., eds. (1999). *Data Mining: A long-term dream*. IEEE Intelligent Systems, November / December 1999, pp. 30ss.

En este número de la revista de IEEE apareció una sección dedicada por completo al creciente interés que despiertan las técnicas de Data Mining, tanto en los centros de investigación como en las empresas, por sus múltiples aplicaciones en biotecnología, compañías aseguradoras, detección de fraude, mantenimiento de aeronaves...

[157] Wang, X., Chen, B., Qian, G. & Ye, F. (2000). *On the optimization of fuzzy decision trees*. Fuzzy Sets and Systems, 112, Elsevier Science B.V., pp. 117-125.

En este artículo se describe la utilización de conjuntos difusos en la construcción de árboles de decisión, dando lugar a árboles en que los caminos de la raíz a las hojas no son mutuamente excluyentes y un caso particular puede corresponder a varias hojas simultáneamente con distintos grados de pertenencia.

[158] Wang, K., Zhou, S. & He, Y. (2000). *Growing decision trees on support-less association rules*. Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 20-23, 2000, Boston, MA USA, pp. 265-269

Trabajo en el que se construye un árbol a partir de reglas de asociación considerando únicamente la confianza de las reglas. El umbral de soporte mínimo se elimina del proceso de extracción de reglas de asociación porque el soporte de una regla no resulta indicativo de su capacidad predictiva.

[159] Wang, K., Zhou, S. & Liew, S.C. (1999). *Building Hierarchical Classifiers Using Class Proximity*. VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK, pp. 363-374

Se emplea una estrategia similar a la de CBA [100] para construir un clasificador contextual que permita clasificar documentos por temas en jerarquías de conceptos.

[160] Wayne, R. (2002). *Peer (to peer) pressure: it's a good thing*. Software Development, 10:4, April 2002, pp. 38-43.

Artículo en el que se le da un repaso a los sistemas P2P existentes en la actualidad y se analiza su posible evolución en el fututo: "the next big thing?".

[161] Widom, J. (1995). *Research Problems in Data Warehousing*. Proceedings of the 1995 International Conference on Information and Knowledge Management, CIKM'95, November 29 - December 2, 1995, Baltimore, MD, USA, pp. 25ss.

Ponencia en la que se exponen algunos de los problemas que hay que resolver para construir aplicaciones OLAP.

[162] Yeager, W. & Williams, J. (2002). *Secure peer-to-peer networking: The JXTA example*. IT Pro, IEEE Computer Society, March / April 2002, pp. 53-57.

En este artículo se describe el proyecto JXTA haciendo especial énfasis en aspectos relativos a temas de seguridad. La plataforma JXTA para el desarrollo de aplicaciones P2P está basada en XML y es independiente tanto del

lenguaje de programación como del sistema operativo y de los protocolos de red subyacentes.

[163] Zheng, Z. (2000). *Constructing X-of-N Attributes for Decision Tree Learning*. Machine Learning 40(1), July 2000, pp. 35-75

Trabajo en el que se muestra la utilidad del uso simultáneo de varios atributos para ramificar un árbol de decisión.

[164] Zwick, R., Carlstein, E., Budescu, D.V. (1987). *Measures of similarity among fuzzy concepts: A comparative analysis*. International Journal of Approximate Reasoning, 1987, 1:221-242.

Interesante estudio que analiza distintas formas de medir la similitud entre dos entidades desde el punto de vista de la Psicología.

Que otros se jacten de las páginas que han escrito; a mí me enorgullecen las que he leído.

JORGE LUIS BORGES