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.

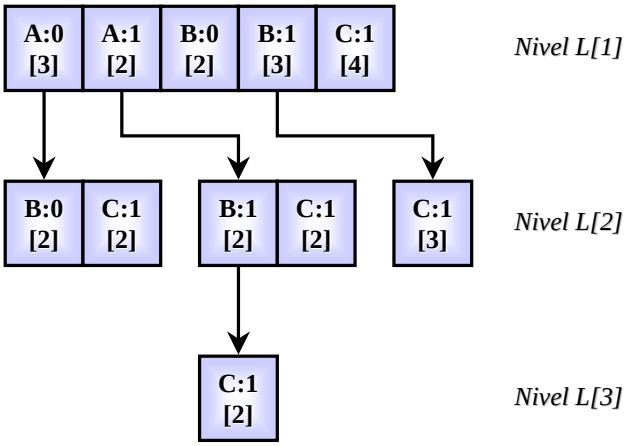

Figura 4.1: Árbol de itemsets correspondiente a los itemsets mostrados en la tabla 4.2

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

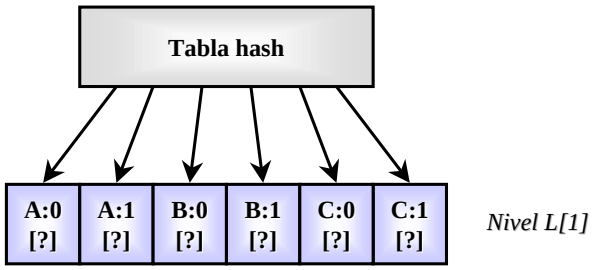

Figura 4.2: Árbol de itemsets tras su inicialización. Una tabla hash asegura el acceso eficiente a los items del nodo raíz.

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

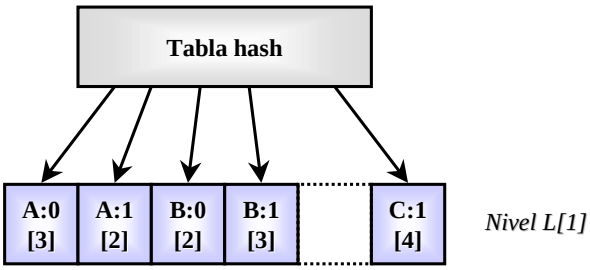

Figura 4.3: Árbol de itemsets tras eliminar los items no relevantes del conjunto de candidatos C[1].

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

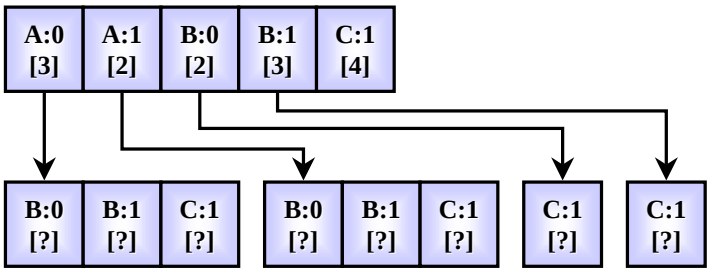

Figura 4.4: Itemsets candidatos de tamaño 2

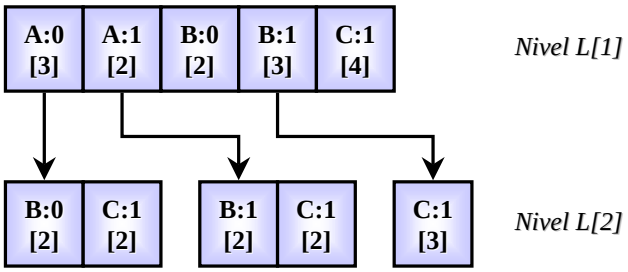

Figura 4.5: Itemsets relevantes de tamaño 2

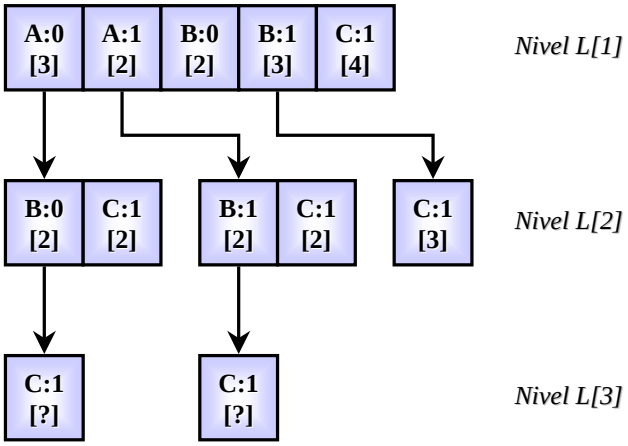

Figura 4.6: Itemsets candidatos de tamaño 3

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

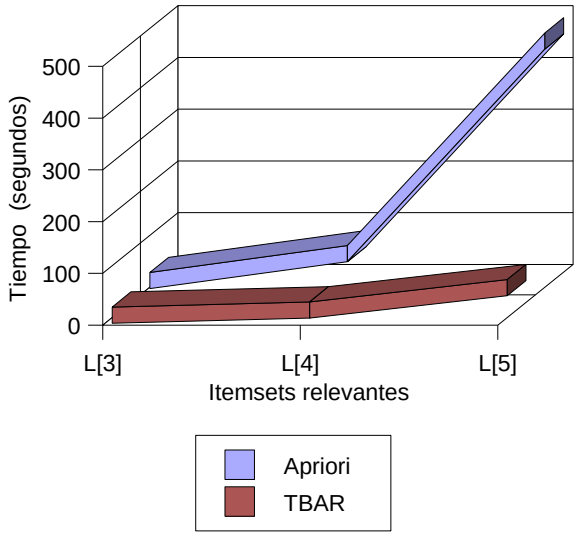

Figura 4.7: TBAR vs. Apriori (SOYBEAN)

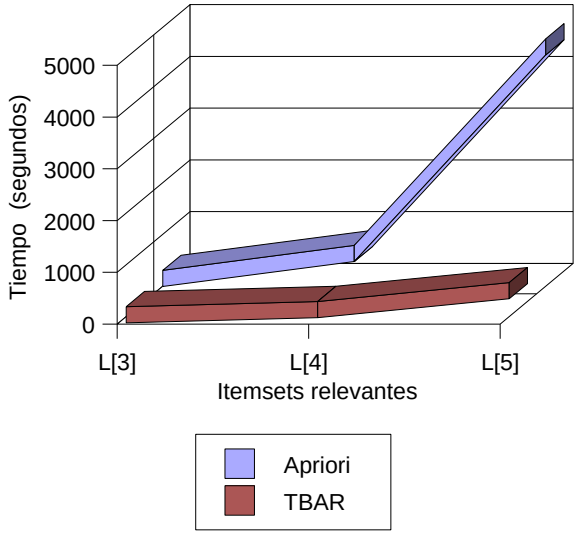

Figura 4.8: TBAR vs. Apriori (MUSHROOM)

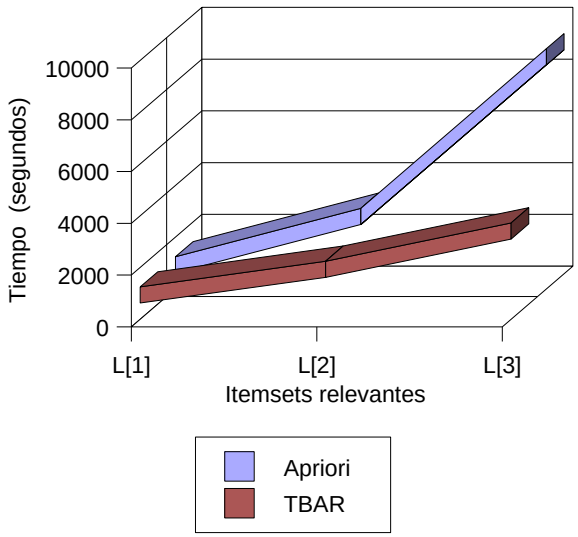

Figura 4.9: TBAR vs. Apriori (CENSUS)

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.

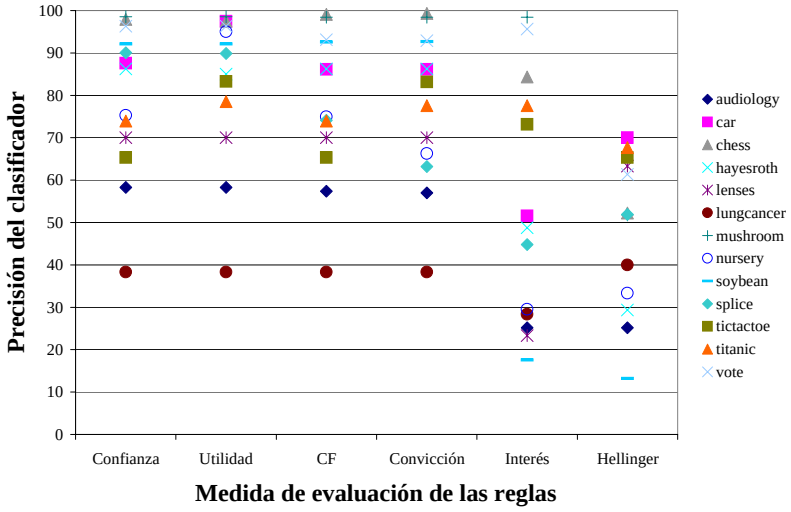

Figura 4.10: Precisión del clasificador ART para distintas medidas de evaluación de las reglas.

- 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.

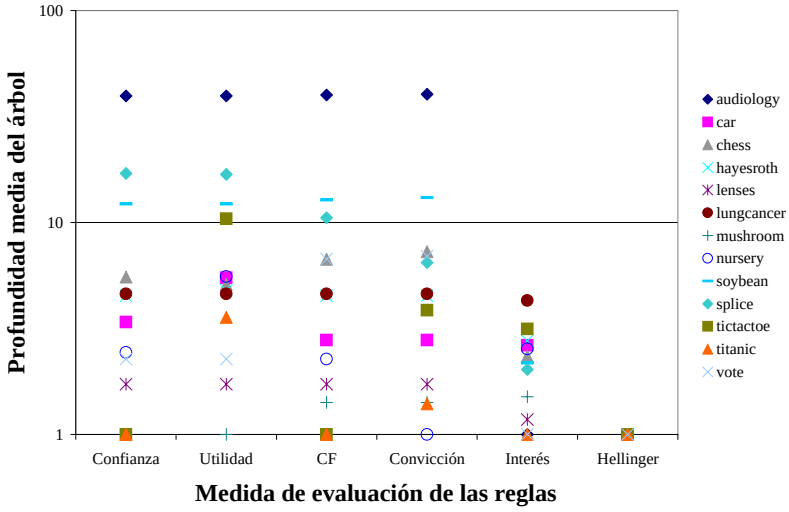

Figura 4.11: Profundidad del árbol ART para distintas medidas de evaluación de las reglas.

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).

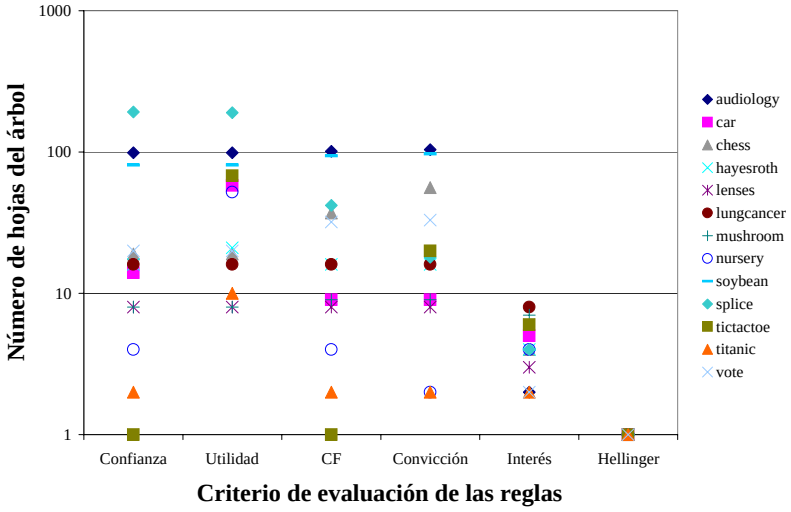

Figura 4.12: Número de hojas del árbol ART para distintas medidas de evaluación de las reglas.

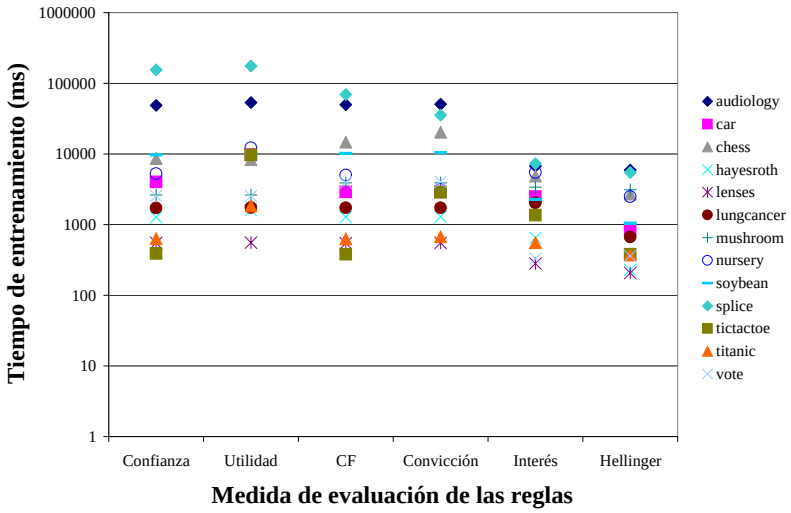

Figura 4.13: Tiempo de entrenamiento para distintas medidas de evaluación de las reglas.

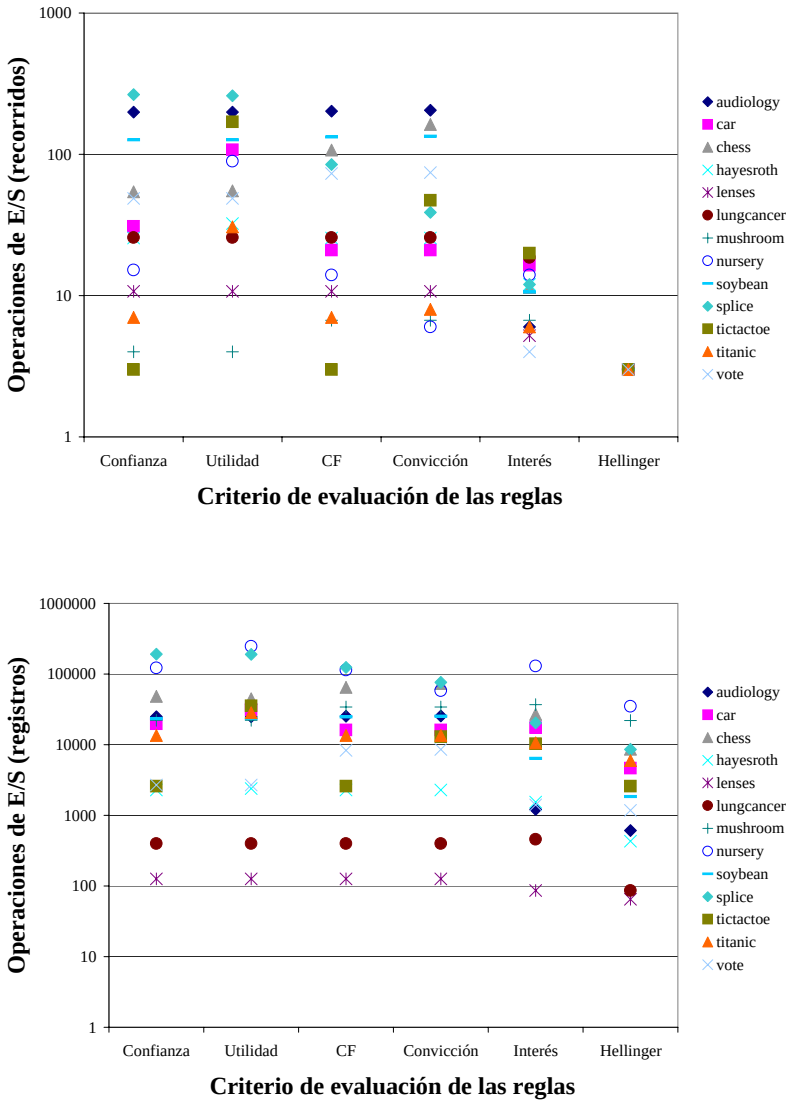

Figura 4.14: Número de operaciones de E/S para distintas medidas de evaluación de las reglas.

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.