Capítulo 2

Propedéutica

Propedéutica:
Enseñanza preparatoria para el estudio de una disciplina.

Diccionario de la Real Academia Española de la Lengua

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

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

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

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

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

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

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

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

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

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

2.1. Árboles de decisión

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

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

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

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

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

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

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

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

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

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

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

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

2.1.1. Reglas de división

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

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

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

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

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

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

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

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

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

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

2.1.1.1. Ganancia de información: Entropía

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

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

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

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

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

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

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

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

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

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

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

2.1.1.2. El criterio de proporción de ganancia

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

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

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

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

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

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

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

2.1.1.3. El índice de diversidad de Gini

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

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

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

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

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

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

2.1.1.4. Otros criterios

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

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

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

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

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

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

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

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

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

- MAXDIF

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

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

- Índice Generalizado de Gini

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

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

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

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

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

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

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

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

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

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

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

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

2.1.2. Reglas de parada

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

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

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

- *Pureza del nodo*

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

- *Cota de profundidad*

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

- *Umbral de soporte*

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

2.1.3. Reglas de poda

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

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

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

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

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

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

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

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

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

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

2.1.3.1. Poda por coste-complejidad

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

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

de T para un parámetro de complejidad α es

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

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

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

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

2.1.3.2. Poda pesimista

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

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

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

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

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

2.1.4. Algoritmos TDIDT

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

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

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

conjunto de árboles construidos.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Tabla 2.1: Algoritmos de construcción de árboles de decisión

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

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

2.1.5. Paso de árboles a reglas

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

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

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

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

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

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

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

2.2. Inducción de reglas y listas de decisión

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

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

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

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

2.2.1. Metodología STAR

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Tabla 2.3: Algunos algoritmos de inducción de reglas

2.2.2. Listas de decisión

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Tabla 2.4: Algunos algoritmos de inducción de listas de decisión

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

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

2.2.3. Algoritmos genéticos

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3. Reglas de asociación

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

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

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

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

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

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

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

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

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

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

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

2.3.1. Algoritmos de extracción de reglas de asociación

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

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

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

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

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

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

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

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

Tabla 2.5: Algunos algoritmos de extracción de reglas de asociación

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

Tabla 2.6: Algoritmos paralelos de extracción de reglas de asociación

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

Tabla 2.7: Otros algoritmos de extracción de reglas de asociación

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.2. Construcción de clasificadores con reglas de asociación

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

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

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

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

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

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

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

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

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

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

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

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

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

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