Capítulo 6

Árboles de decisión
[*Decision Trees*]

1. Introducción

"Knowledge Acquisition is the transfer and transformation of problem-solving expertise from some knowledge source to a program"

Buchanan et al. 1983

Los árboles de decisión, también denominados árboles de clasificación o de identificación, sirven, como su propio nombre indica, para resolver problemas de clasificación. La construcción de árboles de decisión es el método de aprendizaje inductivo supervisado más utilizado. Como forma de representación del conocimiento, los árboles de decisión destacan por su sencillez. A pesar de que carecen de la expresividad de las redes semánticas o de la lógica de primer orden, su dominio de aplicación no está restringido a un ámbito concreto sino que pueden ser utilizados en diversas áreas (desde aplicaciones de diagnóstico médico hasta juegos como el ajedrez o sistemas de predicción meteorológica).

La construcción automática de árboles de clasificación puede servir de alternativa a los métodos manuales de extracción de conocimiento. Conseguir que un experto sobre un tema concreto explique cómo resuelve los problemas a los que se enfrenta es un trabajo arduo. Además, cuanto mejor es el experto peor suele describir su conocimiento (la paradoja de la Ingeniería del Conocimiento [Waterman, 1986]). Por si todo esto fuese poco, los expertos en un tema no siempre están de acuerdo (*Ley de Hiram*: "Si se consultan suficientes expertos, se puede confirmar cualquier opinión").

Para que el aprendizaje inductivo (como proceso de generalización a partir de ejemplos concretos) sea correcto hemos de disponer de numerosos ejemplos. Si las conclusiones obtenidas no están avaladas por muchos ejemplos, entonces la aparición de errores en los datos (algo que es más común de lo deseable) podría conducir al aprendizaje de un modelo erróneo.

Un árbol de decisión es una forma de representar el conocimiento obtenido en el proceso de aprendizaje inductivo. Cada nodo interior del árbol contiene una pregunta sobre un atributo concreto (con un hijo por cada posible respuesta) y cada hoja se refiere a una decisión (una clasificación).

Un árbol de este tipo puede usarse para clasificar un caso comenzando desde su raíz y siguiendo el camino determinado por las respuestas a las preguntas de los nodos internos hasta que encontremos una hoja del árbol. Funciona como una aguja de ferrocarril: cada caso es dirigido hacia una rama u otra de acuerdo con los valores de sus atributos al igual que los trenes cambian de vía según su destino.

Los árboles de clasificación podrán ser útil siempre que los ejemplos a partir de los que se desea aprender se puedan representar mediante un conjunto prefijado de atributos y valores (discretos o numéricos). Sin embargo, no son de gran utilidad cuando la estructura de los ejemplos es variable ni, obviamente, para la predicción de valores continuos.

Una característica interesante de los árboles de decisión es la facilidad con la que se pueden derivar reglas de producción a partir de ellos. Esto es importante para facilitar la comprensión del modelo de clasificación construido cuando el árbol de decisión es complejo. El algoritmo para la obtención de las reglas de producción derivadas del árbol es trivial: de cada camino desde la raíz del árbol hasta un nodo hoja se deriva una regla cuyo antecedente es la conjunción de los valores de los atributos de los nodos internos y cuyo consecuente es la decisión a la que hace referencia la hoja del árbol (la clasificación realizada). Posteriormente, el conjunto de reglas derivado del árbol de decisión puede mejorarse generalizando aquéllas reglas que incluyan condiciones irrelevantes para la clasificación en su antecedente.

Como alternativa a los sistemas clásicos basados en árboles de decisión, 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 (construyendo lo que algunos denominan, incorrectamente, multi-árbol [*multi-tree*]).

RESOLUCIÓN DE UN PROBLEMA DE CLASIFICACIÓN
UTILIZANDO ÁRBOLES DE DECISIÓN

---

1. *Seleccionar el conjunto de datos de entrenamiento sobre el que se aplicará el algoritmo de construcción del árbol de decisión.*

2. *Seleccionar el atributo objetivo, el atributo por el cual se clasificarán los casos de entrenamiento seleccionados.*

3. *Descartar a priori los atributos irrelevantes para la clasificación.*

4. *Construir recursivamente el árbol de decisión:*

   a) *Si todos los casos de entrenamiento corresponden a objetos de una misma clase hemos logrado una buena clasificación. Se ha alcanzado una hoja del árbol de decisión.*

   b) *Si no encontramos un atributo por el que poder ramificar o se cumple alguna condición de parada (**regla de parada**), no se sigue expandiendo el árbol por la rama actual.*

   c) *Usando la tabla de casos de entrenamiento se emplea alguna heurística (**regla de división**) para seleccionar un atributo por el que ramificar. Para cada valor permitido de ese atributo obtenemos el subconjunto de casos en los que el atributo toma dicho valor y se genera el subárbol correspondiente recursivamente.*

5. *Poda a posteriori del árbol de decisión obtenido (según alguna **regla de poda**)*

6. *Generación de reglas a partir del árbol de decisión*

---

2. Construcción de árboles de clasificación

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 (se sigue la navaja o principio de economía de Occam: "los entes no han de multiplicarse innecesariamente").

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). 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 del orden *O(NlogN)*, siendo *N* el número de ejemplos de entrada.

El método de construcción de árboles de decisión mediante particionamiento recursivo del conjunto de casos de entrenamiento tiene su origen en el trabajo de Hunt a finales de los años 50. El algoritmo "divide y vencerás" [*divide&conquer*] es simple y elegante:

- ✓ Si existen uno o más casos en el conjunto de entrenamiento y todos corresponden a una misma clase C, el árbol de decisión es una hoja que identifica a la clase C.

- ✓ Si el conjunto de casos de entrenamiento queda vacío, también nos encontramos en una hoja del árbol. Sin embargo, 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 (instancias de una misma clase). Se elige una pregunta basada en el valor de un atributo que tenga dos o más respuestas alternativas mutuamente exclusivas R_i. El árbol de decisión 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 usa recursivamente para construir los subárboles correspondientes a los hijos del nodo. A cada hijo H_i se le asigna el subconjunto de casos de entrenamiento correspondientes a la alternativa R_i.

En esta forma de construir los árboles de decisión de forma descendente y recursiva se encuentra el origen del acrónimo TDIDT [*Top-Down Induction On Decision Trees*], que se utiliza para referirse a la familia completa de algoritmos de construcción de árboles de decisión.

Cuando se construye un nodo se considera el subconjunto de casos de entrenamiento que pertenecen a cada clase(*estadísticas del nodo*). 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 un atributo (usando una ***regla de división***), se divide el conjunto de entrenamiento en subconjuntos (mutuamente excluyentes) y se aplica el mismo procedimiento a cada subconjunto. A veces se podará el árbol obtenido: proceso de *post-poda* siguiendo alguna ***regla de poda***.

2.1 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.

Cada posible pregunta ha de evaluarse mediante alguna heurística y, dado que los algoritmos suelen ser greedy, ésta desempeña un papel esencial en la construcción del árbol (una vez que se ha escogido una pregunta para un nodo no se vuelven a considerar alternativas). Las heurísticas estadísticas usadas intentan favorecer las divisiones que mejor discriminan unas clases de otras. Ejemplos muy conocidos de estas heurísticas son la ganancia de información (ID3) o el índice de diversidad de Gini (CART).

Los criterios de división o ramificación utilizados generalmente están basados en medidas de la impureza de un nodo. 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 es una función φ definida sobre el conjunto de las J-uplas $(p_1,p_2,..,p_J)$ donde cada p_i indica la probabilidad de que un caso recogido por un nodo del árbol sea de la clase i. Como es lógico, $\Sigma p_j=1$. La función φ ha de poseer las siguientes propiedades:

- φ tiene un único máximo en $(1/J,1/J,..,1/J)$
- φ alcanza su mínimo en $(1,0,..,0)$, $(0,1,..,0)$ ... $(0,0,..,1)$ y el valor de su mínimo es 0
- φ es una función simétrica de $p_1, p_2, ..., p_J$.

La impureza de un árbol de decisión se obtiene a partir de la impureza de sus hojas o nodos terminales de la siguiente forma:

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

donde $p(t)$ es la probabilidad de que un caso corresponda al nodo terminal t y $\varphi(t)$ es la impureza de dicho nodo terminal.

2.1.1 LA GANANCIA DE INFORMACIÓN

Se intenta maximizar la ganancia de información obtenida al ramificar el árbol por un atributo minimizando la función *I*:

$$I(A_i) = \sum_{j=1}^{M_i} P(A_{i,j}) H(C|A_{i,j})$$

$$H(C|A_{i,j}) = -\sum_{k=1}^{N} P(C_k|A_{i,j}) \log_2 P(C_k|A_{i,j})$$

donde A_i es el atributo por el que se ramifica, $P(A_{i,j})$ es la probabilidad de que el atributo A_i tome su valor *j* y $H(C|A_{i,j})$ es la entropía de clasificación del conjunto de casos en los que el atributo A_i toma su valor *j*.

La información transmitida en un mensaje depende de su probabilidad *P* y puede medirse en bits como $-log_2(P)$. Por ejemplo, si tenemos 256 posibles mensajes, la información transmitida por uno de ellos es de 8 bits. Cuando el logaritmo es neperiano la unidad de información se denomina *nat* y cuando el logaritmo es decimal, *hartley*.

La probabilidad de que un caso escogido al azar pertenezca a una clase C_k es $P(C_k)$ y la información que se obtiene es $-log_2(P(C_k))$. La información que esperamos obtener al clasificar un caso cualquiera del conjunto de casos de entrenamiento *S* será igual a $-\Sigma P(C_k) \cdot log_2(P(C_k))$, cantidad a la que se denomina entropía del conjunto.

La información necesaria para transmitir la división del conjunto *T* de casos de entrenamiento en M_i subconjuntos T_j será igual a $\Sigma P(T_j) \cdot H(T_j)$, donde $P(T_j)$ es la probabilidad de que un caso pertenezca a 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 será por lo tanto igual a $H(T)-\Sigma P(T_j) \cdot H(T_j)$, siendo $H(T)$ la entropía de *T*. Al comparar posibles particiones del conjunto *T* se evalúa la ganancia de información obtenida por cada una de ellas. Como H(T) es constante, nos basta con comparar $\Sigma P(T_j) \cdot H(T_j)$, que se corresponde con la expresión de arriba.

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

2.1.2 EL CRITERIO DE PROPORCIÓN DE GANANCIA

Aunque usando la ganancia de información se obtienen buenos resultados al construir árboles de decisión (vg: *ID3*), 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. Se ha obtenido la máxima ganancia de información posible pero el árbol de decisión construido no sirve de nada.

Para normalizar de algún modo la ganancia obtenida podemos seguir usando resultados obtenidos en Teoría de la Información. El contenido de un mensaje que nos indique la respuesta a la pregunta realizada (no la clase a la que pertenece cada caso) será igual a $-\Sigma P(A_{i,j}) \cdot log_2(A_{i,j})$. Con la ayuda de este valor podemos redefinir nuestra función de evaluación:

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

Cuando la división realizada del conjunto de casos de entrenamiento es trivial, el denominador de *R* es cercano a cero. Se ha de escoger el atributo que maximice el cociente R tal que su ganancia sea, 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 haya, el criterio de proporción de ganancia evitará la construcción de árboles de decisión que clasifiquen los casos utilizando su identificador.

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).

2.1.3 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 por un atributo. La función empleada es la siguiente:

$$G(A_i) = \sum_{j=1}^{M_i} P(A_{i,j}) G(C|A_{i,j})$$

$$G(C|A_{i,j}) = \sum_{k=1}^{N} P(C_k|A_{i,j}) P(\neg C_k|A_{i,j}) = 1 - \sum_{k=1}^{N} P^2(C_k|A_{i,j})$$

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 $P(C_k|A_{i,j})$ por el factor $P(\neg C_k|A_{i,j})$, que es igual a $1-P(C_k|A_{i,j})$.

El índice de Gini es una medida de la diversidad de clases en un nodo del árbol. 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.

2.1.4 MAX (© J.C. Cubero, 1998)

"Que sea sencillo, lo más sencillo posible, pero no más"
Albert Einstein

La minimización de la entropía (equivalente a la maximización de la ganancia de información) utilizada por Quinlan en ID3 trata de penalizar aquellas divisiones del conjunto de entrenamiento muy desordenadas.

Como nuestro objetivo es la clasificación, una medida de la bondad de un conjunto dado de casos de entrenamiento es la probabilidad de la clase más común (de hecho este es el término que aumenta menos la entropía del conjunto). En este caso, el objetivo perseguido será la maximización de esta medida en los conjuntos generados al elegir un atributo para ramificar por él. La maximización de la función $K(A_i)$:

$$K(A_i) = \sum_{j=1}^{M_i} P(A_{i,j}) K(C|A_{i,j})$$

$$K(C|A_{i,j}) = \max_k P(C_k|A_{i,j})$$

El problema de maximización anterior se puede expresar como un problema de minimización utilizando la diferencia entre casos bien clasificados y casos mal clasificados como medida de la idoneidad de una ramificación del árbol.

$$D(A_i) = \sum_{j=1}^{M_i} P(A_{i,j}) D(C|A_{i,j})$$

$$D(C|A_{i,j}) = \max_k \left\{ P(C_k|A_{i,j}) - P(\neg C_k|A_{i,j}) \right\}$$

Esto es completamente equivalente a la minimización del número de casos para los que se realiza una clasificación incorrecta (considerando la clase más común la clasificación correcta de un conjunto dado de casos), ya que $P(X)-P(\neg X)=2P(X)-1$ al ser $P(\neg X)=1-P(X)$.

$$E(A_i) = \sum_{j=1}^{M_i} P(A_{i,j}) E(C|A_{i,j})$$

$$E(C|A_{i,j}) = \min_k \left\{ P(\neg C_k|A_{i,j}) \right\}$$

Para evitar que esta medida favorezca la construcción de árboles de decisión en cuya raíz se utilice una clave primaria de la relación (como sucede con la entropía en ID3), se puede redefinir la función K de forma que no se tengan en cuenta pequeñas contribuciones debidas a muchos valores diferentes de un atributo:

$$K(A_i) = \sum_{j \in U} P(A_{i,j}) K(C|A_{i,j})$$

$$U = \{j | Max\{n(C_k|A_{i,j})\} \geq S\}$$

$$K(C|A_{i,j}) = \underset{k}{Max}\ P(C_k|A_{i,j})$$

donde S es un umbral establecido (equivalente a la relevancia mínima exigida a los itemsets en la generación de reglas de asociación) y $n(C_k|A_{i,j})$ es el número de casos correspondientes a la clase C_k tales que el atributo A_i toma su valor j.

Aunque parezca más compleja la expresión, su cálculo es casi directo. Teniendo en cuenta que $P(C_k|A_{i,j})=n(C_k|A_{i,j})/n(A_{i,j})$ y $P(A_{i,j})=n(A_{i,j})/N$, donde N es el número total de casos y $n(A_{ij})$ es el número de casos que toman el valor j del atributo A_i, la función K es igual a:

$$K(A_i) = \frac{1}{N} \sum_{j \in U} K(C|A_{i,j})$$

$$U = \{j | Max\{n(C_k|A_{i,j})\} \geq S\}$$

$$K(C|A_{i,j}) = \underset{k}{Max}\, n(C_k|A_{i,j})$$

El código correspondiente a la evaluación de la función K sería más o menos así:

```
for (j=0; j<MaxJ; j++) {

    max = 0;

    for (k=0; k<MaxK; k++)
        if (info.clase[k][i][j]>max)
           max = n[k][i][j];

    if (max>=S)
       K += max;
}

K /= N;
```

Si no tenemos información incompleta, el factor *1/N* será igual para todos los atributos y podremos eliminarlo, con lo que la función de evaluación será una simple suma de valores. Cuando desconozcamos los valores de algunos atributos, el valor *N* será igual al número de casos para los que esté definido el atributo A_i.

Sin embargo, se observa que utilizando esta función heurística favorecemos en exceso los árboles con un grado de ramificación mínimo, incluso cuando el atributo por el que se ramifica es irrelevante para la clasificación (hemos pasado de un extremo a otro). Esto se podría solventar con facilidad modificando la función heurística de la siguiente forma:

$$K'(A_i) = \#U \cdot \sum_U P(A_{i,j}) K(C|A_{i,j})$$

$$U = \{j | Max\{n(C_k|A_{i,j})\} \geq S\}$$

$$K(C|A_{i,j}) = \max_k P(C_k|A_{i,j})$$

Teniendo en cuenta que $P(C_k|A_{i,j})=n(C_k|A_{i,j})/n(A_{i,j})$ y $P(A_{i,j})=n(A_{i,j})/N$, donde N es el número total de casos y $n(X)$ es el número de casos que verifican X, la función K podemos expresarla como:

$$K'(A_i) = \frac{\#U}{N} \sum_{j \in U} K(C|A_{i,j})$$

$$U = \{j | Max\{n(C_k|A_{i,j})\} \geq S\}$$

$$K(C|A_{i,j}) = \max_k n(C_k|A_{i,j})$$

Intuitivamente, la sumatoria de la expresión de arriba estima cuántos casos se clasifican correctamente de cuantos se encuentran en el conjunto de entrenamiento (al dividir por N obtenemos la proporción de casos supuestamente bien clasificados). El factor $\#U$ se utiliza para favorecer la ramificación del árbol por atributos que consiguen un árbol más plano (más ramas interesantes). Utilizando el umbral S se evita seleccionar atributos que sean claves primarias.

La utilización de $\#U$ carece de una base teórica. Es más, aunque en algunos ejemplos funciona bien (llega a conseguir mejores resultados que el criterio de proporción de ganancia), no garantiza la obtención de buenos árboles de decisión.

2.1.5 MAXDIF

Retomemos de *MAX* la expresión correspondiente a la función *D*, que obtenía resultados equivalentes a los obtenidos en la formulación inicial de la función *K*. La definición inicial de D era la siguiente:

$$D(A_i) = \sum_{j=1}^{M_i} P(A_{i,j}) D(C|A_{i,j})$$

$$D(C|A_{i,j}) = \underset{k}{Max}\left\{ P(C_k|A_{i,j}) - P(\neg C_k|A_{i,j}) \right\}$$

Utilizando esta definición, se consigue una buena regla de división para la construcción de árboles de decisión si incorporamos la idea del umbral mínimo de soporte S a la función D. Se obtiene una buena heurística sin tener que introducir artificialmente el factor #U:

$$D(A_i) = \sum_{j \in U} P(A_{i,j}) D(C|A_{i,j})$$

$$U = \{ j | Max\{ n(C_k|A_{i,j}) - n(\neg C_k|A_{i,j}) \} \geq S \}$$

$$D(C|A_{i,j}) = \underset{k}{Max}\left\{ P(C_k|A_{i,j}) - P(\neg C_k|A_{i,j}) \right\}$$

A esta regla heurística de división la denominaremos *MAXDIF*. Utilizando la relación existente entre la probabilidad *p(x)* y la frecuencia de aparición *n(x) = N p(x)*, obtenemos:

$$D(A_i) = \frac{1}{N} \sum_{j \in U} D(C|A_{i,j})$$

$$U = \{ j | Max\{ n(C_k|A_{i,j}) - n(\neg C_k|A_{i,j}) \} \geq S \}$$

$$D(C|A_{i,j}) = \underset{k}{Max}\left\{ n(C_k|A_{i,j}) - n(\neg C_k|A_{i,j}) \right\}$$

Obsérvese el parecido de esta regla heurística con el índice de diversidad de Gini: la sumatoria se ha sustituido por el máximo y el producto por una diferencia. La similaridad es notable y puede que no sea casual.

2.2 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 probabilidades (según los casos que recoja) o simplemente la clase más común de las recogidas por los casos. Sorprendentemente, se ha demostrado empíricamente que esta última técnica es mejor a la hora de minimizar el error de clasificación.

Las reglas de parada, denominadas originalmente reglas de *pre-poda*, tratan de predecir si merece la pena seguir construyendo el árbol o no. Ejemplos de este tipo de reglas son:

✓ *Pureza del nodo*

Cuando un nodo solamente contiene ejemplos de una clase, obviamente, el proceso de construcción del árbol de decisión ha finalizado. Además, podría 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. Se suele optar por construir el árbol de decisión completo y realizar una poda a posteriori.

✓ *Cota de profundidad*

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.

✓ *Mínimo de casos*

Cuando nos encontramos un nodo con menos de X ejemplos detenemos el proceso de obtención del árbol. Una clasificación avalada por menos de X casos de entrenamiento no se considera fiable (menos de X ejemplos son insuficientes para estimar probabilidades con una precisión aceptable).

2.3 Reglas de poda

Una vez construido completamente el árbol de decisión, las reglas de poda (post-poda para ser precisos) intentan eliminar los subárboles que no contribuyen significativamente a la precisión de la clasificación.

De hecho, 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 aplicar más tests. El resultado suele ser un árbol muy complejo, más de lo deseable, que "sobreajusta" los datos del conjunto de entrenamiento [efecto conocido por el término inglés *overfitting*]. El sobreaprendizaje es un problema bastante importante ya que limita considerablemente la aplicabilidad del modelo de clasificación aprendido.

Ejemplo

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: $P(1-P)$.
- ✓ La probabilidad de que un caso de Y se asigne a la clase X: $(1-P)P$.

El error estimado será igual a $2P(1-P)$, error mayor que $1-P$ si $P>0.5$. Por lo tanto, el clasificador más sencillo posible es el mejor cuando la clase y los atributos de los casos son estadísticamente independiente. En casos reales esto 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.

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*) es muy difícil. La poda ha de realizarse en función de algún estimador honesto (no sesgado) del error de clasificación del árbol de decisión.

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

A continuación se comentarán algunos de los métodos de poda de árboles de decisión más comunes: la poda por estimación del error, la poda por coste-complejidad y la poda pesimista.

2.3.1 PODA POR ESTIMACIÓN DEL ERROR [reduced-error prunning]

Un nodo se poda si el error de resustitución [*resubstitution error*] del nodo considerado como hoja es menor que el error de resustitución del subárbol cuya raíz es el nodo. El método requiere reservar un conjunto de casos para la poda (por lo cual no se podrán utilizar todos los casos disponibles para construir el árbol). Cuando no disponemos de muchos datos, se puede utilizar algún tipo de validación cruzada [*cross-validation*] para obtener mejores resultados.

2.3.2 PODA POR COSTE-COMPLEJIDAD [cost-complexity prunning]

Esta técnica de poda, usada 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 de mínimo error, aquél que minimice 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 poda por coste-complejidad se puede realizar utilizando un conjunto de prueba independiente del conjunto de entrenamiento o validación cruzada [*CV: Cross-Validation*].

❖ *Poda por coste-complejidad con un conjunto de prueba:*

❶ Dividir el conjunto de casos en dos subconjuntos (entrenamiento y prueba).

❷ Construir el árbol con el conjunto de entrenamiento.

❸ Encontrar el subárbol de mínimo error para el conjunto de prueba y calcular su error de sustitución R_0 y el error estándar del estimador SE_0: *árbol 0-SE*.

❹ El subárbol podado será el subárbol que minimiza R_α con el máximo valor de α tal que el estimador del error en el conjunto de prueba sea menor que R_0+SE_0: *árbol 1-SE*.

❖ *Poda por coste-complejidad con validación cruzada:*

❶ Escoger v subconjuntos disjuntos.

❷ Encontrar el valor de α que minimiza el estimador del error por validación cruzada para el subárbol que minimiza R_α (obtener R_0 y SE_0).

❸ Encontrar el máximo nivel de α tal que estimador del error por validación cruzada para el subárbol que minimiza R_α sea menor que R_0+SE_0.

❹ El árbol podado es el subárbol construido sobre el conjunto completo de ejemplos que minimiza R_α. Si ignoramos el tercer paso obtenemos el *árbol 0-SE*, si no tendremos el *árbol 1-SE*.

2.3.3 PODA PESIMISTA [pessimistic prunning, Quinlan]

Esta técnica 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 los clasifica incorrectamente, su error de resustitución es E/N. El estimador del error de resustitución asociado a un subárbol será la suma de los errores estimados para cada una de sus ramas.

La probabilidad real del error cometido no se puede determinar con exactitud pero se puede establecer un intervalo de confianza. Dado un grado de confianza CF, se puede establecer una estimación de la probabilidad del error $U_{CF}(E,N)$ usando una distribución binomial.

Se poda el 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 una hoja. De esta forma se eliminan los subárboles que no mejoran significativamente la precisión del clasificador. El método es cuestionable como cualquier heurística pero suele producir resultados aceptables.

2.4 Tests considerados

Todos los sistemas de construcción automática de clasificadores definen un mecanismo para evaluar la idoneidad de cada test propuesto (p.ej. la regla de división en la construcción de árboles de decisión). Esto implica que se deben generar de alguna forma los distintos tests para que puedan ser evaluados.

Generalmente, se define un formato y se examinan todos los posibles tests de ese formato. Además, es habitual que el test empleado involucre a un único atributo (para facilitar su comprensibilidad y simplificar el proceso de búsqueda evitando una explosión combinatoria).

Por ejemplo, C4.5 utiliza tres formatos diferentes de tests:

- ✓ El típico test sobre atributos discretos, con una rama del árbol para cada posible valor del atributo discreto considerado.
- ✓ Un test más complejo sobre atributos discretos en el que se agrupan los valores del atributo en subgrupos.
- ✓ Uno binario de la forma $atributo \leq valor$ para atributos numéricos.

Por su parte, CART sólo utiliza expresiones lógicas de forma que el árbol resultante siempre es binario. Los atributos de tipo numérico son tratados igual que en C4.5, sin embargo los tests aplicados sobre atributos discretos (de tipo categórico) son siempre del tipo $¿X \in \{v_1..v_n\}?$. Obviamente, la aparición de este tipo de preguntas sobre un atributo en el árbol de decisión dificulta la generación de reglas *IF-THEN* a partir del árbol de decisión.

Otra posibilidad consiste en evaluar el atributo discreto construyendo un subárbol de decisión para aquellos valores avalados por un número suficiente de casos y enviando todos los demás valores a un subárbol común (una rama de tipo *else*). La construcción de árboles de decisión con ramas de tipo *else* es una aportación de este proyecto a la familia TDIDT de algoritmos de construcción de árboles de decisión. Esta técnica puede utilizarse en algoritmos como C4.5 para reducir en algunos casos el tamaño del árbol de decisión, si bien es verdad que suele incrementar la dificultad de comprensión del árbol de decisión por parte de un experto.

2.5 Información incompleta

Clasificar un caso utilizando un árbol de decisión requiere, en principio, que se conozcan todas sus características (sus atributos) para poder elegir la rama del árbol correcta en cada nodo pregunta. Por desgracia, los datos recogidos en la vida real suelen ser incompletos (ya sea porque el valor de un atributo era desconocido, se consideró irrelevante, no se mecanizó o simplemente el atributo no era aplicable al caso concreto).

Hay que elegir entre descartar todos aquellos casos con información incompleta o adaptar adecuadamente los algoritmos de clasificación para poder tratar con ellos. La primera alternativa no es aceptable normalmente, por lo que hemos de abordar el problema del manejo de información incompleta: modificar el algoritmo de construcción del árbol de decisión y establecer un mecanismo para clasificar casos con información incompleta

El problema se puede resolver rellenando atributos desconocidos con valores por defecto (p.ej. el valor más común del atributo), construyendo árboles de decisión para determinar el valor del atributo desconocido (Shapiro), teniendo árboles de clasificación auxiliares (*surrogate split [CART]*), utilizando la distribución de probabilidad de los valores de los atributos (*ASSISTANT*) o la teoría de Dempster-Shafer.

EJEMPLO: Manejo de valores desconocidos en C4.5

La modificación de la regla de división utilizada en *C4.5* (el criterio de proporción de ganancia) para tratar información incompleta es bastante sencilla. Sólo se tienen en cuenta los atributos de los casos cuyos valores son conocidos:

$$Ganancia = P(A_{conocido}) \times (H(T) - \sum_{i=1}^{N} P(T_i)H(T_i)) + P(A_{desconocido}) \times 0$$

$$Ratio = \frac{Ganancia}{\sum_{i=0}^{N} P(T_i)\log_2 P(T_i)}$$

donde T_0 es el conjunto de casos de entrenamiento cuyo valor de A es desconocido.

C4.5 utiliza la misma técnica probabilística que *ASSISTANT* a la hora de dividir el conjunto de casos de entrenamiento durante la construcción del árbol de decisión. Cuando el valor del atributo A_i de un caso del conjunto T es $A_{i,j}$, se le asigna al conjunto T_j con probabilidad 1. Cuando el valor del atributo A_i es desconocido, se le asigna a un conjunto T_j con la probabilidad de que un caso de T pertenezca a Tj.

Algo parecido se hace al clasificar un caso. El resultado de la clasificación será una distribución de la probabilidad y el caso será asignado a la clase con mayor probabilidad.

2.6 Algoritmos estándar de construcción de árboles de decisión

2.6.1 CLS [*Concept Learning System*]

El sistema *CLS,* descrito por Hunt, Martin y Stone en *"Experiments in Induction"* (1966), es el origen de gran familia *TDIDT* [*Top-Down Induction of Decision Trees*] de algoritmos para la construcción de árboles de decisión.

Los tests encontrados en los árboles de decisión construidos con *CLS* son siempre de la forma *¿atributo=valor?* que, obviamente, sólo tienen dos posibles respuestas (sí y no).

CLS intenta minimizar el coste de clasificación de un objeto considerando el coste de establecer el valor de una propiedad del objeto y el coste debido a una mala clasificación del objeto (el coste debido a la decisión de que un objeto pertenece a una clase determinada cuando en realidad es una instancia de otra clase distinta).

CLS utiliza una técnica similar al minimax. Explora el espacio de posibles árboles de decisión (con una cota de profundidad) y elige la acción que minimice la función de costo en el conjunto de árboles construidos. Como es evidente, CLS requiere gran capacidad de cómputo y no siempre encuentra buenos árboles de decisión

2.6.2 ID3 [Iterative Dichotomizer 3]

Este algoritmo greedy de Quinlan (1979) es el método más famoso de todos los que existen para la creación de árboles de decisión. Usa una poda pesimista y utiliza el criterio de proporción de ganancia. Extensiones de ID3 le permiten tratar con datos erróneos (ruido) e información incompleta.

Para que el árbol de decisión generado sea lo más sencillo posible, este algoritmo evalúa la capacidad de discriminación de cada uno de los atributos mediante el cálculo de las entropías de los distintos atributos en los casos de entrenamiento empleados. La entropía nos da una idea de la desorganización de la información, es decir, nos indica la capacidad de discriminación de cada atributo. Se trata de minimizar la función definida por la siguiente ecuación:

$$H(C|A_i)=\sum_{j=1}^{M_i} p(A_{i,j})\cdot[-\sum_{k=1}^{K} P(C_k|A_{i,j})\cdot\log_2 P(c_k|A_{i,j})]$$

- $H(C|A_i)$ es la entropía de clasificación de C utilizando el atributo A_i.
- $P(A_{i,j})$ es la probabilidad del atributo k para su valor j.
- $P(C_k|A_{i,j})$ es la probabilidad del valor i de la clase sea c cuando el atributo k toma su valor j.
- M_i es el número total de valores permitidos para el atributo A_i.
- K es el número total de clases.

ID3 utiliza un método iterativo para construir árboles de decisión y prefiere los árboles sencillos frente a los más complejos (ya que, en principio, aquéllos que tienen sus caminos hasta las hojas más cortos son más útiles a la hora de clasificar entradas). 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.

2.6.3 ACLS [*Analogue Concept Learning System*]

ACLS (Patterson y Niblett, 1983) es una generalización de ID3 que permite tratar propiedades con valores enteros y ha sido aplicado con éxito en problemas complejos (vg: tareas de reconocimiento de imágenes). *Expert-Ease* (1983), *EX-TRAN* (1984) y *RuleMaster* (1984) son productos comerciales derivados de *ACLS*.

2.6.4 ASSISTANT

Otro derivado de ID3. Propuesto por Kononenko, Bratko y Roskar en 1984, permite atributos con valores continuos (reales). Las clases no han de ser disjuntas, aunque tienen que formar una jerarquía. Para tratar valores desconocidos utiliza un enfoque bayesiano. Puede manejar información con ruido, ya que emplea una heurística para detener la construcción del árbol cuando se estima que seguir ramificando no mejora la precisión de la clasificación.

ASSISTANT se caracteriza porque todos los tests que se realizan sobre los atributos son binarios. El conjunto de valores de cada atributo se divide en dos subconjuntos disjuntos para evitar la propensión del criterio de ganancia de información a ramificar por atributos con muchos valores diferentes. Para cada atributo han de comprobarse $2^{v-1}-1$ posibles tests (siendo v el número de valores diferentes del atributo), lo que es inviable para valores no demasiado grandes de v.

2.6.5 CART (Classification And Regression Trees)

Diseñado por Breiman y sus colaboradores en 1984, usa el índice de diversidad de Gini y puede agrupar las clases en dos superclases ("twoing criterion", muy ineficiente). Permite realizar una poda costo-complejidad con validación cruzada sobre 10 conjuntos (10-CV). Puede obtener tanto el árbol 0-SE como el 1-SE.

2.6.6 C4

Un descendiente más de ID3 que permite atributos continuos, sobre los que se aplican tests de la forma $atributo \leq valor$. Ideado por Quinlan (1987), utiliza la poda pesimista.

2.6.7 ID4 & ID5

Descendientes de ID3 orientados al aprendizaje incremental desarrollados por Thrun y sus colaboradores en 1991.

2.6.8 C4.5

Híbrido entre CART y C4. Construido también por Quinlan (1993). Permite usar como regla de división la ganancia de información, el índice de diversidad de Gini o el criterio de proporción de ganancia. Se puede realizar una post-poda pesimista del árbol (sustituyendo un subárbol por una hoja o por una de sus ramas).

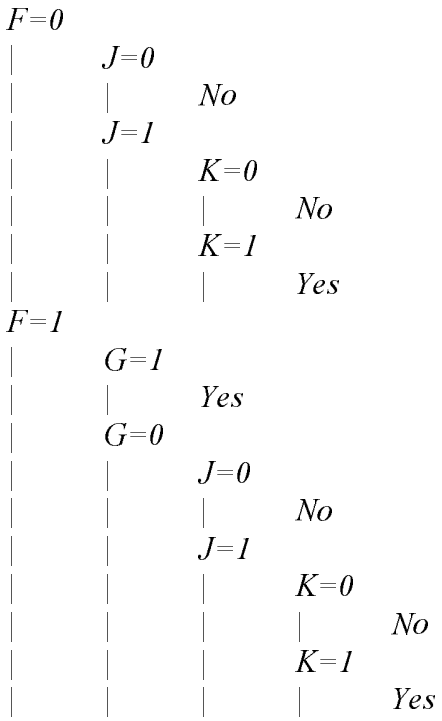
2.6.9 SLIQ & SPRINT

SLIQ [*Supervised Learning in Quest*] es un algoritmo de construcción de árboles de decisión que maneja tanto atributos categóricos (discretos) como atributos numéricos (continuos) y destaca por su escalabilidad. Forma parte de los trabajos realizados en el proyecto Quest de Data Mining del IBM Almaden Research Center. SLIQ construye el árbol en anchura (los demás algoritmos lo hacen en profundidad) usando el índice de Gini. Utiliza sólo tests binarios de la forma $¿atributo \leq valor?$ para atributos numéricos o $¿atributo \in \{conjunto\ de\ valores\}?$ para atributos no numéricos. Finalmente, emplea un criterio de poda basado en el principio MDL [*Minimum Description Length*].

No obstante, SLIQ requiere mantener cierta información de cada tupla en memoria, lo que limita su escalabilidad. SPRINT [*Scalable PaRallelizable Induction of decision Trees*], otro algoritmo ideado por los miembros del proyecto Quest, soluciona este problema utilizando estructuras de datos diferentes a las empleadas por SLIQ.

3. Generación de reglas

Aun tras podarlos, los árboles de decisión pueden ser muy complejos y difíciles de comprender. Además, la estructura del árbol puede dividir un mismo concepto en varias ramas. Por ejemplo el árbol de decisión para *F=G=1 o J=K=1* es:



Cualquier árbol de decisión para *F=G=1 o J=K=1* dividirá *F=G=1* o *J=K=1*. Si obtenemos una regla de producción para cada camino de la raíz a una hoja del árbol, el conjunto de reglas así generado clasifica los casos tal como lo hace el árbol. Las partes *IF* de las reglas serán exhaustivas y mutuamente exclusivas. Por ejemplo:

if F=1 ∧ G=0 ∧ J=1 ∧ K=1 then Yes

3.1 Generalización de reglas

Al convertir el árbol de decisión en una colección de reglas (una por hoja del árbol), algunas de las reglas pueden contener condiciones irrelevantes en su antecedente. En la regla anterior, la conclusión no se ve afectada por los valores de F y G. La regla puede generalizarse eliminando esas condiciones superfluas:

if J=1 ∧ K=1 then Yes

Si tenemos una regla *if A then C* y una generalización suya *if B then C* donde *B* se obtiene eliminando una condición *X* perteneciente a *A*, hemos de decidir si la generalización es válida. La importancia de *X* se calcula a partir del conjunto de casos de entrenamiento usados al construir el árbol. Se crea una tabla de contingencia para los casos que satisfacen el antecedente *B*:

	Clase C	Otras clases
Satisface X	Y_1	E_1
No satisface X	Y_2	E_2

Los casos que satisfacen la condición *X* están cubiertos por la regla original, E_1 de los cuales están mal clasificados. La regla generalizada cubre además los casos que no satisfacen *X*, lo que introduce E_2 nuevos errores de clasificación.

Para decidir si una regla ha de generalizarse o no, pueden utilizarse técnicas estadísticas (como la prueba exacta de Fisher). No obstante, Quinlan prefiere utilizar la misma técnica que en la poda pesimista del árbol de decisión. Se estima el error de la regla original $U_{CF}(E_1,Y_1+E_1)$ y el de la regla generalizada $U_{CF}(E_1+E_2,Y_1+Y_2+E_1+E_2)$. Si la estimación pesimista del error de la regla generalizada no es superior a la de la regla original, se elimina la condición *X* (se generaliza).

Más de una condición podría eliminarse al generalizar una regla. En vez de mirar todos los posibles subconjuntos de condiciones susceptibles de ser eliminados, se utiliza un algoritmo greedy: mientras se pueda eliminar alguna condición de la regla, se elimina aquella que produce la regla generalizada con menor estimación de error.

Tras realizar la generalización de las reglas individuales surge un pequeño problema: las reglas dejan de ser exhaustivas y mutuamente excluyentes. Un caso podría ser cubierto por varias reglas o, si se eliminan las reglas poco fiables, por ninguna.

Para seleccionar el subconjunto de reglas más adecuado para representar cada clase Quinlan utiliza *simulated annealing*. Se trata de minimizar el número de bits necesarios para codificar el modelo de clasificación (principio MDL [*Minimum Description Length*] de Rissanen: la mejor teoría derivable de los casos de entrenamiento minimiza el número de bits requeridos para codificar el mensaje que incluye la teoría con sus excepciones).

4. Resultados

Para representar un árbol de decisión se utilizará la notación de Quinlan, en la cual las etiquetas de cada hoja del árbol de decisión van acompañadas por un número *(X)* que indica el número de casos que corresponden a cada nodo terminal del árbol. Cuando el nodo terminal no es puro (es decir, cuando le corresponden casos de distintas clases), la etiqueta va acompañada de una pareja *(X|Y)* donde *X* tiene el mismo significado que antes e *Y* indica el número de errores cometidos al asignarle a la hoja del árbol la clase más común entre los ejemplos de entrenamiento que abarca.

Para comparar las distintas reglas de división vistas utilizaremos el típico ejemplo de Quinlan que construye un árbol de clasificación para decidir si se juega o no al golf en función de las condiciones atmosféricas presentes:

OUTLOOK[1]	TEMPERATURE[2]	HUMIDITY[3]	WINDY[4]	PLAY[5]
sunny	75	70	true	yes
sunny	80	90	true	no
sunny	85	85	false	no
sunny	72	95	false	no
sunny	69	70	false	yes
overcast	72	90	true	yes
overcast	83	78	false	yes
overcast	64	65	true	yes
overcast	81	75	false	yes
rain	71	80	true	no
rain	65	70	true	no
rain	75	80	false	yes
rain	68	80	false	yes
rain	70	96	false	yes

[1] Este atributo sirve para expresar estado general del tiempo, si hace un día soleado [*sunny*], llueve [*rain*] o el cielo está encapotado [*overcast*].

[2] La temperatura medida en grados Fahrenheit (*ºF*): $T(^{\circ}C) = \frac{5}{9}\left(T(^{\circ}F) - 32\right)$

[3] La humedad relativa en tanto por ciento (%)

[4] Indica si hace o no viento

[5] El objetivo de la clasificación: saber si se va a jugar al golf o no

Considerando tanto la temperatura como la humedad atributos continuos (al estilo de *C4.5*), todas las reglas de división analizadas (entropía, criterio de proporción de ganancia, índice de Gini, *MAX* incluyendo el factor #U y *MAXDIF*) obtienen el siguiente árbol de decisión a partir de la tabla de catorce casos de arriba:

```
OUTLOOK = overcast
        PLAY = yes (4.0)
OUTLOOK = rain
        WINDY = false
                PLAY = yes (3.0)
        WINDY = true
                PLAY = no  (2.0)
OUTLOOK = sunny
        HUMIDITY <= 70
                PLAY = yes (2.0)
        HUMIDITY > 70
                PLAY = no  (3.0)
```

Intentemos complicar un poco la construcción del árbol de decisión introduciendo algún valor desconocido (representado por ?) para poder comparar las distintas heurísticas:

OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	75	70	true	yes
sunny	80	90	true	no
sunny	85	85	false	no
sunny	72	95	false	no
sunny	69	70	false	yes
?	72	90	true	yes
overcast	83	78	false	yes
overcast	64	65	true	yes
overcast	81	75	false	yes
rain	71	80	true	no
rain	65	70	true	no
rain	75	80	false	yes
rain	68	80	false	yes
rain	70	96	false	yes

Utilizando la ganancia de información (entropía), el criterio de proporción de ganancia, el índice de diversidad de Gini o *MAX* (incluyendo el factor *#U*) se obtiene el siguiente árbol:

```
OUTLOOK = overcast
        PLAY = yes (3.2)
OUTLOOK = rain
        WINDY = false
                PLAY = yes (3.0)
        WINDY = true
                PLAY = no  (2.4|0.4)
OUTLOOK = sunny
        HUMIDITY <= 70
                PLAY = yes (2.0)
        HUMIDITY > 70
                TEMPERATURE <= 72
                        WINDY = false
                                PLAY = no  (1.0)
                        WINDY = true
                                PLAY = yes (0.4)
                TEMPERATURE > 72
                        PLAY = no  (2.0)
```

Si fijamos en uno el mínimo númeto de casos que ha de cubrir cada hoja del árbol (uno de los posibles criterios de pre-poda) obtenemos directamente el árbol siguiente, idéntico al obtenido con la información completa:

```
OUTLOOK = overcast
        PLAY = yes (3.2)
OUTLOOK = rain
        WINDY = false
                PLAY = yes (3.0)
        WINDY = true
                PLAY = no  (2.4|0.4)
OUTLOOK = sunny
        HUMIDITY <= 70
                PLAY = yes (2.0)
        HUMIDITY > 70
                PLAY = no  (3.4|0.4)
```

Este árbol conserva la estructura del árbol de decisión "ideal", de hecho es el ejemplo con el que Quinlan defiende las cualidades de sus algoritmos de construcción de árboles de decisión (ID3 y *C4.5*). Nótese que las hojas etiquetadas con *(X|0.4)* se derivan de considerar los casos "*?=rain*" y "*?=sunny*".

Sin embargo, al utilizar el criterio *MAXDIF* a la hora de seleccionar por qué atributo se ha de ramificar, obtenemos un árbol de decisión bastante diferente:

```
HUMIDITY <= 80
        OUTLOOK = overcast
                PLAY = yes (3.0)
        OUTLOOK = rain
                WINDY = false
                        PLAY = yes (2.0)
                WINDY = true
                        PLAY = no  (2.0)
        OUTLOOK = sunny
                PLAY = yes (2.0)
HUMIDITY > 80
        OUTLOOK = rain
                PLAY = yes (1.25)
        OUTLOOK = sunny
                PLAY = no  (3.75|0.75)
```

Este árbol de clasificación es radicalmente distinto al obtenido con los demás métodos y merece algún comentario. Si consideramos que el mejor árbol es el más pequeño (el que contiene menos nodos internos), el método *MAXDIF* parece superior a *MAX* y a los criterios basados en la entropía o el índice de Gini (aunque es cierto que el árbol que se obtiene con esos métodos se podría podar fácilmente para obtener el árbol generado por *C4.5*).

No obstante, aunque en este caso particular el árbol de *MAXDIF* es algo más complejo, se ve más afectado por la existencia de información incompleta que el construido por *C4.5*. De hecho, no clasificaría bien el ejemplo cuyo atributo *OUTLOOK* se desconoce.

A pesar de ello, el método *MAXDIF* parece comportarse de forma interesante. Aparentemente, la heurística *MAXDIF* se acopla mejor a los datos de entrada que el criterio de proporción de ganancia (lo que bien podría ser negativo si tenemos errores en el conjunto de entrenamiento) e intenta reducir al máximo las imprecisiones cometidas en la clasificación (un arma de doble filo que se vuelve en su contra en el ejemplo expuesto).

Es destacable que sólo en una hoja aparece una posible mala clasificación (siempre según el conjunto de entrenamiento de entrada al clasificador), que en este caso podría atribuirse al sobreaprendizaje del conjunto de entrenamiento (el cual no es deseable).

Si podamos el árbol obtenido anteriormente (poda pesimista con CF=25%) con cualquiera de las otras reglas de división, el árbol de decisión se nos queda en un simple nodo: cualquier caso se clasificaría siempre como *PLAY (14|5)*, con un error estimado del 39%. No obstante, al podar el árbol obtenido con MAXDIF se obtiene un árbol con un error estimado menor (una estimación pesimista del 33% con CF=0.25, del 25% con CF=0.99):

```
HUMIDITY <= 80
        PLAY = yes (9|2)
HUMIDITY > 80
        OUTLOOK = rain
                PLAY = yes (1.25)
        OUTLOOK = sunny
                PLAY = no  (3.75|0.75)
```

Por otro lado, el conjunto de reglas que se puede obtener de ambos árboles es similar y, de hecho, clasifica de la misma forma los distintos ejemplos del conjunto de entrenamiento:

Reglas derivadas utilizando MAXDIF	Reglas derivadas con las demás reglas de división
IF HUMIDITY <= 80 THEN PLAY = yes (9.0|2.0)	IF OUTLOOK = overcast THEN PLAY = yes (3.2)
IF HUMIDITY > 80 AND OUTLOOK = rain THEN PLAY = yes (1.25)	IF OUTLOOK = rain AND WINDY = false THEN PLAY = yes (3.0)
IF HUMIDITY > 80 AND OUTLOOK = sunny THEN PLAY = no (3.75|0.75)	IF OUTLOOK = sunny AND HUMIDITY <= 70 THEN PLAY = yes (2.0)
IF HUMIDITY <= 80 AND OUTLOOK = rain AND WINDY = true THEN PLAY = no (2.0) NB: Esta última regla permite clasificar correctamente los ejemplos de PLAY=no correspondientes a HUMIDITY<=80	IF OUTLOOK = rain AND WINDY = true THEN PLAY = no (2.3|0.3) IF OUTLOOK = sunny AND HUMIDITY > 70 THEN PLAY = no (3.3|0.3)

En definitiva, la elección del criterio de división parece depender del problema, aunque empíricamente se ha comprobado que el clasificador generado no es muy sensible a esta elección. Los árboles de decisión permiten la construcción de clasificadores simples e intuitivos con un error comparable al de métodos más complejos (generalmente inferior), robustos y poco sensibles a la presencia de *outliers*.

5. Bibliografía

Michael J.A. Berry & Gordon Linoff
"Data Mining Techniques for Marketing, Sales, and Customer Support"
USA: John Wiley & Sons, 1997

Se analizan distintas técnicas empleadas en Data Mining y sus aplicaciones "comerciales". Se incluye un capítulo, puramente descriptivo, dedicado a la construcción de árboles de decisión. Sin embargo, el libro en sí no parece estar escrito para informáticos, sino más bien para gerentes y administradores de empresas.

Wray Buntine & Rich Caruana
"Introduction to IND and Recursive Partitioning"
Research Institute for Advanced Computer Science

Manual del paquete IND, una implementación en C de distintos algoritmos de construcción de árboles de decisión (CART esencialmente, aunque también incluye C4 y distintas variaciones bayesianas y basadas en el principio MDL de Rissanen).

"An Introduction to CART Methodology"
California Statistical Software, Inc., 1985

Describe las características esenciales de CART™ desde un punto de vista práctico y poco riguroso (de forma que pueda entenderlo cualquier persona). Viene a ser un resumen del libro de Breiman, Friedmanm, Olsten y Stone titulado "Classification And Regression Trees" (Wadsworth, 1984), donde se describe detalladamente el programa CART™.

Manish Mehta, Rakesh Agrawal & Jorma Rissanen
"SLIQ: A Fast Scalable Classifier for Data Mining"
5th International Conference on Extending Database Technology, Avignon, France, 1996

En este artículo se presenta el algoritmo SLIQ, un algoritmo de construcción de árboles de decisión diseñado específicamente para aplicaciones de Data Mining en el IBM Almaden Research Center para el proyecto Quest.

Manish Mehta, Jorma Rissanen & Rakesh Agrawal
"MDL-based Decision Tree Prunning"
International Conference on KDD and Data Mining (KDD'95), Montreal, Canada, 1995

Este artículo propone la aplicación del principio MDL [Minimum Description Length] a la poda de árboles de decisión. Los resultados obtenidos con el algoritmo propuesto son comparables a los obtenidos con la poda pesimista en C4.5.

J. Ross Quinlan
"Induction of Decision Trees"
Machine Learning 1:81-106, Kluwer Academic Publishers, 1986

Uno de los artículos clásicos en Machine Learning. En él se describe el algoritmo *ID3*. Este algoritmo se enmarca dentro de la familia *TDIDT* [*Top-Down Induction of Decision Trees*] de sistemas de aprendizaje, cuyo "patriarca" es *CLS* [*Concept Learning System*].

J. Ross Quinlan
"C4.5: Programs for Machine Learning"
Morgan Kaufmann Publishers, San Francisco, California, 1993

El libro del *C4.5*, uno de los muchos derivados de *ID3*. En realidad, el algoritmo C4.5 se puede considerar un híbrido entre CART y C4. De hecho, permite usar como regla de división la ganancia de información de ID3, el índice de diversidad de Gini de CART o el criterio de proporción de ganancia.

Elaine Rich & Kevin Knight
"Inteligencia Artificial"
McGraw-Hill, 1994 [2ª edición]

Uno de los libros de texto más populares de IA. Incluye un capítulo dedicado a técnicas de aprendizaje. En él se tratan, a un nivel casi meramente informativo, distintas formas de afrontar el problema del aprendizaje automático (entendido éste como un proceso por el cual un ordenador acrecienta su conocimiento y mejora su habilidad).

Sabrina Sestito & Tharam S. Dillon
"Automated Knowledge Adquisition"
Prentice Hall Series in Computer Science and Engineering, Australia, 1994

Otro libro sobre técnicas de aprendizaje automático que incluye, entre otros, un capítulo dedicado por completo a la construcción de árboles de decisión.

John Shafer, Rakesh Agrawal & Manish Mehta
"SPRINT: A Scallable Parallel Clasifier for Data Mining"
Proceedings of the 22nd VLDB Conference, India, 1996

Un artículo sobre un algoritmo eficiente y escalable de construcción de árboles de decisión diseñado para ser paralelizado, lo cual lo hace idóneo para trabajar con grandes bases de datos. SPRINT es un algoritmo derivado de SLIQ que, al igual que éste, ha sido desarrollado por los participantes en el proyecto Quest del IBM Almaden Research Center.