# Aprendizaje Automático: Arboles de Decisión.

Dr. Alejandro Guerra Hernández
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Maestría en Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México 91000
aguerra@uv.mx
www.uv.mx/aguerra

Marzo 15, 2004

## 1. Introducción

El aprendizaje de árboles de decisión es una de las técnicas de inferencia inductiva más usadas. Se trata de un método para aproximar funciones de valores discretos, capaz de expresar hipótesis disyuntivas y robusto al ruido en los ejemplos de entrenamiento. La descripción que presento en este capítulo, cubre una familia de algoritmos para la inducción de árboles de decisión que incluyen ID3 [3] y C4.5 [5]. Estos algoritmos llevan a cabo su búsqueda de hipótesis en un espacio completamente expresivo, evitando así los problemas mencionados en la sesión pasada, con respecto a espacios de hipótesis incompletos. El sesgo inductivo en este caso, consiste en la preferencia por árboles pequeños, sobre árboles grandes. Un árbol aprendido puede representarse también como un conjunto de reglas *si-entonces*, más fáciles de entender para un usuario.

## 2. Representación de los árboles de decisión

La figura 1 muestra un árbol de decisión típico. Cada nodo del árbol está conformado por un *atributo* y puede verse como la pregunta: ¿Qué valor tiene este atributo en el ejemplar a clasificar? Las ramas que salen de los nodos, corresponden a los posibles valores del atributo correspondiente.

Un árbol de decisión clasifica a un ejemplar, filtrandolo de manera descendente, hasta encontrar una hoja, que corresponde a la clasificación buscada. Consideren el proceso de clasificación del siguiente ejemplar, que describe un día en partícular:

$$\langle\ cielo = soleado, temperatura = caliente, humedad = alta, viento = fuerte\ \rangle$$

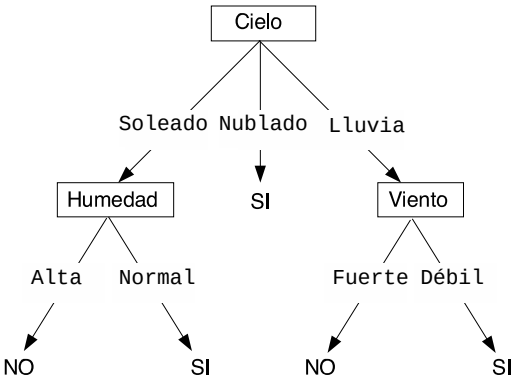


Figura 1: *Un ejemplo de arbol de decisión para el concepto "buen día para jugar tenis". Los nodos representan un atributo a ser verificado por el clasificador. Las ramas son los posibles valores para el atributo en cuestión. Los textos en mayúsculas, representan las clases consideradas, i.e., los valores posibles del atributo objetivo.*

Como el atributo *cielo*, tiene el valor *soleado* en el ejemplar, éste es filtrado hacía abajo del árbol por la rama de la derecha. Como el atributo *humedad*, tiene el valor *alta*, el ejemplo es filtrado nuevamente por rama de la derecha, lo cual nos lleva a la hoja que indica la clasificación del ejemplar: *Buen día para jugar tenis = no*. El algoritmo de clasificación se muestra en el cuadro 1.

La función `toma-valor` encuentra el valor de un atributo, en el ejemplar que se está clasificando. El predicado `hoja` es verdadero si su argumento es un nodo terminal del árbol y falso si se trata de un nodo interno. La función `sub-árbol` se mueve por la rama del árbol que corresponde al valor del atributo probado en el ejemplar. De esta forma, obtiene un sub-árbol. En nuestro ejemplo, a partir del nodo raíz *cielo*, esta función obtiene el sub-árbol que resulta de moverse por la rama *soleado*, etc.

En general, un árbol de decisión representa una disyunción de conjunciones de restricciones en los posibles valores de los atributos de los ejemplares. Cada rama que va de la raíz del árbol a una hoja, representa una conjunción de tales restricciones y el árbol mismo representa la disyunción de esas conjunciones. Por ejemplo, el árbol de la figura 1, puede expresarse como sigue:

$$\begin{array}{ll} & (cielo = soleado \ \wedge \ humedad = normal) \\ \vee & (cielo = nublado) \\ \vee & (cielo = lluvia \ \wedge \ viento = débil) \end{array}$$

```
fun clasifica(ejemplar, árbol)
    input:  ejemplar: el ejemplar a ser clasificado;
            árbol: el árbol de decisión usado para clasificar;
    output:  clase: la clase del ejemplar.

    clase ← toma-valor(raíz(árbol), ejemplar);
    if hoja(raíz(árbol)) then return clase
    else clasifica(ejemplar, sub-árbol(árbol, clase));
    endif
endfun
```

Cuadro 1: El algoritmo clasifica, para árboles de decisión

# 3. Problemas apropiados para la aplicación de árboles de decisión

Aun cuando se han desarrollado diversos métodos para la inducción de árboles de decisión, y cada uno de ellos ofrece diferentes capacidades, en general estos algoritmos son apropiados para solucionar problemas de aprendizaje conocidos como *problemas de clasificación*. Estos problemas presentan las siguientes características:

**Ejemplares representados por pares atributo-valor.** Los ejemplares del problema están representados como un conjunto fijo de atributos, por ejemplo *Cielo* y sus valores, por ej. *Soleado*. El caso más sencillo es cuando cada atributo toma valores de un pequeño conjunto discreto y cada valor es disjunto, por ejemplo {*Soleado, Nublado, Lluvia*}. Como sea, existen extensiones para trabajar con atributos de valores reales, por ejemplo, *Temperatura* expresado numéricamente.

**La función objetivo tiene valores discretos.** El árbol de decisión de la figura 1, asigna una clasificación binaria, por ejemplo *SI* o *NO* a cada ejemplar. Un árbol de decisión puede ser extendido fácilmente, para representar funciones objetivos con más de dos valores posibles. Una extensión menos simple consiste en considerar funciones objetivo de valores discretos. Como sea, la aplicación del método en dominios discretos es menos común.

**Se necesitan descripciones disyuntivas.** Como se mencionó, los árboles de decisión representan naturalmente conceptos disyuntivos.

**Ruido en los ejemplos de entrenamiento.** El método es robusto al ruido en los ejemplos de entrenamiento, tanto errores de clasificación, como errores en los valores de los atributos.

**Valores faltantes en los ejemplos.** El método puede usarse aún cuando algunos ejemplos de entrenamiento tengan valores desconocidos para algunos atributos.

# 4. El algoritmo básico de aprendizaje de árboles de decisión

La mayoría de los algoritmos para inferir árboles de decisión son variaciones de un algoritmo básico que emplea una búsqueda descendente (*top-down*) y egoista (*greedy*) en el espacio de posibles árboles de decisión. La presentación de estos algoritmos se centra en ID3 y C4.5.

El algoritmo básico ID3, construye el árbol de decisión de manera descendente, comenzando por preguntarse: Qué atributo debería ser colocado en la raíz del árbol? Para responder esta pregunta, cada atributo es evaulado usando un test estadístico para determinar que tan bien clasifica él solo los ejemplos de entrenamiento. El mejor atributo es seleccionado y colocado en la raíz del árbol. Una rama y su nodo correspondiente es entonces creada para cada valor posible del atributo en cuestión. Los ejemplos de entrenamiento son repartidos en los nodos descendentes de acuerdo al valor que tengan para el atributo de la raíz. El proceso entonces se repite con los ejemplos ya distribuidos, para seleccionar un atributo que será colocado en cada uno de los nodos generados. Generalmente, el algoritmo se detiene si los ejemplos de entrenamiento comparten el mismo valor para el atributo que está siendo probado. Sin embargo, otros criterios para finalizar la búsqueda son posibles: i) Covertura mínima, el número de ejemplos cubiertos por cada nodo está por abajo de cierto umbral; ii) Pruebas de significancia[1] estadística, usando $\chi^2$ para probar si las distribuciones de las clases en los sub-árboles difiere significativamente. Este algoritmo lleva a cabo una búsqueda egoista de un árbol de decisión aceptable, sin reconsiderar nunca las elecciones pasadas (*backtracking*). Una versión simplificada de él se muestra en el cuadro 2[2]:

## 4.1. ¿Qué atributo es el mejor clasificador?

La decisión central de ID3 consiste en seleccionar qué atributo colocará en cada nodo del árbol de decisión. En el algoritmo presentado, esta opción la lleva a cabo la función `mejor-partición`, que toma como argumentos un conjunto de ejemplos de entrenamiento y un conjunto de atributos, regresando la partición inducida por el atributo, que sólo, clasifica mejor los ejemplos de entrenamiento. Considere los ejemplos de entrenamiento del cuadro 3 para el concepto objetivo: buen día para jugar tenis? El encabezado del cuadro indica los atributos usados para describir estos ejemplos, siendo *jugar-tenis?* el atributo objetivo.

Si queremos particionar este conjunto de ejemplos con respecto al atributo *temperatura*, obtendríamos:

[1] Aunque, como veremos, la poda del árbol se prefiere a las pruebas de significacia.

[2] Su implementación en Lisp se encuentra en la página web del curso

```
fun ID3(ejemplos, atributos, cible)
    input:  ejemplos: conjunto de entrenamiento;
            atributos: atributos usados;
            cible: el atributo a predecir;
    static: default: valor por default del atributo c;
            mejor-partición: la mejor partición;
            mejor-atributo: el mejor atributo;
            valor-atributo: el valor de un atributo;
            partición-ejs: una particion de ejemmplos;
            sub-ejemplos: un subconjunto de los ejemplos de entrenamiento;
    output: árbol: un árbol de decisión;

    árbol ← {};
    if ejemplos = {} then return default
    else if mismo-valor(ejemplos,cible) then
            return árbol ← toma-valor(first(ejemplos).cible);
    else if atributos = {} then
            return árbol ← valor-más-común(ejemplos,cible)
    else
        mejor-partición ← mejor-partición(ejemplos, atributos);
        mejor-atributo ← first(mejor-partición);
        árbol ← mejor-atributo;
        foreach partición-ejs in rest(mejor-partición) do
                valor-atributo ← first(partición-ejs);
                sub-ejemplos ← rest(partición-ejs);
                agregar-rama(árbol, valor-atributo,
                    ID3(sub-ejemplos, {atributos \ mejor-atributo}, cible));
    return árbol
endfun
```

Cuadro 2: El algoritmo ID3

| día | cielo | temperatura | humedad | viento | jugar-tennis? |
|---|---|---|---|---|---|
| d1 | soleado | calor | alta | débil | no |
| d2 | soleado | calor | alta | fuerte | no |
| d3 | nublado | calor | alta | débil | si |
| d4 | lluvia | templado | alta | débil | si |
| d5 | lluvia | frío | normal | débil | si |
| d6 | lluvia | frío | normal | fuerte | no |
| d7 | nublado | frío | normal | fuerte | si |
| d8 | soleado | templado | alta | débil | no |
| d9 | soleado | frío | normal | débil | si |
| d10 | lluvia | templado | normal | débil | si |
| d11 | soleado | templado | normal | fuerte | si |
| d12 | nublado | templado | alta | fuerte | si |
| d13 | nublado | calor | normal | débil | si |
| d14 | lluvia | templado | alta | fuerte | no |

Cuadro 3: *Conjunto de ejemplos de entrenamiento para el concepto objetivo jugar-tenis? en ID3, por Tom M. Mitchel [2].*

```
> (partition 'temperatura *examples*)
(TEMPERATURA (FRIO D5 D6 D7 D9) (CALIENTE D1 D2 D3 D13)
 (TEMPLADO D4 D8 D10 D11 D12 D14))
```

Lo que significa que el atributo *temperatura* tiene tres valores diferentes en el conjunto de entrenamiento: *frío*, *caliente*, y *templado*. Los ejemplares *d5*, *d6*, *d7*, y *d9*, tienen como valor del atributo *temperatura= frío*. La función *mejor-partición* encuentra el atributo que mejor separa los ejemplos de entrenamiento de acuerdo al atributo objetivo. En qué consiste una buena medida cuantitativa de la bondad de un atributo? Definiremos una propiedad estadística llamada *ganancia de información*.

## 4.2. Entropía y ganancia de información

Una manera de cuantificar la bondad de un atributo en este contexto, consiste en considerar la cantidad de información que proveerá este atributo, tal y como ésto es definido en teoría de información por Claude E. Shannon [6]. Un bit de información es suficiente para determinar el valor de un atributo booleano, por ejemplo, si/no, verdader/falso, 1/0, etc., sobre el cual no sabemos nada. En general, si los posibles valores del atributo $v_i$, ocurren con probabilidades $P(v_i)$, entonces en contenido de información, o entropia, $E$ de la respuesta actuale está dado por:

$$E(P(v_i), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \ \log_2 \ P(v_i)$$

Consideren nuevamente el caso booleano, aplicando esta ecuación a un volado

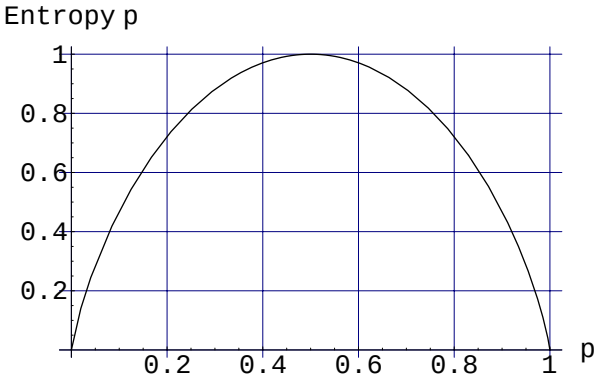


Figura 2: *Gráfica de la función entropia para clasificaciones booleanas.*

con una moneda confiable, tenemos que la probabilidad de obtener aguila o sol es de 1/2 para cada una:

$$E(\frac{1}{2},\frac{1}{2}) = -\frac{1}{2}\ \log_2\ \frac{1}{2} - \frac{1}{2}\ \log_2\ \frac{1}{2} = 1$$

Ejecutar el volado nos provee 1 bit de información, de hecho, nos provee la clasificación del experimento: si fue aguila o sol. Si los volados los ejecutamos con una moneda cargada que da 99 % de las veces sol, entonces $E(1/100, 99/100) =$ 0,08 bits de información, menos que en el caso de la moneda justa, porque ahora tenemos más evidencia sobre el posible resultado del experimento. Si la probabilidad de que el volado de sol es del 100 %, entonces $E(0,1) = 0$ bits de información, ejecutar el volado no provee información alguna. La gráfica de la función de entropía se muestra en la figura 2.

Consideren nuevamente los ejemplos de entrenamiento del cuadro 3. De 14 ejemplos, 9 son positivos (si es un buen día para jugar tenis) y 5 son negativos. La entropia de este conjunto de entrenamiento es:

$$E(\frac{9}{14},\frac{5}{14}) = 0{,}940$$

Si todos los ejemplos son positivos o negativos, por ejemplo, pertencen todos a la misma clase, la entropia será 0. Una posible interpretación de ésto, es considerar la entropia como una medida de ruido o desorden en los ejemplos. Definimos la *ganancia de información* (GI) como la reducción de la entropía causada por particionar un conjunto de entrenamiento $S$, con respecto a un atributo $A$:

$$Ganancia(S,A) = E(S) - \sum_{v\in A} \frac{|S_v|}{|S|} E(S_v)$$

Observen que el segundo término de *Ganancia*, es la entropía con respecto al atributo $A$. Al utilizar esta medida en ID3, sobre los ejemplos del cuadro 3, obtenemos:

```
Ganancia de informacion del atributo CIELO : 0.24674976
Ganancia de informacion del atributo TEMPERATURA : 0.029222548
Ganancia de informacion del atributo HUMEDAD : 0.15183544
Ganancia de informacion del atributo VIENTO : 0.048126936
Maxima ganancia de informacion: 0.24674976
Particion:
(CIELO (SOLEADO D1 D2 D8 D9 D11) (NUBLADO D3 D7 D12 D13)
 (LLUVIA D4 D5 D6 D10 D14))
```

Esto indica que el atributo con mayor ganancia de información fue *cielo*, de ahí que esta parte del algoritmo genera la partición de los ejemplos de entrenamiento con respecto a este atributo. Si particionamos recursivamente los ejemplos que tienen el atributo *cielo* = *soleado*, obtendríamos:

```
Ganancia de informacion del atributo TEMPERATURA : 0.5709506
Ganancia de informacion del atributo HUMEDAD : 0.9709506
Ganancia de informacion del atributo VIENTO : 0.01997304
Maxima ganancia de informacion: 0.9709506
Particion:
(HUMEDAD (NORMAL D11 D9) (ALTA D8 D2 D1))
```

Lo cual indica que en el nodo debajo de *soleado* deberíamos incluir el atributo *humedad*. Todos los ejemplos con *humedad* = *normal*, tienen valor *si* para el concepto objetivo. De la misma forma, todos los ejemplos con valor *humedad* = *alta*, tiene valor *no* para el concepto objetivo. Así que ambas ramas descendiendo de nodo *humedad*, llevarán a clases terminales de nuestro problema de aprendizaje. El algoritmo terminará por construir el árbol de la figura 1.

# 5. Espacio de hipótesis en el aprendizaje inductivo de árboles de decisión

Como los otros métodos de aprendizaje, ID3 puede concebirse como un proceso de búsqueda en un espacio de hipótesis, para encontrar aquella hipótesis que se ajusta mejor a los datos de entrenamiento. El espacio de hipótesis explorado por ID3 es el espacio de todos los árboles de decisión posibles. El algoritmo lleva a cabo una búsqueda de lo simple a lo complejo, comenzando por el árbol vacío, para considerar cada vez hipótesis más complejas. La medida *ganancia de información* guía esta búsqueda de ascenso de colina (*hill-climbing*), como ejemplificamos en la sección anterior.

Considerando ID3 en términos de su espacio y estrategias de búsqueda, es posible analizar sus capacidades y limitaciones:

- El espacio de hipótesis de ID3 es *completo* con respecto a las funciones de valores discretos que pueden definirse a partir de los atributos considerados. De manera que no existe el riesgo que la función objetivo no se encuentre en el espacio de hipótesis.

- ID3 mantiene sólo una hipótesis mientras explora el espacio de hipótesis posibles. Esto contrasta, por ejemplo, con el algoritmo *eliminación de candidatos*, que mantiene el conjunto de todas las hipótesis consistentes con el conjunto de entrenamiento. Es por ello que ID3 es incapaz de determinar cuantos árboles de decisión diferentes son consistentes con los datos.

- El algoritmo básico ID3 no ejecuta vuelta atrás (*backtracking*) en su búsqueda. Una vez que el algoritmo selecciona un atributo, nunca reconsiderará esta elección. Por lo tanto, es suceptible a los mismos riesgos que los algoritmos estilo ascenso de colina, por ejemplo, caer máximos o mínimos locales. Como veremos, la vuelta atrás puede implementarse con alguna técnica de poda.

- ID3 utiliza todos los ejemplos de entrenamiento en cada paso de su búsqueda guiada por el estadístico *ganancia de información*. Esto contrasta con los métodos que usan los ejemplos incrementalmente, por ejemplo *encuentra-S* o *eliminación de candidatos*. Una ventaja de usar propiedades estadísticas de todos los ejemplos es que la búsqueda es menos sensible al ruido en los datos.

# 6. Sesgo inductivo en el aprendizaje de árboles de decisión

Recuerden que el sesgo inductivo es el conjunto de afirmaciones que, junto con los datos de entrenamiento, justifican deductivamente la clasificación realizada por un sistema de aprendizaje inductivo sobre ejemplares futuros. Dado un conjunto de entrenamiento, por lo general hay muchos árboles de decisión consistentes con éste. Describir el sesgo inductivo de ID3 equivale a explicar porqué este algoritmo prefiere ciertos árboles a otros, qué árbol eligirá.

Puesto que ID3 encontrará el primer árbol consistente con el conjunto de entrenamiento, producto de una búsqueda de ascenso de colina, de lo simple a lo complejo, el algoritmo tiene preferencia por: i) árboles pequeños sobre árboles grandes, que indican que la búsqueda termino en proximidad a la raíz del árbol; y ii) debido a su caracter egoista, árboles que colocan atributos más informativos cerca de la raíz del árbol. Sin embargo, observen que este sesgo es aproximado. Un algoritmo que tuviera un sesgo idéntico al descrito aquí, tendría que realizar una búsqueda primero en amplitud y preferir los árboles de menor profundidad. ID3 busca primero en profundidad.

### 6.1. Sesgo por restricción y sesgo por preferencia

Existe una diferencia interesante entre los sesgos que exhiben ID3 y el algoritmo *eliminación de candidatos*, discutido en la sesión anterior. El sesgo de ID3 es producto de su estratégia de búsqueda, mientras que el sesgo de *eliminación de candidatos* es resultado de la definición del espacio de búsqueda. Por lo tanto, el sesgo de ID3 es exhibe una preferencia por ciertas hipótesis, sobre otras, por ejemplo, hipótesis compactas. Este tipo de sesgo, que no impone restricciones sobre las hipótesis que serán eventualmente consideradas, recibe el nombre de *sesgo por preferencia*. Por otra parte, el sesgo de *eliminación de candidatos* que restringe el conjunto de hipótesis a considerar, recibe el nombre de *sesgo por restricción* o *sesgo del lenguaje*.

En general, es preferible trabajar con un sesgo por preferencia, puesto que éste permite al sistema de aprendizaje explorar un espacio de hipótesis completo, asegurando que la representación del concepto objetivo se encuentra ahí. Consideren que es posible trabajar con ambos sesgos a la ves, por ejemplo, el sistema aprendiz de damas chinas de la sesión de introducción, introduce un sesgo por restricciones cuando se decide que la hipótesis tiene la forma de una combinación lineal de los atributos del tablero, y un sesgo por preferencia cuando se introduce la búsqueda ordenada por mínimos cuadrados ($LMS$) en el espacio de posibles parámetros $w_i$.

### 6.2. ¿Porqué preferir hipótesis más compactas?

Es el sesgo inductivo de ID3, preferir las hipótesis más compactas, lo suficientemente robusto para generalizar más allá de los datos observados? Este es un debate no resuelto iniciado por William de Occam[3] *circa* 1320. Un argumento intuitivo es que existen mucho menos hipótesis compactas que extensas, por lo que es más difícil que una hipótesis compacta coincida accidentalmente con los datos observados. En cambio, hay muchas hipótesis extensas que se puede, ajustar a los datos de entrenamiento, pero fallarán al generalizar. Aunque este argumento no es del todo convincente, dejaremos la discusión sobre la navaja de Occam para la sesión destinada a aprendizaje Bayesiano.

## 7. Consideraciones sobre el aprendizaje inductivo de árboles de decisión

Algunas consideraciones sobre la aplicación práctica del algoritmo básico de ID3 presentado aquí, incluyen: mecanismos para determinar que tanto debe crecer el árbol en profundidad; para procesar atributos con valores contínuos; para procesar ejemplos de entrenamiento con valores faltantes; para introducir costos diferentes asociados a los atributos; así como para determinar una buena métrica de selección de los atributos y mejorar la eficiencia computacional del

[3]El enunciado exacto de la navaja de Occam es: *Non sunt multiplicanda entia prater necessitatem* (las entidades no deben multiplicarse más allá de lo necesario).

algoritmo. Cabe mencionar que, muchos de estos aspectos han sido incorporados en el sistema C4.5 [5].

## 7.1. Evitando un sobreajuste con los datos de entrenamiento

El algoritmo básico de ID3 crece cada rama del árbol en profundidad hasta que logra clasificar perfectamente los ejemplos de entrenamiento. Esta estrategia es razonable, pero puede introducir dificultades si los datos de entrenamiento presentan ruido, o cuando el conjunto de entrenamiento es demasiado pequeño, como para ofrecer un muestreo significativo del concepto objetivo. En estos casos, ID3 puede producir árboles que se sobreajustan a los datos de entrenamiento. Formalmente definimos el sobreajuste como:

**Def 1** *Dado un espacio de hipótesis $H$, se dice que una hipótsis $h \in H$ está sobreajustada a los ejemplos de entrenamiento, si existe una hipótesis alternativa $h' \in H$, tal que $h'$ tiene un error de clasificación más pequeño que $h$ sobre la distribución completa de los ejemplares del problema.*

Es común observar que a medida que el tamaño del árbol crece, en término del número de nodos usado[4], su precisión sobre el conjunto de entrenamiento mejora monotonicamente, pero, sobre el conjunto de prueba primero crece y luego decae.

Como es esto posible que un árbol $h$ que tiene mayor precisión que $h'$ sobre el conjunto de entrenamiento, luego tenga un desempeño menor sobre el conjunto de prueba? Una situación en la que esto ocurre es cuando el conjunto de entremiento contiene ruido, por ejemplo, elementos mal clasificados. Consideren agregar el siguiente ejemplar mal clasificado (clasificado como *jugar-tenis? = no*) al conjunto de entrenamiento del cuadro 3:

$$\langle\ cielo = soleado, temperatura = caliente, humedad = normal, viento = fuerte\ \rangle$$

Al ejecutar ID3 sobre el nuevo conjunto de entrenamiento, éste construirá un árbol más complejo. En partícular, el ejemplo con ruido será filtrado junto con los ejemplos *d9* y *d11* (*cielo = soleado* y *humedad = normal*), que son ejemplos positivos. Dado que el nuevo ejemplo es negativo, ID3 buscará refinar el árbol a partir del nodo *humedad*, agregando un atributo más al árbol. Este nuevo árbol $h'$ tiene mayor precisión sobre los ejemplos de entrenamiento que $h$, puesto que se ha ajustado al ejemplo con ruido. Pero $h$ tendrá mejor desempeño al clasificar nuevos ejemplares, tomados de una misma distribución que los ejemplos de entrenamiento.

Existe la posibilidad de sobreajuste, aún cuando el conjunto de entrenamiento esté libre de ruido, por ejemplo, si el conjunto de entrenamiento tiene pocos

[4] Obseven que esto refleja el número de atributos usado en la hipótesis, esto es, árboles más grandes imponen más restricciones.

elementos. En conjuntos de entrenamiento pequeños es fácil encontrar regularidades accidentales en donde un atributo puede particionar muy bien los ejemplos dados, aunque no esté relacionado con el concepto objetivo.

Puesto que el sobreajuste puede reducir la precisión de un árbol inducido por ID3 entre un 10 a 25 %, diferentes enfoques han sido propuestos para evitar este fenómeno. Los enfoques pueden agruparse en dos clases:

- Enfoques que detienen el crecimiento del árbol anticipadamente, antes de que alcance un punto donde clasifique perfectamente los ejemplos de entrenamiento.
- Enfoques en donde se deja crecer el árbol para después podarlo.

Aunque el primer enfoque parezca más directo, la poda posterior del árbol ha demostrado tener más éxito en la práctica. Esto se debe a la dificultad de estimar en que momento debe detenerse el crecimiento del árbol. Independientemente del enfoque usado, una pregunta interesante es: ¿Cual es el tamaño correcto de un árbol? Algunos enfoques para responder a esta pregunta incluyen:

- Usar un conjunto de ejemplos, diferentes de los usados en el entrenamiento, para evaluar la utilidad de eliminar nodos del árbol.
- Usar los ejemplos disponibles para el entrenamiento, pero aplicando una prueba para estimar cuando agregar o eliminar un nodo, podría producir una mejora al clasificar nuevos ejemplares, por ejemplo, usar el test $\chi^2$ para evaluar si al expandir un nodo, el cambio mejorará la clasificación sobre los ejemplos de entrenamiento, o sobre toda la distribución.
- Usar explícitamente una medida de complejidad para codificar los ejemplos de entrenamiento y el árbol de decisión, deteniéndo el crecimiento cuando el tamaño codificado sea minimizado. Por ejemplo, el principio de descripción mínima (*MDL*).

### 7.1.1. Reduciendo el error por poda

¿Como podemos usar un conjunto de ejemplos de validación para prevenir el sobre ajuste? Un enfoque llamado *reduced-error pruning* [4], consiste en considerar cada nodo del árbol como candidato a ser podado. La poda consiste en eliminar todo el subárbol que tiene como raíz el nodo en cuestión, convirtiéndolo así en una hoja, cuya clase corresponde a valor más común de los ejemplares asociados a ese nodo.

Un nodo solo es eliminado si el árbol podado que resulta de ello, no presenta un desempeño peor que el árbol original sobre el conjunto de validación. El efecto de esto, es que los nodos que se han colocado en el árbol por coincidencias fortuitas en los datos del entrenamiento, generalmente son eliminados debido a que las coincidencias suelen no estar presentes en el conjunto de validación.

Este método es unicamente efectivo si contamos con suficientes ejemplos, de tal forma que el conjunto de entrenamiento y el conjunto de validación sean

significativos estadísticamente. De otra forma, tomar ejemplos para el conjunto de validación reduce aún más el tamanõ del conjunto de entrenamiento, aumentando así la posibilidad de sobre ajuste.

### 7.1.2. Poda de reglas

En la práctica, un método exitoso para encontrar el árbol de mayor precisión se conoce como *rule post-prunning* [5] y está incorporado en el sistema C4.5 de Ross Quinlan. El procedimiento es el siguiente:

1. Inducir el árbol de decisión permitiendo sobre ajuste, por ejemplo, con nuestro algoritmo básico ID3.

2. Convertir el árbol aprendido en un conjunto de reglas equivalente, esto es, una conjunción por cada rama del árbol que va de la raíz a una hoja.

3. Podar (generalizar) cada regla, eliminando las precondiciones que resulten en una mejora de la precisión estimada.

4. Ordenar las reglas por su precisión estimada, y aplicarlas en ese orden al clasificar nuevos ejemplares.

Cabe mencionar que el método aplicado por C4.5 no es estadísticamente valido, aunque ha demostrado ser una heurística útil. En la sesión de evaluación de hipótesis, estudiamos técnicas estadísticamente robustas para estimar medias e intervalos de confianza. Lo relevante aquí es que la conversión del árbol en reglas ayuda a distinguir los diferentes contextos en los que un atributo participa en la clasificación, es decir, reglas diferentes; elimina la diferencia entre nodos ubicados cerca de la raíz y aquellos ubicados cerca de las hojas; y aumenta la fácilidad de comprehensión por parte del usuario.

## 7.2. Incorporando valores contínuos

En el algoritmo básico de ID3 tanto el concepto objetivo, como los atributos usados para describir los ejemplares, deben tener valores discretos. La segunda restricción puede ser eliminada fácilmente, permitiendo el uso de atributos con valores contínuos. Esto se logra definiéndo dinámicamente nuevos atributos discretos que particionan los atributos de valores contínuos, en intervalos discretos. Para un atributo contínuo $A$, el algoritmo puede crear dinámicamente un atributo discreto $A_c$ que es verdadero si $A > c$ y falso en cualquier otro caso. La única consideración es como seleccionar el mejor valor para el umbral $c$. Supongan que el atributo *temperatura* toma valores discretos y que su relación con el concepto objetivo es la siguiente:

| temperatura | 40 | 48 | 60 | 72 | 80 | 90 |
|---|---|---|---|---|---|---|
| jugar-tenis? | No | No | Si | Si | Si | No |

Qué valor booleano basado en un umbral debemos definir para el atributo temperatura? Obviamente, necesitamos un umbral $c$, tal que éste produzca la mayor ganancia de información posible. Es posible generar candidatos a umbral, ordenando los ejemplos de acuerdo a su valor en el atributo *temperatura* e identificando ejemplos adyacentes que difieren en el valor de su atributo objetivo. Se puede demostrar que los umbrales $c$ que máximiza la ganancia de información, se encuentran en estos sitios. Para el ejemplo presentado, dos umbrales pueden localizarse en los puntos $(48 + 60)/2$ y $(80 + 90/2)$. La ganancia de información puede entonces calcularse para los atributos $temperatura_{>54}$ y $temperatura_{>85}$. El atributo con mayor ganancia de información, en este caso el primero, puede ser usado entonces para competir con otros atributos en la construcción del árbol de decisión. Por supuesto, es posible también mantener ambos atributos dinámicamente creados, usando multiples intervalos [1].

## 7.3. Medidas alternativas para la selección de atributos

Existe un sesgo natural en la medida de ganancia de información, el cual favorece atributos con muchos valores, sobre aquellos que tienen poco valores. Por ejemplo, un atributo *fecha*, tendría mayor ganancia de información que cualquiera de los atributos en nuestro ejemplo. Esto se debe a que este atributo predice perfectamente el valor del atributo objetivo. El problema es que este atributo tiene tantos valores distintos que tiende a separar perfectamente los ejemplos de entrenamiento en pequeños subconjuntos, que se ajustan al concepto buscado. Por esto, el atributo *fecha* tiene una ganancia de información elevada, a pesar de ser un predictor pobre.

Una solución a este problema es usar una métrica alternativa a la ganancia de información. Quinlan [3], propone una medida alternativa que ha sido usada con éxito, *gain ratio*. Esta métrica penaliza atributos como *fecha* incorporando un término conocido como *split information*, que es sensible a qué tan amplia y uniforme es la partición que un atributo induce en los datos:

$$splitInformation(S, A) = -\sum_{i=1}^{c} \frac{|Si|}{|S|} \; log_2 \frac{|Si|}{|S|}$$

Observen que este término es la entropia de $S$ con respecto al atributo $A$. La medida *gain radio* está definida como:

$$gainRatio(S, A) = \frac{gain(S, A)}{splitInformatio(S, A)}$$

Un problema práctico con este enfoque es que el denominador de esta medida puede ser 0 o muy pequeño, si $|Si| \approx |S|$, lo cual hace que la medida sea indefinida para atributos que tienen casi el mismo valor para todos los ejemplos.

# Referencias


[1] U.M. Fayad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan-Kaufmann, 1993.

[2] T.M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill International Editions, Singapore, 1997.

[3] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[4] J.R. Quinlan. Rule induction with statistical data -a comparison with multiple regression. *Journal of the Operational Research Society*, 38:347–352, 1987.

[5] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA., USA, 1993.

[6] C. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, IL, USA, 1948.