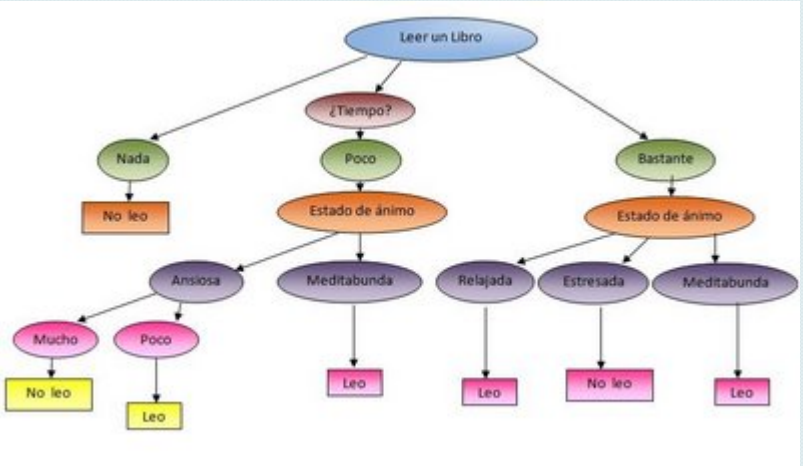
Árboles de decisión

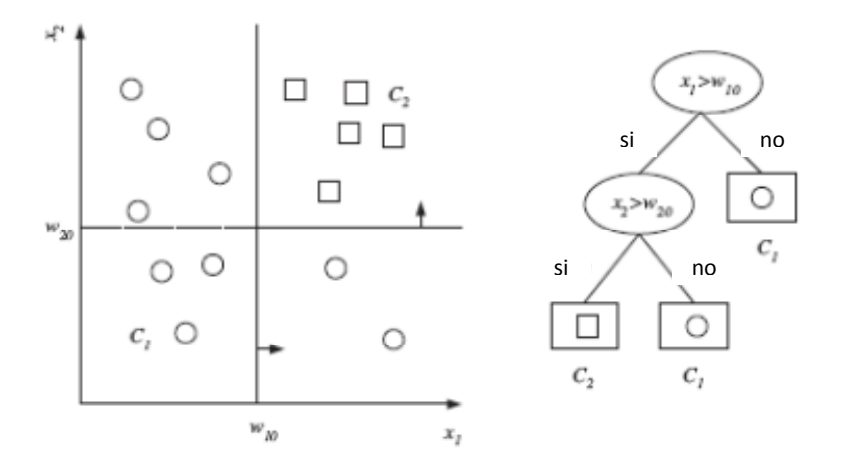
- En la estimación paramétrica definíamos un modelo sobre todo el espacio de input y aprendíamos los parámetros a partir de todos los datos de entrenamiento.
- Luego se usaba el mismo modelo y el mismo conjunto de parámetros para el testeo de cualquier input.

- En la estimación no paramétrica, se divide el espacio de input en regiones locales, definidas por una medida de distancia, por ejemplo la distancia euclídea, y para cada input se computa el correspondiente modelo local a partir de los datos de entrenamiento de la región usada.



- El aprendizaje por árboles de decisiones es un método no paramétrico para aproximar funciones objetivos de valores discretos, en los cuales la función aprendida se representa por un árbol de decisión.

- Un árbol de decisión es un modelo jerárquico para aprendizaje supervisado donde la región local se identifica con una secuencia de separaciones recursiva en un número pequeño de pasos.



- Los árboles de decisión clasifican instancias ordenándolas hacia abajo en el árbol desde la raíz hasta algún nodo hoja

- Cada nodo del árbol m específica una función test $f_m(\boldsymbol{x})$ para algún atributo de la instancia

- Cada rama descendiente de ese nodo corresponde a uno de los valores posibles para este atributo.

- Una instancia se clasifica comenzando con el nodo raíz del árbol, testeando el atributo especificado en ese nodo, luego moviéndose hacia abajo en la rama del árbol correspondiente al valor del atributo en el ejemplo dado.

- Un árbol de decisión es un modelo no paramétrico en el sentido de que no se asume ninguna forma paramétrica para las densidades de las clases y la estructura del árbol no está fija a priori, pero a medida que el árbol crece, se agregan ramas y hojas durante el aprendizaje dependiendo de la complejidad del problema inherente a los datos.

- Cada $f_m(\boldsymbol{x})$ define un discriminante en el espacio p –dimensional dividiéndolo en regiones que a su vez se subdividen a medida que se toma un camino desde la raíz hacia abajo.

- Cada nodo hoja tiene un rótulo, el cual en el caso de la clasificación, es la clase.
    - Cada hoja define una región localizada dentro del espacio de input donde las instancias que caen en esta región tienen los mismos rótulos.
- Las fronteras de las regiones están definidas por los discriminantes que están codificados en los nodos internos del camino desde la raíz al nodo hoja.

Ejemplo (**jugar-al-tenis**)

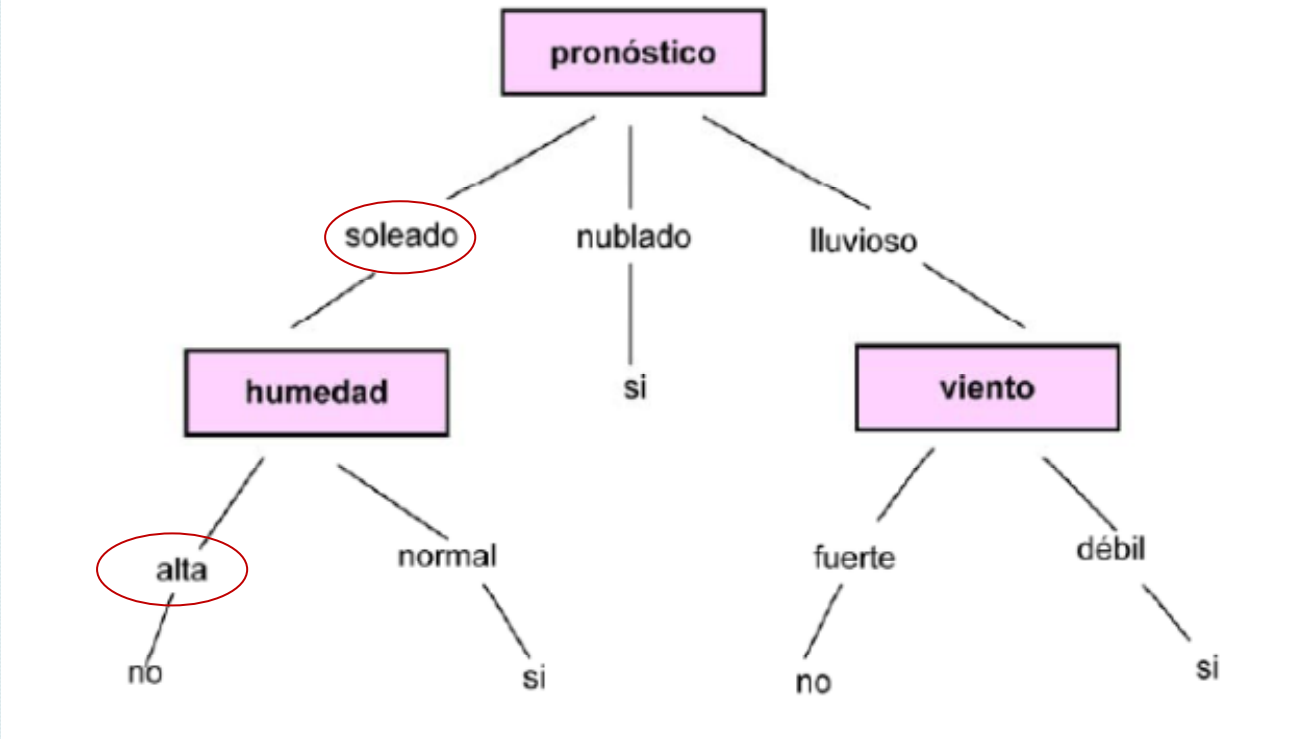
Consideremos la tarea de aprender el concepto objetivo:

"sábados en los cuales Pepe juega al tenis por la mañana"

- El atributo objetivo es **jugar-al-tenis,** el cual puede tomar los valores "si" o "no" para distintos sábados por la mañana y puede predecirse basado en otros atributos de la mañana en cuestión: **pronóstico**, **temperatura**, **humedad** y **viento**

Día	pronóstico	temperatura	humedad	viento	Juega al tenis
D1	soleado	cálido	alta	débil	no
D2	soleado	cálido	alta	fuerte	no
D3	nublado	cálido	alta	débil	si
D4	lluvioso	templado	alta	débil	si
D5	lluvioso	frío	normal	débil	si
D6	lluvioso	frío	normal	fuerte	no
D7	nublado	frío	normal	fuerte	si
D8	soleado	templado	alta	débil	no
D9	soleado	frío	normal	débil	si
D10	lluvioso	templado	normal	débil	si
D11	soleado	templado	normal	fuerte	si
D12	nublado	templado	alta	fuerte	si
D13	nublado	cálido	normal	débil	si
D14	lluvioso	templado	alta	fuerte	no

- La figura muestra un ejemplo de árbol de decisión en el cual clasifica los sábados a la mañana de acuerdo a si el clima es adecuado para jugar al tenis



Por ejemplo, la instancia

<pronóstico = soleado, temperatura = cálida, humedad = alta, viento = fuerte>

debería ordenarse hacia abajo desde la rama más hacia la izquierda del árbol de decisión y por lo tanto debería clasificarse cómo una instancia negativa (i.e. el árbol predice **jugar-al-tenis** = no).

- En general, los árboles de decisión representan una disyunción de conjunciones de restricciones de los valores atributos de las instancias.

- Cada camino desde la raíz del árbol a la hoja corresponde a una conjunción de tests de atributos, y la hoja en sí misma una disyunción de esas conjunciones.

- Por ejemplo, el árbol de decisión de la figura corresponde a la expresión

*(***pronóstico** = *soleado* ^ **humedad** = *normal)*

v *(***pronóstico** = *nublado)*

v *(***pronóstico** = *lluvia* ^ **viento** = *débil)*

(casos en que las instancias son positivas)

- Una ventaja de los árboles de decisión es la interpretabilidad.

- Veremos que los árboles se pueden convertir en un conjunto de reglas SI-ENTONCES que se entienden fácilmente.

Problemas apropiados para el aprendizaje mediante árboles de decisión

- Las instancias son representadas por pares de atributos:
  - Las instancias se describen mediante un conjunto fijo de atributos (por ejemplo, **temperatura**) y sus valores (por ejemplo, cálida).
  - La situación más sencilla es cuando cada atributo toma un pequeño número de valores disjuntos (eg. cálido, templado, frío).
  - Sin embargo, se puede extender el algoritmo para tratar con valores reales (e.g. representando la temperatura numéricamente).

- La función objetivo tiene como output valores discretos:
  - El árbol de decisión de la figura anterior asigna una clasificación booleana (e.g. si o no) a cada ejemplo
  - Los métodos de árboles de decisión pueden extenderse fácilmente para aprender funciones que tienen más de dos posibles valores de output
- Se requieren descripciones disyuntivas
  - Los árboles de decisión representan naturalmente expresiones disyuntivas

- Los datos de entrenamiento no deben contener errores
  - Estos métodos no son robustos a errores tanto en la clasificación de los ejemplos de entrenamiento como en los errores de los valores de los atributos que describen esos ejemplos
- Los ejemplos de entrenamiento pueden tener valores faltantes de los atributos
  - Los métodos de árboles de decisión pueden usarse aún cuando algunos ejemplos entrenamiento tengan valores desconocidos (e.g. si la humedad del día se conoce sólo para algunos de los ejemplos de entrenamiento)

- Algunos problemas que se pueden resolver mediante el aprendizaje de árboles de decisión son:

- clasificar a pacientes por sus enfermedades

- clasificar equipos que funcionan mal por sus causas,

o clasificar pedidos de préstamos por sus fallas en los pagos.

- Tales problemas en los que la tarea es clasificar ejemplos en un conjunto discreto de categorías posibles se denominan problemas de clasificación.

Árboles univariados

- En un árbol univariado, en cada nodo interno, se usa el test sólo en una de las dimensiones del input.

- Si la dimensión del input usada x_j es discreta, y toma uno de n valores posibles, el nodo de decisión verifica el valor de x_j y sigue la rama correspondiente, implementando una división en n formas.

- Por ejemplo, si un atributo es **color** $\in$ {rojo, azul, verde}, entonces un nodo de este atributo tiene 3 ramas, cada una correspondiente a los distintos colores.

- Un nodo de decisión tiene ramas discretas y un input continuo debería ser discretizado.

  - Si x_j es continuo, el test es una comparación

$$f_m(\boldsymbol{x}): x_j > w_{m_0} \quad (1)$$

    donde w_{m_0} es un valor apropiado elegido para el umbral.

- El espacio de decisión divide al espacio en dos:

$$L_m = \{\boldsymbol{x} \mid x_j > w_{m_0}\} \quad \text{y} \quad R_m = \{\boldsymbol{x} \mid x_j \leq w_{m_0}\}$$

  - esto se llama una división binaria.

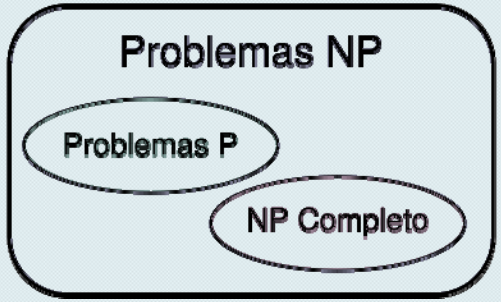
- Los sucesivos nodos de decisión en el camino desde la raíz hacia una hoja dividen a estos dos usando otros atributos.

- Para un conjunto de entrenamiento dado, existen muchos árboles que se pueden codificar sin error, y por simplicidad, estamos interesados en encontrar el más pequeño entre ellos, donde la medida está dada por el número de nodos en el árbol y la complejidad de los nodos de decisión.

- Encontrar el menor árbol es NP-completo y estamos forzados a usar procedimientos de búsqueda heurísticos de búsqueda local para encontrar los árboles en un tiempo razonable.

  o Un problema de decisión *C* es NP-completo si:

  1. *C* es un problema NP (no polinomial)
  2. Todo problema de NP se puede transformar polinómicamente en *C*.

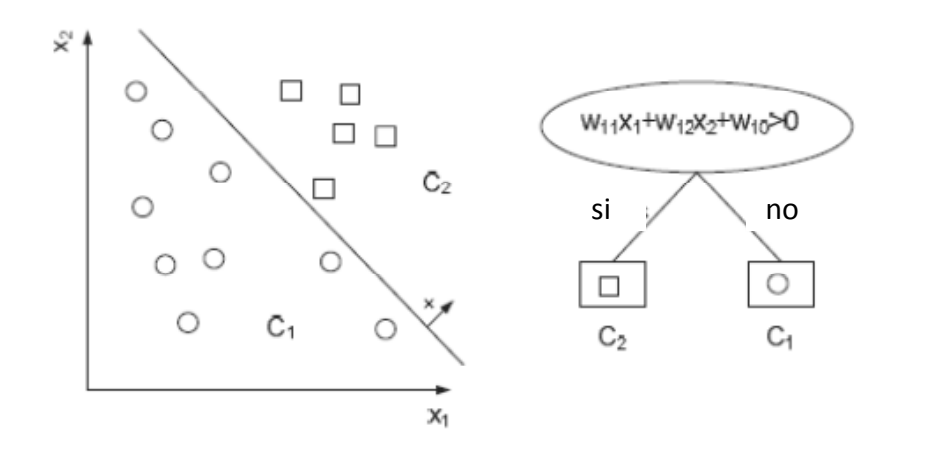


Árboles multivariados

- En un árbol multivariado todas las dimensiones del output se usan para las divisiones.

- Cuando todos los inputs son numéricos, un nodo lineal multivariado se define como

$$f_m(\boldsymbol{x}):\ \boldsymbol{w}_m^t \boldsymbol{x} + w_{m_0} > 0$$



- Como el nodo multivariado es una suma pesada, los atributos discretos podrían representarse con variables numéricas 0/1.

El algoritmo básico de aprendizaje mediante árboles de decisión

- El algoritmo básico, ID3, aprende árboles de decisión construyéndolos de arriba hacia abajo, comenzando con la pregunta:

  ¿qué atributos deben testearse en la raíz del árbol?

- Para contestar esta pregunta, se evalúa cada instancia del atributo usando un test estadístico para determinar cuán bien clasifica por ella sola los ejemplos de entrenamiento.

- Se selecciona el mejor atributo y se usa como test en el nodo raíz del árbol

- Se crea un descendiente del nodo raíz para cada valor posible de ese atributo, y se ordenan los ejemplos de entrenamiento en el nodo apropiado

- Se repite el proceso entero usando los ejemplos de entrenamiento asociados con cada nodo descendiente para elegir el mejor atributo para tratar en ese punto del árbol.

- Esto constituye una búsqueda codiciosa (greedy) para obtener un árbol de decisión aceptable, en el cual el algoritmo nunca vuelve para atrás para reconsiderar decisiones anteriores.

Algoritmo ID3

Input:

- **ejemplos**: son los ejemplos de entrenamiento
- **atributo_objetivo**: es el atributo cuyos valores van a ser predichos por el árbol
- **atributos**: es una lista de otros atributos que pueden testearse por el árbol de decisión aprendido

Output:

- devuelve un árbol de decisión que clasifica correctamente los ejemplos dados

ID3 (**ejemplos**, **atributo_objetivo**, **atributos**)

Crear un nodo **raíz** para el árbol

a. Si todos los ejemplos son positivos, devolver un árbol de un único nodo, **raíz,** con rótulo = +

b. Si todos los ejemplos son negativos, devolver un árbol de un único nodo **raíz,** con rótulo = **-**

c. Si los atributos están vacíos, devolver un árbol de un único nodo **raíz,** con rótulo = el valor más frecuente del **atributo_objetivo** en **ejemplos**

d. En otro caso: comenzar

i. $A \leftarrow$ el atributo entre los **atributos** que mejor clasifique a **ejemplos**

ii. el atributo decisión para **raíz** $\leftarrow A$

iii. para cada valor posible v_i de A,

a. agregar una nueva rama debajo de **raíz,** correspondiente al test $A = v_i$

b. hacer **ejemplos**$_{v_i}$ el mejor subconjunto de **ejemplos** que tiene el valor v_i para A

c. si **ejemplos**$_{v_i}$ está vacío

- entonces

  debajo de esta nueva rama, agregar un nodo hoja con rótulo = valor más frecuente del **atributo_objetivo** en **ejemplos**

  en caso contrario,

  debajo de esta nueva rama, agregar un subárbol

  ID3(**ejemplos**$_{v_i}$, **atributo_objetivo**, **atributos –** $\{A\}$)

e. terminar

f. devolver **raíz**

¿Cuál es el atributo que clasifica mejor?

- La elección central de este algoritmo es elegir el atributo para testear cada nodo del árbol.
- Se quiere elegir el atributo que sea más útil para clasificar los ejemplos.
- ¿Cuál es una medida de la bondad de un atributo?
- Definimos una propiedad estadística, llamada *ganancia de información*, que mide cuán bien un atributo separa los ejemplos de entrenamiento de acuerdo a la meta de la clasificación.

- IE3 usa esta medida de la ganancia de información para seleccionar entre los atributos candidatos en cada paso mientras va creciendo el árbol.

La entropía mide la homogeneidad de los ejemplos

- Para definir la ganancia de información con precisión, comencemos definiendo una medida que se usa en teoría de la información, llamada entropía, que caracteriza la (im)pureza de una colección arbitraria de ejemplos.

- Una división es pura si después de la división, para todas las ramas, todas las instancias que eligen una rama pertenecen a la misma clase.

- Supongamos que para el nodo m, N_m es el número de instancias de entrenamiento que llegan a dicho nodo.
  - Para el nodo raíz es n.

- N_m^i es el número de instancias de entrenamiento que llegan al nodo m y pertenecen a la clase C_i, con $\sum_{i=1}^{k} N_m^i = N_m$.

- Dada una instancia que llega al nodo m, el estimador para la probabilidad de la clase C_i es

$$\hat{P}(C_i|x,m) \equiv p_m^i = \frac{N_m^i}{N_m}$$

- El nodo m es puro si p_m^i para todo i es 0 ó 1.
  - Es 0 si ninguna de las instancias que llegan al nodo m es de la clase C_i
  - Es 1 si todas las instancias que llegan al nodo m son de la clase C_i
- Si la división es pura, no es necesario seguir separando y se puede agregar un nodo hoja rotulado con la clase para la cual $p_m^i = 1$
- Una posible función para medir la impureza es la entropía

$$\mathcal{I}_m = -\sum_{i=1}^{k} p_m^i \log_2 p_m^i \quad (2)$$

donde vamos a considerar que $0 \log 0 \equiv 0$.

Otras medidas de la impureza de una división

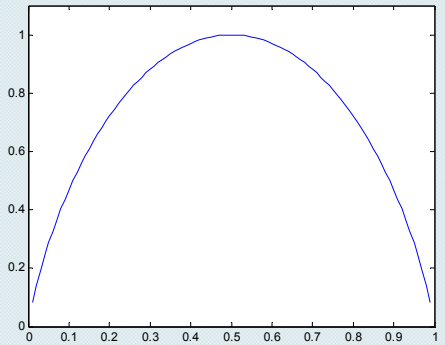
- Para el problema de dos clases, donde $p^1 \equiv p$ y $p^2 \equiv 1 - p$, $\phi(p, 1-p)$ es una función no negativa que mide la impureza de una división, si satisface las siguientes propiedades:

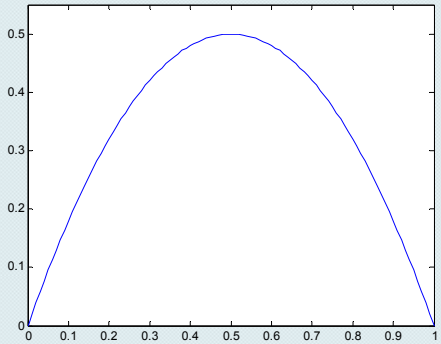
1. $\phi\left({}^1/_2, {}^1/_2\right) \geq \phi(p, 1-p) \quad \forall p \in [0,1]$
2. $\phi(0,1) = \phi(1,0)$
3. $\phi(p, 1-p)$ es creciente en p en $\left[0, {}^1/_2\right]$ y decreciente en $\left[{}^1/_2, 1\right]$

- Algunas funciones posibles son:

1. Entropía

$$\phi(p, 1-p) = -p \log_2(p) - (1-p) \log_2(1-p)$$



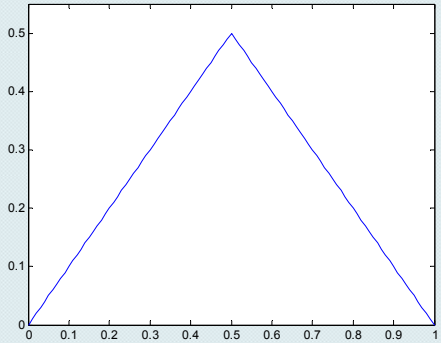


2. Índice de Gini

$$\phi(p, 1-p) = 2p(1-p)$$

3. Error de mala clasificación

$$\phi(p, 1-p) = 1 - \max(p, 1-p)$$



Ejemplo (**jugar-al-tenis**)

- Supongamos que tenemos una colección de 14 ejemplos de algún concepto booleano, que incluye 9 ejemplos positivos y 5 negativos (adoptaremos la notación [9+,5-] para resumir este ejemplo).

- Entonces la entropía relativa a su clasificación booleana es:

$$\mathcal{I}([9+,5-]) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

- Una interpretación de la entropía a partir de la teoría de la información es que especifica el número mínimo de bits de información necesaria para codificar el código de clase de una instancia.

  - En un problema de dos clases, si $p^1 = 1$ y $p^2 = 0$, todos los ejemplos son de C_1, y no es necesario enviar nada y la entropía es 0.

  - Si $p^1 = p^2 = 0.5$, es necesario enviar un bit de señal 1 de uno de los 2 casos, y la entropía es 1.

  - Entre estos dos extremos, podemos crear código y usar menos de un bit por mensaje teniendo códigos más cortos para la clase más probable y más largos para la clase menos probable.

- Cuando hay más de $k > 2$ clases, ocurre algo similar y la mayor entropía es $\log_2 k$ cuando $p^i = {}^1/_k$.

- Si el nodo m no es puro, entonces las instancias pueden dividirse para disminuir la impureza, y hay múltiples posibles atributos en los cuales puede dividirse.

- Para un atributo numérico, hay múltiples posiciones posibles de división.

  - Entre todas ellas buscamos la división que minimiza la impureza después de la división porque queremos generar los árboles más pequeños.

- Si los subconjuntos después de la división están más cercanos a ser puros, serán necesarios pocas (o ninguna) divisiones después.

- Por supuesto, esto es un óptimo local, y no hay garantías de haber encontrado el árbol de decisión más pequeño.

- Supongamos que en el nodo m, N_{m_j} de N_m toma la rama j

  - o estas son las $\boldsymbol{x}^l$ para las cuales el test $f_m(\boldsymbol{x}^l)$ da como output j.

    - Para un atributo discreto con n valores, hay n resultados
    - Para un atributo continuo, hay dos resultados ($n = 2$)

  - o en ambos casos se satisface $\sum_{j=1}^{n} N_{m_j} = N_m$

- $N_{m_j}^i$ de N_{m_j} pertenece a la clase C_i: $\sum_{i=1}^{k} N_{m_j}^i = N_{m_j}$
- Análogamente $\sum_{j=1}^{n} N_{m_j}^i = N_m^i$
- Luego, dado que en el nodo m, el test devuelve el output j, el estimador de la probabilidad de la clase C_i es:

$$\hat{P}(C_i|x, m, j) \equiv p_{m_j}^i = \frac{N_{m_j}^i}{N_{m_j}}$$

y el total de impurezas después de la división está dado por:

$$\mathcal{I}_m^{'} = -\sum_{j=1}^{n} \frac{N_{m_j}^i}{N_{m_j}} \sum_{i=1}^{k} p_{m_j}^i \log_2 p_{m_j}^i \quad (3)$$

- En el caso de un atributo continuo, para calcular $p_{m_j}^{i}$ usando la ecuación (1) se necesita conocer w_{m_0} para ese nodo.

- En el caso de un atributo discreto, no es necesaria esta iteración.

- Así para todos los atributos, discretos y continuos, se calcula la impureza y se elige aquella que tiene mínima entropía.

- Luego la construcción del árbol continúa recursivamente y en paralelo para todas las ramas que no son puras, hasta que sean todas puras.

- Se puede decir también que en cada paso durante la construcción del árbol, se elige la división que produce el mayor decrecimiento en la impureza de los datos alcanzando el modo m (ecuación (2)) y el total de entropía de los datos alcanzando todas las ramas después de la división (ecuación (3)).

- Cuando hay ruido, hacer crecer al árbol hasta que haya alcanzado su mayor pureza, puede hacerlo crecer demasiado y hacer que sobrestime

    - Por ejemplo, consideremos el caso de rotular mal una instancia dentro de un grupo de instancias bien rotuladas.

- Para paliar tal sobrestimación, la construcción del árbol termina cuando un nodo es lo suficientemente puro, digamos, un subconjunto de los datos no se divide más si $\mathcal{I} < \theta_I$.

- Esto implica que si no se requiere que $p_{m_j}^i$ sea exactamente 0 ó 1 pero esté cerca, con un umbral θ_p.

- En este caso, se crea un modo hoja y se lo rotula con la clase que tenga el valor más alto de $p_{m_j}^i$.

o θ_I (o θ_p) es un parámetro de complejidad

- Cuando son pequeños, la varianza es alta y el árbol crece para reflejar el conjunto de entrenamiento con precisión, y cuando son grandes, la varianza es menor y se puede representar aproximadamente al conjunto de datos de entrenamiento con un árbol menor y puede tener mayor sesgo.

- El valor ideal depende tanto del costo de mala clasificación como del costo en memoria y cómputo.

Ganancia de información medida como reducción en la entropía esperada

- Dada la entropía como una medida de la impureza en una colección de ejemplos de entrenamiento, podemos definir una medida de la efectividad de un atributo en clasificar a los datos de entrenamiento.

- La medida que usaremos, llamada *ganancia de información*, es simplemente la reducción en la entropía causada al dividir los ejemplos de acuerdo a este atributo.

- La ganancia de información, ganancia(S, A) de un atributo A relativa a una colección de ejemplos S, se define como:

$$\text{ganancia}(S, A) \equiv \mathcal{I}(S) - \sum_{v \in \text{valores}(A)} \frac{|S_v|}{|S|} \mathcal{I}(S_v)$$

donde:

  - o valores(A): conjunto de todos los posibles valores del atributo *A*
  - o S_v: subconjunto de *S* para el cual el atributo *A* tiene valor *v* (i.e. $S_v = \{s \in S \mid A(s) = v\}$)

- Notemos que el primer término de la ecuación es la entropía del conjunto original *S*, y el segundo término es el valor esperado de la entropía después de que se divide a *S* usando el atributo *A*.

- La entropía esperada descripta por el segundo término es simplemente la suma de las entropías de cada subconjunto S_v, pesadas por la fracción de ejemplos $|S_v|/|S|$ que pertenece a S_v.

- ganancia(S, A) es por lo tanto la reducción esperada en la entropía causada por saber el valor del atributo *A*.

  - El valor de ganancia(S, A) es el número de bits que se ahorran cuando se codifica el valor objetivo de un miembro arbitrario de *S*, sabiendo el valor del atributo *A.*

Ejemplo (**jugar-al-tenis**)

- *S* es una colección de ejemplos de entrenamiento descripta por atributos que incluyen **viento**, que puede tener los valores *fuerte* o *débil.*

- Dicha colección consta de 14 ejemplos, [9+,5-].

- De esos 14 ejemplos, supongamos que 6 de los positivos y 2 de los negativos tienen **viento** = *débil* y los restantes **viento** = *fuerte*.

- La información ganada al ordenar los 14 ejemplos originales con el atributo **viento**, se puede calcular como:

$$\text{valores}(\mathbf{viento}) = débil, fuerte$$

$$S = [9+, 5-]$$

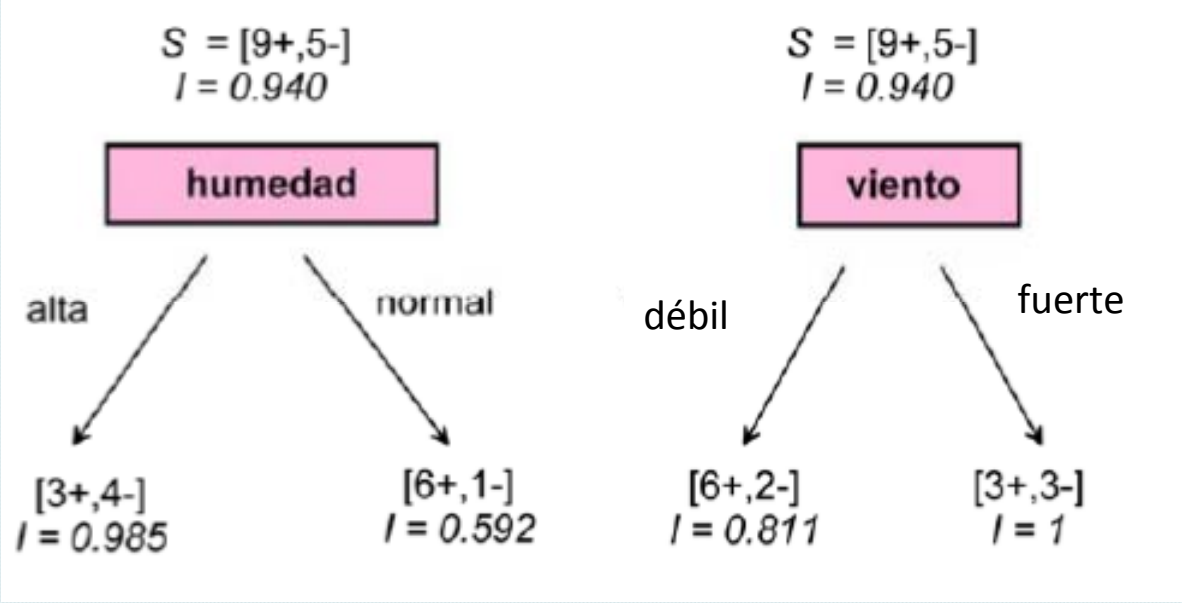
$$S_{débil} = [6+, 2-]$$

$$S_{fuerte} = [3+, 3-]$$

$$\begin{aligned}
\text{ganancia}(S, \textbf{viento}) &= \mathcal{I}(S) - \sum_{v \in [débil, fuerte]} \frac{|S_v|}{|S|} \mathcal{I}(S_v) \\
&= \mathcal{I}(S) - \frac{8}{14} \mathcal{I}(S_{débil}) - \frac{6}{14} \mathcal{I}(S_{fuerte}) \\
&= 0.940 - \frac{8}{14} * 0.811 - \frac{6}{14} * 1 = 0.048
\end{aligned}$$

- La ganancia de información es precisamente la medida usada por ID3 para seleccionar el mejor atributo en cada paso de crecimiento del árbol.

- El uso de información ganada para evaluar la relevancia de atributos se resume en la figura



- Consideremos el primer paso del algoritmo, en el cual se crea el nodo más alto del árbol de decisión.

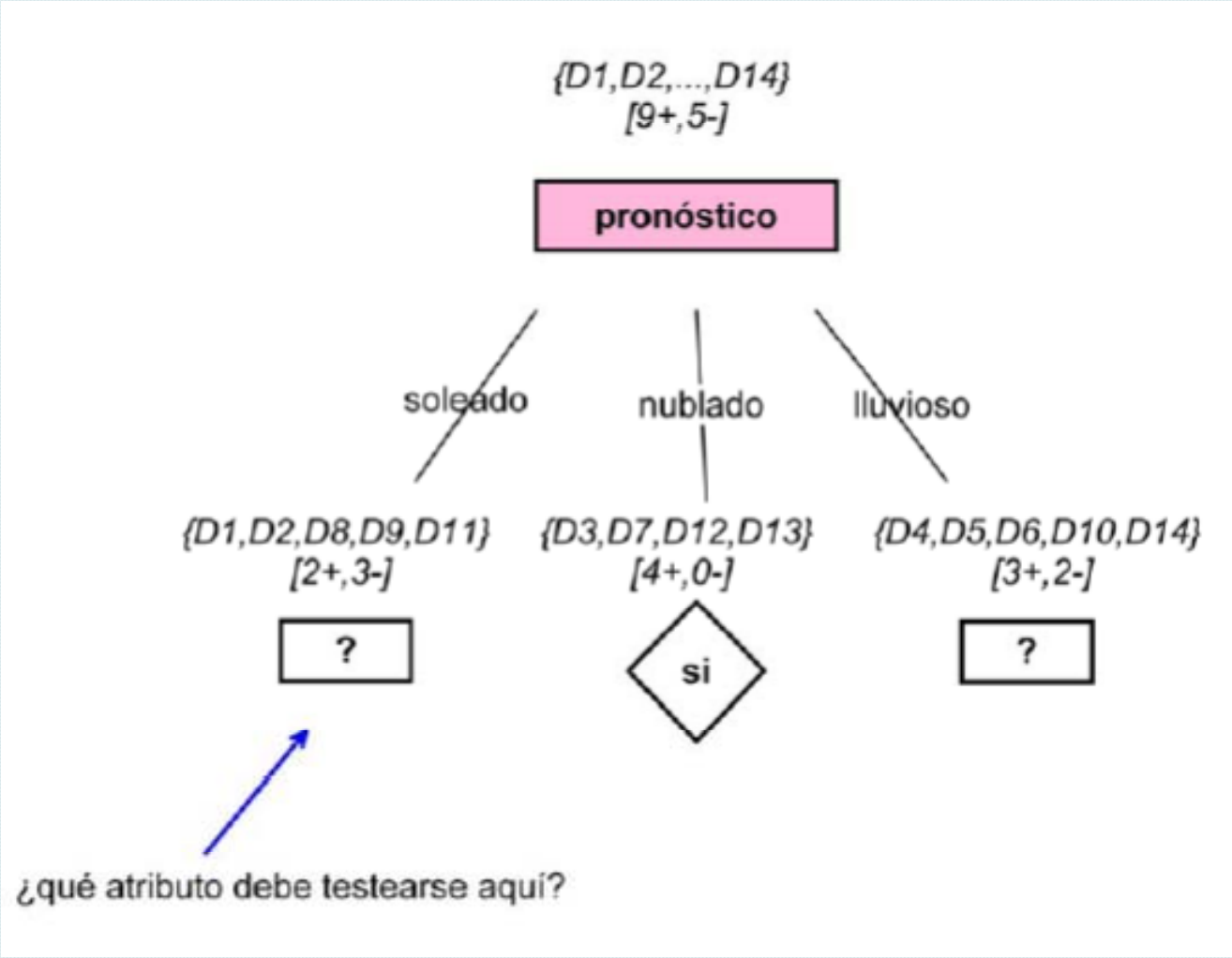
  - ¿Cuál es el atributo que debería testearse primero?

  - ID3 determina la ganancia de información de cada atributo candidato (i.e. **pronóstico**, **temperatura**, **humedad** y **viento**) entonces selecciona el que produce la mayor ganancia de información.

- La ganancia de información para los cuatro atributos es:

$$\text{ganancia}(S, \mathbf{pronóstico}) = 0.246$$
$$\text{ganancia}(S, \mathbf{humedad}) = 0.151$$
$$\text{ganancia}(S, \mathbf{viento}) = 0.048$$
$$\text{ganancia}(S, \mathbf{temperatura}) = 0.029$$

- De acuerdo con la medida de la ganancia de información, el **pronóstico** proporciona el mejor predictor del atributo objetivo, **jugar-al-tenis** sobre los ejemplos de entrenamiento.
  - Por lo tanto **pronóstico** se elige como el atributo de decisión para el nodo raíz, y se crean ramas debajo de la raíz para cada uno de sus posibles valores (soleado, nublado y lluvioso).



$S_{\text{soleado}} = \{D1, D2, D8, D9, D11\}$

$$\text{ganancia}(S_{\text{soleado}}, \textbf{humedad}) = 0.970 - \frac{3}{5} * 0 - \frac{2}{5} * 0 = 0.970$$

$$\text{ganancia}(S_{\text{soleado}}, \textbf{temperatura}) = 0.970 - \frac{2}{5} * 0 - \frac{2}{5} * 1 - \frac{1}{5} * 0$$
$$= 0.570$$

$$\text{ganancia}(S_{\text{soleado}}, \textbf{viento}) = 0.970 - \frac{2}{5} * 1 - \frac{3}{5} * 0.918 = 0.019$$

- Notemos que todos los ejemplos con **pronóstico** = nublado son ejemplos positivos de **jugar-al-tenis**
    - Por lo tanto, este nodo del árbol se convierte en un nodo hoja con la clasificación **jugar-al-tenis** = si.

- En contraste, los descendientes correspondientes a:

  pronóstico = soleado y **pronóstico =** lluvioso

  tienen todavía entropía no nula, y el árbol de decisión debe continuar debajo de esos nodos.

- El proceso de elegir un nuevo atributo y dividir a los ejemplos de entrenamiento se repite ahora para cada nodo descendiente que no sea terminal, esta vez usando sólo los ejemplos de entrenamiento asociados con ese nodo.

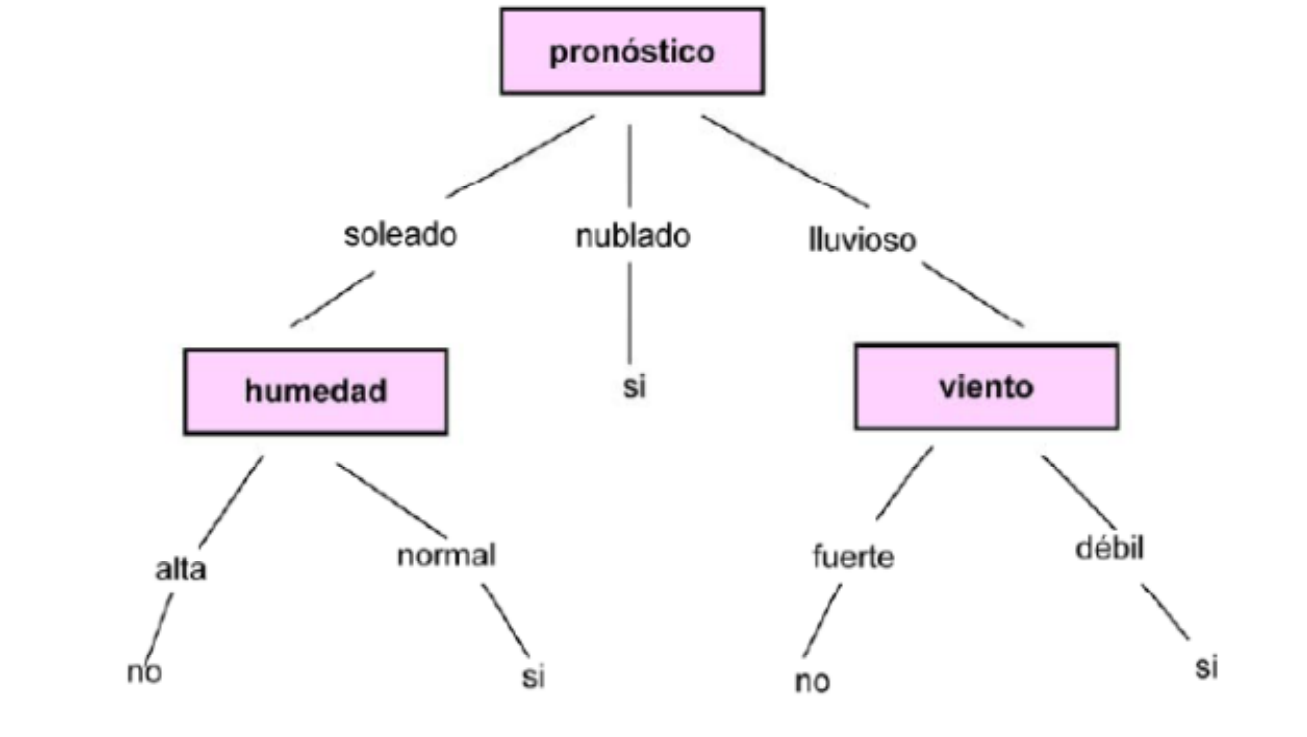
- Los atributos que se incorporaron más arriba en el árbol se excluyen, de modo que cualquier atributo puede aparecer a lo sumo una vez en cualquier camino del árbol.

- Este proceso continúa para cada nuevo nodo hoja hasta que se cumple alguna de las siguientes dos condiciones:

  (1) cada atributo ya ha sido incluído en algún camino del árbol

  (2) los ejemplos de entrenamiento asociados con este nodo hoja tienen todos el mismo valor del atributo objetivo
  (i.e. sus entropías son nulas)

El árbol final de decisión aprendido mediante el ID3 se muestra en la figura



Evitando el sobreajuste de los datos

- El algoritmo descripto anteriormente hace crecer cada rama del árbol la profundidad suficiente como para clasificar perfectamente a los ejemplos de entrenamiento.

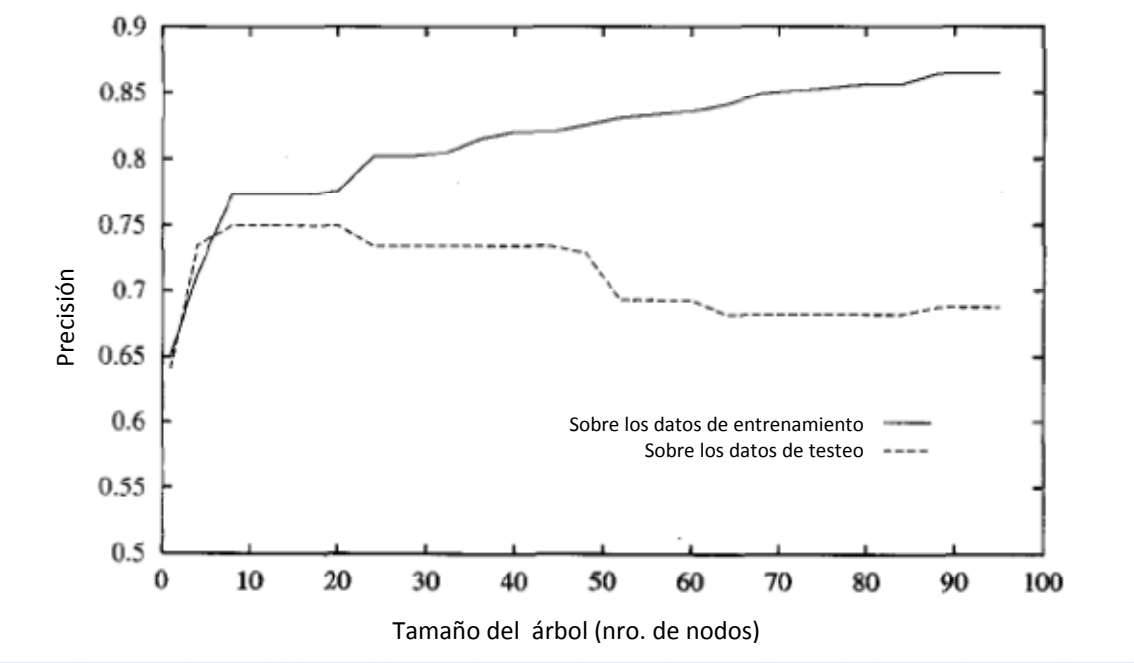
- Mientras que algunas veces es una estrategia razonable, puede ser inexacta cuando hay ruido en los datos, o cuando el número de ejemplos de entrenamiento es muy pequeño para producir una muestra representativa de la verdadera función objetivo.

- En cualquiera de los casos el algoritmo puede producir un sobre ajuste de los datos de entrenamiento.

- Diremos que una hipótesis sobre-ajusta los datos de entrenamiento si cualquier otra hipótesis que ajusta los datos de entrenamiento no tan bien como esa tiene una performance mejor sobre toda la distribución de las instancias
  (i.e. incluyendo las instancias más allá del conjunto de entrenamiento).

Definición:
Dado un espacio de hipótesis *H*, se dice que una hipótesis $h \in H$ **sobre-ajusta** los datos de entrenamiento si existe alguna hipótesis alternativa $h' \in H$, tal que h tiene menor error que h' sobre los ejemplos de entrenamiento, pero h' tiene menor error que h sobre toda la distribución de las instancias.

- La figura ilustra el impacto de sobre ajustar una aplicación típica de árboles decisión.



- En este caso se aplica el algoritmo ID3 a la tarea de aprender qué pacientes tienen cierto tipo de diabetes.

- El eje horizontal del plot indica el número total de nodos en el árbol de decisión.

- Se observa que a medida que crece el árbol, la precisión crece monótonamente.

- Sin embargo, la precisión medida sobre un conjunto independiente de ejemplos de testeo primero crece y luego decrece.

- Como se observa, una vez que el árbol excede los 25 nodos, el hacer crecer más al árbol provoca que decrezca su precisión sobre los ejemplos de testeo a pesar de que aumenta la precisión sobre el conjunto de entrenamiento.

- ¿Cómo se puede ser que el árbol *h* ajuste los ejemplos de entrenamiento mejor que *h'*, pero que el desempeño sea peor sobre los ejemplos subsiguientes?

- Una forma de que esto ocurra es cuando los datos de entrenamiento contienen errores aleatorios o ruido.

Ejemplo (**jugar-al-tenis**)

- Consideremos el efecto de agregar el siguiente ejemplo de entrenamiento positivo, que fue incorrectamente rotulado como negativo

Día	pronóstico	temperatura	humedad	viento	juega al tenis
D15	soleado	alta	normal	fuerte	no

- Dados los datos originales sin errores, el algoritmo ID3 produce el árbol de decisión que vimos

- Sin embargo, el agregado de este ejemplo incorrecto causará que el ID3 construya un árbol más complejo.

- En particular, el nuevo ejemplo será ordenado en el segundo nodo hoja de la izquierda del árbol aprendido de la figura junto con los ejemplos positivos previos D9 y D11.

- Como este ejemplo está rotulado como negativo, ID3 buscará más refinamientos del árbol por debajo de este nodo.

- El resultado es que ID3 dará como output un árbol de decisión *h* que será más complejo que el árbol original (*h'*)
- Por supuesto, *h* ajustará a los ejemplos de entrenamiento perfectamente, mientras que el más simple *h'* no lo hará.

- Sin embargo, dado que el nuevo nodo de decisión es simple debido al ajuste de un ejemplo de entrenamiento con ruido, se espera que *h* tenga un mejor desempeño que *h'* sobre datos subsiguientes que correspondan a la misma distribución de las instancias.

- Hay varios enfoques para evitar el sobre-ajuste en los árboles de decisión. Se agrupan en dos clases:

  - aproximaciones que paran el crecimiento del árbol antes de que alcance el punto donde clasifican perfectamente los datos de entrenamiento
  - aproximaciones que permiten el sobre-ajuste de los datos, y después hacen una post poda del árbol

- A pesar de que la primera aproximación puede ser más directa, la segunda es más exitosa en la práctica.
  Esto se debe a la dificultad de la primera aproximación de determinar precisamente cuando detener el crecimiento del árbol

- También aparece la cuestión de cuál es el criterio para determinar el tamaño final del árbol correcto.
  Las aproximaciones incluyen:

  - usar un conjunto separado de ejemplos, distinto de los ejemplos de entrenamiento, para evaluar la utilidad de la post poda de nodos del árbol
    - o usar los datos disponibles para entrenamiento, pero aplicar un test estadístico para estimar si la expansión (o po-

da) de un nodo particular producirá una mejora más allá del conjunto de entrenamiento

- usar una medida explícita de la complejidad para codificar los ejemplos de entrenamiento y el árbol de decisión, deteniendo el crecimiento del árbol cuando el tamaño de codificación se minimice.

- El primer método es el más común y se conoce generalmente como entrenamiento y validación.

- En esta aproximación, los datos disponibles se dividen en dos conjuntos: un conjunto de *entrenamiento*, que se usa para aprender la hipótesis, y un conjunto separado de *validación*, que se usa para evaluar la precisión de esta hipótesis sobre los datos subsiguientes, y en particular para evaluar el impacto de la poda de esta hipótesis.

- Es importante que el conjunto de validación sea lo suficientemente grande como para ser una muestra estadísticamente significativa de las instancias.

Poda

- Frecuentemente, un nodo no se divide más si el número de instancias de entrenamiento que alcanzan al nodo es menor que un cierto porcentaje del conjunto de entrenamiento, por ejemplo 5%, sin importar la impureza del error.

- La idea es que cualquier decisión basada en tan pocas instancias causa varianza y por lo tanto generalización del error.

- Detener la construcción del árbol antes de que esto ocurra se llama pre-poda del árbol.

- Otra posibilidad de lograr árboles simples es la post-poda, que en la práctica funciona mejor que la pre-poda.

- En la post-poda:
  - hacemos crecer al árbol hasta que todas las hojas son puras y no hay error de entrenamiento.
  - encontramos luego los subárboles que causan sobre-ajuste y los podamos.
  - para el conjunto rotulado inicial, dejamos de lado un conjunto de poda, que no se ha usado durante el entrenamiento.
  - para cada subárbol, se lo reemplaza con un nodo hoja rotulado con las instancias de entrenamiento cubiertas por el subárbol.

- si la performance del nodo hoja no es peor que la del subárbol del conjunto de poda:
  - se poda el subárbol y se mantiene el nodo hoja porque la complejidad adicional del subárbol no está justificada;

  en caso contrario:

  - se mantiene el subárbol.

- En la práctica, un método útil de encontrar hipótesis con mayor precisión es una técnica conocida como post-poda.

- Una variante de este método es el C4.5, el cual se basa en el algoritmo ID3 original.

Algoritmo C4.5

1. Inferir el árbol de decisión del conjunto de entrenamiento, haciendo crecer al árbol hasta que los datos ajusten los datos tan bien como sea posible y permitiendo el sobreajuste cuando ocurra

2. Convertir al árbol aprendido en un conjunto equivalente de reglas creando una regla para cada camino del árbol del nodo raíz a un nodo hoja

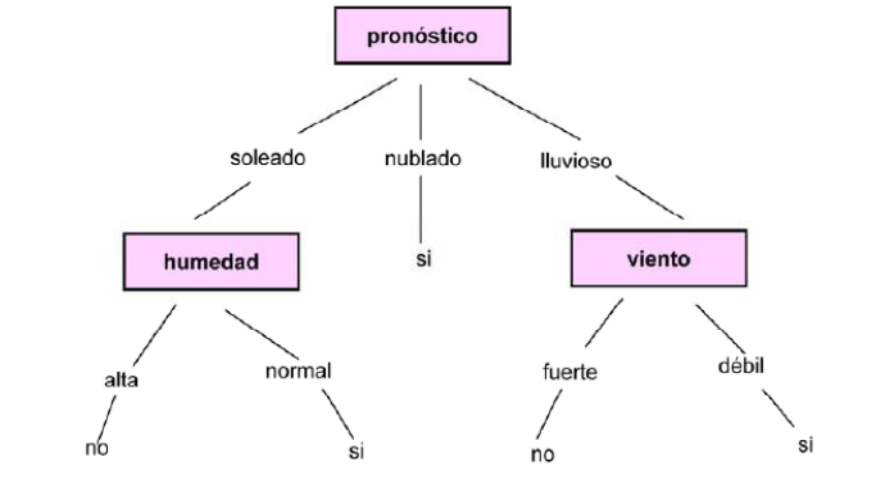
3. Podar (generalizar) cada regla removiendo cualquier precondición que resulte en un mejoramiento de su precisión estimada

4. Ordenar las reglas podadas por su precisión estimada, y considerarlas en la secuencia de clasificación cuando se clasifiquen instancias subsecuentes.

Ejemplo (**jugar-al-tenis**)

- En la post-poda, se genera una regla por cada nodo hoja del árbol.

- Cada atributo testeado a lo largo del camino desde la raíz hasta la hoja se convierte en una regla antecedente (pre condición) y la clasificación del nodo hoja se convierte en una regla consecuente (post condición).



- Por ejemplo, el camino más hacia la izquierda del árbol se traduce en la regla

SI $(\textbf{pronóstico} = \text{soleado}) \wedge (\textbf{humedad} = \text{alta})$

ENTONCES **jugar-al-tenis** = no

- Luego, tal regla es podada removiendo cualquier antecedente, o pre condición, tal que su remoción no empeore la precisión estimada.

- Dada la regla anterior, por ejemplo, la regla de post-poda consideraría remover las precondiciones $(\textbf{pronóstico} = \text{soleado})$ y $(\textbf{humedad} = \text{alta})$

- Se debería seleccionar cuáles de estos pasos de poda producen el mayor mejoramiento en la precisión de la regla estimada, luego se considera podar la segunda pre condición en un siguiente paso de poda.

- No se poda si la precisión de la regla estimada disminuye.

- Como hemos visto, un método para estimar la regla de precisión es usar un conjunto de ejemplos de validación disjunto del conjunto de entrenamiento.

- Otro método es evaluar la performance basada en el conjunto de entrenamiento mismo, usando un estimador pesimista para tener en cuenta que los datos de entrenamiento dan un estimador sesgado a favor de estas reglas.

- Más precisamente, C4.5 calcula un estimador pesimista calculando la precisión de la regla sobre los ejemplos de entrenamiento a los cuales se aplica, luego calcula el desvío estándar de esta precisión estimada suponiendo una distribución binomial.

- Para un nivel de confianza dado, se toma la cota inferior del estimador como medida de performance
  (e.g. para el intervalo de confianza del 95% la precisión de la regla se estima de forma pesimista con la precisión observada sobre el conjunto de entrenamiento, menos 1.96 veces el desvío estándar).

- Para conjunto de datos grandes, este estimador pesimista es cercano a la precisión observada
  (e.g. el desvío estándar es pequeño).

- ¿Por qué se convierten los árboles de decisión en reglas después de la poda?

  Tiene las siguientes ventajas:

  - La conversión en reglas permite distinguir entre diversos contextos en los cuales se usa el nodo de decisión.

    - Debido a que cada camino distinto del árbol de decisión produce una regla distinta, la decisión de poda de acuerdo al atributo testeado puede ser diferente para cada camino.

    - En contraste, si se podase el árbol en sí mismo, habría sólo dos elecciones: remover el nodo de decisión completamente, o conservarlo en su forma original.

- La conversión en reglas remueve la distinción entre atributos testeados que ocurren cerca de la raíz del árbol y aquellos que ocurren cerca de las hojas.
  - Así se evitan confusiones tales como reorganizar el árbol si se poda el nodo raíz mientras se conserva parte del subárbol debajo.
- La conversión en reglas facilita la lectura.
  Las reglas son generalmente más fáciles para entender por las personas.

Incorporando atributos con valores continuos

- El algoritmo ID3 descripto anteriormente está restringido a atributos que toman un conjunto discreto de valores.
  - Primero, el atributo objetivo cuyos valores se predicen por el árbol aprendido deben tener valores discretos.
  - Segundo, los atributos testeados en los nodos de decisión del árbol también deben tener valores discretos.
- Esta segunda restricción puede removerse fácilmente de modo de que puedan incorporarse atributos de decisión continuos al árbol aprendido.

- Esto puede lograrse definiendo nuevos atributos de valores discretos que dividan los valores de los atributos continuos en un conjunto discreto de intervalos.

- En particular, para un atributo *A* que tiene valores continuos, el algoritmo puede crear dinámicamente un nuevo atributo booleano A_c que sea verdadero si $A < c$ y falso en otro caso.

- La única cuestión es cómo elegir mejor valor para el umbral *c.*

Ejemplo (**jugar-al-tenis**)

- Supongamos que queremos incluir un atributo con valores continuos **temperatura** en la descripción de los días del ejemplo de entrenamiento

- Supongamos además que los ejemplos de entrenamiento asociados con un nodo particular en el árbol de decisión tiene los siguientes valores

temperatura	40	48	60	72	80	90
juega al tenis	no	no	si	si	si	no

- ¿Cuál es el atributo booleano que debería definirse basado en **temperatura?**
  Se quiere encontrar el umbral *c* que produzca la mayor ganancia de información.

- Ordenando los ejemplos de acuerdo al atributo continuo *A* e identificando ejemplos adyacentes que difieran en su clasificación objetivo, podemos generar un conjunto de umbrales candidatos a mitad de camino entre los combatientes valores de *A*.

- Se puede probar que el valor de c que maximiza la ganancia de información está en dicha frontera.

- En nuestro ejemplo, hay dos candidatos a umbrales, correspondientes a los valores de temperatura para los cuales el valor de jugar al tenis cambia: (48+60)/2 y (80+90)/2.

- La ganancia de información puede calcularse para cada uno de los candidatos $\mathbf{temperatura}_{>54}$ y $\mathbf{temperatura}_{>85}$

- Se elige el mejor: $\mathbf{temperatura}_{>54}$

- Esto crea dinámicamente un atributo booleano que puede entonces competir con otros atributos con valores discretos en el crecimiento del árbol

Datos faltantes

- En algunos casos, en los datos pueden faltar los valores de algunos atributos.

- Por ejemplo, en caso de que se quieren predecir el alta de pacientes basados en varios test de laboratorio, puede ocurrir que el test **análisis-de-sangre** sólo esté disponible para un subconjunto de pacientes.

- En tal caso es común estimar los valores faltantes de los atributos basándose en otros ejemplos para los cuales el atributo tiene un valor conocido.

- Consideremos la situación en la cual ganancia(S, A) puede calcularse en el nodo *n* en el árbol de decisión para evaluar si el atributo *A* es el mejor atributo para testear ese nodo de decisión.

- Supongamos que $\langle x, c(x) \rangle$ es uno de los ejemplos de entrenamiento de *S* y que el valor $A(x)$ es desconocido.

- Nuestra estrategia para tratar con el valor faltante es asignar el valor que sea más frecuente entre los ejemplos de entrenamiento del nodo *n.*

- Alternativamente, podríamos asignarle el valor más frecuente ejemplos de entrenamiento del nodo *n* que tenga la clasificación $c(x)$.

- El ejemplo de entrenamiento elaborado usando este valor estimado para $A(x)$ puede usarse directamente en el algoritmo del árbol de decisión.