

Programación Lógica y Funcional

Prolog en la IA

Dr. Alejandro Guerra-Hernández

Instituto de Investigaciones en Inteligencia Artificial

Universidad Veracruzana

*Campus Sur, Calle Paseo Lote II, Sección Segunda No 112,
Nuevo Xalapa, Xalapa, Ver., México 91097*

<mailto:aguerra@uv.mx>

<https://www.uv.mx/personal/aguerra/plf>

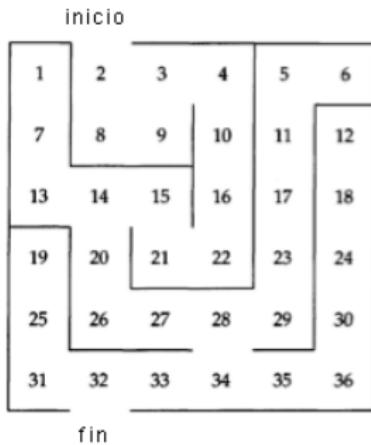
Maestría en Inteligencia Artificial 2025



Universidad Veracruzana

Laberinto

- ▶ ¿Cómo **buscar** la salida de este laberinto?



- ▶ Un **movimiento válido** es ir a una casilla **adyacente**, horizontal o vertical.
- ▶ ¿Cuál será la **estrategia** para salir?



Universidad Veracruzana

Representación de Conocimiento

► Adyacencia:

```
1  conecta(inicio,2).  
2  conecta(1,7).  
3  conecta(2,8).  
4  conecta(2,3). %%% agrega varias soluciones.  
5  conecta(3,4).  
6  conecta(3,9).  
7  conecta(4,10).  
8  ...  
9  conectado(Pos1,Pos2) :- conecta(Pos1,Pos2).  
10 conectado(Pos1,Pos2) :- conecta(Pos2,Pos1).
```



Universidad Veracruzana

Estrategia

- ▶ Una **búsqueda en profundidad**, evitando **ciclos**:

```

1  sol :- 
2      camino([inicio],Sol), write(Sol).
3
4  camino([fin|RestoDelCamino],[fin|RestoDelCamino]).
5  camino([PosActual|RestoDelCamino],Sol) :-
6      conectado(PosActual,PosSiguiente),
7      \+ member(PosSiguiente,RestoDelCamino),
8      camino([PosSiguiente,PosActual|RestoDelCamino],Sol).

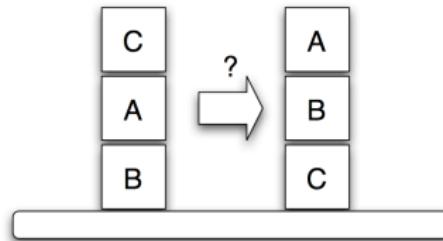
```

- ▶ ¿Qué sucede si eliminamos la línea 7?
- ▶ ¿Dónde estamos representando la meta?



Estados y Acciones

- ▶ ¿Cómo podemos definir un **problema** y sus **soluciones** sistemáticamente de manera declarativa?
- ▶ Ejemplo:



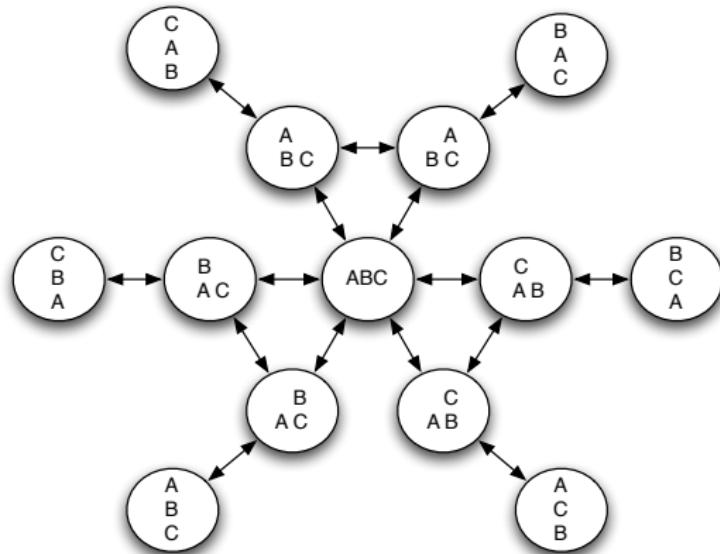
- ▶ Las **acciones** son del tipo `pon_en(a,b)`, etc.
- ▶ Los **estados** son configuraciones de cubos.



Universidad Veracruzana

Espacio de estados de un problema

- ▶ Estados y acciones configuran un **grafo dirigido**:



Universidad Veracruzana

Adaptado de Bratko [1], p. 262.

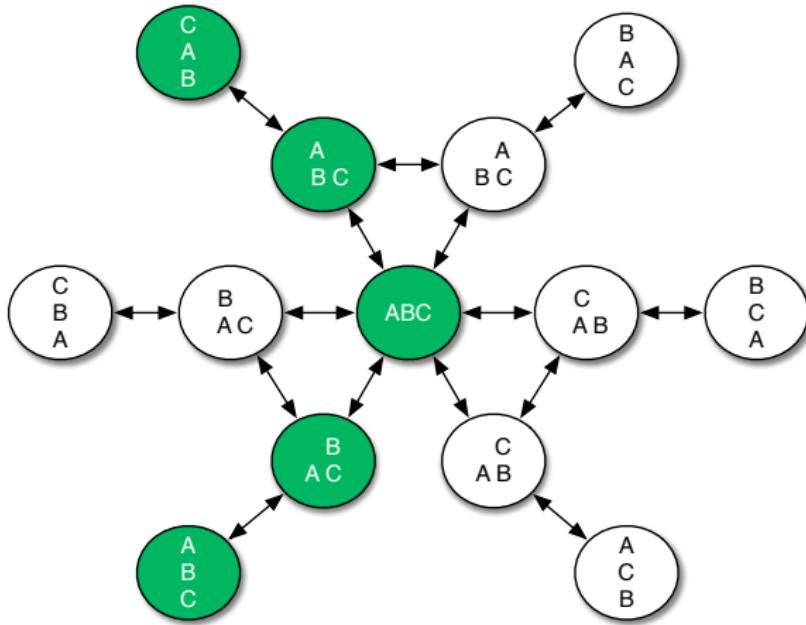
Soluciones en un espacio de estados

- ▶ La solución a un problema dado consiste en encontrar un **camino** desde un nodo representando el estado inicial, hasta el estado final deseado.
- ▶ Cada paso en ese camino está dado por una **acción válida**.
- ▶ ¿Cómo podemos **representar** todo esto en Prolog?



Universidad Veracruzana

Gráficamente



Universidad Veracruzana

Sucesor

- ▶ El espacio de estados se representa mediante la **relación** $s/2$ cuyos argumentos son **estados** del problema.
- ▶ $s(E_0, E_1)$ denota que existe un movimiento o **acción válida** para ir del estado E_0 al estado E_1 .
- ▶ Podemos definir esta relación por **extensión**, p. ej., *conectado/2*, aunque para problemas interesantes esto no suele ser posible.
- ▶ Así, la definición de sucesor suele ser **intensional**.



Universidad Veracruzana

Estados

- ▶ La representación de los estados debe ser **compacta** y favorecer el computo **eficiente** de los sucesores.
- ▶ **Ejemplo.** En el mundo de los cubos representaremos los **estados** como una lista de pilas, que a su vez son listas de cubos.
- ▶ Asumiendo que solo caben tres pilas en la mesa, el **estado inicial** es:
[[c, a, b], [], []]
- ▶ **Nota.** También asumimos que la posición de la pila de interés no es importante, *i.e.*, está a la izquierda, pero podría estar al centro o a la derecha en la mesa.



Universidad Veracruzana

Metas

- ▶ Bajo estos supuestos, la **meta** puede representarse como:
 - ▶ $[[a, b, c], [], []]$, ó
 - ▶ $[], [a, b, c], []$ ó
 - ▶ $[], [], [a, b, c]$.
- ▶ Intencionalmente:

```
1 meta(E) :-  
2     member([a,b,c], E).
```



Universidad Veracruzana

Cómputo de sucesores

- Si quito el *Tope1* de la *Pila1* y lo pongo en la pila *Pila2*, obtengo un sucesor:

```
1 s(Pilas, [Pila1,[Tope1|Pila2]|OtrasPilas]) :-  
2     quitar([Tope1|Pila1], Pilas, Pilas1),  
3     quitar(Pila2, Pilas1, OtrasPilas).  
4  
5 quitar(X, [X|Ys] ,Ys).  
6 quitar(X, [Y|Ys] ,[Y|Ys1]) :-  
7     quitar(X,Ys,Ys1).  
8 quitar(_,[],[]).
```

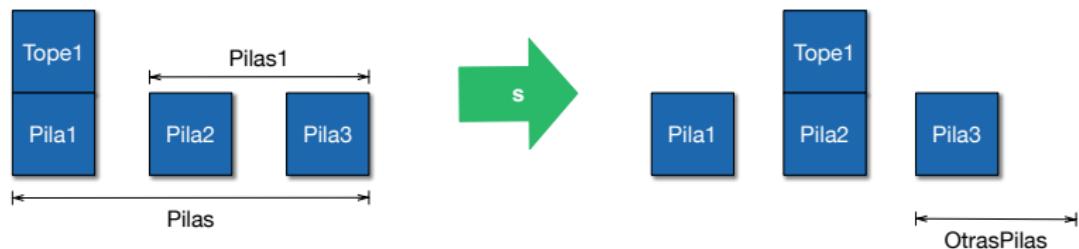
- Corrida:

```
1 ?- s([[a],[b],[c]],Succ).  
2 Succ = [[], [a, b], [c]] ;  
3 Succ = [[], [a, c], [b]] ;  
4 ...  
5 false.
```



Universidad Veracruzana

Gráficamente



Universidad Veracruzana

Slowing Down

- ▶ ¿Qué hace *quitar/3*? En general:

```
1  ?- quitar(2,[1,2,3],L).
2  L = [2,3]
3  ?- quitar(E,[1,2,3],L).
4  E = 1,
5  L = [2,3] ;
6  E = 2,
7  L = [1,3] ...
```

- ▶ En este caso:

```
1  ?- quitar([T1|P1],[[a],[b],[c]],Pilas1), quitar(P2,Pilas1,OtrasP),
2  T1 = a,
3  P1 = [],
4  Pilas1 = [[b], [c]],
5  P2 = [b],
6  OtrasP = [[c]].
```



Universidad Veracruzana

Estrategia

- ▶ Para encontrar un **camino solución** Sol , que va de un nodo dado N a un nodo meta:
 1. Si N es un nodo meta, entonces $Sol = [N]$, o
 2. Si existe un nodo $N1$, sucesor de N , tal que existe un camino $Sol/1$ de $N1$ a un nodo meta, entonces $Sol = [N|Sol/1]$. El valor de $Sol/1$ se encuentra recursivamente.
- ▶ Lo cual traduce a Prolog como:

```
1  solucion(N, [N]) :-  
2      meta(N).  
3  solucion(N, [N|Sol1]) :-  
4      s(N,N1),  
5      solucion(N1,Sol1).
```



Universidad Veracruzana

Corrida

- ▶ Usando el mundo de los bloques:

```

1  ?- solucion([[a,b,c],[],[]],S).
2  S = [[[a, b, c], [], []]]
3
4  ?- solucion([[c,b,a],[],[]],S).
5  S = [[[c, b, a], [], []],
6  [[b, a], [c], []],
7  [[a], [b, c], []],
8  [[]], [a, b, c], []]]

```

- ▶ Pero, los **ciclos** pueden dar problemas:

```

1  ?- solucion([[b,c,a],[],[]],S).
2  ERROR: Out of local stack
3  % Execution Aborted

```

- ▶ Nota. Usen los archivos `mundobloques.pl` y `busquedaProfundidad.pl` del capítulo 4.



Universidad Veracruzana

Evitando ciclos

- ▶ La idea es la que usamos al resolver el laberinto –No visites estados previamente visitados (línea 8):

```
1 solucion2(Nodo,Sol) :-  
2     primeroProfundidad([],Nodo,Sol).  
3  
4 primeroProfundidad(Camino,N,[N|Camino]) :-  
5     meta(N).  
6 primeroProfundidad(Camino,Nodo,Sol) :-  
7     s(Nodo,N1),  
8     \+ (member(N1, Camino)),  
9     primeroProfundidad([Nodo|Camino],N1,Sol).
```



Universidad Veracruzana

Corrida

- ▶ Claro que la salida está en orden inverso:

```

1  ?- solucion2([[b,c,a],[],[]],S).
2  S = [[[], [a, b, c], []], [[a], [b, c], []],
3        [[], [b, a], [c]], [[c], [b], [a]], [[], [b, c], [a]],
4        [[b], [c], [...]], [[], [...|...]|...],
```

- ▶ y es muy larga. Podemos formatearla con un *imprime_edos/1*:

```

1  impr_edos([]) :- write('Fin'),nl.
2  impr_edos([Edo|Edos]) :- write(Edo), nl, impr_edos(Edos).
```

```

1  ?- solucion2([[b,c,a],[],[]],S), reverse(S,Saux), impr_edos(Saux).
2  [[b, c, a], [], []]
3  [[c, a], [b], []]
4  [[a], [c, b], []]...
```



Limitando la profundidad

- ▶ Agregamos un argumento para definir la **máxima profundidad** de búsqueda *MaxProf* y agregamos la **restricción** correspondiente:

```
1 solucion3(Nodo,Sol,MaxProf) :-  
2     primeroProfundidad2(Nodo,Sol,MaxProf).  
3  
4 primeroProfundidad2(Nodo,[Nodo],_) :-  
5     meta(Nodo).  
6 primeroProfundidad2(Nodo,[Nodo|Sol],MaxProf) :-  
7     MaxProf > 0,  
8     s(Nodo,Nodo1),  
9     \+ member(Nodo1,Sol),  
10    Max1 is MaxProf-1,  
11    primeroProfundidad2(Nodo1,Sol,Max1).
```

- ▶ **Problema:** ¿Qué profundidad?



En profundidad iterativa

- Una solución elegante y declarativa para el camino más corto en profundidad, buscar **inversamente** desde la meta:

```

1 camino(Nodo,Nodo,[Nodo]). 
2 camino(PrimerNodo,UltimoNodo,[UltimoNodo|Camino]) :- 
3     camino(PrimerNodo,MenosElUltimo,Camino), 
4     s(MenosElUltimo,UltimoNodo), 
5     \+ member(UltimoNodo,Camino). 
6 
7 solucion4(NodoInicial,Sol) :- 
8     camino(NodoInicial,NodoMeta,Sol), meta(NodoMeta).

```

```

1 ?- solucion4([[b,c,a],[],[]],S), impr_edos(S). 
2 [[], [a,b,c], []] 
3 [[], [b,c], [a]] 
4 [[a], [c], [b]] 
5 [[c,a], [b], []] 
6 [[b,c,a], [], []] 
7 Fin 
8 S = [[[], [a, b, c], ...

```



Estrategia

- ▶ Partimos de un solo **camino candidato** [[`NodoInicial`]].
- ▶ Luego, dado un conjunto de caminos candidatos **sucesores**:
 - ▶ Si la cabeza del primer camino es un nodo **meta**, entonces esa es la solución al problema. De otra forma
 - ▶ Eliminar el primer camino del conjunto de caminos candidatos y generar el conjunto de todas las posibles **extensiones** de un paso de este camino.
 - ▶ Agregar este conjunto de extensiones al **final** del conjunto de candidatos.
 - ▶ Buscar **recursivamente** en este nuevo conjunto de caminos candidatos.
- ▶ Para generar las extensiones de un sólo paso, usemos el predicado predefinido *bagof/3*



El código

```

1   solucion5(Inicio,Sol) :-
2       primeroEnAmplitud([[Inicio]],Sol).
3
4   primeroEnAmplitud([[Nodo|Camino]|_], [Nodo|Camino]) :- 
5       meta(Nodo).
6   primeroEnAmplitud([Camino|Caminos],Sol) :- 
7       extender(Camino,NuevosCaminos),
8       append(Caminos,NuevosCaminos,Caminos1),
9       primeroEnAmplitud(Caminos1,Sol).
10
11  extender([Nodo|Camino],NuevosCaminos) :- 
12      bagof([NuevoNodo,Nodo|Camino],
13             (
14                 s(Nodo,NuevoNodo),
15                 \+ (member(NuevoNodo, [Nodo|Camino])))
16             ),
17      NuevosCaminos,
18 !.
19  extender(_,[]).

```



Corrida

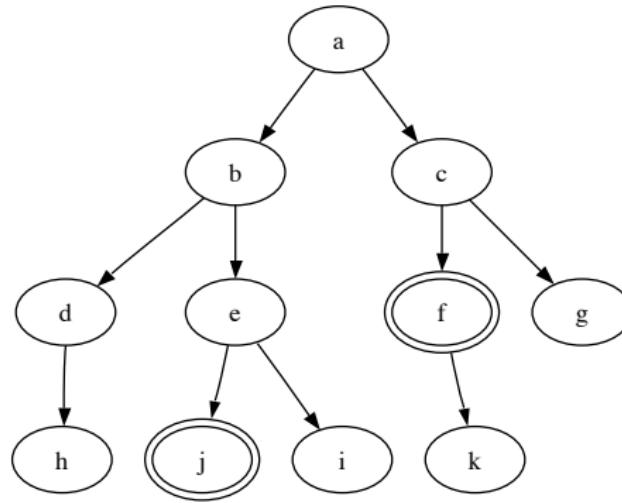
- ▶ Nota. Use el archivo `busquedaAmplitud.pl`:

```
1  ?- solucion5([[c,b,a],[],[]], S).
2  S = [[[], [a, b, c], []], [[a], [b, c], []],
3        [[b, a], [c], []], [[c, b, a], [], []]]
```



Otro ejemplo

- ▶ Comparemos las dos estrategias implementadas hasta ahora sobre el grafo:



Universidad Veracruzana

El programa

- ▶ Nota. El archivo graflo.pl:

```
1  s(a,b). s(a,c).
2  s(b,d). s(b,e).
3  s(d,h). s(e,i).
4  s(e,j). s(c,f).
5  s(c,g). s(f,k).
6
7  meta(j). meta(f).
```



Universidad Veracruzana

Las búsquedas

- ▶ La búsqueda en amplitud encuentra primero el nodo **más cercano a la raíz**.

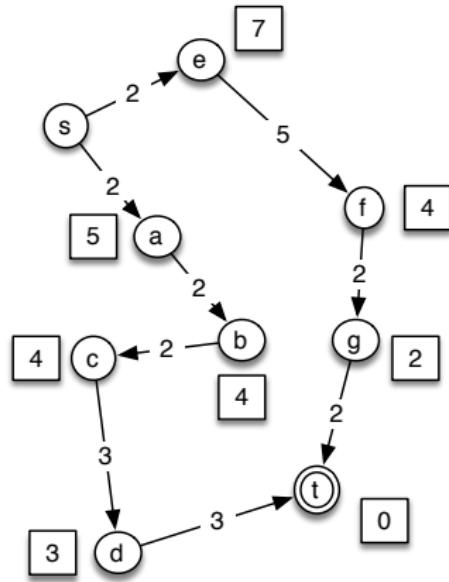
```
1  ?- solucion5(a,R).  
2  R = [f, c, a] ;  
3  R = [j, e, b, a] ;  
4  false.  
5  
6  ?- solucion4(a,R).  
7  R = [a, b, e, j] ;  
8  R = [a, c, f] ;  
9  false.
```



Universidad Veracruzana

Búsquedas optimizadas

- Dado el siguiente mapa, la tarea es encontrar la ruta más **corta** entre una ciudad inicial s y una ciudad meta t .



- Los círculos son **estados** a visitar.
- Las flechas representan **sucesores**.
- Las etiquetas en las flechas son **distancias**.
- Los cuadros son **distancias lineales** a la meta.



Universidad Veracruzana

Heurísticas

- ▶ Las búsquedas anteriores son **ciegas** ante los criterios de optimización.
- ▶ Un algoritmo primero el mejor afina la búsqueda en amplitud, expandiendo el **mejor** candidato.
- ▶ Una función **costo** $s(n, m, c)$ representa el costo c de moverse de un nodo n al nodo m en el espacio de estados.
- ▶ Se puede diseñar una función **heurística** h que estime la facilidad de llegar de un nodo n a la meta.
- ▶ La búsqueda puede guiarse entonces **minimizando** el costo recorrido y la distancia por recorrer.



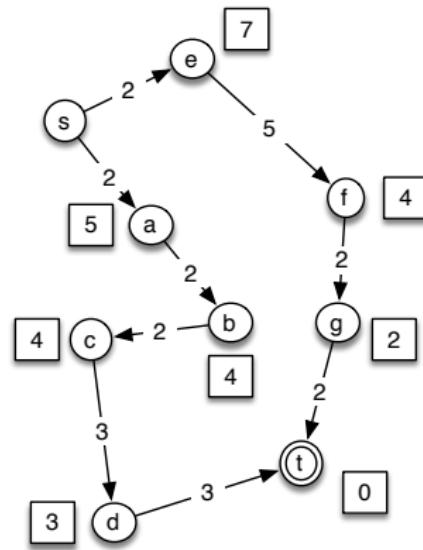
Grafo, costo, heurística y meta

- ▶ El grafo del ejemplo puede representarse, considerando costo, heurística y metas como:

```

1   s(s,a,2). s(s,e,2).
2   s(e,f,5). s(a,b,2).
3   s(f,g,2). s(b,c,2).
4   s(g,t,2). s(c,d,3).
5   s(d,t,3).
6
7   h(a,5). h(b,4).
8   h(c,4). h(d,3).
9   h(e,7). h(f,4).
10  h(g,2). h(t,0).
11
12  meta(t).

```



Costo y costo estimado

- ▶ $f(n)$ será construida para **estimar** el costo del mejor camino solución entre el nodo inicial s y el nodo meta t , con la restricción de que el camino pase por el nodo n .
- ▶ Supongamos que tal camino existe y que un nodo meta que minimiza su costo es n . Entonces el estimado de $f(n)$ puede calcularse como la suma de dos términos:

$$f(n) = g(n) + h(n)$$

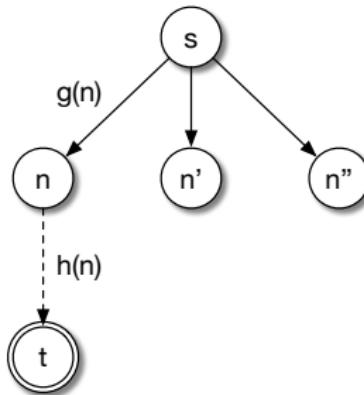
donde $g(n)$ es el **costo** del camino óptimo de s a n ; y $h(n)$ es el **costo estimado** de un camino óptimo de n a la meta t .



Universidad Veracruzana

Gráficamente

- ▶ La heurística **computa** el costo hasta el momento $g(n)$ y **estima** el costo por venir $h(n)$:



- ▶ Evidentemente $g(n)$ es fácil de computar, mientras que $h(n)$ requiere de **ingeniería del conocimiento**.



Búsqueda

- ▶ Al inicio el proceso 1 (vía a) está más **activo**, porque los valores f en ese camino son más bajos que los del otro.
- ▶ Cuando llega a c y el proceso 2 (vía e) sigue en e , la situación cambia:

$$f(c) = g(c) + h(c) = 6 + 4 = 10$$

$$f(e) = g(e) + h(e) = 2 + 7 = 9$$

- ▶ $f(e) < f(c)$ y ahora el proceso 2 procede al nodo f y el proceso 1 espera. Pero entonces:

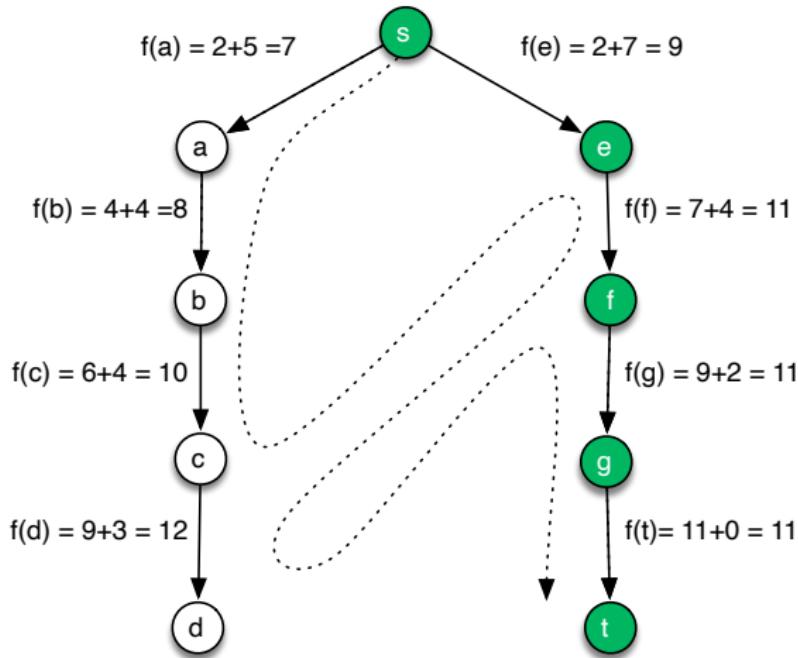
$$f(f) = 7 + 4 + 11, f(c) = 10, f(c) < f(f)$$

- ▶ El proceso 2 es detenido y se le permite al proceso 1 continuar...



Universidad Veracruzana

Gráficamente



Universidad Veracruzana

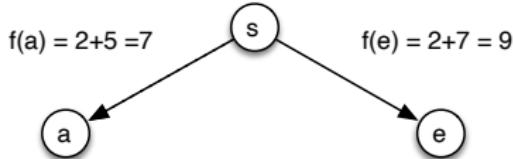
Representación

- ▶ Los **nodos de búsqueda** son de dos tipos:

Hoja: $I(N, F/G)$, donde N es un nodo en el espacio de estados, G es $g(N)$ y F es $f(N) = G + h(N)$.

Nodo interno: $t(N, F/G, Subárboles)$, donde F es el valor $f(N)$ actualizado por el valor para $f(N)$ del sucesor más prometedor.

- ▶ Nota. Los sub-árboles de t se **ordenan** de acuerdo a su f -valor.
- ▶ Ejemplo: $t(s, 7/0, [l(a, 7/2), l(e, 9/2)])$



Universidad Veracruzana

Actualización de los F -valores

- ▶ A partir del árbol $t(s, 7/0, [l(a, 7/2), l(e, 9/2)])$, el proceso continua alcanzando el árbol:

$t(s, 9/0, [l(e, 9/2), t(a, 10/2, [t(b, 10/4, [l(c, 10/6)])])])$

- ▶ Para una hoja N del árbol, mantenemos la definición original:

$$f(N) = g(N) + h(N)$$

- ▶ Para un nodo interno N con sub-árboles S_1, S_2, \dots :

$$f(N) = \min_i f(S_i)$$



Universidad Veracruzana

El código

► Interfaz:

```

1  % primeroMejor(Inicio, Sol): Sol es un camino de Inicio
2  % a la meta. Asumimos 9999 > que todo f-valor
3
4  primeroMejor(Inicio, Sol) :-
5      expandir([], l(Inicio,0/0), 9999, _, si, Sol).

```

► expandir(Camino,Arbol,Umbral,Arbol1,Solucionado,Sol):

Camino. Recorrido entre *Inicio* y *Arbol*.

Arbol. El proceso de búsqueda actual.

Umbral. El límite-*f* para la expansión de *Arbol*.

Arbol1. es *Arbol* expandido bajo el límite-*f*, i.e., el *f*-valor de *Arbol1* > *Umbral* (al menos que una meta se haya encontrado).

Solucionado. Puede ser *si*, *no*, o *nunca*, refleja si se ha encontrado la solución.

Sol. Una solución desde *Inicio* a un nodo meta, que pasa por *Arbol1* bajo el *Umbral*.



Universidad Veracruzana

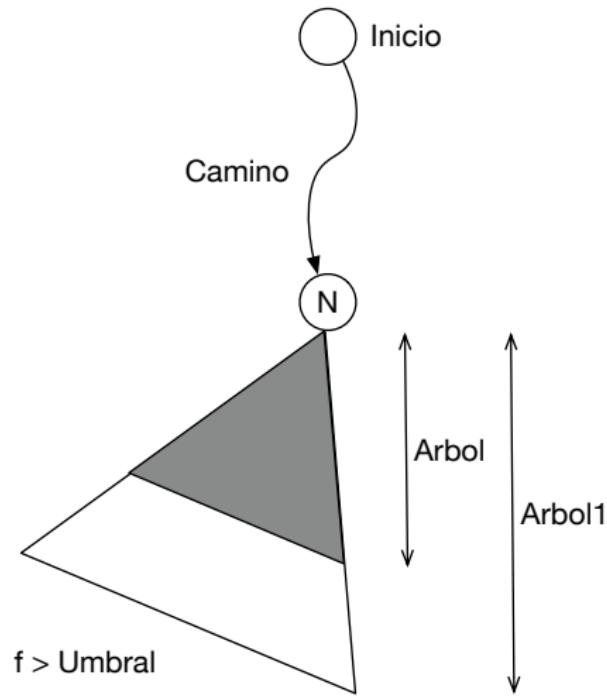
Resultados de expandir

- ▶ *Solucionado = si.* *Sol* tiene un camino solución encontrado al expandir *Arbol* bajo el *Umbral*. *Arbol1* no tiene valor.
- ▶ *Solucionado = no.* *Arbol1* es la expansión de *Arbol* con un *f*-valor mayor que el umbral. *Sol* no tiene valor.
- ▶ *Solucionado = nunca.* *Arbol1* y *Sol* no tienen valor.



Universidad Veracruzana

Gráficamente



Universidad Veracruzana

Expandir I

- ▶ Hay cinco **casos** posibles de expansión:

1. *Nodo* es una hoja meta del espacio de soluciones:

```
1 % Caso 1: nodo hoja meta, construir camino solución
2
3 expandir(Camino,l(Nodo,_),_,_,si,[Nodo|Camino]) :-_
4     meta(Nodo).
```



Universidad Veracruzana

Expandir II

2. Nodo es una hoja, cuyo F -valor es menor o igual que el *Umbral*.

```

1  % Caso 2: nodo hoja, f-valor < Umbral
2  % Generar sucesores y expandir bajo el Umbral.
3
4  expandir(Camino, l(Nodo, F/G), Umbral, Arbol1, Resuelto, Sol) :-
5      F =< Umbral,
6      ( bagof(Nodo2/C, (s(Nodo, Nodo2, C), \+ member(Nodo2, Camino)), Succ),
7          !,                                     % Nodo tiene sucesores
8          listaSucs(G, Succ, As),             % Construir subárboles As
9          mejorF(As, F1),                   % f-valor del mejor sucesor
10         expandir(Camino, t(Nodo, F1/G, As), Umbral, Arbol1, Resuelto, Sol)
11     ;
12     Resuelto = nunca                  % Nodo sin sucesores
13 ).
```



Expandir III

3. Nodo es interno con F -valor menor o igual que el *Umbral*.

```
1  % Caso 3: no hoja, f-valor < Umbral
2  % Expandir el subárbol más promisorio; dependiendo de
3  % resultados, continuar decidirá como proceder.
4
5  expandir(Camino,t(Nodo,F/G,[A|As]),Umbral,Arbol1,Resuelto,Sol) :-
6      F =< Umbral,
7      mejorF(As,MejorF),min(Umbral,MejorF,Umbral1),
8      expandir([Nodo|Camino],A,Umbral1,A1,Resuelto1,Sol),
9      continuar(Camino,t(Nodo,F/G,[A1|As]),Umbral,Arbol1,
10      Resuelto1,Resuelto).
```



Expandir IV

4. Cubre los puntos muertos, cuando no hay solución:

```

1  % Caso 4: Nodo interno sin subárboles
2  % No hay solución por esta vía
3
4  expandir(_,t(_,_,[]),_,_,nunca,_) :- !.
```

5. El *f*-valor es mayor que el *Umbral*, el árbol no crece:

```

1  % Caso 5: f-valor > Umbral
2  % Arbol no debe crecer.
3
4  expandir(_,Arbol,Umbral,Arbol,no,_) :- 
5      f(Arbol,F), F > Umbral.
```



Continuar

- ▶ *continuar/7* decide como procede la búsqueda de acuerdo al árbol expandido:

```
1  % continuar(Camino,Arbol,Umbral,NuevoArbol,SubarbolResuelto,
2  % ArbolResuelto,Sol)
3
4  continuar(_,_,_,_,si,si,Sol).
5
6  continuar(Camino,t(Nodo,_/G,[A1|As]),Umbral,Arbol1,no,Resuelto,Sol) :-
7      insertar(A1,As,NAs),
8      mejorF(NAs,F1),
9      expandir(Camino,t(Nodo,F1/G,NAs),Umbral,Arbol1,Resuelto,Sol).
10
11 continuar(Camino,t(Nodo,_/G,[_|As]),Umbral,Arbol1,nunca,Resuelto,Sol) :-
12     mejorF(As,F1),
13     expandir(Camino,t(Nodo,F1/G,As),Umbral,Arbol1,Resuelto,Sol).
```



Corridas iniciales

- ▶ Se llama a $\text{expandir}([], I(s, 0/0), 9999, \text{si}, Solucion)$
- ▶ La primer llamada no satisface $\text{meta}(Nodo)$.
- ▶ El primer caso es expandir la hoja $I(s, 0/0)$.
- ▶ La primera parte del trabajo, lo hace $\text{bagof}/3$ que computa los sucesores de s con sus distancias asociadas $\text{Succ}/[a/2, e/2]$.
- ▶ El predicado $\text{listaSuccs}/3$ construye la lista de posibles árboles sucesores $As/[I(a, 7/2), I(e, 9/2)]$.
- ▶ Con esa lista $\text{mejorF}/2$ encuentra que el mejor $F1/7$ y
- ▶ Se llama recursivamente a expandir con el árbol expandido a $t(s, 7/0, [I(a, 7/2), I(e, 9/2)])$.
El resto no ha cambiado.



Continuación de la corrida

- ▶ La tercer llamada es
 $expandir([], t(s, 7/0, [I(a, 7/2), I(e, 9/2)]), 9999, _, si, Solucion).$
- ▶ Como $F/7 \leq 9999$ la evaluación continua.
- ▶ Se actualiza el umbral, obteniendo el mejor F-valor del resto de los hijos de s . Como $MejorF/9 < 999$ el nuevo umbral es 9.
- ▶ Y se llama recursivamente a expandir el camino recorrido aumentado con $s|Camino$, $Arbol/I(a, 7/2)$, el $Umbral/1/9$ y el resto sigue igual.



Funciones auxiliares I

```

1  % listaSucs(G0,[Nodo1/Costo1,...],[l(MejorNodo,MejorF/G),...]): 
2  % ordena la lista de hojas por F-valor
3
4  listaSucs(_,[],[]).
5  listaSucs(G0,[N/C|NCs],As) :- 
6      G is G0 + C,
7      h(N,H),                                % Heuristica h(N)
8      F is G + H,
9      listaSucs(G0,NCs,As1),
10     insertar(l(N,F/G),As1,As).
11
12  % Insertar A en una lista de arboles As, preservando orden por
13  % f-valor.
14
15  insertar(A,As,[A|As]) :- 
16      f(A,F), mejorF(As,F1),
17      F =< F1, !.
18  insertar(A,[A1|As],[A1|As1]) :- 
19      insertar(A,As,As1).
20
21  % Extraer f-value

```



Funciones auxiliares II

```
22
23  f(l( _,F/_),F).           % f-valor de una hoja
24  f(t( _,F/_,_),F).         % f-valor de un árbol
25
26  mejorF([A|_],F) :-          % mejor F de una lista de árboles
27      f(A,F).
28
29  mejorF([],9999).
30
31  min(X,Y,X) :-              %
32      X <= Y, !.
33  min(_,Y,Y).
```



Corrida

- ▶ Primero consultamos la base de conocimiento con el mapa de las ciudades:

```
1  ?- [ciudades].  
2  true
```

- ▶ Luego corremos la consulta:

```
1  ?- primeroMejor(s,Sol).  
2  Sol = [t, g, f, e, s] ;  
3  Sol = [t, d, c, b, a, s] ;  
4  false.
```



Universidad Veracruzana

Introducción

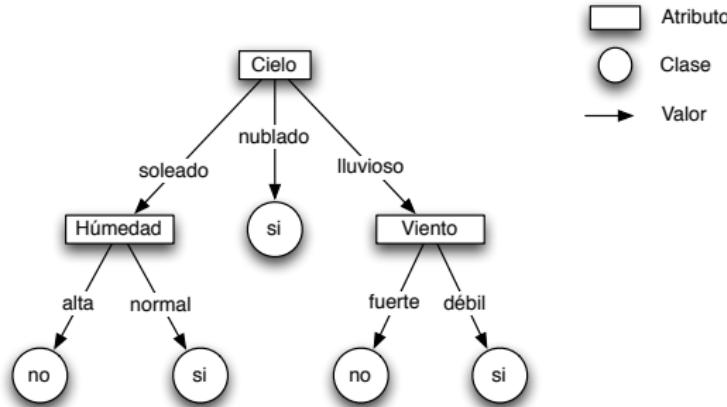
- ▶ El aprendizaje de árboles de decisión es una de las técnicas de **inferencia inductiva** más usadas.
- ▶ Se trata de un método para **aproximar** funciones de valores discretos, capaz de expresar hipótesis disyuntivas y robusto al ruido en los ejemplos de entrenamiento.
- ▶ La descripción que presento en este capítulo, cubre el algoritmo **ID3** de Quinlan [2].
- ▶ Aunque también aplica a **C4.5** [3] y su implementación en Weka J48, en cierta medida.



Universidad Veracruzana

Representación del árbol

- ▶ Cada nodo del árbol denota un **atributo** y puede verse como la pregunta: ¿Qué valor tiene este atributo en el caso a clasificar?
- ▶ Las ramas denotan los posibles **valores** del atributo correspondiente.



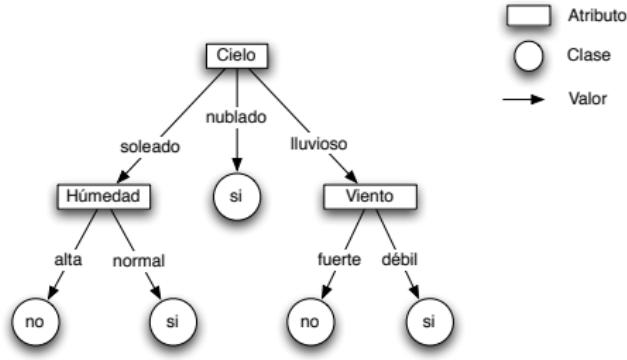
- ▶ Las hojas denotan los **valores de clase** inducidos.



Universidad Veracruzana

Representación de casos

- ▶ Los **casos** a clasificar se representan como tuplas finitas de **pares atributo–valor** (proposicional):
 $\langle \text{cielo} = \text{soleado}, \text{temperatura} = \text{caliente}, \text{humedad} = \text{alta}, \text{viento} = \text{fuerte} \rangle$
- ▶ Un árbol de decisión **clasifica** un caso, filtrándolo ascendentemente, hasta encontrar una hoja, *i.e.*, valor de clasificación buscado.



Universidad Veracruzana

Algoritmo clasificación

- ▶ El algoritmo es como sigue:

```
1: function CLASIFICA(Caso, Arbol)
2:   Aux  $\leftarrow$  valor_atributo(raiz(Arbol), Caso);
3:   if hoja(raiz(Arbol)) then
4:     return Aux
5:   else
6:     clasifica(Caso, sub_arbol(Arbol, Aux));
7:   end if
8: end function
```

- ▶ Se asumen las siguientes funciones:

valor_atributo(A,C). Regresa el valor del atributo *A* en el caso *C*.

sub_arbol(A,V). Regresa el sub-árbol de *A* obtenido
a partir de la rama *V*.

raiz(A). Regresa el atributo que es la raíz del árbol.

hoja(A). Verdadero si el atributo *A* es una hoja.



Semántica del árbol de decisión

- ▶ En general, un árbol de decisión representa una **disyunción de conjunciones** de restricciones en los posibles valores de los atributos de los casos.
- ▶ Cada rama que va de la raíz del árbol a una hoja, representa una **conjunción** de tales restricciones.
- ▶ El árbol mismo representa la **disyunción** de esas conjunciones.
- ▶ **Ejemplo:** Uno juega tenis si:

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



Problemas adecuados

- ▶ Representación basada en pares **atributo-valor**, donde cada atributo tiene un dominio **discreto** de valores disjuntos. Hay extensiones para valores **continuos**.
- ▶ **Función objetivo discreta**. Además de los casos binarios, un árbol de decisión puede ser extendido fácilmente, para más de dos valores de clase posibles.
- ▶ Se necesitan descripciones **disyuntivas**.
- ▶ El método es robusto al **ruido** en los ejemplos de entrenamiento, tanto errores de clasificación, como errores en los valores de los atributos.
- ▶ También es robusto a valores **faltantes** en los ejemplos.



Universidad Veracruzana

Conjunto de entrenamiento y clase

día	cielo	temperatura	humedad	viento	jugar-tenis?
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



Particiones

- ▶ La decisión central de ID3 consiste en **seleccionar** qué atributo colocará en cada nodo del árbol de decisión.
- ▶ La función **mejor-particion**, toma como argumentos un conjunto de ejemplos de entrenamiento, regresando la **partición** inducida por el atributo que, sólo, clasifica **mejor** los ejemplos de entrenamiento:

```
1  ?- mejor_particion(Ejs,MejorPart).  
2  MejorPart = [cielo [soleado d1 d2 d8 d9 d11]  
3                      [nublado d3 d7 d12 d13]  
4                      [lluvia d4 d5 d6 d10 d14]]
```

- ▶ La función *mejor-partición* encuentra el atributo que mejor **separa** los ejemplos con respecto a la **clase**.



Universidad Veracruzana

Entropía

- ▶ Para **cuantificar** la bondad de un atributo, se puede considerar la cantidad de información que éste proveerá, tal y como lo define la teoría de la información de Shannon y Weaver [4].
- ▶ Un bit de información es suficiente para **determinar** el valor de un atributo booleano, por ejemplo, si/no, verdadero/falso, 1/0, etc., sobre el cual no sabemos nada.
- ▶ Si los posibles valores del atributo v_i , ocurren con probabilidades $P(v_i)$, entonces en contenido de información, o **entropía**, E de la respuesta actual está dado por:

$$E(P(v_1), \dots, P(v_n)) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$



Universidad Veracruzana

Ejemplo (volados)

- ▶ Consideren el caso booleano de un volado con una moneda **confiable**
–La probabilidad de obtener aguila o sol es $1/2$:

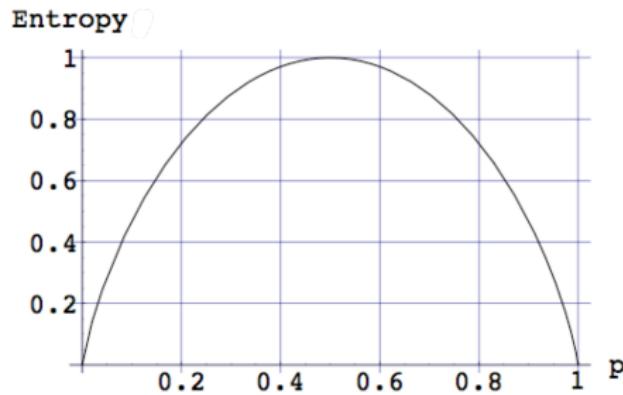
$$E\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1$$

- ▶ Ejecutar el volado nos provee 1 bit de información, de hecho, nos provee la clasificación del experimento: si fue águila o sol.
- ▶ Si usamos una moneda cargada que da 99 % de las veces sol, $E(1/100, 99/100) = 0,08$ bits de información. **Menos** que en el caso de la moneda justa, porque ahora tenemos más evidencia sobre el posible resultado del experimento.



Gráficamente

- ▶ 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 es:



Universidad Veracruzana

Regresando al tenis

- ▶ De 14 ejemplos, 9 son positivos (si) y 5 son negativos. La **entropía** de este conjunto de entrenamiento es:

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

- ▶ Si **todos** los ejemplos son positivos o negativos, p. ej., pertenecen todos a la misma clase, la entropía será 0. Una posible interpretación de esto, es considerar la entropía como una medida de ruido o **desorden** en los ejemplos.



Universidad Veracruzana

Ganancia de Información

- ▶ La **reducción** de la entropía en el conjunto de entrenamiento S causada por particionar S con respecto a un atributo A , se calcula así:

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

Observen que el segundo término es la entropía con respecto al atributo A .



Universidad Veracruzana

Ganancia y tenis

- ▶ Al utilizar esta medida en ID3, sobre los ejemplos del tenis:

```
1 Ganancia del atributo CIELO : 0.24674976
2 Ganancia del atributo TEMPERATURA : 0.029222548
3 Ganancia del atributo HUMEDAD : 0.15183544
4 Ganancia del atributo VIENTO : 0.048126936
5 Máxima ganancia de informacion: 0.24674976
6 Partición:
7 (CIELO (SOLEADO D1 D2 D8 D9 D11) (NUBLADO D3 D7 D12 D13)
8 (LLUVIA D4 D5 D6 D10 D14))
```

- ▶ Esto indica que el **atributo con mayor ganancia** de información fue *cielo*, de ahí las particiones de los ejemplos.



Universidad Veracruzana

Algoritmo de inducción

```
1: function ID3(Ejs, Atrbs, Clase)
2:   Default ← valMasComun(Ejs, Clase);
3:   if CriterioDeParo() then
4:     return A ← hoja(Default);
5:   else if Atrbs = ∅ then
6:     return A ← hoja(Default);
7:   else
8:     MejorParticion ← mejorParticion(Ejs, Atrbs);
9:     A ← hoja(first(MejorParticion));
10:    for all P ∈ rest(MejorParticion) do
11:      VAtrb ← first(P);
12:      SubEjs ← rest(P);
13:      agregaRama(A, VAtrb, ID3(SubEjs, {Atrbs \ A}, Clase));
14:    end for
15:    return A
16:   end if
17: end function
```



Consideraciones

- ▶ 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.
- ▶ ID3 mantiene **sólo una hipótesis** mientras explora el espacio de hipótesis posibles.
- ▶ El algoritmo básico ID3 **no reconsidera** (*backtracking*) en su búsqueda. Algo similar puede lograrse con alguna técnica de **poda**.
- ▶ ID3 utiliza **todos** los ejemplos en cada paso de su búsqueda basada en *ganancia de información*.
- ▶ Una ventaja es que la búsqueda es **menos sensible al ruido** en los datos.



Interfaz

```
1  :- dynamic
2      ejemplo/3,
3      nodo/3.
4
5  id3(ArchCSV) :-
6      id3(ArchCSV,1). % Umbral = 1, por default.
7
8  id3(ArchCSV,Umbral) :-
9      reset,
10     cargaEjs(ArchCSV,Atrs),
11     findall(N,ejemplo(N,_,_),Inds), % Obtiene índices de los ejemplos
12     inducir(Inds,raiz,Atrs,Umbral),
13     imprimeArbol, !.
```



Universidad Veracruzana

Ejemplos de entrenamiento: tenis.pl

```
1 ejemplo(1,no,[cielo=soleado,temperatura=alta,humedad=alta,viento=debil]).  
2 ejemplo(2,no,[cielo=soleado,temperatura=alta,humedad=alta,viento=fuerte]).  
3 ejemplo(3,si,[cielo=nublado,temperatura=alta,humedad=alta,viento=debil]).  
4 ejemplo(4,si,[cielo=lluvioso,temperatura=templada,humedad=alta,viento=debil]).  
5 ejemplo(5,si,[cielo=lluvioso,temperatura=fresca,humedad=normal,viento=debil]).  
6 ejemplo(6,no,[cielo=lluvioso,temperatura=fresca,humedad=normal,viento=fuerte]).  
7 ejemplo(7,si,[cielo=nublado,temperatura=fresca,humedad=normal,viento=fuerte]).  
8 ejemplo(8,no,[cielo=soleado,temperatura=templada,humedad=alta,viento=fuerte]).  
9 ejemplo(9,si,[cielo=soleado,temperatura=fresca,humedad=normal,viento=debil]).  
10 ejemplo(10,si,[cielo=lluvioso,temperatura=templada,humedad=normal,viento=debil]).  
11 ejemplo(11,si,[cielo=soleado,temperatura=nublado,humedad=normal,viento=fuerte]).  
12 ejemplo(12,si,[cielo=nublado,temperatura=templada,humedad=alta,viento=fuerte]).  
13 ejemplo(13,si,[cielo=nublado,temperatura=alta,humedad=normal,viento=debil]).  
14 ejemplo(14,no,[cielo=lluvioso,temperatura=templada,humedad=alta,viento=fuerte]).
```



Ejemplos de entrenamiento: tenis.csv

```
1  cielo, temperatura, humedad, viento, jugarTenis
2  soleado, alta, alta, debil, no
3  soleado, alta, alta, fuerte, no
4  nublado, alta, alta, debil, si
5  lluvioso, templada, alta, debil, si
6  lluvioso, fresca, normal, debil, si
7  lluvioso, fresca, normal, fuerte, no
8  nublado, fresca, normal, fuerte, si
9  soleado, templada, alta, debil, no
10 soleado, fresca, normal, debil, si
11 lluvioso, templada, normal, debil, si
12 soleado, templada, normal, fuerte, si
13 nublado, templada, alta, fuerte, si
14 nublado, alta, normal, debil, si
15 lluvioso, templada, alta, fuerte, no
```



leeCSV

```
1  ?- leeCSV("tenis.csv",Atrs,Ejs), dominios(Atrs,Ejs,Doms).
2  14 ejemplos de entrenamiento cargados.
3  Atrs = [cielo, temperatura, humedad, viento, jugarTenis],
4  Ejs = [[soleado, alta, alta, debil, no], [soleado, alta, alta,
5          fuerte, no], [nublado, alta, alta, debil, si], ...],
6  Doms = [[cielo, [soleado, nublado, lluvioso]], [temperatura,
7          [alta, templada, fresca]], [humedad, [alta, normal]],
8          [viento, [debil, fuerte]], [jugarTenis, [no, si]]].
```



Universidad Veracruzana

Auxiliares: last y butlast

```
1  %%% last(L,E): E es el último elemento de la lista L.  
2  
3  last([],[]).  
4  last([L|Ls],E) :-  
5      append(_, [E], Ls).  
6  
7  %%% butLast(L1,L2): L2 es L1 sin el último elemento.  
8  
9  butLast([],[]).  
10 butLast([L1|L2],L2) :-  
11     last([L1|L2],Last),  
12     append(L2, [Last], L1).  
  
1  ?- last([1,2,3],L).  
2  L = 3  
3  ?- butLast([1,2,3],BL).  
4  BL = [1,2]
```



csv2ejs: corrida

```
1  ?- csv2ejs("tenis.csv",Ejs,Atrs).  
2  14 ejemplos de entrenamiento cargados.  
3  Ejs = [ejemplo(1, no, [cielo=soleado, temperatura=alta,  
4                humedad=alta, viento=debil]), ejemplo(2, no,  
5                [cielo=soleado, temperatura=alta, humedad=alta,  
6                viento=fuerte]), ejemplo(3, si, [cielo=nublado,  
7                temperatura=alta, humedad=alta, viento=debil]), ...],  
8  Atrs = [cielo, temperatura, humedad, viento, jugarTenis]
```



Universidad Veracruzana

csv2ejs: Definición

```
1  % csv2ejs(ArchCSV,Ejs,Atrs): transforma las lista de salida de
2  % leerCSV/2 en una lista de ejemplos(Ind,Clase,[Attr=Val]) y otra
3  % lista de atributos.
4
5  csv2ejs(ArchCSV,Ejs,Atrs) :-
6      leeCSV(ArchCSV,Atrs,EjsCSV),
7      butlast(Atrs,AtrsSinClase),
8      procEjs(1,AtrsSinClase,EjsCSV,Ejs).
9
10 procEjs(_,_,[],[]).
11
12 procEjs(Ind,AtrsSinClase,[Ej|Ejs],[ejemplo(Ind,Clase,EjAtrsVals)|Resto]) :-
13     last(Ej,Clase),
14     butlast(Ej,EjSinValorClase),
15     maplist(procAtrVal,AtrsSinUltimo,EjSinValorClase,EjAtrsVals),
16     IndAux is Ind + 1,
17     procEjs(IndAux,AtrsSinClase,Ejs,Resto).
18
19 procAtrVal(Attr,Value,Attr=Value).
```



cargaEjs

```
1  % cargaEjs(ArchCSV): carga ArchCSV como hechos estilo
2  % ejemplo(Ind,Clase,[Attr=Val]).  
3  
4  cargaEjs(ArchCSV,Atrs) :-  
5      csv2ejs(ArchCSV,Ejs,Atrs),  
6      maplist(assertz,Ejs).  
  
1  ?- listing(ejemplo).  
2  :- dynamic ejemplo/3.  
3  
4  ejemplo(1, no, [cielo=soleado, temperatura=alta, humedad=alta,  
5  viento=debil]).  
6  ejemplo(2, no, [cielo=soleado, temperatura=alta, humedad=alta,  
7  viento=fuerte]).  
8  ...
```



Arbol de decisión: Representación

```
1  ?- listing(nodo).
2  :- dynamic nodo/3.
3
4  nodo(1, cielo=lluvioso, raiz).
5  nodo(2, viento=fuerte, 1).
6  nodo(hoja, [no/2], 2).
7  nodo(3, viento=debil, 1).
8  nodo(hoja, [si/3], 3).
9  nodo(4, cielo=nublado, raiz).
10 nodo(hoja, [si/4], 4).
11 nodo(5, cielo=soleado, raiz).
12 nodo(6, humedad=normal, 5).
13 nodo(hoja, [si/2], 6).
14 nodo(7, humedad=alta, 5).
15 nodo(hoja, [no/3], 7).
```



imprimeArbol

```
1  imprimeArbol :-  
2      imprimeArbol(raiz,0).  
3  
4  imprimeArbol(Padre,_) :-  
5      nodo(hoja,Clase,Padre), !,  
6      write(' => '), write(Clase).  
7  
8  imprimeArbol(Padre,Pos) :-  
9      findall(Hijo,nodo(Hijo,_,Padre),Hijos),  
10     Pos1 is Pos+2,  
11     imprimeLista(Hijos,Pos1).  
12  
13 imprimeLista([],_) :- !.  
14  
15 imprimeLista([N|T],Pos) :-  
16     nodo(N,Test,_),  
17     nl, tab(Pos), write(Test),  
18     imprimeArbol(N,Pos),  
19     imprimeLista(T,Pos).
```



Arbol impreso

```
1  cielo=lluvioso
2    viento=fuerte => [no/2]
3    viento=debil => [si/3]
4  cielo=nublado => [si/4]
5  cielo=soleado
6    humedad=normal => [si/2]
7    humedad=alta => [no/3]
```



Universidad Veracruzana

Inducción: Caso 1

```
1 % Caso 1. El número de ejemplos a clasificar es menor que el
2 % umbral. Se crea un nodo hoja con la distribución Distr,
3 % apuntando al padre del nodo.
4
5 inducir(Ejs,Padre,_,Umbral) :-  
6   length(Ejs,NumEjs),  
7   NumEjs=<Umbral,  
8   distr(Ejs, Distr),  
9   assertz(nodo(hoja,Distr,Padre)), !. % al final de los nodos.
```



Universidad Veracruzana

Inducción: Caso 2

```
1 % Caso 2. Todos los ejemplos a clasificar son de la misma clase.  
2 % Se crea un nodo hoja con la distribución [Clase], apuntando al  
3 % padre del nodo.  
4  
5 inducir(Ejs,Padre,_,_) :-  
6   distr(Ejs, [Clase]),  
7   assertz(nodo(hoja,[Clase],Padre)).
```



Universidad Veracruzana

Inducción: Caso 3

```
1 % Caso 3. Se debe decidir que atributo es el mejor clasificador
2 % para los ejemplos dados.
3
4 inducir(Ejs,Padre,Atrs,Umbral) :-  
    eligeAtr(Ejs,Atrs,Atr,Vals,Resto), !,  
    particion(Vals,Atr,Ejs,Padre,Resto,Umbral).
```



Inducción: Caso 4

```
1 % Caso 4. Los datos no se pueden particionar.  
2  
3 inducir(Ejs,Padre,_,_) :- !,  
4     nodo(Padre,Test,_),  
5     write('No es posible construir partición de '),  
6     write(Ejs), write(' en el nodo '), writeln(Test).
```



Distribución de clases: Ejemplos

```
1  ?- findall(E, ejemplo(E,_,_), Ejs), distr(Ejs, Dist).  
2  Ejs = [1, 2, 3, 4, 5, 6, 7, 8, 9|...],  
3  Dist = [no/5, si/9].  
4  ?- distr([1,2,6,8], Dist).  
5  Dist = [no/4].
```



Universidad Veracruzana

Distribución de clases: Definición I

```

1  %%% distr(+Ejs,-DistClaseEjs)
2  %%% Computa la Distribución de clases para el conjunto de
3  %%% Ejs. La notación X^Meta causa que X no sea instanciada
4  %%% al solucionar la Meta.
5
6  distr(Ejs,DistClaseEjs) :-  

7      % Extrae Valores de Clase de los Ejs
8      setof(Clase,Ej^AVs^(member(Ej,Ejs),ejemplo(Ej,Clase,AVs)),Clases),
9      % Cuenta la distribución de los valores para la Clase
10     cuentaClases(Clases,Ejs,DistClaseEjs).
  
  

1  ?- setof(Clase,
2          Ej^AVs^(member(Ej,[1,2,3,4,5,6,7,8,9,10,11,12,13,14]),
3                  ejemplo(Ej,Clase,AVs)),
4                  Clases).
5  Clases = [no, si].

```



Distribución de clases: Definición II

```
1  cuentaClases([],_,[]) :- !.  
2  cuentaClases([Clase|Clases],Ejs,[Clase/NumEjsEnClase|RestoCuentas]) :-  
3      % Extrae los ejemplos con clase Clase en la lista Cuentas  
4      findall(Ej,(member(Ej,Ejs),ejemplo(Ej,Clase,_)),EjsEnClase),  
5      % Computa cuantos ejemplos hay en la Clase  
6      length(EjsEnClase,NumEjsEnClase), !,  
7      % Cuentas para el resto de los valores de la clase  
8      cuentaClases(Clases,Ejs,RestoCuentas).
```

```
1  ?- numlist(1,14,Ejs), cuentaClases([no,si],Ejs,Distr).  
2  Ejs = [1, 2, 3, 4, 5, 6, 7, 8, 9|...],  
3  Distr = [no/5, si/9].
```



Universidad Veracruzana

setof vs findall

- ▶ Alternativamente, se puede usar `findall/3` y `sort/2` para obtener las clases:

```
1  ?- findall(C, ejemplo(_,C,_), TodasClases), sort(TodasClases, Clases).
2  TodasClases = [no, no, si, si, si, no, si, no, si|...]
3  Clases = [no, si].
```

- ▶ `setof/3` está basado en `bagof/3`, por lo que es necesario usar existenciales para obtener el mismo resultado que `findall/3`:

```
1  ?- bagof(C,I^AV^ejemplo(I,C,AV),Clases).
2  Clases = [no, no, si, si, si, no, si, no, si|...].
```



Universidad Veracruzana

El mejor atributo I

```
1  % eligeAtr(+Ejs,+Atrs,-Atr,-Vals,-Resto)
2  % A partir de un conjunto de ejemplos Ejs y atributos Atrs,
3  % computa el atributo Atr en Atrs con mayor ganancia de
4  % información, sus Vals y el Resto de atributos en Atrs.
5
6  eligeAtr(Ejs,Atrs,Atr,Vals,RestoAtrs) :-
7      length(Ejs,NumEjs),
8      contenidoInformacion(Ejs,NumEjs,I), !,
9      findall((Atr-Vals)/Gain,
10             ( member(Atr,Atrs),
11               vals(Ejs,Atr,[],Vals),
12               separaEnSubConjs(Vals,Ejs,Atr,Parts),
13               informacionResidual(Parts,NumEjs,IR),
14               Gain is I - IR
15             ),
16             Todos ),
17      maximo(Todos,(Atr-Vals)/_), !
18      eliminar(Atr,Atrs,RestoAtrs), !.
```



El mejor atributo II

```
1  ?- numlist(1,14,Ejs),
2      Atrs = [cielo, temperatura, humedad, viento, jugarTenis],
3      butlast(Atrs, AtrsSinClase),
4      eligeAtr(Ejs,AtrsSinClase, BestAttr, Dom, RestoAtrs).
5      [(cielo-[lluvioso,nublado,soleado])/0.246749819774439,
6      (temperatura-[fresca,templada,alta])/0.029222565658954647,
7      (humedad-[normal,alta])/0.15183550136234136,
8      (viento-[fuerte,debil])/0.04812703040826927 ]
9  Ejs = [1, 2, 3, 4, 5, 6, 7, 8, 9|...],
10 AtrsSinClase = [cielo, temperatura, humedad, viento],
11 BestAttr = cielo,
12 Dom = [lluvioso, nublado, soleado],
13 RestoAtrs = [temperatura, humedad, viento],
14 Atrs = [cielo, temperatura, humedad, viento, jugarTenis]
```



El mejor atributo III

```
1 contenidoInformacion(Ejs,NumEjs,I) :-  
2     setof(Clase,  
3             Ej^AVs^(member(Ej,Ejs),ejemplo(Ej,Clase,AVs)),  
4             Clases), !,  
5     sumaTerms(Clases,Ejs,NumEjs,I).  
6  
7 sumaTerms([],_,_,0) :- !.  
8 sumaTerms([Clase|Clases],Ejs,NumEjs,Info) :-  
9     findall(Ej,(member(Ej,Ejs),ejemplo(Ej,Clase,_)),EjsEnClase),  
10    length(EjsEnClase,NumEjsEnClase),  
11    sumaTerms(Clases,Ejs,NumEjs,I),  
12    Info is I -  
13        (NumEjsEnClase/NumEjs)*(log(NumEjsEnClase/NumEjs)/log(2)).
```



Universidad Veracruzana

El mejor atributo IV

```
1  ?- findall(N, ejemplo(N,_,_),E), length(E,NumEjs),
2      contenidoInformacion(E,NumEjs,I).
3  E = [1, 2, 3, 4, 5, 6, 7, 8, 9|...],
4  NumEjs = 14,
5  I = 0.9402859586706309.
```



Universidad Veracruzana

El mejor atributo V

```
1 informacionResidual([],_,0) :- !.
2 informacionResidual([Part|Parts],NumEjs,IR) :- 
3     length(Part,NumEjsPart),
4     contenidoInformacion(Part,NumEjsPart,I), !,
5     informacionResidual(Parts,NumEjs,R),
6     IR is R + I * NumEjsPart/NumEjs.
7
8 separaEnSubConjs([],_,[],[]) :- !.
9 separaEnSubConjs([Val|Vals],Ejs,Atr,[Part|Parts]) :- 
10    subconj(Ejs,Atr=Val,Part), !,
11    separaEnSubConjs(Vals,Ejs,Atr,Parts).
12
13 subconj([],_,[]) :- !.
14 subconj([Ej|Ejs],Atr,[Ej|RestoEjs]) :- 
15     ejemplo(Ej,_,AVs),
16     member(Atr,AVs), !,
17     subconj(Ejs,Atr,RestoEjs).
18 subconj([_|Ejs],Atr,RestoEjs) :- 
19     subconj(Ejs,Atr,RestoEjs).
20
21 vals([],_,Vals,Vals) :- !.
```



El mejor atributo VI

```
22  vals([Ej|Ejs],Atr,Vs,Vals) :-  
23      ejemplo(Ej,_,AVs),  
24      member(Atr=V,AVs), !,  
25      (  
26          member(V,Vs), !, vals(Ejs,Atr,Vs,Vals);  
27          vals(Ejs,Atr,[V|Vs],Vals)  
28      ).
```



Alternativa con findall

```
1 separaEnSubConjs2(Atr, Vals, Ejs, Partes) :-  
2     findall(Atr=V, member(V, Vals), AVs),  
3     findall(Parte, (member(AV, AVs), subConj2(Ejs, AV, Parte)), Partes).  
4  
5 subConj2(Ejs, AV, SubConj) :-  
6     findall(Id, (member(Id, Ejs), ejemplo(Id, _, AVs), member(AV, AVs)), SubConj).
```



Ganancias de información: Corrida

```
1  ?- findall(N, ejemplo(N,_,_),E),
2    findall((A-Valores)/Gain,
3      (member(A,[cielo,temperatura,humedad,viento]),
4       vals(E,A,[],Valores)),
5       separaEnSubConjs(Valores,E,A,Ess),
6       informacionResidual(Ess,14,R),
7       Gain is 0.940286 - R),
8       All).
9   |   E = [1, 2, 3, 4, 5, 6, 7, 8, 9|...],
10  All = [(cielo-[lluvioso, nublado, soleado])/0.24674986110380803,
11        (temperatura-[fresca, templada, alta])/0.029222606988323685,
12        (humedad-[normal, alta])/0.1518355426917104,
13        (viento-[fuerte, debil])/0.048127071737638305].
```



Partición

```
1  % particion(+Vals,+Atr,+Ejs,+Padre,+Resto,+Umbral)
2  % Por cada Valor del atributo Atr en Vals, induce una partición
3  % en los ejemplos Ejs de acuerdo a Valor, para crear un nodo del
4  % árbol y llamar recursivamente a inducir.
5
6  particion([],_,_,_,_,_) :- !.
7  particion([Val|Vals],Atr,Ejs,Padre,RestoAtrs,Umbral) :-  
8      subconj(Ejs,Atr=Val,SubEjs), !,  
9      generaNodo(Nodo),  
10     assertz(nodo(Nodo,Atr=Val,Padre)),  
11     inducir(SubEjs,Nodo,RestoAtrs,Umbral), !,  
12     particion(Vals,Atr,Ejs,Padre,RestoAtrs,Umbral).
```



Ejecutando todo el experimento

```
1  ?- id3('tenis.csv').  
2  cielo=lluvioso  
3      viento=fuerte => [no/2]  
4      viento=debil => [si/3]  
5  cielo=nublado => [si/4]  
6  cielo=soleado  
7      humedad=normal => [si/2]  
8      humedad=alta => [no/3]  
9  true.
```



Universidad Veracruzana

Predicados auxiliares I

```
1  genera_nodo_id(M) :-  
2      retract(id(N)),  
3      M is N+1,  
4      assert(id(M)), !.  
5  
6  genera_nodo_id(1) :-  
7      assert(id(1)).  
8  
9  eliminar(X,[X|T],T) :- !.  
10  eliminar(X,[Y|T],[Y|Z]) :-  
11      eliminar(X,T,Z).  
12  
13  subconjunto([],_) :- !.  
14  subconjunto([X|T],L) :-  
15      member(X,L), !,  
16      subconjunto(T,L).  
17  
18  maximo([X],X) :- !.  
19  maximo([X/M|T],Y/N) :-  
20      maximo(T,Z/K),  
21      (M>K,Y/N=X/M ; Y/N=Z/K), !.
```



Universidad Veracruzana

Predicados auxiliares II

```
22  
23 % elimina ejemplo y nodo en el espacio de trabajo  
24  
25 reset :-  
26   retractall(ejemplo(_,_,_)),  
27   retractall(nodo(_,_,_)).
```



Evitando un sobre ajuste

- ▶ Dado un espacio de hipótesis H , se dice que una hipótesis $h \in H$ está **sobre ajustada** 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 casos del problema.
- ▶ Por **ruido** o por conjunto de entrenamiento demasiado **pequeño**.



Universidad Veracruzana

Sobre ajuste por ruido

- ▶ Al agregar el siguiente caso mal clasificado (*jugarTenis = no*):
 ⟨ *cielo = soleado, temperatura = alta, humedad = normal, viento = fuerte* ⟩
- ▶ El ejemplo con ruido será filtrado junto con los ejemplos 9 y 11 (*cielo = soleado* y *humedad = normal*), que son ejemplos positivos. Dado que el nuevo ejemplo es negativo, ID3 buscará refine el árbol a partir del nodo *humedad*, **agregando** un atributo más al árbol.



Universidad Veracruzana

Estrategias de solución

- ▶ 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**.
- ▶ Los segundos han sido más exitosos.



Universidad Veracruzana

¿Cuál es el tamaño correcto del árbol?

- ▶ Usar un conjunto de ejemplos no usados en el entrenamiento, para evaluar la utilidad de eliminar nodos del árbol. **Por ejemplo:** Poda.
- ▶ Usar los ejemplos disponibles para el entrenamiento, aplicando una prueba para estimar cuando expandir el árbol (o detener su crecimiento). **Por ejemplo:** El test χ^2 .
- ▶ Usar explícitamente una medida de complejidad para codificar los ejemplos de entrenamiento y el árbol de decisión, deteniendo el crecimiento del árbol cuando el tamaño de la codificación sea minimizado. **Por ejemplo:** Descripción de longitud mínima (*MDL*).



Universidad Veracruzana

Poda del árbol

- ▶ Un enfoque llamado *reduced-error pruning* [2], 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 casos 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.
- ▶ Se eliminan **coincidencias fortuitas** en los datos del entrenamiento.
- ▶ Únicamente efectivo si contamos con **suficientes ejemplos**



Universidad Veracruzana

Poda de reglas

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 cualquier condición que resulte en una mejora de la precisión estimada.
4. Ordenar las reglas por su precisión estimada, y aplicarlas en ese orden al clasificar nuevos casos.



Universidad Veracruzana

Atributos continuos

- ▶ Para un atributo continuo A , el algoritmo puede crear dinámicamente un atributo discreto A_c que es verdadero si $A > c$ y falso en cualquier otro caso.
- ▶ Ejemplo:

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

- ▶ Dos umbrales pueden localizarse en los puntos $(48 + 60)/2$ y $(80 + 90/2)$. La ganancia de información puede entonces calcularse para los atributos $\text{temperatura}_{>54}$ y $\text{temperatura}_{>85}$.



Medidas alternativas: Gain Ratio

- ▶ Split information:

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

- ▶ Gain ratio:

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

- ▶ Problemas: denominador indefinido cuando $|S_i| \approx |S|$.



Referencias I

- [1] I Bratko. *Prolog programming for Artificial Intelligence*. Fourth. Essex, England: Pearson, 2012.
- [2] JR Quinlan. "Induction of Decision Trees". En: *Machine Learning* 1 (1986), págs. 81-106.
- [3] JR Quinlan. *C4. 5: programs for machine learning*. Vol. 1. San Mateo, CA, USA: Morgan kaufmann, 1993.
- [4] C Shannon y W Weaver. "The mathematical theory of communication". En: *The Bell System Technical Journal* 27 (jul. de 1948), págs. 623-656.



Universidad Veracruzana