

# Metodología de Programación I

## Inducción de Árboles de Decisión

Dr. Alejandro Guerra-Hernández

Departamento de Inteligencia Artificial  
Facultad de Física e Inteligencia Artificial  
aguerra@uv.mx  
<http://www.uv.mx/aguerra>

Maestría en Inteligencia Artificial 2009



Universidad Veracruzana

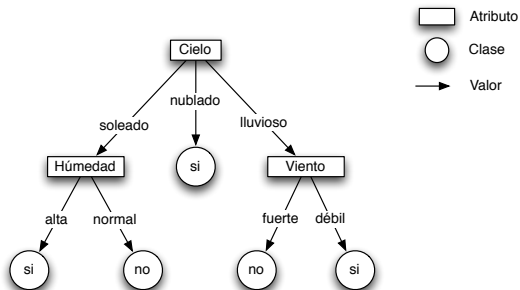
# Introducción

- ▶ 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 una familia de algoritmos para la inducción de árboles de decisión que incluyen **ID3** y **C4.5**.



# Representación del árbol

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





# Algoritmo clasificación

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



# 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 ejemplares.
- ▶ Cada rama que va de la raíz del árbol a una hoja, representa una **conjunción** de tales restricciones y el árbol mismo representa la **disyunción** de esas conjunciones. Por ejemplo, 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 pares **atributo-valor**, donde cada atributo tiene un dominio **discreto** y cada valor es disjunto. 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 posibles.
- ▶ Se necesitan descripciones **disyuntivas**.
- ▶ **Ruido** en los ejemplos de entrenamiento. El método es robusto al ruido en los ejemplos de entrenamiento, tanto errores de clasificación, como errores en los valores de los atributos.
- ▶ Valores **faltantes** en los ejemplos.



# Conjunto de entrenamiento y clase

- ▶ Consideren los siguientes ejemplos de **entrenamiento**:

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

- ▶ La **meta** es *jugar-tenis*.



# 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 y un conjunto de atributos, regresando la **partición** inducida por el atributo que, sólo, clasifica **mejor** los ejemplos de entrenamiento. Ejemplo:

```
?- mejor_particion(Ejs,MejorPart).  
MejorPart = [temperatura [frio d5 d6 d7 d9  
                        [caliente d1 d2 d3  
                        d13]  
            [templado d4 d8 d10  
            d11 d12 d14]]]
```

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



# 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 información por Claude E. Shannon.
- ▶ 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:

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



## Ejemplo (volados)

- ▶ Consideren nuevamente el caso booleano de un volado con una moneda **confiable**, tenemos que la probabilidad de obtener aguilá o sol es de  $1/2$  para cada una:

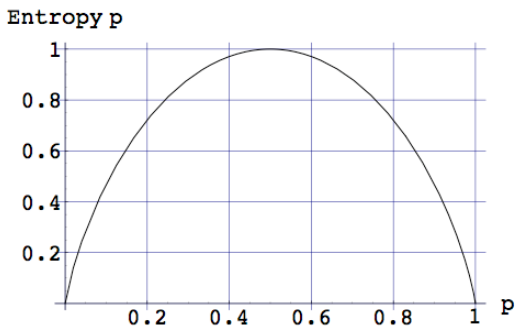
$$E\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1$$

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



# 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:



## Regresando al tenis

- ▶ Consideren nuevamente los ejemplos de entrenamiento para el 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, por ejemplo, pertenecen todos a la misma clase, la entropía será 0. Una posible interpretación de ésto, es considerar la entropía como una medida de ruido o **desorden** en los ejemplos.



# Ganancia de Información

- Definimos la **ganancia de información** (Ganancia) como la reducción de la entropía causada por particionar un conjunto de entrenamiento  $S$ , con respecto a un atributo  $A$ :

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

Observen que el segundo término de *Ganancia*, es la entropía con respecto al atributo  $A$ .



# Ganancia y tenis

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

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

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



# Algoritmo de inducción

```
1: function ID3(Ejs, Atrbs, Clase)
2:    $A \leftarrow \emptyset$ ; Default  $\leftarrow$  valMasComun(Ejs, Clase);
3:   if Ejs =  $\emptyset$  then
4:     return Default;
5:   else if mismoVal(Ejs, Clase) then
6:     return  $A \leftarrow$  valor(first(Ejs).Clase);
7:   else if Atrbs =  $\emptyset$  then
8:     return  $A \leftarrow$  valMasComun(Ejs, Clase);
9:   else
10:    MejorParticion  $\leftarrow$  mejorParticion(Ejs, Atrbs);
11:    MejorAtrb  $\leftarrow$  first(MejorParticion);
12:     $A \leftarrow$  MejorAtrb;
13:    for all  $P \in$  rest(MejorParticion) do
14:       $V_{Atrb} \leftarrow$  first( $P$ );
15:      SubEjs  $\leftarrow$  rest( $P$ );
16:      rama( $A$ ,  $V_{Atrb}$ , ID3(SubEjs, {Atrbs \ MejorAtrb}, Clase));
17:    end for
18:    return  $A$ 
19:  end if
20: 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. La vuelta atrás puede implementarse con alguna técnica de **poda**.
- ▶ ID3 utiliza **todos** los ejes. 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.



# Clases, Atributos y Ejemplos

## ► Ejemplo, crédito bancario:

```
1  clases([si,no]).
2  atrbs([cuenta,saldo,empleado]).
3  ej(1,si,[cuenta = banco, empleado = si, saldo = 300]).
4  ej(2,si,[cuenta = banco, empleado = si, saldo = 300]).
5  ej(3,no,[cuenta = banco, empleado = no, saldo = 300]).
6  ej(4,si,[cuenta = banco, empleado = no, saldo = 4000]).
7  ej(5,si,[cuenta = ninguna, empleado = si, saldo = 4000]).
8  ej(6,no,[cuenta = ninguna, empleado = si, saldo = 300]).
9  ej(7,no,[cuenta = ninguna, empleado = no, saldo = 300]).
10 ej(8,no,[cuenta = ninguna, empleado = no, saldo = 4000]).
11 ej(9,no,[cuenta = ninguna, empleado = no, saldo = 4000]).
12 ej(10,no,[cuenta = otra, empleado = no, saldo = 300]).
13 ej(11,si,[cuenta = otra, empleado = no, saldo = 4000]).
14 ej(12,no,[cuenta = otra, empleado = no, saldo = 300]).
```

## ► El archivo se carga con *consult/1*.

# Memoria de Trabajo e ID3

- ▶ Se debe limpiar la memoria de trabajo al consultar *id3*:

```
1  iniciar_kb :-
2      abolish(nodo,3), abolish(arbol,1),
3      abolish(examinado,3), abolish(atrbs,1),
4      abolish(clases,1), abolish(nodo_actual,1)
      , !.
```

- ▶ ID3 será algo como:

```
1  id3 :-
2      repeat,
3          nl, write('¿Qué archivo voy a usar? '),
4              read(Archivo),nl,
5              iniciar_kb,
6              consult(Archivo),
7              construye_arbol, muestra_arbol, nl,
8              write('¿Salir (s/n)? '),
9              read(s).
```



# Construye árbol

- ▶ Extrae los atributos, clases y ejemplos para correr *id3/3*:

```
1  construye_arbol :-
2      genera_id_nodo(_),
3      clause(atrbs(Atrbs),true),
4      findbag(Ej,clause(ej(Ej,_,_),true),Ejs),
5      id3(Ejs,Atrbs,Nodo),
6      assert(arbol(Nodo)), !.
7
8  genera_id_nodo(Y) :-
9      clause(nodo_actual(X),true), !,
10     retract(nodo_actual(X)),
11     Y is X + 1,
12     assert(nodo_actual(Y)).
13  genera_id_nodo(0) :-
14     assert(nodo_actual(0)).
```

- ▶ *clause/2* es verdadero si sus argumentos pueden unificar con una cláusula en el programa.



## ID3

- ▶ *id3/3* recibe como argumentos los ejemplos y los atributos y regresa el *id* árbol generado:

```
1 id3([],_,[]).
2 id3(Ejs,_,[hoja(Clase)]) :-
3     criterio_terminal(Ejs,Clase).
4 id3(Ejs,Atrbs,ID) :-
5     mejor_atrb(Atrbs,Ejs,MejorAtrb),
6     parte_vals(MejorAtrb,Ejs,PartVals),
7     borra(MejorAtrb,Atrbs,AtrbsNuevos),
8     genera_subarboles(PartVals,AtrbsNuevos,
9         SubarbolIDs),
10    genera_id_nodo(ID),
11    assert(nodo(ID,MejorAtrb,SubarbolIDs)).
```



# Criterio terminal

- ▶ El criterio terminal dictamina cuando parar el crecimiento del árbol:

```
1  criterio_terminal([Ej|Ejs],Clase) :-
2      clause(ej(Ej,Clase,_),true),
3      !,
4      misma_clase(Ejs,Clase).
5
6  misma_clase([],_).
7  misma_clase([Ej|Ejs],C) :-
8      clause(ej(Ej,C,_),true),
9      !,
10     misma_clase(Ejs,C).
```



# Mejor atributo

- ▶ Se usan tablas para computar el mejor atributo:

```
1 mejor_atrb(Atrbs,Ejs,MejorAtrb) :-
2     construye_tablas(Atrbs,Ejs),
3     calculos_comunes(MC,N),
4     calculo_param_clasif(Atrbs,MC,N,Vals),
5     mejor(Atrbs,Vals,MejorAtrb).
6
7 construye_tablas(Atrbs,Ejs) :-
8     clause(clases(Lc),true),
9     length(Lc,NroColTab),
10    abolish(tabla,3),
11    inicia_tablas(Atrbs,NroColTab),
12    construye_tablas(Atrbs,Ejs).
13
14 construye_tablas([],_).
15 construye_tablas([Atrb|Atrbs],ListaEjs) :-
16     tabla(Atrb,ListaEjs),
17     !,
18     construye_tablas(Atrbs,ListaEjs).
```



# Inicia Tablas

- ▶ Se necesita un predicado para generar listas de ceros:

```
1  inicia_tablas([],_).
2  inicia_tablas([A|As],NoCol) :-
3      crea_lista_ceros(NoCol,L),
4      assert(tabla(A,[],L)),
5      inicia_tablas(As,NoCol).
6
7  crea_lista_ceros(0,[]).
8  crea_lista_ceros(N,[0|R]) :-
9      N1 is N-1,
10     crea_lista_ceros(N1,R).
```



# Computo de las tablas

- ▶ Se busca el val. del atrib. en los ejcs. y su pos. en la clase:

```
1  tabla(_, []).
2  tabla(Atrb, [Ej|Ejs]) :-
3      valor(Atrb, Ej, V),
4      pos_de_clase(Ej, Pc),
5      actualiza_tabla(Atrb, V, Pc),
6      !, tabla(Atrb, Ejs).
7
8  valor(A, [A = V|_], V) :- !.
9  valor(A, [_|RestoAVs], V) :- valor(A, RestoAVs, V).
10 valor(A, Ej, V) :-
11     ej(Ej, _, AVs), valor(A, AVs, V).
12
13 pos_de_clase(Ej, Pc) :-
14     clause(ej(Ej, C, _), true),
15     clause(clase(Clases), true),
16     pos(C, Clases, Pc).
```



# Actualiza tablas

```
1  actualiza_tabla(Atrb,V,Pc) :-
2      retract(tabla(Atrb,TabLineas,TotClase)),
3      modifica_tabla(TabLineas,V,Pc,
4          LineasNuevo),
5      incrementa_pos_lista(1,Pc,TotClase,
6          TotalNuevo),
7      assert(tabla(Atrb,LineasNuevo,TotalNuevo
8          )).
9
10  modifica_tabla([],V,Pc,[(V,Vals,1)]) :-
11      clause(clases(Clases),true),
12      length(Clases,NoColumnas),
13      crea_lista_ceros(NoColumnas,L),
14      incrementa_pos_lista(1,Pc,L,Vals).
15  modifica_tabla([(V,Nums,Tot)|Resto],V,Pc,
16      [(V,NumsNvo,TotNvo)|Resto]) :-
17      TotNvo is Tot+1,
18      incrementa_pos_lista(1,Pc,Nums,NumsNvo).
19  modifica_tabla([X|Resto1],V,Pc,[X|Resto2]) :-
20      modifica_tabla(Resto1,V,Pc,Resto2).
```



# Incrementando pos en las listas

```
1 incrementa_pos_lista(N,N,[X|R],[Y|R]) :-  
2   Y is X+1.  
3 incrementa_pos_lista(N1,N,[X|R1],[X|R2]) :-  
4   N2 is N1+1,  
5   incrementa_pos_lista(N2,N,R1,R2).
```



# Cálculos

```
1  calculos_comunes(MC,N) :-
2      clause(tabla(_,_,Xjs),true),
3      calculos_comunes(Xjs,0,0,MC,N).
4
5  calculos_comunes([],TotalSum,N,MC,N) :-
6      MC is (-1 / N) * ( TotalSum - N * log(N)
7          ).
8
9  calculos_comunes([Xj|Xjs],Ac1,Ac2,MC,N) :-
10     NAc1 is Ac1 + Xj * log(Xj),
11     NAc2 is Ac2 + Xj,
12     calculos_comunes(Xjs,NAc1,NAc2,MC,N).
13
14 calculo_param_clasif([],_,_,[]).
15 calculo_param_clasif([A|As],MC,N,[V|Vs]) :-
16     gain_ratio(A,MC,N,V),
17     calculo_param_clasif(As,MC,N,Vs).
```



# Métrica: gain ratio

```
1 gain_ratio(A,MC,N,GR) :-
2     clause(tabla(A,Lineas,_),true),
3     calcula_factores_B_IV(Lineas,N,0,0,B,IV),
4     IM is MC - B,
5     GR is IM / IV.
6
7 calcula_factores_B_IV([],N,Sum1,Sum2,B,IV) :-
8     B is (-1 / N) * ( Sum1 - Sum2 ),
9     IV is (-1 / N) * ( Sum2 - N * log(N) ).
10 calcula_factores_B_IV([( _,L,TotL)|Resto],N,
11     Ac1,Ac2,B,IV) :-
12     sum_lineas(L,0,SL),
13     NAc1 is Ac1 + SL,
14     NAc2 is Ac2 + TotL * log(TotL),
15     calcula_factores_B_IV(Resto,N,NAc1,NAc2,B,IV).
```



# Suma líneas

```
1 sum_lineas([],X,X).
2 sum_lineas([0|Ns],Ac,Tot) :-
3     sum_lineas(Ns,Ac,Tot).
4 sum_lineas([N|Ns],Ac,Tot) :-
5     Nac is Ac + N * log(N),
6     sum_lineas(Ns,Nac,Tot).
```



# Mejor valor

```
1 mejor([A|As],[V|Vs],Res) :-  
2     mejor_valor(As,Vs,(A,V),Res).  
3  
4 mejor_valor([],[],(A,V),A).  
5 mejor_valor([A|As],[V|Vs],(_,TV),Res) :-  
6     V > TV,  
7     mejor_valor(As,Vs,(A,V),Res).  
8 mejor_valor(_|As,_|Vs,(TA,TV),Res) :-  
9     mejor_valor(As,Vs,(TA,TV),Res).
```



# Particiones

```
1 parte_vals(Atrb,Ejs,Res) :-
2     valores(Atrb,Ejs,Vals),
3     parte_ejs(Atrb,Vals,Ejs,Res).
4
5 valores(Atrb,Ejs,Vals) :-
6     findbag(V,(member(Ej,Ejs),value(Atrb,Ej,V
7         )),Vs),
8     remove_duplicates(Vs,Vals).
9
10 parte_ejs(_, [V],Ejs, [(V,Ejs)]).
11 parte_ejs(A, [V|Vs],Ejs, [(V,VEjs)|Resto]) :-
12     findbag(Ex,(member(Ex,Ejs),value(A,Ex,V))
13         ,VEjs),
14     dif(VEjs,Ejs,RestEx),
15     parte_ejs(A,Vs,RestEx,Resto).
```



# Subarboles

```
1  genera_subarboles([],_,[]).
2  genera_subarboles([(V,Ejs)|Resto1],Atrbs,[(V,
   Id)|Resto2]) :-
3      id3(Ejs,Atrbs,Id),
4      !,
5      genera_subarboles(Resto1,Atrbs,Resto2).
```

