Apprendimento PAC

Usando la disuguaglianza precedente ed altre considerazioni è possibile mostrare che alcune classi di concetti non sono PAC-apprendibili dato uno specifico algoritmo di apprendimento L e $\mathcal{H}$.

In particolare è possibile mostrare che:

- se $\mathcal{H}$ contiene tutte le funzioni booleane definite su X allora $C = \mathcal{H}$ non è PAC-apprendibile da algloritmi consistenti
- esistono classi di concetti C che non sono PAC-apprendibili da algoritmi consistenti che usano C come Spazio delle Ipotesi, tuttavia diventano PAC-apprendibili se uno Spazio delle Ipotesi "più grande" è usato, cioè $C \subset \mathcal{H}$

Il problema con la disuguaglianza data è che questa non può essere usata se $|\mathcal{H}| = \infty$

Tuttavia, il fattore chiave non è quante funzioni diverse sono contenute in $\mathcal{H}$, ma quante funzioni "utili" sono in $\mathcal{H}$: VC-dimension dà una risposta a questa domanda

Apprendimento PAC

Si può mostrare che, se assumiamo:

- $c \in \mathcal{H}$
- L consistente

allora con probabilità almeno $(1 - \delta)$, l'algoritmo di apprendimento L restituisce una ipotesi $h \in \mathcal{H}$ tale che $error_{\mathcal{D}}(h) \leq \epsilon$ se il numero di esempi di apprendimento n soddisfa la seguente disuguaglianza:

$$n \geq \frac{1}{\epsilon}(4 \log_2(\frac{1}{\delta}) + 8VC(\mathcal{H}) \log_2(\frac{13}{\epsilon}))$$

Notare che $VC(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$

$$VC(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$$

Mostriamo che $VC(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$

- per ogni S tale che $\mathcal{H}$ frammenta S abbiamo $|\mathcal{H}| \geq 2^{|S|}$, infatti $\mathcal{H}$ può implementare tutte le possibili dicotomie di S, che sono esattamente $2^{|S|}$.
- scegliendo un S tale che $|S| = VC(\mathcal{H})$, otteniamo $|\mathcal{H}| \geq 2^{VC(\mathcal{H})}$

Quindi, applicando $\log_2$ ad entrambi i lati della disuguaglianza, possiamo concludere che $\log_2(|\mathcal{H}|) \geq VC(\mathcal{H})$

Mistake Bounds per algoritmi On-line

Quando si considerano algoritmi on-line per l'apprendimento di concetti, è ragionevole essere interessati ad un limite superiore al numero di errori commessi prima di apprendere *esattamente* il concetto target

Il **Modello Mistake Bound** è stato defi nito per questo scopo:

- le istanze x_i sono presentate ad L una alla volta
- data una istanza x, L deve "indovinare" il valore target $c(x)$
- solo dopo, il valore corretto è fornito ad L ai fi ni dell'apprendimento
- se la predizione di L era sbagliata, allora si ha un errore (mistake)

Bisogna rispondere alla seguente domanda:

> "Quanti errori farà L prima di apprendere esattamente il concetto target ?"

Mistake bound per la versione on-line di Find-S

L'algoritmo **Find-S** può essere usato in versione on-line !

/* versione on-line di Find-S per congiunzione di m letterali */

inizializza h alla ipotesi pi`u specifi ca $h \equiv l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \wedge \cdots \wedge l_m \wedge \neg l_m$

do forever /* in effetti, fi no a quando arrivano esempi */

leggi la nuova istanza x e predici $h(x)$

leggi $c(x)$, cioè il valore target per x

/* errore ! */

if $(h(x) \neq c(x)$ rimuovi da h ogni letterale che non è soddisfatto da x

Assumendo $c \in \mathcal{H}$, e assenza di rumore negli esempi

E' possibile dare un limite superiore al numero di errori ?

Mistake bound per la versione on-line di Find-S

E' possibile dare un limite superiore al numero di errori ?

Si!

- l'ipotesi iniziale contiene $2m$ letterali
- dopo il primo sbaglio (che occorre subito!), solo m letterali rimangono nella ipotesi corrente
- dopo ogni altro errore, almeno un letterale è rimosso dalla ipotesi corrente

Quindi il numero totale di errori che **Find-S** commette prima di convergere alla ipotesi corretta è $\leq m + 1$

Mistake bound per Halving

L'algoritmo **Halving** è una versione on-line di **Candidate-Elimination**

/* Aggiorna il Version Space (VS) on-line */

inizializza S e G come in **Candidate-Elimination**

do forever /* in effetti, fino a che VS contiene più di 1 ipotesi */

leggi una nuova istanza x e predici tramite voto a maggioranza delle ipotesi in VS

leggi $c(x)$, cioè il valore target per x

aggiorna S e G in modo da rimuovere da VS le ipotesi h per cui $h(x) \neq c(x)$

/* un errore occorre solo se la maggioranza delle ipotesi in VS sono sbagliate */

Assumendo $c \in \mathcal{H}$, $|\mathcal{H}| < \infty$, e nessun rumore negli esempi

E' possibile dare un limite superiore al numero di errori ?

Mistake bound per Halving

E' possibile dare un limite superiore al numero di errori ?

Si!

- il VS iniziale contiene $|\mathcal{H}|$ ipotesi
- dopo un errore (che occorre quando il voto a maggioranza è sbagliato), ALMENO $|VS|/2$ ipotesi sono rimosse da VS

Quindi il numero totale di errori che **Halving** commette prima di convergere alla ipotesi corretta è $\leq log_2|\mathcal{H}|$

ATTENZIONE: **Halving** può convergere alla ipotesi corretta senza errori !! Accade quando il voto a maggioranza è sempre corretto e solo le ipotesi minoritarie sono rimosse da VS

Optimal Mistake Bounds

Fino ad ora abbiamo considerato limiti al numero di errori (caso pessimo) per algoritmi *specifici*.

E' possibile dare il più basso fra i limiti di errore su tutti i possibili algoritmi di apprendimento (*optimal mistake bound*) ?

- Assumiamo $C = \mathcal{H}$
- Sia $M_L(c)$ il massimo su tutte le possibili sequenze di esempi di apprendimento del numero di errori commesso da L per apprendere esattamente il concetto $c \in C$
- Sia $M_L(C) \equiv \max_{c \in C} M_L(c)$

Definizione: Sia C una classe non vuota di concetti. L' **optimal mistake bound** per C, denotato *Opt(C)*, è il minimo su tutti i possibili algoritmi di apprendimento L di $M_L(C)$:

$$Opt(C) \equiv \min_{L \in AlgoritmiApprendimento} M_L(C)$$

Optimal Mistake Bounds

Littlestone (1987) ha mostrato che

$$VC(C) \leq Opt(C) \leq M_{Halving}(C) \leq log_2(|C|)$$

Un esempio fi nale di Mistake Bound...

Fino ad ora abbiamo visto esempi di algoritmi di apprendimento capaci di trattare con esempi di apprendimento consistenti (cioè che non danno luogo a contraddizioni)

Non è diffi cile modifi care **Halving** in modo da ottenere un algoritmo capace di trattare dati inconsistenti: Algoritmo **Weighted-Majority**

- l'idea base è di assegnare ad ogni ipotesi (ma possiamo estendere l'idea ad ogni insieme di predittori) un peso che è usato per pesare il voto associato ad ogni ipotesi
- quindi, invece di considerare il voto a maggioranza, consideriamo il voto a maggioranza pesata
- quando una ipotesi commette un errore, invece di rimuoverla, si moltiplica il peso a lei associata per un fattore $\beta < 1$ (riduzione del peso)

Weighted-Majority

/* Weighted-Majority: l' algoritmo Halving... con pesi! */

per ogni ipotesi (o predittore) h_j definire il peso w_j, inizialmente uguale a 1

do forever /* in effetti, fino a che ci sono esempi */

leggi una nuova istanza x

calcola $q_0 \leftarrow \sum_{j:h_j(x)=0} w_j$ e $q_1 \leftarrow \sum_{j:h_j(x)=1} w_j$

if $q_0 > q_1$ **then** predici 0

if $q_1 > q_0$ **then** predici 1

if $q_0 = q_1$ **then** predici 0 o 1 a caso

read $c(x)$, i.e. the target value for x

for each h_j **do**

if $h_j(x) \neq c(x)$ **then** $w_j \leftarrow \beta w_j$ /* $0 \leq \beta < 1$ */

Cosa accade se $\beta = 0$?

Mistake bound per Weighted-Majority

Teorema: Sia $seq \equiv (x_1, c(x_1)), (x_2, c(x_2)), \ldots$ una qualunque sequenza di esempi di allenamento e sia k il numero minimo di errori commesso su seq da una qualunque ipotesi dello Spazio delle Ipotesi. Allora il numero di errori su seq commesso da **Weighted-Majority** usando $\beta = \frac{1}{2}$ è al più

$$2.4(k + \log_2(|\mathcal{H}|))$$

(lo proviamo!)

Per $0 \leq \beta < 1$, Littlestone e Warmuth (1991) hanno provato che il bound di sopra diventa

$$\frac{k \log_2 \frac{1}{\beta} + \log_2(|\mathcal{H}|)}{\log_2 \frac{2}{1+\beta}}$$

(questo non lo proviamo...)

Prova...

Per provare il teorema confrontiamo il peso finale w^* della migliore ipotesi h^*, cioè quella che produce il numero minore di errori k, con la somma finale dei pesi su tutte le ipotesi $W = \sum_{j=1}^{|\mathcal{H}|} w_j$ (naturalmente abbiamo $w^* \leq W$)

- **fatto 1:** $w^* = (\frac{1}{2})^k$, infatti h^* commette esattamente k errori
- **fatto 2:** per ogni errore di **Weighted-Majority**, W si riduce di al più $\frac{3}{4}W$, infatti se **Weighted-Majority** commette un errore, tutte le ipotesi h_j che hanno contribuito alla maggioranza pesata avranno il loro peso ridotto di metà; quindi almeno $\frac{1}{2}W$ (il voto a maggioranza pesata è $\geq \frac{1}{2}W$) del peso totale è ridotto a metà, cioè il nuovo peso totale è **minore o uguale a**

$$\underbrace{\frac{1}{2}W}_{\text{minoranza}} + \underbrace{\frac{1}{2}(\frac{1}{2}W)}_{\text{maggioranza}} = \frac{3}{4}W$$

- **fatto 3:** se M è il numero totale di errori prodotti da **Weighted-Majority**, allora per il peso totale finale $W \leq |\mathcal{H}|(\frac{3}{4})^M$, a causa del fatto 2 e la condizione iniziale $W = |\mathcal{H}|$

Prova...

Ricordando che $w^* \leq W$, ed a causa dei fatti 1-3, abbiamo

$$(\tfrac{1}{2})^k = w^* \leq W \leq |\mathcal{H}|(\tfrac{3}{4})^M$$

Prendendo il logaritmo (in base 2) dei termini pi`u a sinistra e pi`u a destra, otteniamo

$$k \log_2(\tfrac{1}{2}) \leq \log_2(|\mathcal{H}|) + M \log_2(\tfrac{3}{4})$$

ed isolando M otteniamo (notare che $\log_2(\frac{3}{4})$ è una quantità negativa)

$$M \leq \frac{k + \log_2(|\mathcal{H}|)}{-\log_2(\frac{3}{4})} \leq 2.4(k + \log_2(|\mathcal{H}|))$$

Alberi di Decisione

In molte applicazioni del mondo reale non è suffi ciente apprendere funzioni booleane con ingressi binari.

Gli Alberi di Decisione sono particolarmente adatti a trattare:

- istanze rappresentate da coppie attributo-valore;
- funzioni target con valori di output discreti (in generale pi`u di 2 valori);
- esempi di apprendimento che possono contenere errori e/o avere valori mancanti.

Inoltre, algoritmi di apprendimento per Alberi di Decisione sono in genere molto veloci.

Per questi motivi gli Alberi di Decisione sono molto utilizzati in applicazioni pratiche.

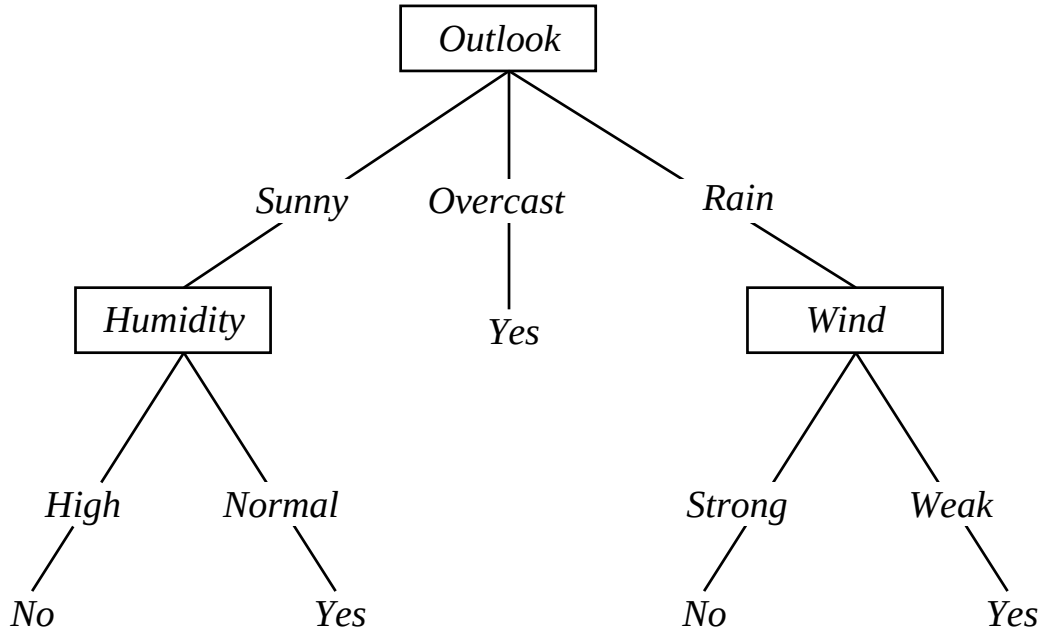
Giocare a Tennis!!

E’ la giornata ideale per giocare a Tennis ?

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Alberi di Decisione

Decidere se è la giornata ideale per giocare a Tennis !!



Es. ingresso: (Outlook= *Sunny* , Temperature=*Hot*, Humidity=*High*, Wind=*Strong*).

attributo *valore attributo*

Alberi di Decisione (cont.)

In un Albero di decisione:

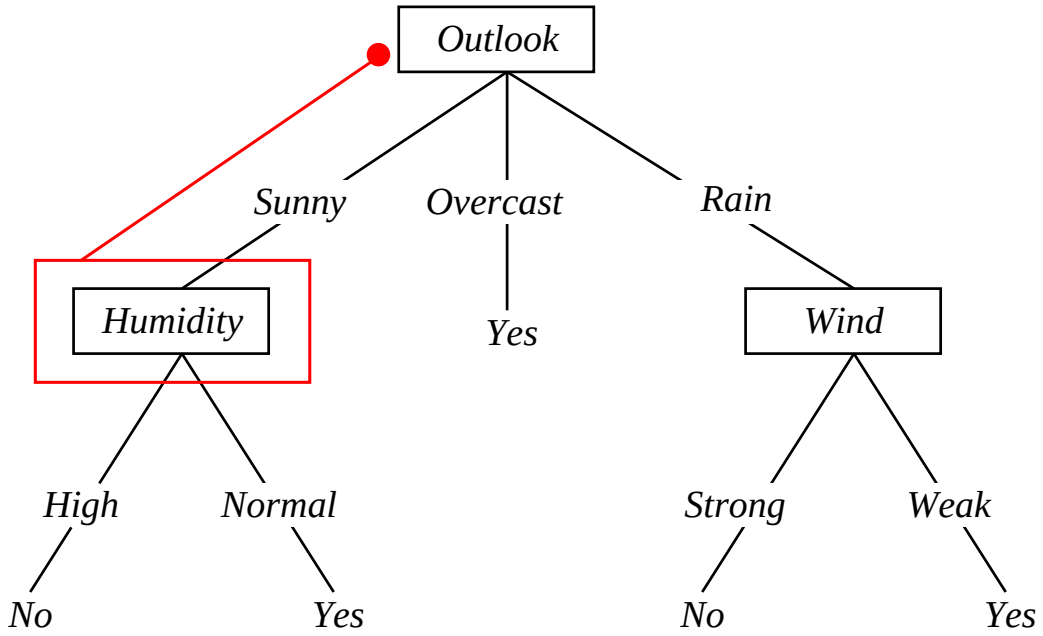
- Ogni nodo interno effettua un test su un attributo;
- Ogni ramo uscente da un nodo corrisponde ad uno dei possibili valori che l'attributo può assumere;
- Ogni foglia assegna una classificazione.

Per classificare una istanza con un Albero di Decisione bisogna

1. partire dalla radice;
2. selezionare l'attributo della istanza associato al nodo corrente;
3. seguire il ramo associato al valore assegnato a tale attributo nella istanza;
4. se si raggiunge una foglia restituire l'etichetta associata alla foglia, altrimenti a partire dal nodo corrente ripetere dal passo 2.

Alberi di Decisione: Classifi cazione

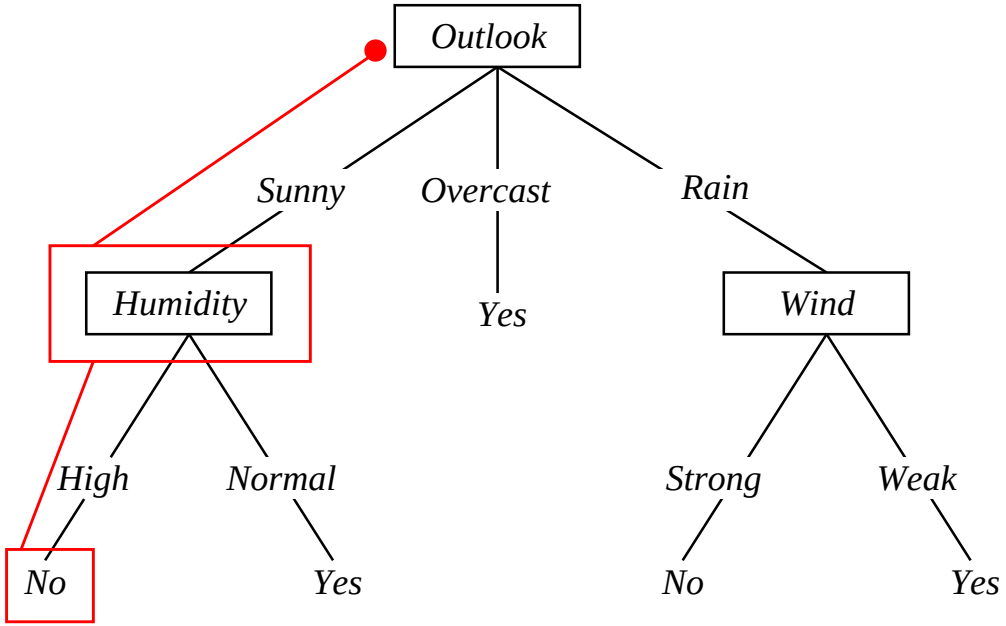
Alla radice è associato Outlook e quindi bisogna seguire il ramo *Sunny*



Es. ingresso: (Outlook=*Sunny*, Temperature=*Hot*, Humidity=*High*, Wind=*Strong*).

Alberi di Decisione: Classifi cazione

Al nodo raggiunto è associato Humidity e quindi bisogna seguire il ramo *High*



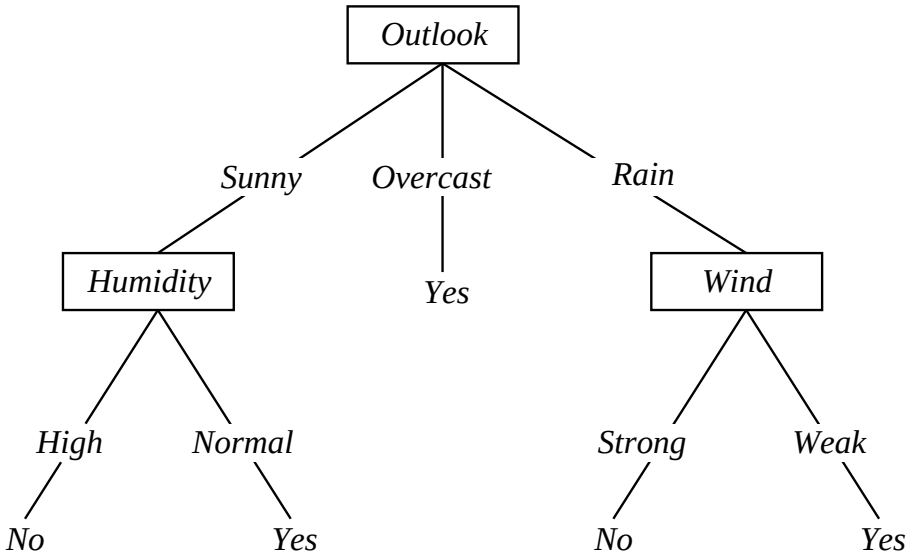
Es. ingresso: (Outlook=*Sunny*, Temperature=*Hot*, Humidity=*High*, Wind=*Strong*).

Si raggiunge una foglia: quindi l'istanza è classifi cata *No*

Alberi di Decisione e Funzioni Booleane

Con Alberi di Decisione ogni funzione booleana può essere rappresentata:

- Ogni cammino dalla radice ad una foglia codifica una congiunzione di test su attributi;
- Più cammini che conducono allo stesso tipo di classificazione codificano una disgiunzione di congiunzioni;



(Outlook=*Sunny* **and** Humidity=*Normal*)

or

Outlook=*Overcast*

or

(Outlook=*Rain* **and** Wind=*Weak*)

Esempio di Algoritmo di Apprendimento: ID3

L'apprendimento di Alberi di Decisione tipicamente procede attraverso una procedura di tipo (divide et impera) che costruisce l'albero top-down:

1. crea il nodo radice, $\hat{Tr} \leftarrow Tr$ e inserisci tutti gli attributi nell'insieme $\mathcal{A}$;
2. se gli esempi in $\hat{Tr}$ sono tutti della stessa classe (- o +), assegna al nodo l'etichetta della classe e fermati;
   altrimenti
   (a) se $\mathcal{A}$ è vuoto, assegna al nodo l'etichetta della classe più frequente e fermati;
   altrimenti assegna al nodo $A \leftarrow$ l'attributo "ottimo" in $\mathcal{A}$;
3. partiziona $\hat{Tr}$ secondo i possibili valori che A può assumere:
   $\hat{Tr}_{A=val_1}, \ldots, \hat{Tr}_{A=val_{m(A)}}$, dove $m(A) =$ numero valori distinti di A;
4. $\forall\, \hat{Tr}_{A=val_j} = \emptyset$ crea una foglia figlio con l'etichetta della classe più frequente in $\hat{Tr}$;
5. $\forall\, \hat{Tr}_{A=val_i} \neq \emptyset$ crea nodo figlio e vai a 2 con $\hat{Tr} \leftarrow \hat{Tr}_{A=val_i}$ e $\mathcal{A} \leftarrow \mathcal{A} \backslash A$.

Esempio di Algoritmo di Apprendimento: ID3

L'apprendimento di Alberi di Decisione tipicamente procede attraverso una procedura di tipo (divide et impera) che costruisce l'albero top-down:

1. crea il nodo radice, $\hat{Tr} \leftarrow Tr$ e inserisci tutti gli attributi nell'insieme $\mathcal{A}$;
2. se gli esempi in $\hat{Tr}$ sono tutti della stessa classe (- o +), assegna al nodo l'etichetta della classe e fermati;
   altrimenti
   (a) se $\mathcal{A}$ è vuoto, assegna al nodo l'etichetta della classe più frequente e fermati;
   altrimenti assegna al nodo $A \leftarrow$ l'attributo "ottimo" in $\mathcal{A}$;
3. partiziona $\hat{Tr}$ secondo i possibili valori che A può assumere:
   $\hat{Tr}_{A=val_1}, \ldots, \hat{Tr}_{A=val_{m(A)}}$, dove $m(A) =$ numero valori distinti di A;
4. $\forall\, \hat{Tr}_{A=val_j} = \emptyset$ crea una foglia figlio con l'etichetta della classe più frequente in $\hat{Tr}$;
5. $\forall\, \hat{Tr}_{A=val_i} \neq \emptyset$ crea nodo figlio e vai a 2 con $\hat{Tr} \leftarrow \hat{Tr}_{A=val_i}$ e $\mathcal{A} \leftarrow \mathcal{A} \backslash A$.

Esempio di Algoritmo di Apprendimento: ID3

L'apprendimento di Alberi di Decisione tipicamente procede attraverso una procedura di tipo (divide et impera) che costruisce l'albero top-down:

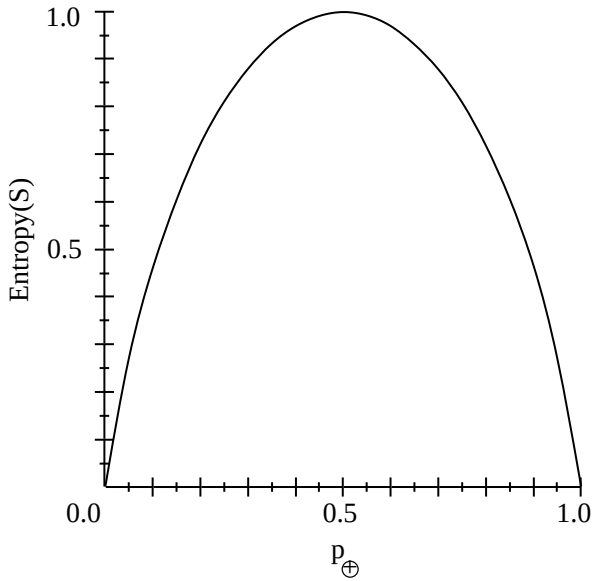
1. crea il nodo radice, $\hat{Tr} \leftarrow Tr$ e inserisci tutti gli attributi nell'insieme $\mathcal{A}$;
2. se gli esempi in $\hat{Tr}$ sono tutti della stessa classe (- o +), assegna al nodo l'etichetta della classe e fermati;
   altrimenti
   (a) se $\mathcal{A}$ è vuoto, assegna al nodo l'etichetta della classe più frequente e fermati;
   altrimenti assegna al nodo $A \leftarrow$ l'attributo "ottimo" in $\mathcal{A}$;
3. partiziona $\hat{Tr}$ secondo i possibili valori che A può assumere:
   $\hat{Tr}_{A=val_1}, \ldots, \hat{Tr}_{A=val_{m(A)}}$, dove $m(A) =$ numero valori distinti di A;
4. $\forall\, \hat{Tr}_{A=val_j} = \emptyset$ crea una foglia figlio con l'etichetta della classe più frequente in $\hat{Tr}$;
5. $\forall\, \hat{Tr}_{A=val_i} \neq \emptyset$ crea nodo figlio e vai a 2 con.. se necessario risalire ai nodi avi $\Uparrow$

ID3: Selezione Attributo Ottimo

Vari algoritmi di apprendimento si differenziano soprattutto (ma non solo) dal modo in cui si seleziona l'attributo ottimo: ID3: utilizza il concetto di *Entropia* e *Guadagno Entropico*

$$Entropia(S) = -p_- log_2(p_-) - p_+ log_2(p_+)$$

dove p_- (p_+) è la proporzione di esempi negativi (positivi) nell'insieme S



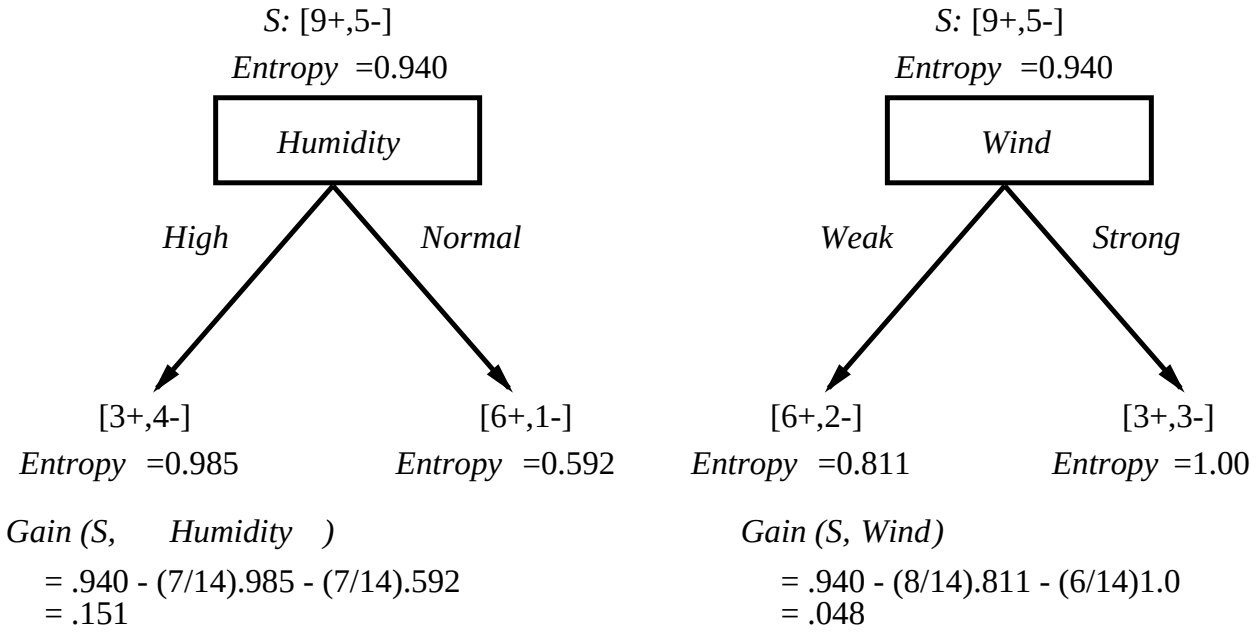
L' *Entropia* misura il "grado di purezza" dell'insieme degli esempi!

ID3: Selezione Attributo Ottimo

Si sceglie l'attributo A che massimizza il *Guadagno Entropico*:

$$Guadagno(S, A) = Entropia(S) - \sum_{v \in Valori(A)} \frac{|S_{A=v}|}{|S|} Entropia(S_{A=v})$$

Il *Guadagno* misura la riduzione aspettata della entropia nel partizionare i dati usando A



Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048