Apprendimento PAC

PAC Learning (Probably Approximately Correct Learning)

Assume che: input ed output sono binari, non c'è rumore, le istanze sono estratte da X concordemente ad una distribuzione di probabilità $\mathcal{D}$ *arbitraria* ma *stazionaria*.

Il framework di apprendimento PAC cerca di rispondere alle seguenti domande:

- Sotto quali condizioni apprendere con successo è possibile o impossibile ?
- Sotto quali condizioni si può assicurare che un particolare algoritmo di apprendimento apprenda con successo ?

Apprendimento PAC

Consideriamo una classe C di possibili concetti target (funzioni che vogliamo apprendere) definita su uno Spazio delle Istanze X (con istanze di dimensione m), ed un algoritmo di apprendimento L che utilizza uno Spazio delle Ipotesi $\mathcal{H}$.

Def.: C è PAC-apprendibile da L usando $\mathcal{H}$ se per ogni

- $c \in C$,
- distribuzione $\mathcal{D}$ su X,
- ϵ tale che $0 < \epsilon < 1/2$,
- δ tale che $0 < \delta < 1/2$,

l'algoritmo di apprendimento L con probabilità almeno $(1 - \delta)$ restituisce una ipotesi $h \in \mathcal{H}$ tale che $error_{\mathcal{D}}(h) \leq \epsilon$, in tempo che è **polinomiale** in $1/\epsilon$, $1/\delta$, m, e $size(c)$ (spazio di memoria necessario per rappresentare c).

Apprendimento PAC

Si può dimostrare che assumendo:

- $c \in \mathcal{H}$
- L consistente, cioè che restituisce una ipotesi h consistente con Tr, o equivalentemente $h \in VS_{\mathcal{H},Tr}$ (ad esempio, **Find-S** è consistente)

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

$$n \geq \frac{1}{\epsilon}(ln(|\mathcal{H}|) + ln(\frac{1}{\delta}))$$

dove $|\mathcal{H}|$ è la cardinalità dello Spazio delle Ipotesi e $ln()$ è il logaritmo naturale

Prova...

Idea base: bisogna dare un limite al numero di esempi necessario ad assicurare che il Version Space (ricordiamo che L è consistente) non contiene ipotesi non "accettabili":

$$(\forall h \in VS_{\mathcal{H},Tr})\ error_{\mathcal{D}}(h) < \epsilon$$

Prova...

$$(\forall h \in VS_{\mathcal{H},Tr})\ error_{\mathcal{D}}(h) < \epsilon$$

Primo risultato: se $|\mathcal{H}| < \infty$, $|Tr| = n > 0$ e Tr è costituito da esempi di un concetto target c estratti indipendentemente ed a caso, allora per ogni $0 \leq \epsilon \leq 1$, la probabilità che la disuguaglianza **NON** sia soddisfatta è minore od uguale a $|\mathcal{H}|e^{-\epsilon n}$:

- siano $h_1, \ldots, h_k$ tutte le ipotesi con $error_{\mathcal{D}}(h) \geq \epsilon$; la disuguaglianza NON è soddisfatta se e solo se ALMENO una di tali ipotesi è consistente con Tr
- la probabilità che una di tali ipotesi sia consistente con un singolo esempio è $(1-\epsilon)$, e quindi per n esempi indipendenti la probabilità è $(1-\epsilon)^n$
- poiché esistono k di tali ipotesi, la probabilità che ci interessa è a più $k(1-\epsilon)^n \leq |\mathcal{H}|(1-\epsilon)^n$ (poiché $k \leq |\mathcal{H}|$)
- infine, poiché $(1-\epsilon) \leq e^{-\epsilon}$ se $0 \leq \epsilon \leq 1$, abbiamo $|\mathcal{H}|(1-\epsilon)^n \leq |\mathcal{H}|e^{-\epsilon n}$

Prova...

Cosa abbiamo ottenuto: abbiamo un limite superiore alla probabilità che usando un Tr con n esempi, il Version Space contenga qualche ipotesi CATTIVA, cioè che L restituisca h, una delle ipotesi cattive, per cui $error_{\mathcal{D}}(h) \geq \epsilon$!

Quindi, se vogliamo che tale probabilità sia inferiore a livello desiderato δ

$$|\mathcal{H}|e^{-\epsilon n} \leq \delta$$

bisogna usare un valore per n per cui

$$n \geq \frac{1}{\epsilon}(ln(|\mathcal{H}|) + ln(\frac{1}{\delta}))$$

o, alternativamente, se si usa un tale valore per n, con probabilità $1 - \delta$ l'ipotesi h restituita da L avrà $error_{\mathcal{D}}(h) < \epsilon$

Apprendimento PAC: esempio $\mathcal{H}_4$

Congiunzione di m letterali

- Spazio delle Istanze $\rightarrow$ stringhe di m bit: $X = \{s | s \in \{0,1\}^m\}$
- Spazio delle Ipotesi $\rightarrow$ tutte le sentenze logiche che riguardano i letterali $l_1, \ldots, l_m$ (anche in forma negata, $\neg l_i$) e che contengono solo l'operatore $\wedge$ (**and**):

$$\mathcal{H} = \{f_{\{i_1,\ldots,i_j\}}(s) | f_{\{i_1,\ldots,i_j\}}(s) \equiv L_{i_1} \wedge L_{i_2} \wedge \cdots \wedge L_{i_j},$$
$$\text{dove } L_{i_k} = l_{i_k} \text{ oppure } \neg l_{i_k}, \{i_1, \ldots, i_j\} \subseteq \{1, \ldots, 2m\}\}$$

Notare che se in una formula un letterale compare sia affermato che negato, allora la formula ha sempre valore di verità $false$ (formula non soddisfacibile)

Quindi, tutte le formule che contengono almeno un letterale sia affermato che negato sono equivalenti alla funzione che vale sempre $false$

Apprendimento PAC: esempio $\mathcal{H}_4$

E' la congiunzione di m letterali PAC-apprendibile usando **Find-S** con $\mathcal{H}_4$?

Apprendimento PAC: esempio $\mathcal{H}_4$

E' la congiunzione di m letterali PAC-apprendibile usando **Find-S** con $\mathcal{H}_4$? Si !

Infatti:

- ogni congiunzione di m letterali è inclusa in $\mathcal{H}_4$

Apprendimento PAC: esempio $\mathcal{H}_4$

E' la congiunzione di m letterali PAC-apprendibile usando **Find-S** con $\mathcal{H}_4$? Si !

Infatti:

- ogni congiunzione di m letterali è inclusa in $\mathcal{H}_4$
- **Find-S** è consistente

Apprendimento PAC: esempio $\mathcal{H}_4$

E' la congiunzione di m letterali PAC-apprendibile usando **Find-S** con $\mathcal{H}_4$? Si !

Infatti:

- ogni congiunzione di m letterali è inclusa in $\mathcal{H}_4$
- **Find-S** è consistente
- $|\mathcal{H}_4| = 3^m + 1$ e poiché $\frac{1}{\epsilon}(ln(3^{m+1}) + ln(\frac{1}{\delta})) > \frac{1}{\epsilon}(ln(3^m + 1) + ln(\frac{1}{\delta}))$

$$n \geq \frac{1}{\epsilon}((m+1)ln(3) + ln(\frac{1}{\delta}))$$

quindi n è polinomiale in $1/\epsilon$, $1/\delta$, m, e $size(c)$ (che non compare)

Apprendimento PAC: esempio $\mathcal{H}_4$

E' la congiunzione di m letterali PAC-apprendibile usando **Find-S** con $\mathcal{H}_4$? Si !

Infatti:

- ogni congiunzione di m letterali è inclusa in $\mathcal{H}_4$
- **Find-S** è consistente
- $|\mathcal{H}_4| = 3^m + 1$ e poiché $\frac{1}{\epsilon}(ln(3^{m+1}) + ln(\frac{1}{\delta})) > \frac{1}{\epsilon}(ln(3^m + 1) + ln(\frac{1}{\delta}))$

$$n \geq \frac{1}{\epsilon}((m+1)ln(3) + ln(\frac{1}{\delta}))$$

quindi n è polinomiale in $1/\epsilon$, $1/\delta$, m, e $size(c)$ (che non compare)

- per ogni esempio di apprendimento **Find-S** impiega tempo lineare nella dimensione della ipotesi corrente (e tale dimensione è $\geq size(c)$) e nella dimensione dell'input (m), quindi di nuovo polinomiale, e globalmente è polinomiale per Tr

Apprendimento PAC: esempio $\mathcal{H}_4$

E' la congiunzione di m letterali PAC-apprendibile usando **Find-S** con $\mathcal{H}_4$? Si !

Infatti:

- ogni congiunzione di m letterali è inclusa in $\mathcal{H}_4$
- **Find-S** è consistente
- $|\mathcal{H}_4| = 3^m + 1$ e poiché $\frac{1}{\epsilon}(ln(3^{m+1}) + ln(\frac{1}{\delta})) > \frac{1}{\epsilon}(ln(3^m + 1) + ln(\frac{1}{\delta}))$

$$n \geq \frac{1}{\epsilon}((m+1)ln(3) + ln(\frac{1}{\delta}))$$

  quindi n è polinomiale in $1/\epsilon$, $1/\delta$, m, e $size(c)$ (che non compare)
- per ogni esempio di apprendimento **Find-S** impiega tempo lineare nella dimensione della ipotesi corrente (e tale dimensione è $\geq size(c)$) e nella dimensione dell'input (m), quindi di nuovo polinomiale, e globalmente è polinomiale per Tr

Quindi tutte le condizioni per la PAC-apprendibilità sono soddisfatte !

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 definito 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 fini 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 più specifica $h \equiv l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \wedge \cdots \wedge l_m \wedge \neg l_m$

do forever /* in effetti, fino 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 finale 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 difficile modificare **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 più a sinistra e più 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 sufficiente 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 più 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.

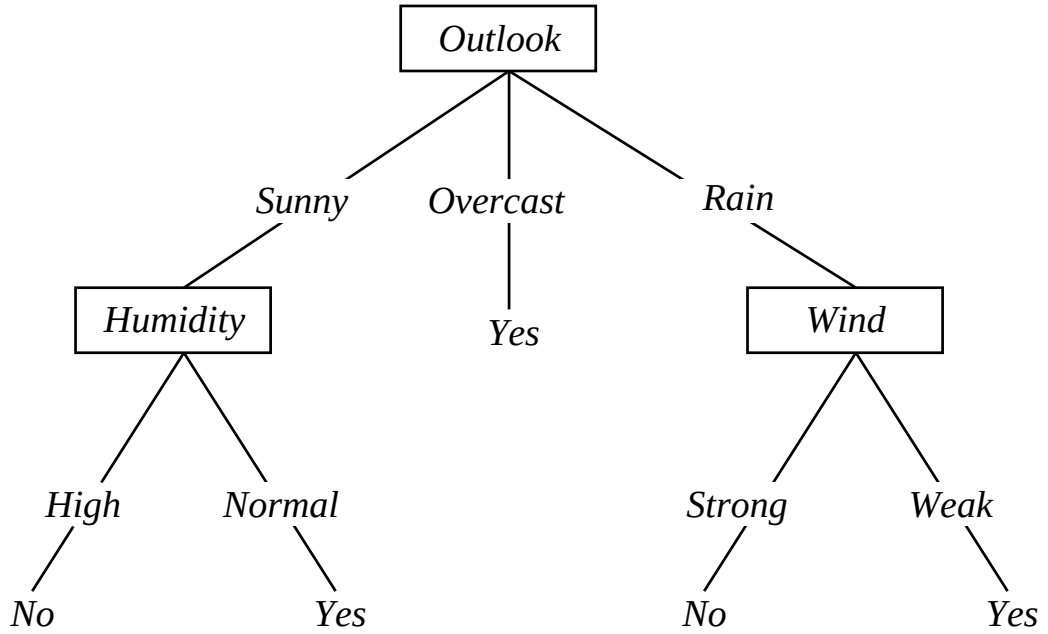
Giocare a Tennis!!

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:

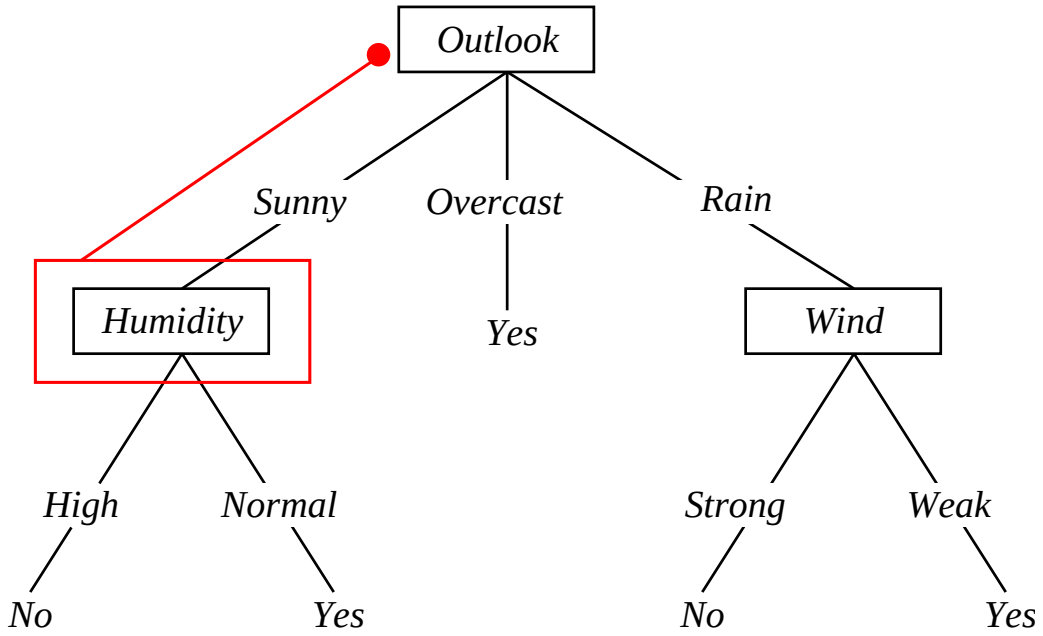
- Ogni nodo interno effettua un test su un attributo;
- Ogni ramo uscente da un nodo corrisponde ad uno dei possibili valori che l'attributo può assumere;
- Ogni foglia assegna una classificazione.

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: Classificazione

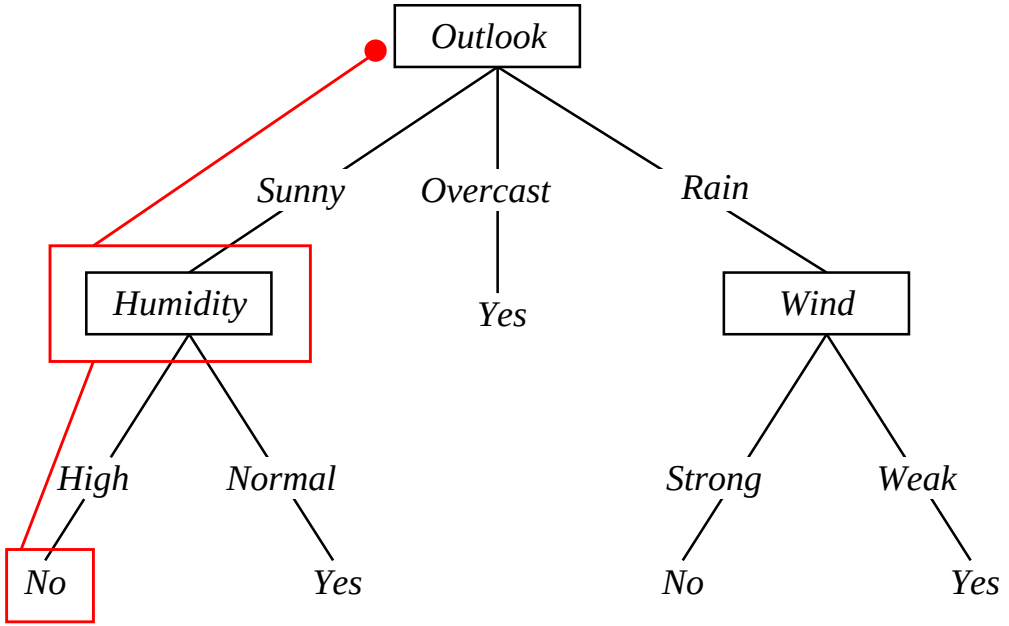
Alla radice è associato Outlook e quindi bisogna seguire il ramo *Sunny*



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

Alberi di Decisione: Classificazione

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



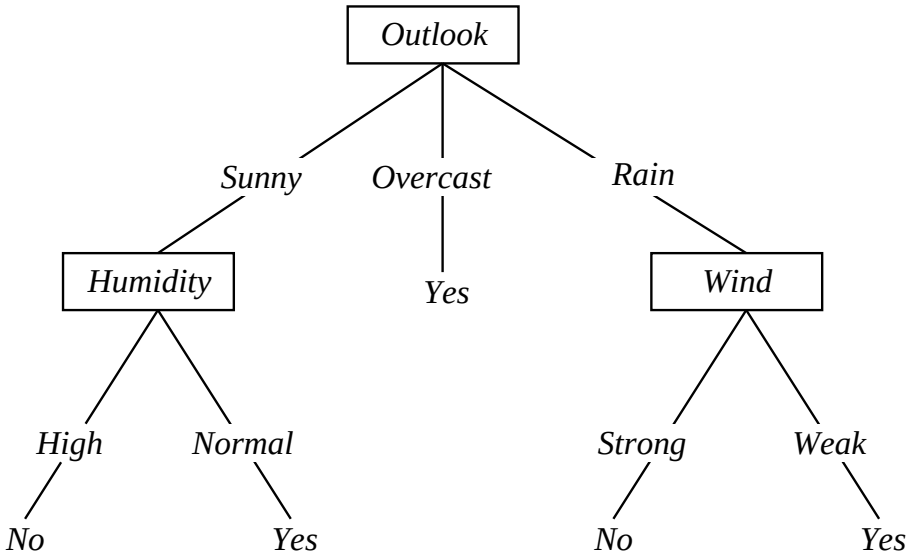
Es. ingresso: (Outlook=*Sunny*, Temperature=*Hot*, Humidity=*High*, Wind=*Strong*).

Si raggiunge una foglia: quindi l'istanza è classificata *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:

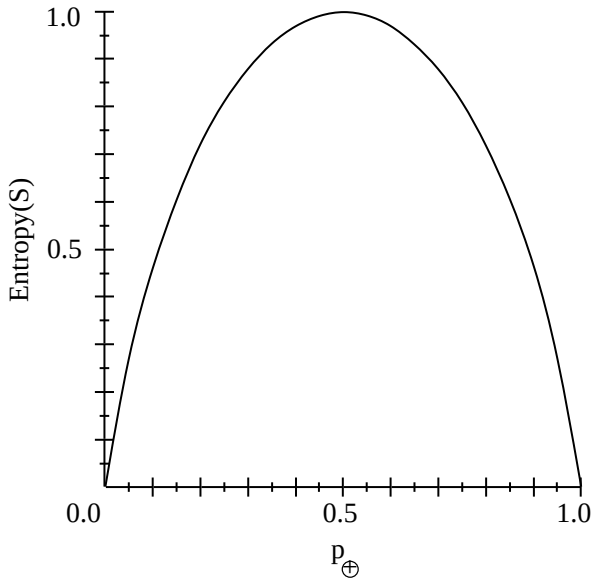
1. crea il nodo radice, $\hat{Tr} \leftarrow Tr$ e inserisci tutti gli attributi nell'insieme $\mathcal{A}$;
2. se gli esempi in $\hat{Tr}$ sono tutti della stessa classe (- o +), assegna al nodo l'etichetta della classe e fermati;
   altrimenti
   (a) se $\mathcal{A}$ è vuoto, assegna al nodo l'etichetta 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



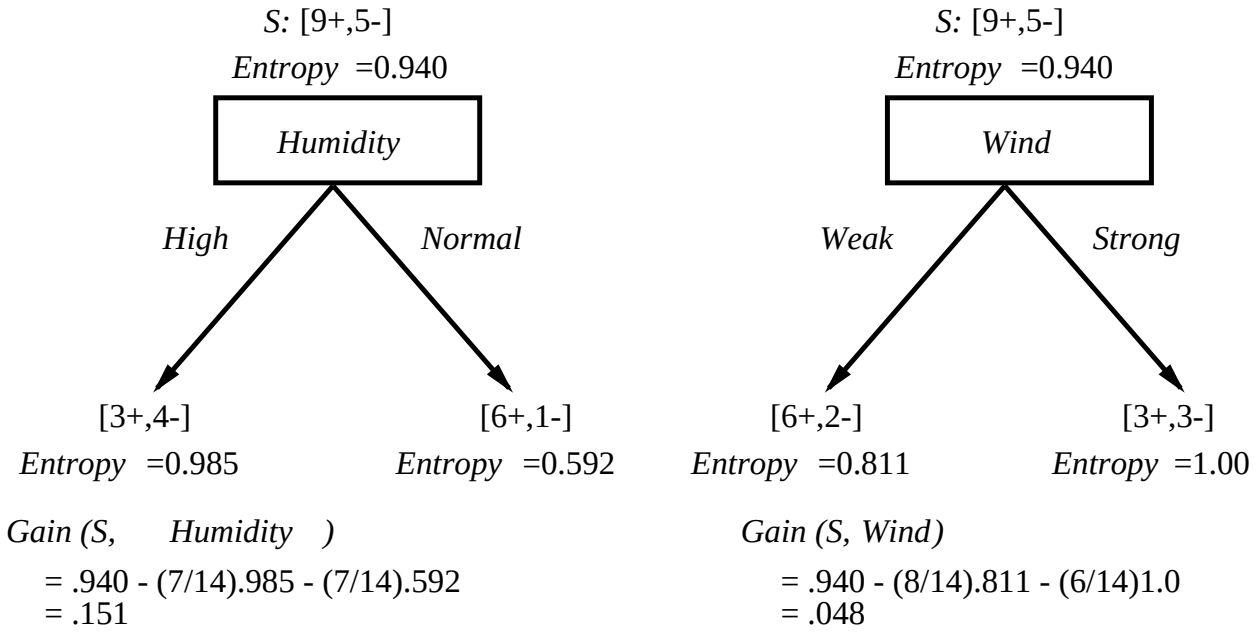
L' *Entropia* misura il "grado di purezza" dell'insieme degli esempi!

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



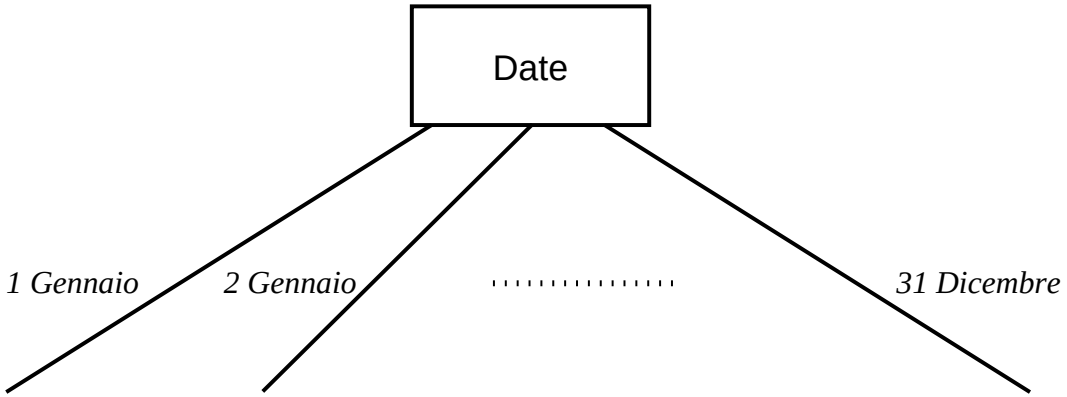
Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

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

Selezione Attributo Ottimo: Problema

Problema: Il *Guadagno* favorisce troppo attributi che possono assumere tanti valori diversi.

Esempio: se al problema di decidere quando giocare a tennis si aggiunge un attributo che consiste nella data del giorno considerato (es. Data=*"11 Novembre"*), allora l'attributo Date è quello che avrà guadagno massimo (ogni giorno costituirà un sottoinsieme diverso e puro, quindi con entropia a 0), anche se in realtà non è significativo.



Selezione Attributo Ottimo: Gain Ratio

Per rimediare a questo problema si definisce il *Gain Ratio*:

$$GainRatio(S, A) = \frac{Guadagno(S, A)}{SplitInformation(S, A)}$$

dove

$$SplitInformation(S, A) = -\sum_{v \in Valori(A)} \frac{|S_{A=v}|}{|S|} \log_2 (\frac{|S_{A=v}|}{|S|})$$

La $SplitInformation$ misura quanti, e quanto uniformi sono, i sottoinsiemi generati dall'attributo A a partire dall'insieme S.

$SplitInformation$ corrisponde alla entropia di S dati i valori di A.

Si osservi che il termine $SplitInformation$ sfavorisce attributi che suddividono S in molti sottoinsiemi tutti della stessa cardinalità.

Selezione Attributo Ottimo: Gain Ratio

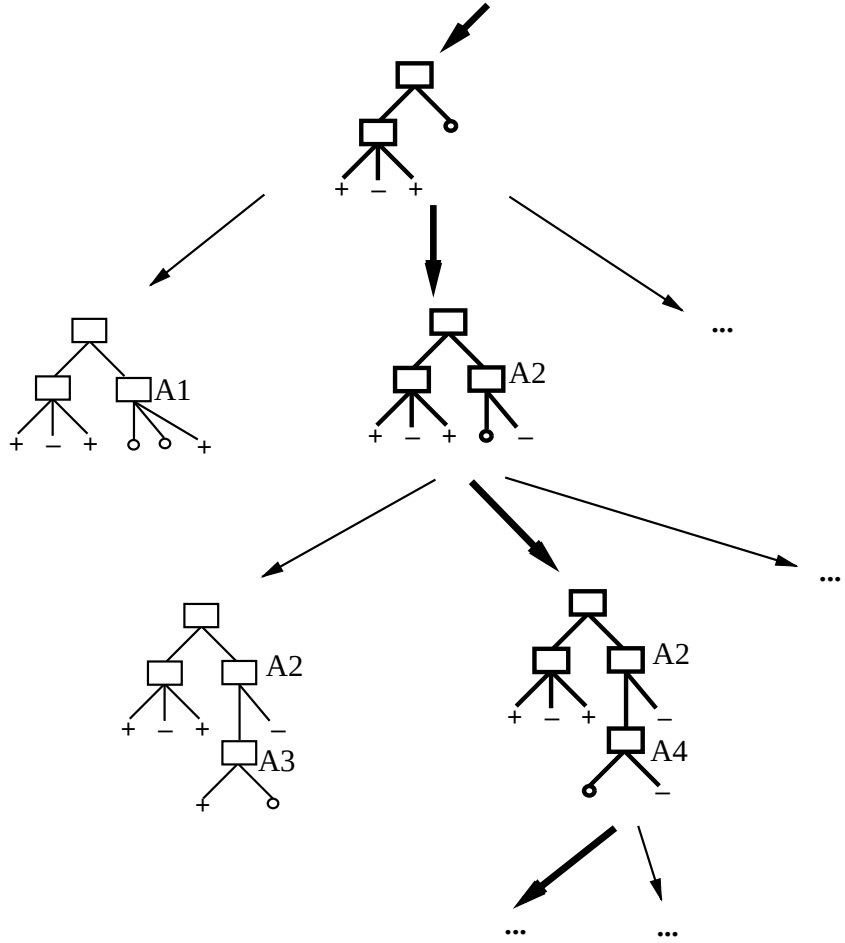
$GainRatio$ non risolve tutti i problemi. Infatti potrebbe succedere che attibuti significativi, ma che assumono qualche valore in più rispetto agli altri, potrebbero essere sfavoriti.

Di solito la strategia utilizzata è la seguente:

1. si calcola il $Guadagno$ per ogni attributo;
2. si calcola la media dei guadagni calcolati;
3. si selezionano SOLO gli attributi che hanno $Guadagno$ al di sopra della media;
4. si sceglie, fra gli attributi selezionati, quello che ha $GainRatio$ maggiore.

Apprendimento di Alberi di Decisione: Bias Induttivo

Il Bias Induttivo è sulla ricerca !



Attributi Continui

Fino ad ora abbiamo considerato attributi a valori discreti.

Cosa succede se un attributo è a valori continui ?

Esempio

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	90 (Hot)	High	Weak	No
D2	Sunny	80 (Hot)	High	Strong	No
D3	Overcast	72 (Hot)	High	Weak	Yes
D4	Rain	60 (Mild)	High	Weak	Yes
D5	Rain	40 (Cool)	Normal	Weak	Yes
D6	Rain	48 (Cool)	Normal	Strong	No
D7	Overcast	40 (Cool)	Normal	Strong	Yes
D8	Sunny	60 (Mild)	High	Weak	Yes
D9	Sunny	40 (Cool)	Normal	Weak	Yes
. . .	. . .	. . .	. . .	. . .	. . .

Attributi Continui

Soluzione: a partire dall'attributo A, creare dinamicamente l'attributo booleano

$$A_c = \begin{cases} true & \text{se } A < c \\ false & \text{altrimenti} \end{cases}$$

Come selezionare il valore "giusto" per c ?

Possibile soluzione: scegliere il c che corrisponde al $Guadagno$ massimo !

E' stato dimostrato che il valore ottimo (massimo del $Guadagno$) si localizza nel valore di mezzo fra due valori con target diverso:

Esempio: tagli candidati

esempi	{D5,D7,D9}	{D6}	{D4,D8}	{D3}	{D2}	{D1}
valore	40	48	60	72	80	90
target	yes	no	yes	yes	no	no
		↑ ↑		↑		
valore taglio		(44) (54)		(76)		

Attributi Continui

Quindi basta calcolare il $Guadagno$ per

$c = 44$ $\underbrace{\{D5, D7, D9\}}_{Temp<44}$ $\underbrace{\{D1, D2, D3, D4, D6, D8\}}_{Temp\geq 44}$ $Guadagno \simeq \boxed{0.379}$

$c = 54$ $\underbrace{\{D5, D6, D7, D9\}}_{Temp<54}$ $\underbrace{\{D1, D2, D3, D4, D8\}}_{Temp\geq 54}$ $Guadagno \simeq 0.091$

$c = 76$ $\underbrace{\{D3, D4, D5, D6, D7, D9\}}_{Temp<76}$ $\underbrace{\{D1, D2, D8\}}_{Temp\geq 76}$ $Guadagno \simeq 0.093$

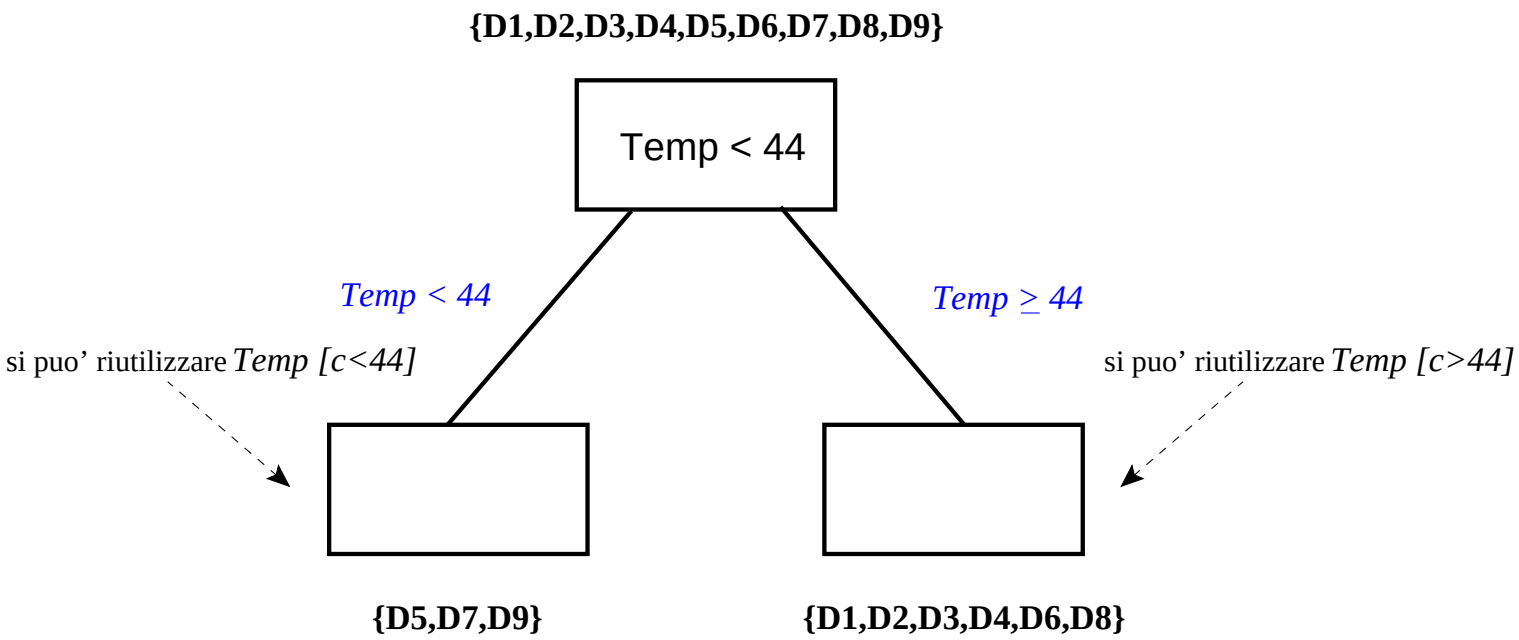
Quindi si seleziona $c = 44$ poiché ottiene guadagno massimo !

Test finale: $Temp < 44$?

Attributi Continui

ATTENZIONE !

Lo stesso attributo può essere riutilizzato sullo stesso cammino ...



Ovviamente, invece di utilizzare $Guadagno$ si può anche utilizzare $GainRatio$

Attributi con Costi

Osservazione: verificare il valore che un attributo assume può avere un costo !

Esempio: Diagnosi Medica

Attributo	Costo
esami sangue	$\in 7$
radiografia	$\in 15$
visita medica	$\in 40$
T.A.C.	$\in 150$

E' possibile tener conto di tali costi nel costruire l'albero di decisione ?

Preferire attributi poco costosi: se possibile posizionarli in prossimità della radice (probabilità di verificarli più alta; es. probabilità 1 per la radice)

Usare un criterio di selezione dell'attributo (ottimo) che inglobi i costi

Esempio per la Diagnosi Medica: $\frac{2^{Guadagno(S,A)}-1}{(Costo(A)+1)^w}$ con $w \in [0,1]$

Esempio per la Percezione Robotica: $\frac{Guadagno^2(S,A)}{Costo(A)}$

Attributi con Valori Mancanti

Problema: nelle applicazioni pratiche può succedere che per alcuni esempi, alcuni attributi non abbiano alcun valore assegnato (valore mancante)

Esempio: Diagnosi Medica

- per il paziente 38 manca il risultato della T.A.C.;
- per il paziente 45 mancano gli esami del sangue e la radiografia;

Possibili soluzioni: dato un nodo n con associato l'insieme di esempi $\hat{Tr}$, quando per un esempio $(x, target) \in \hat{Tr}$ manca il valore di A

a) utilizzare per A il valore in $Valori(A)$ più frequente in $\hat{Tr}$;

b) come a), però considerare solo esempi con target uguale a quello dell'esempio corrente

c) considerare tutti i valori $a_i \in Valori(A)$ e la loro probabilità di occorrere $prob(a_i|\hat{Tr})$, stimata su $\hat{Tr}$, e sostituire l'esempio $(x, target)$ con $|Valori(A)|$ istanze "frazionarie", una per ogni valore $a_i \in Valori(A)$ (dove $A = a_i$) e peso uguale a $prob(a_i|\hat{Tr})$

Attributi con Valori Mancanti

Esempio a)

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	_	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

Consideriamo l'attributo Outlook e l'esempio D5:

esempio originario → nuovo esempio

$$D5 \equiv (\_, Cool, Normal, Weak) \rightarrow \quad D5' \equiv (\boxed{Sunny}, Cool, Normal, Weak)$$

infatti: $\boxed{prob(Sunny|\hat{Tr}) = 4/8}, prob(Overcast|\hat{Tr}) = prob(Rain|\hat{Tr}) = 2/8$

Attributi con Valori Mancanti

Esempio b)

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	_	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

Consideriamo l'attributo Outlook e l'esempio D5:

esempio originario → nuovo esempio

$$D5 \equiv (\_, Cool, Normal, Weak) \rightarrow \quad D5' \equiv (\boxed{Overcast}, Cool, Normal, Weak)$$

infatti: $\boxed{prob(Overcast|target = yes, \hat{Tr}) = 2/4}$,

$$prob(Sunny|target = yes, \hat{Tr}) = prob(Rain|target = yes, \hat{Tr}) = 1/4$$

Attributi con Valori Mancanti

Esempio c)

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	_	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

Consideriamo l'attributo Outlook e l'esempio D5:

esempio originario		esempio frazionario	peso=$prob(a_i\|\hat{Tr})$
	$\nearrow$	$D5_S \equiv (Sunny, Cool, Normal, Weak)$	$4/8$
$D5 \equiv (\_, Cool, Normal, Weak)$	$\rightarrow$	$D5_O \equiv (Overcast, Cool, Normal, Weak)$	$2/8$
	$\searrow$	$D5_R \equiv (Rain, Cool, Normal, Weak)$	$2/8$

Attributi con Valori Mancanti

Esempio c)

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’	_	Cool	Normal	_	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

Consideriamo l’esempio D5’ e gli attributi Outlook e Wind

Se più attributi hanno valori mancanti, si continua a frazionare gli esempi già frazionati ed il peso associato all’esempio frazionato è dato dal prodotto dei pesi ottenuti considerando un solo attributo a turno

Attributi con Valori Mancanti

Esempio c)

esempio originario		esempio frazionario	$prob(a_i, b_j \mid \hat{Tr})$
	$\nearrow$	$D5'_{S,W} \equiv (Sunny, Cool, Normal, Weak)$	$4/8 \cdot 5/8$
	$\nearrow$	$D5'_{S,S} \equiv (Sunny, Cool, Normal, Strong)$	$4/8 \cdot 3/8$
	$\nearrow$	$D5'_{O,W} \equiv (Overcast, Cool, Normal, Weak)$	$2/8 \cdot 5/8$
$D5' \equiv (\_, Cool, Normal, \_)$			
	$\searrow$	$D5'_{O,S} \equiv (Overcast, Cool, Normal, Strong)$	$2/8 \cdot 3/8$
	$\searrow$	$D5'_{R,W} \equiv (Rain, Cool, Normal, Weak)$	$2/8 \cdot 5/8$
	$\searrow$	$D5'_{R,S} \equiv (Rain, Cool, Normal, Strong)$	$2/8 \cdot 3/8$

dove $a_i \in Valori(Outlook)$ e $b_j \in Valori(Wind)$

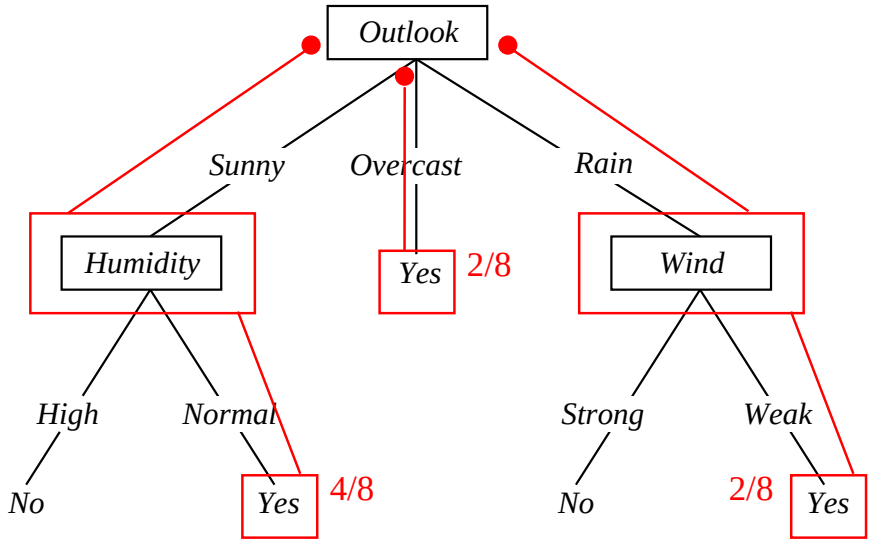
Si fa una assunzione di *indipendenza* degli attributi:

$$prob(a_i, b_j \mid \hat{Tr}) = prob(a_i \mid \hat{Tr}) \cdot prob(b_j \mid \hat{Tr})$$

Attributi con Valori Mancanti

Classificazione di D5 con metodo c)

Per ogni esempio frazionario si esegue la classificazione e per ogni possibile etichetta di classificazione si sommano i pesi degli esempi che raggiungono foglie con quella etichetta:



Prob(Yes) = 4/8 + 2/8 + 2/8 = 1

Prob(No) = 0

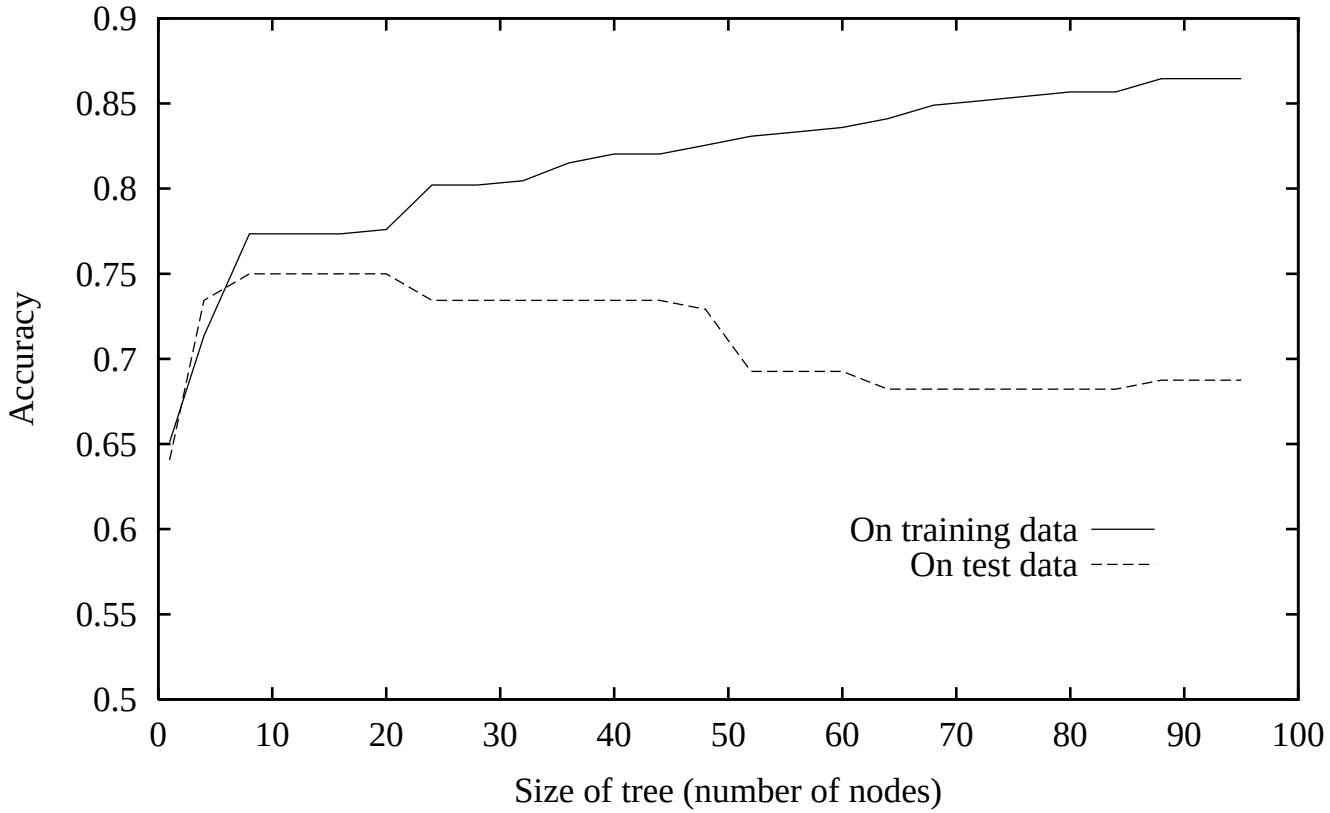
Attributi con Valori Mancanti

Apprendimento con metodo c)

- modificare il concetto di cardinalità di un insieme in modo da considerare i pesi frazionari;
- modificare la definizione di $Guadagno$ conseguentemente;

Apprendimento di Alberi di Decisione: Overfitting

Problema: Overfitting !



Soluzione: Algoritmi di Potatura (Pruning)...

Potatura

Problema 1: Come effettuare la potatura ?

Problema 2: Quando fermarsi con la potatura (o alternativamente con l'apprendimento) ?

Quando Fermarsi...

Problema 1: Come effettuare la potatura ?

Problema 2: Quando fermarsi con la potatura (o alternativamente con l'apprendimento) ?

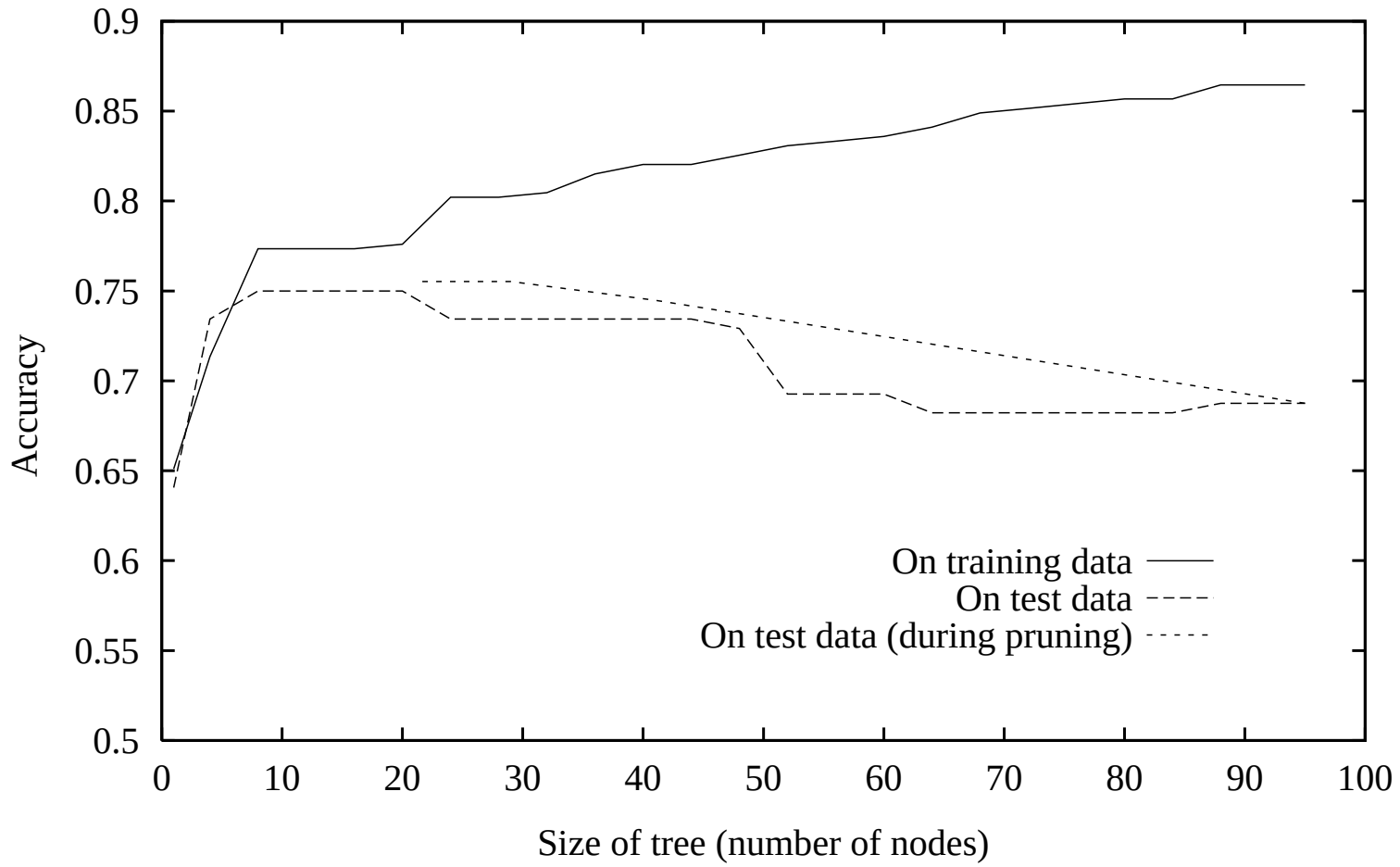
Consideriamo il Problema 2. Le principali "soluzioni" sono:

1. Valutare le prestazione sull'insieme di apprendimento (usando un test statistico);
2. Valutare le prestazioni su un insieme (separato) di validazione;
3. Usare un principio di minimizzazione della lunghezza di descrizione (MDL)

$$\min_{\text{Tree}} \left[size(\text{Tree}) + size(errori(\text{Tree})) \right]$$

In ogni caso è difficile pervenire ad una soluzione ottima...

Usando l'insieme di validazione



Come Potare...

Problema 1: Come effettuare la potatura ? Le principali “soluzioni” sono:

1. Reduced Error Pruning

   - Dividere Tr in $Tr\prime$ e Vs (insieme di validazione);
   - Ripetere fino a quando le prestazioni peggiorano

   (a) per ogni nodo (interno) n valutare l’impatto su Vs avendo potato il nodo (e i suoi discendenti);

   (b) effettuare la potatura che porta alle prestazioni migliori su Vs;

   (potatura: al sotto-albero radicato in n si sostituisce una foglia con etichetta uguale alla classe più frequente nell’insieme degli esempi associati al nodo n. In alternativa si può usare la tecnica del “subtree raising”.)

2. Rule-Post Pruning

Rule-Post Pruning

L'idea di base è quella di trasformare un albero di decisione in un insieme di regole, e poi effettuare la potatura delle regole:

1. Si genera una regola R_i per ogni cammino $path(r, f_i)$ dalla radice r alla foglia i-esima f_i; R_i sarà nella forma

   IF $\underbrace{(Attr_{i_1} = v_{i_1}) \wedge (Attr_{i_2} = v_{i_2}) \wedge \cdots \wedge (Attr_{i_k} = v_{i_k})}_{precondizioni}$ THEN $label_{f_i}$

2. Si effettua la potatura indipendentemente su ogni regola R_i:
   - si stimano le prestazioni ottenute utilizzando SOLO R_i come classificatore;
   - si rimuovono le precondizioni (una o più) che conducono ad un aumento della stima delle prestazioni usando un approccio greedy;
3. Si ordinano le R_i potate per ordine decrescente di prestazione (evita conflitti); eventualmente, aggiunge come classificazione di default la classe più frequente;

Classificare una nuova istanza

La classificazione di una nuova istanza da parte delle regole ordinate avviene seguendo l'ordine stabilito per le regole:

- la prima regola la cui precondizione è soddisfatta dalla istanza è usata per generare la classificazione
- se nessuna regola ha le condizioni soddisfatte, si utilizza la regola di default per classificare l'istanza (cioè si ritorna la classe più frequente nell'insieme di apprendimento);

Considerazioni

Alcune considerazioni sul Rule-Post Pruning:

- la stima delle prestazioni necessaria per effettuare la potatura può essere fatta sia usando un insieme di validazione che utilizzando un test statistico sui dati di apprendimento;
- la trasformazione Albero $\rightarrow$ Regole permette di generare regole dove si possono considerare contesti per un nodo che non necessariamente contengono i suoi nodi avi (e in particolare la radice);
- di solito le regole sono più semplici da comprendere per un umano;

In genere il Rule-Post Pruning riesce a migliorare le prestazioni dell'albero di decisione di partenza e si comporta meglio del Reduced-Error Pruning

Criteri alternativi all'Entropia

Altri criteri suggeriti al posto dell'Entropia:

- **Variance Impurity** (per due classi)

$$p_- \cdot p_+$$

- **(Weighted) Gini Impurity** (generalizzazione di Variance Impurity a più di 2 classi)

$$\sum_{c,c' \in Classi, c \neq c'} \lambda_{c,c'} p_c \cdot p_{c'}$$

  dove $\lambda_{c,c'}$ sono parametri di costo per classificazioni errate

- **Misclassification Impurity**

$$1 - \max_{c \in Classi} p_c$$