



Intelligenza Artificiale

**Apprendimento tramite
osservazioni**



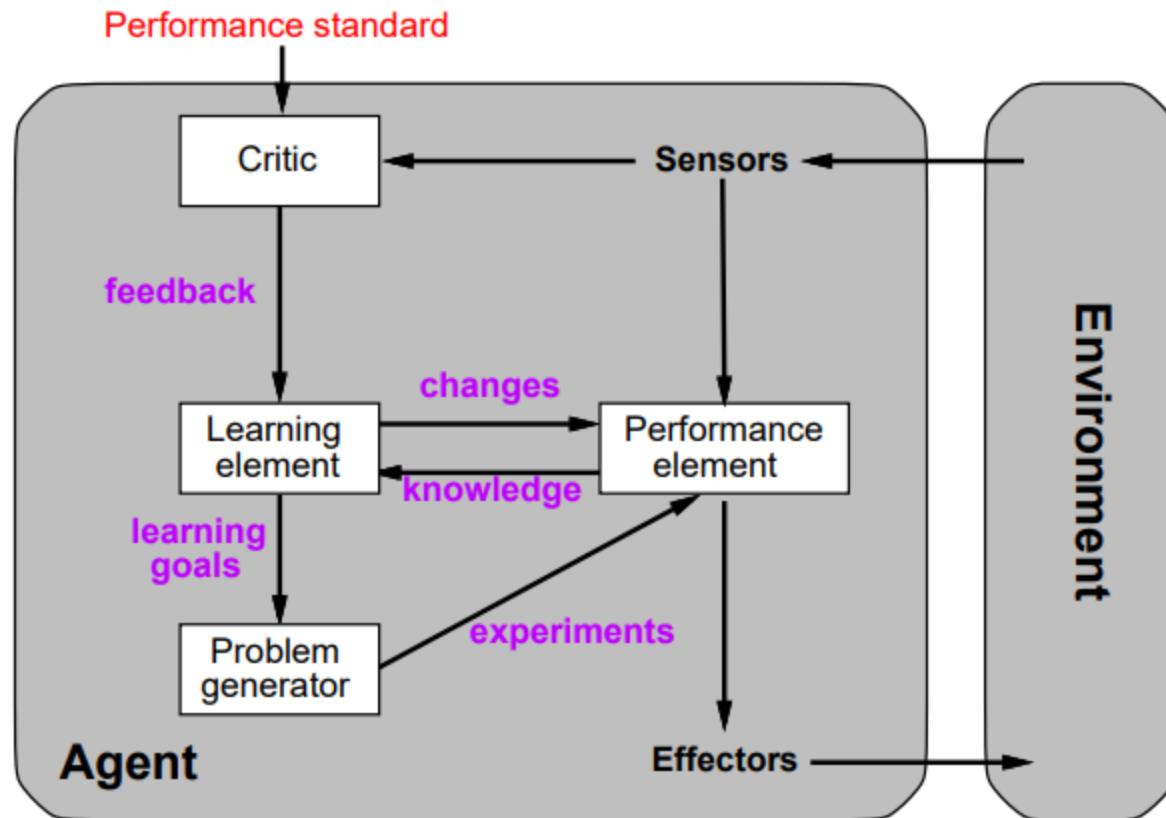
Outline

- ▶ Agenti di apprendimento
- ▶ Apprendimento induttivo
- ▶ Apprendere alberi di decisione
- ▶ Valutare le prestazioni di apprendimento
- ▶ Selezione del modello e ottimizzazione
- ▶ Regressione lineare e classificazione
- ▶ Modelli non parametrici

Apprendimento

- ▶ L'apprendimento è **essenziale** in ambienti sconosciuti
 - ▶ quando il progettista manca di onniscienza
- ▶ L'apprendimento è utile come metodo di costruzione del sistema,
 - ▶ esporre l'agente alla realtà piuttosto che cercare di descriverla
- ▶ L'apprendimento modifica i meccanismi di decisione dell'agente al fine di migliorarne le prestazioni

Agenti di apprendimento



Elemento di apprendimento

Il design di un ***Learning Element*** è dettato:

- ▶ dal tipo di ***Performance Element*** usato
- ▶ da quali componenti del ***Performance Element*** devono essere appresi
- ▶ da come è **rappresentato** quel particolare componente funzionale
- ▶ dal tipo di **feedback** disponibile per l'apprendimento

Scenari d'esempio:

Performance element	Componente	Rappresentazione	Feedback
Alpha–beta pruning search	Eval. Function	Weighted linear function	Win/loss
Logical agent	Transition model	Successor–state axioms	Outcome
Utility-based agent	Transition model	Rete Bayesiana	Outcome
Simple reflex agent	Percept–action function	Rete Neurale	Correct action

Apprendimento supervisionato: risposte corrette per ogni istanza

Apprendimento rinforzato: ricompense occasionali

Feedback (tipi di apprendimento)

- ▶ Apprendimento supervisionato
 - ▶ Apprendere una funzione da esempi input/output
 - ▶ In ambienti completamente osservabili può vedere i risultati delle azioni ed imparare a predirli
 - ▶ In ambienti parzialmente osservabili gli effetti potranno essere invisibili
- ▶ Apprendimento non supervisionato
 - ▶ Imparare a riconoscere pattern nell'input senza alcuna indicazione dell'output
- ▶ Apprendimento per rinforzo
 - ▶ l'agente apprende basandosi sul rinforzo (ricompense)

Apprendimento induttivo

Forma più semplice: apprendere una funzione dagli esempi (**tabula rasa**)

f è la **funzione obiettivo**

Un **esempio** è la coppia $x, f(x)$, es. $\left(\begin{array}{|c|c|c|} \hline o & o & x \\ \hline & x & \\ \hline x & & \\ \hline \end{array}, +1 \right)$

Problema apprendimento induttivo: trovare un **ipotesi** $h \in H$ spazio ipotesi
tale che $h \approx f$
dato un **training set** di esempi

(Questo è un modello altamente semplificato di apprendimento reale, infatti:

- Ignora la conoscenza precedente
- Assume la presenza di un «ambiente» deterministico ed osservabile
- Assume che siano forniti degli esempi
- Assume che l'agente voglia apprendere f – perché?)

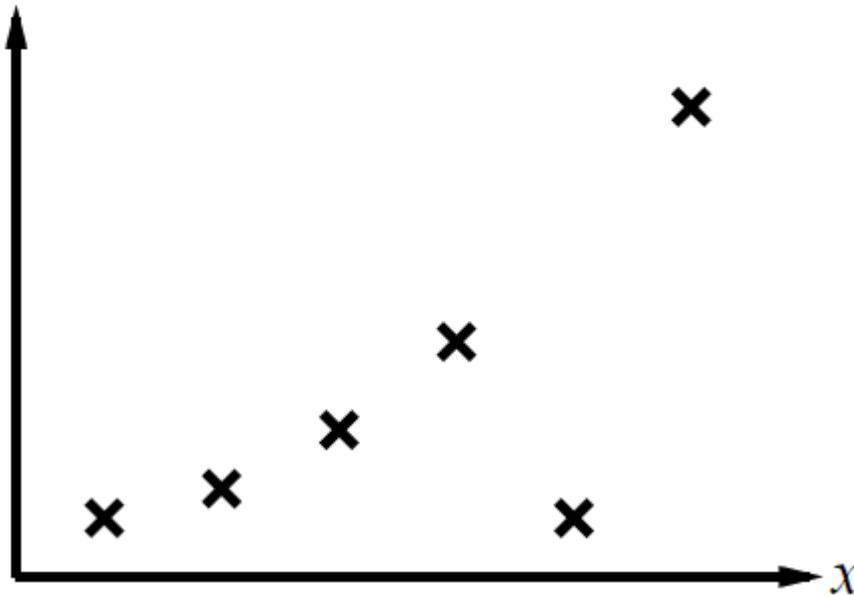
Metodo di apprendimento induttivo

Costruire/adattare h a concordare con f sulla base del **training set**

(h è **consistente** se concorda con f su tutti gli esempi)

Es., curve fitting: $f(x)$

Training Set



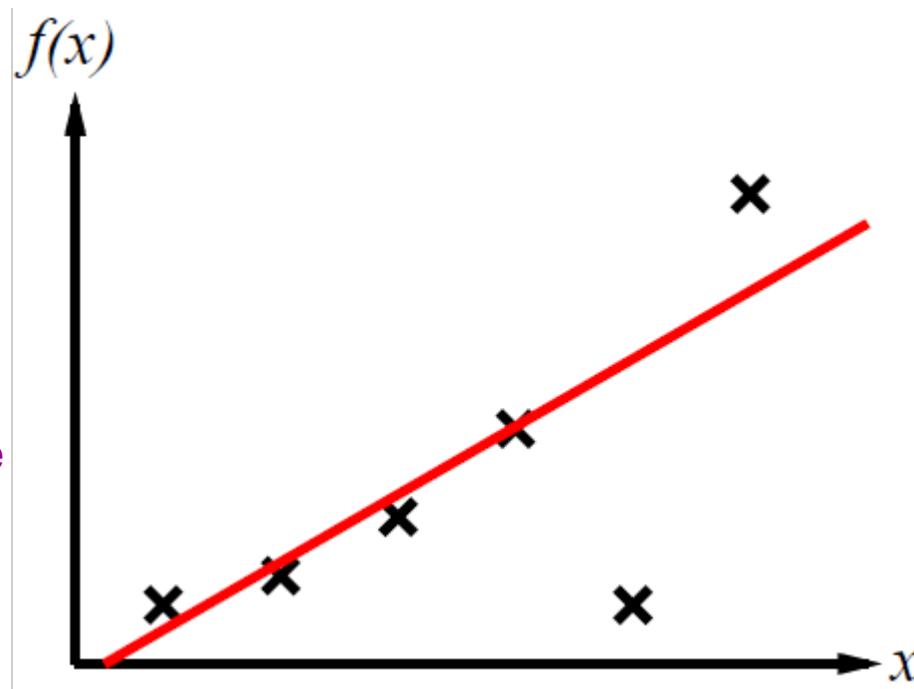
Metodo di apprendimento induttivo

Costruire/adattare h a concordare con f sulla base del training set

(h è consistente se concorda con f su tutti gli esempi)

Es., curve fitting:

Ipotesi lineare
parzialmente,
approssimativamente
consistente



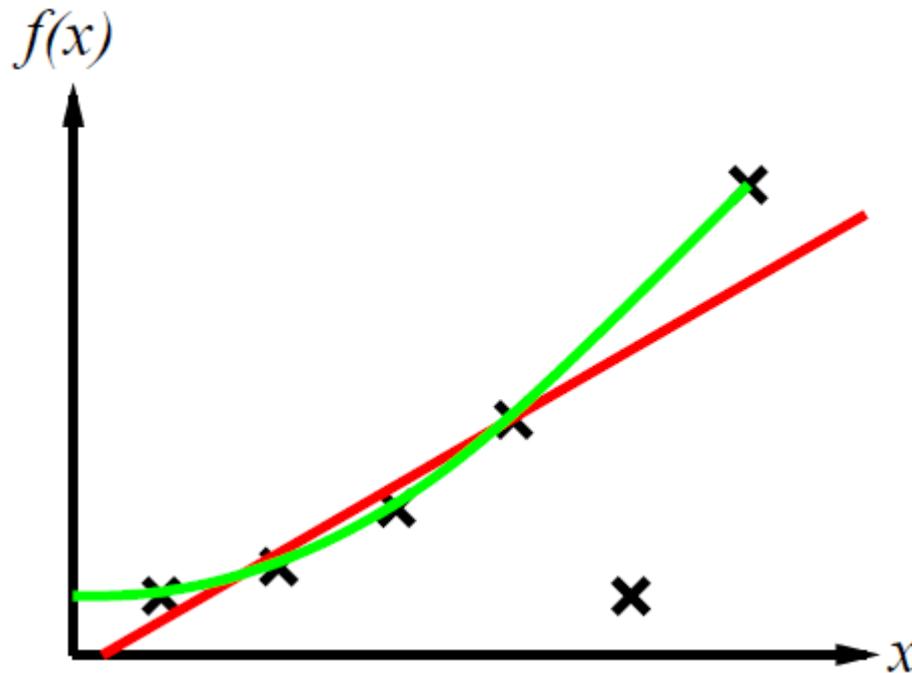
Metodo di apprendimento induttivo

Costruire/adattare h a concordare con f sulla base del training set

(h è consistente se concorda con f su tutti gli esempi)

Es., curve fitting:

Ipotesi quadratica
parzialmente
consistente



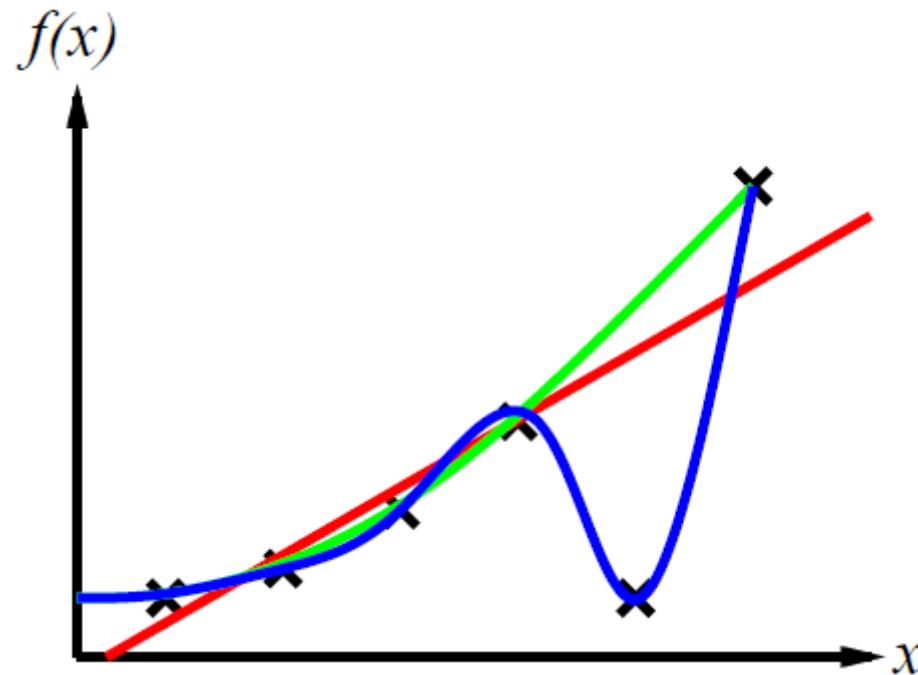
Metodo di apprendimento induttivo

Costruire/adattare h a concordare con f sulla base del training set

(h è consistente se concorda con f su tutti gli esempi)

Es., curve fitting:

Ipotesi di grado 4
consistente



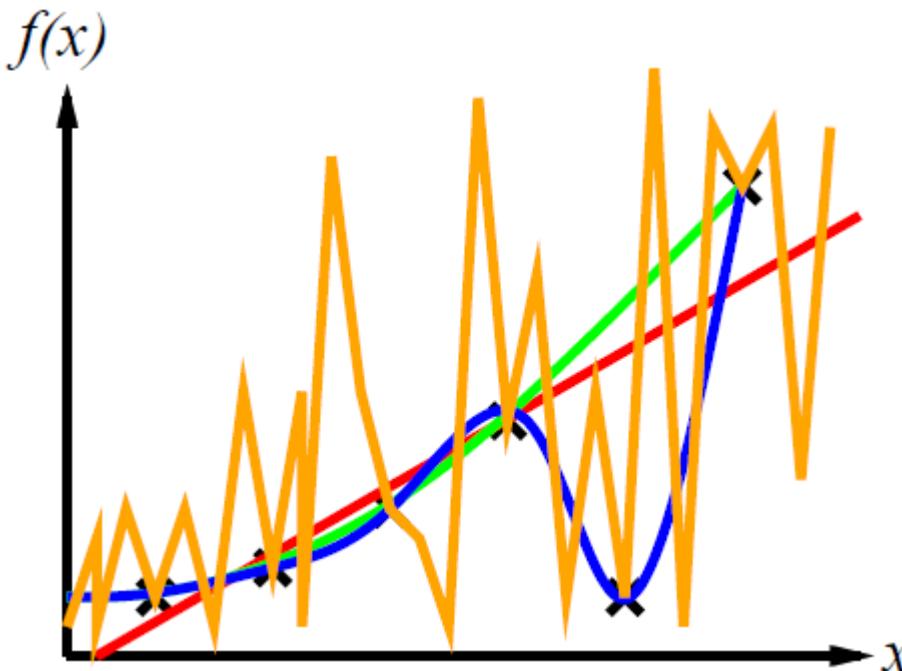
Metodo di apprendimento induttivo

Costruire/adattare h a concordare con f sulla base del training set

(h è consistente se concorda con f su tutti gli esempi)

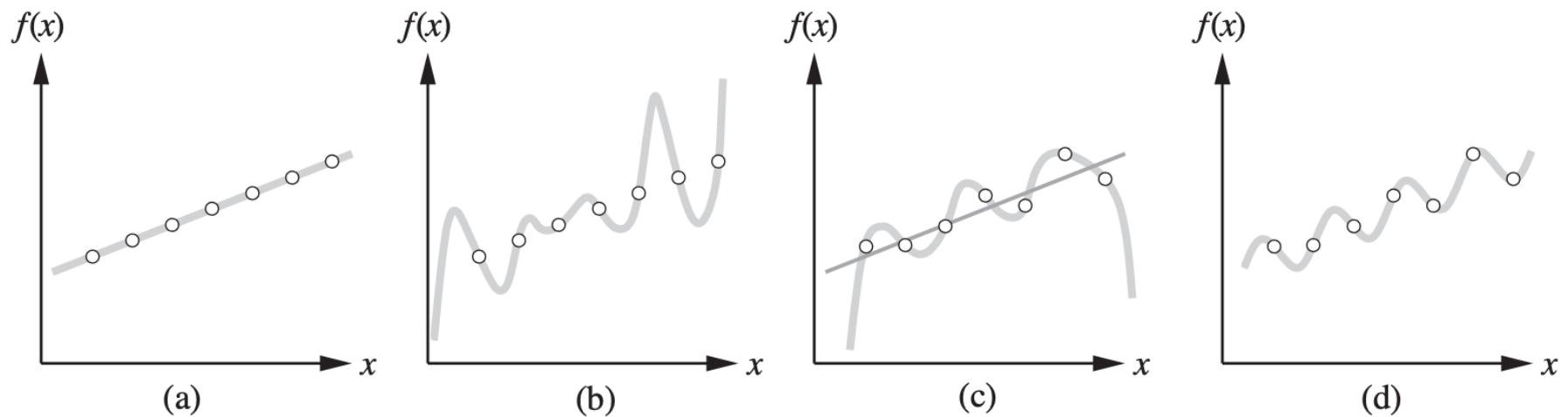
Es., curve fitting:

Ipotesi di alto grado
consistente



Quale scegliere? Si deve preferire l'ipotesi più semplice consistente con i dati
(Rasoio di Occam)

Realizzabilità



Problema di apprendimento è **realizzabile** se lo spazio delle ipotesi contiene la funzione reale.

Non sappiamo se un dato problema di apprendimento è realizzabile, a meno che non abbiamo una conoscenza preliminare.

Tradeoff: la complessità computazionale del problema di apprendimento induttivo è legata alla **dimensione** dello spazio delle ipotesi. Per esempio, la consistenza **non è nemmeno decidibile** per le macchine di Turing.

Training Set

Esempi descritti dai **valori degli attributi** (booleano, discreto, continuo, ecc.)

Es., situazioni dove converrebbe aspettare/non aspettare per un tavolo:

Example	Attributes										Target WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

La **classificazione** degli esempi è **positiva (T)** o **negativa (F)**

Trai

1. *Alternate*: whether there is a suitable alternative restaurant nearby.
2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
3. *Fri/Sat*: true on Fridays and Saturdays.
4. *Hungry*: whether we are hungry.
5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
6. *Price*: the restaurant's price range (\$, \$\$, \$\$\$).
7. *Raining*: whether it is raining outside.
8. *Reservation*: whether we made a reservation.
9. *Type*: the kind of restaurant (French, Italian, Thai, or burger).
10. *WaitEstimate*: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).

Esempi descritti dai **valori degli attributi**

Es., situazioni dove converrebbe andare:

Example	Attributes										Target WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0–10</i>	<i>T</i>
X_2	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30–60</i>	<i>F</i>
X_3	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0–10</i>	<i>T</i>
X_4	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10–30</i>	<i>T</i>
X_5	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>>60</i>	<i>F</i>
X_6	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0–10</i>	<i>T</i>
X_7	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0–10</i>	<i>F</i>
X_8	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0–10</i>	<i>T</i>
X_9	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>>60</i>	<i>F</i>
X_{10}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10–30</i>	<i>F</i>
X_{11}	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0–10</i>	<i>F</i>
X_{12}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30–60</i>	<i>T</i>

La **classificazione** degli esempi è **positiva (T)** o **negativa (F)**

Training Set

- ▶ Possiamo riorganizzare i campioni come segue:

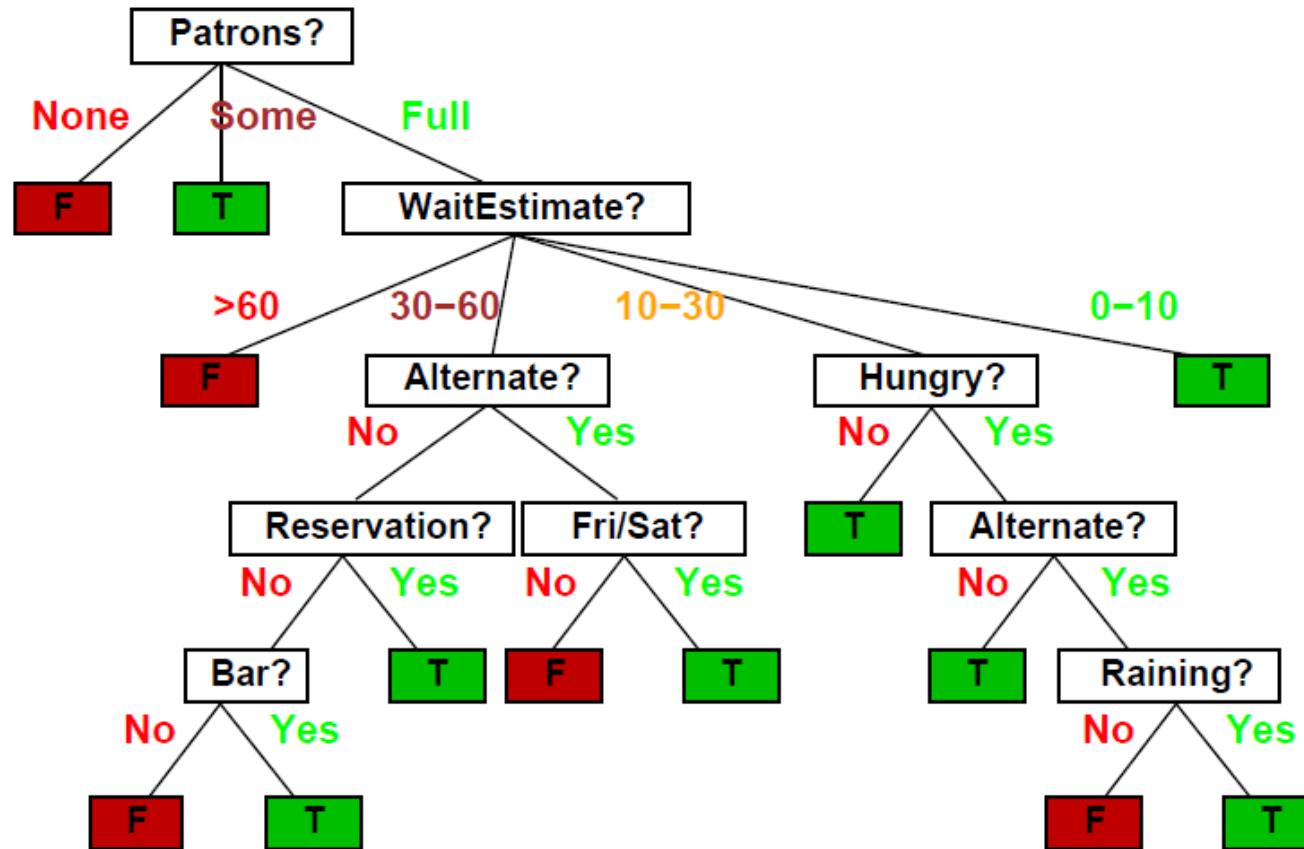
$$\mathbf{x}_1 = \left\{ \begin{array}{l} T \\ F \\ F \\ T \\ Some \\ $$$ \\ F \\ T \\ French \\ 0-10 \end{array} \right\}, \quad y_1 = T \quad \text{sample 1}$$
$$\mathbf{x}_2 = \left\{ \begin{array}{l} T \\ F \\ F \\ T \\ Full \\ \$ \\ F \\ F \\ Thai \\ 30-60 \end{array} \right\}, \quad y_2 = F \quad \dots \quad \text{sample 2}$$

- ▶ cioè, ogni campione è costituito da un vettore x_i a 10 dimensioni di valori discreti delle caratteristiche e un'etichetta di destinazione y_i che è booleana.

Alberi di decisione

Una possibile rappresentazione per le ipotesi

Es. in questo albero, «T» si riferisce alla decisione di aspettare:



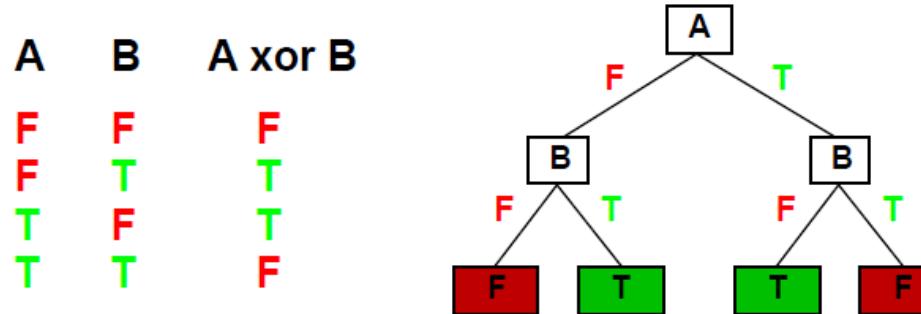
Espressività

Gli alberi di decisione possono esprimere qualsiasi funzione degli attributi in input. Un albero definisce un predicato obiettivo (ad es. WillWait)

$\forall s \text{ WillWait}(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \dots \vee P_n(s))$ con P_i path per arrivare alla foglia T

Esempio per le funzioni booleane:

riga nella tabella della verità → path per arrivare alla foglia



In sostanza, esiste un albero decisionale coerente per qualsiasi training set con un path verso la foglia per ogni esempio (a meno che f sia non deterministico in x), ma probabilmente non si generalizzerà a nuovi esempi

È preferibile ricercare alberi di decisione più **compatti**

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

= numero di funzioni booleane con n attributi

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

- = numero di funzioni booleane con n attributi
- = numero di tabelle di verità distinte con 2^n righe

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

= numero di funzioni booleane con n attributi

= numero di tabelle di verità distinte con 2^n righe = 2^{2^n}

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

= numero di funzioni booleane con n attributi

= numero di tabelle di verità distinte con 2^n righe = 2^{2^n} funzioni differenti

Es., con 6 attributi booleani, ci sono 18,446,744,073,709,551,616 alberi di decisione

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

= numero di funzioni booleane con n attributi

= numero di tabelle di verità distinte con 2^n righe = 2^{2^n}

Es., con 6 attributi booleani, ci sono 18,446,744,073,709,551,616 alberi di decisione

Quante ipotesi puramente congiuntive si possono formulare

(es., *Hungry* \wedge \neg *Rain*)?

Spazio delle ipotesi

Quanti alberi di decisione si possono creare con n attributi booleani?

= numero di funzioni booleane

= numero di tabelle di verità distinte con 2^n righe = 2^{2^n}

Es., con 6 attributi booleani, ci sono 18,446,744,073,709,551,616 alberi di decisione

Quante ipotesi puramente congiuntive si possono formulare

(es., $\text{Hungry} \wedge \neg \text{Rain}$)?

Ogni attributo può essere vero, falso, o ignorato

→ 3^n ipotesi congiuntive distinte

Uno spazio delle ipotesi più espressivo

- Aumenta la possibilità che la funzione obiettivo possa essere espressa
- Aumenta il numero di ipotesi consistenti con il training set
 - può portare a delle predizioni peggiori

Apprendimento degli alberi di decisione

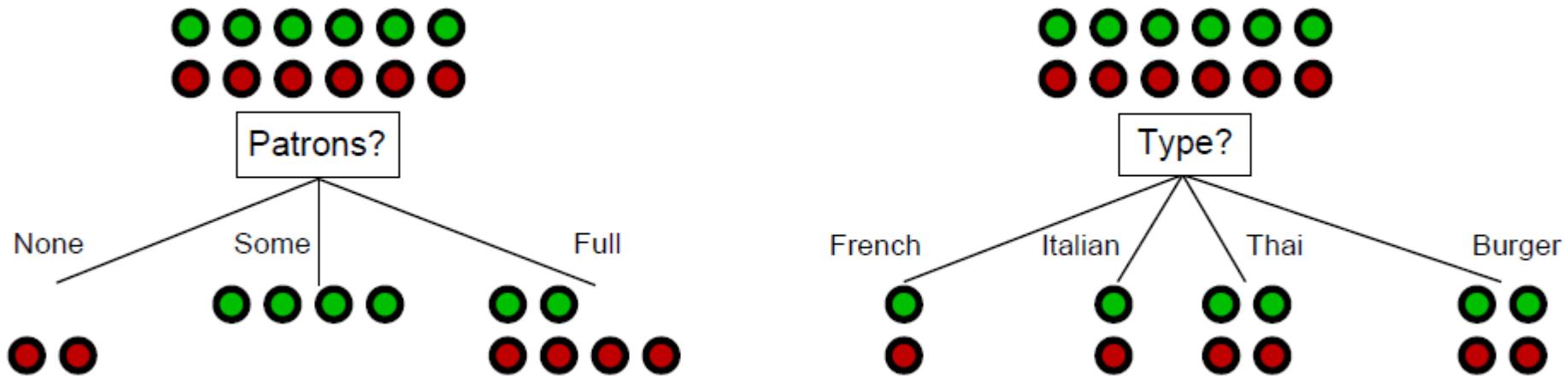
Soluzione banale per la creazione di alberi decisionali: costruisci un percorso fino ad una foglia per ogni campione.

- ▶ Memorizza semplicemente le osservazioni (non estrae nessun modello dai dati).
- ▶ Produce un'ipotesi consistente ma eccessivamente complessa (ricorda il rasoio di Occam).
- ▶ È improbabile che si generalizzi bene a nuove osservazioni.

Dovremmo mirare a costruire l'albero più piccolo che sia coerente con le osservazioni. Un modo per farlo è testare in modo ricorsivo prima gli attributi più importanti.

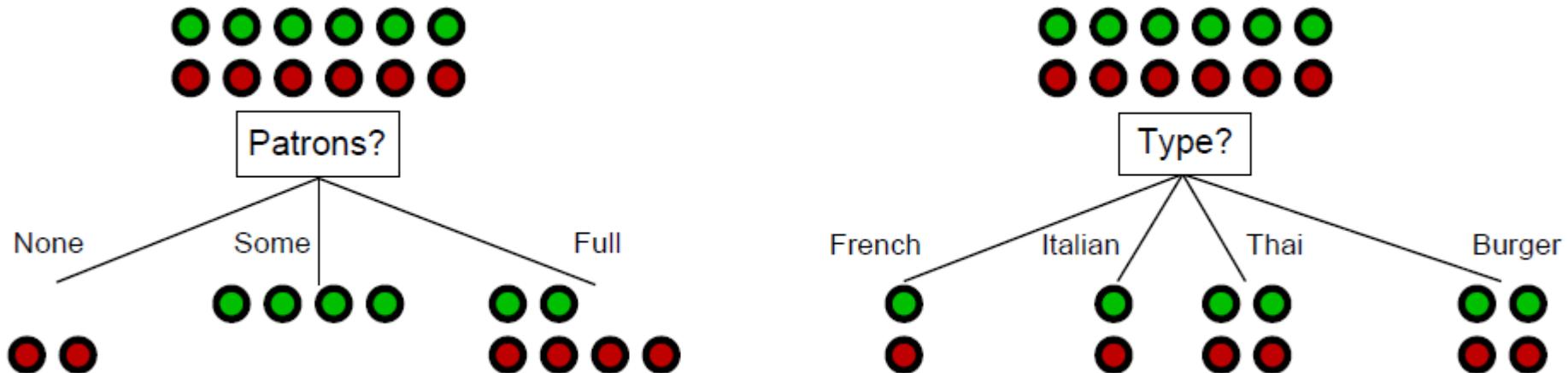
Scegliere un attributo

Ad esempio: è meglio testare prima *Patrons* oppure *Type*



Scegliere un attributo

Idea: un buon attributo divide gli esempi in sottoinsiemi che sono (idealmente) «tutti positivi» o «tutti negativi»



Patrons? è una scelta migliore, fornisce maggiori **informazioni** riguardo la classificazione

Apprendimento degli alberi di decisione

Obiettivo: trovare un albero piccolo e consistente con esempi di training

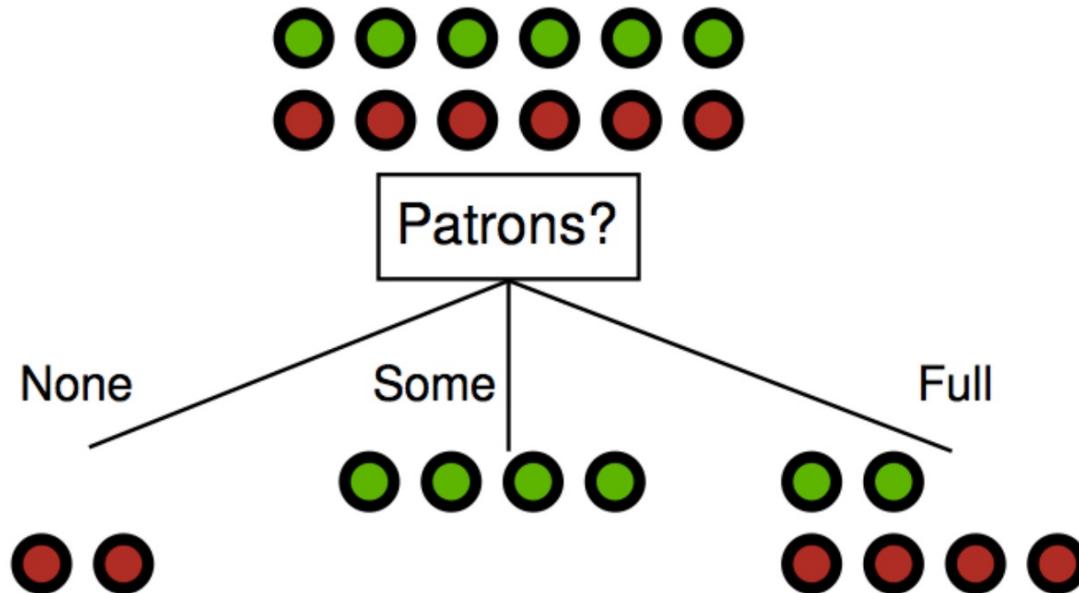
Idea: scegliere ricorsivamente l'attributo «più significativo» come radice del (sotto)albero

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value vi of best do
            examplesi ← {elements of examples with best = vi}
            subtree ← DTL(examplesi, attributes – best, MODE(examples))
            add a branch to tree with label vi and subtree subtree
    return tree
```

Algoritmo DTL (Decision Tree Learning)

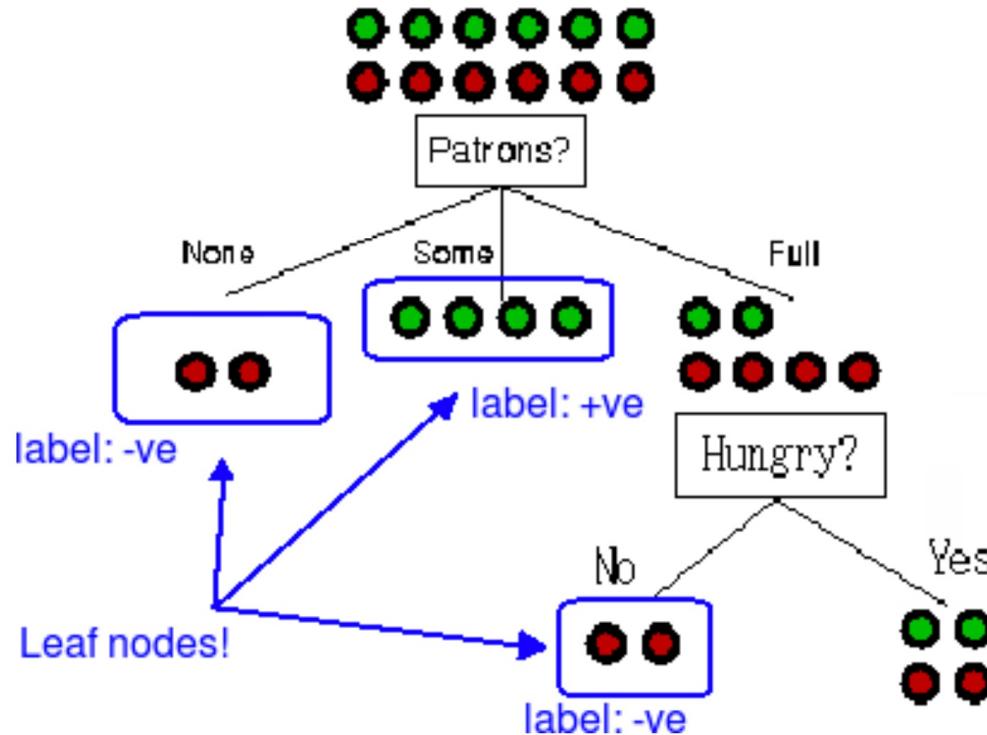
Quattro idee alla base dell'algoritmo:

1. Se sono presenti campioni positivi e negativi, scegliere l'attributo migliore per dividerli, ad esempio, prova **Patrons** nella radice.



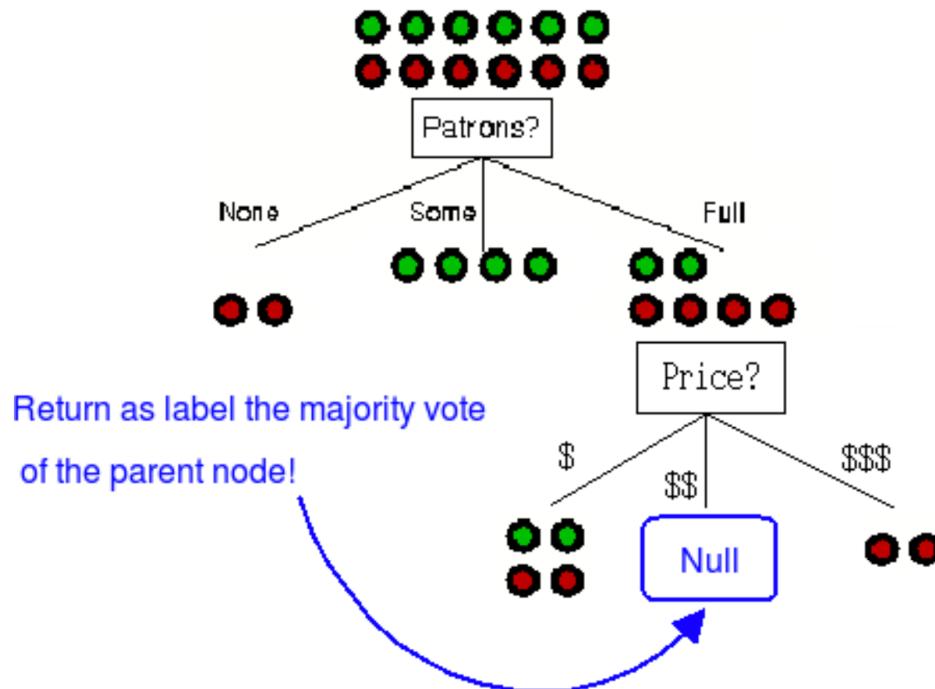
Algoritmo DTL

2. Se tutti i campioni rimanenti sono tutti positivi o tutti negativi, abbiamo raggiunto un nodo foglia. Assegna l'etichetta come positiva (o negativo), ad es.



Algoritmo DTL

3. Se non sono rimasti campioni, significa che non è stato osservato quel campione. Restituisce un valore predefinito calcolato da classificazione maggioritaria al genitore del nodo, ad es.



Algoritmo DTL

4. Se non ci sono attributi rimasti, ma sia campioni positivi che negativi, significa che questi campioni hanno esattamente lo stesso valore delle caratteristiche ma classificazioni diverse. Può succedere perché

- ▶ alcuni dei dati potrebbero essere errati o
- ▶ gli attributi non forniscono informazioni sufficienti per descrivere completamente la situazione (cioè ci mancano altri attributi utili), o
- ▶ il problema è veramente **non-deterministico**, cioè dati due campioni descrivendo esattamente le stesse condizioni, possiamo renderle diverse decisioni.

Soluzione: chiamarlo nodo foglia e assegnargli il voto di maggioranza come l'etichetta.

Informazione

- ▶ Abbiamo bisogno di una misura per giudicare un attributo come **buono** o **cattivo**.
- ▶ Un modo possibile è calcolare il contenuto dell'informazione ad un nodo del Decision Tree.
- ▶ Ad esempio per il nodo R abbiamo
$$I(R) = \sum_{i=1}^L -P(c_i) \log_2 P(c_i)$$
 dove $\{c_1, \dots, c_L\}$ sono le L classi presenti nel nodo, e $P(c_i)$ è la probabilità di ottenere la classe c_i al nodo.
- ▶ Ad esempio, al nodo radice del problema del *Ristorante* $c_1=True$ e $c_2=False$ e ci sono 6 campioni veri e 6 campioni falsi. Pertanto:

$$P(c_1) = \frac{\text{no. of samples } = c_1}{\text{total no. of samples}} = \frac{6}{6+6} = 0.5$$

$$P(c_2) = \frac{\text{no. of samples } = c_2}{\text{total no. of samples}} = \frac{6}{6+6} = 0.5$$

$$I(\text{Root}) = -0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 1 \text{ bit}$$

Informazione

- ▶ In generale, la quantità di informazioni sarà massima quando tutte le classi sono ugualmente probabili e sono minime quando il nodo è omogeneo (tutti i campioni hanno le stesse etichette)
- ▶ Scala: 1 bit = risposta ad un quesito booleano a priori $\langle 0.5, 0.5 \rangle$
- ▶ L'informazione in una risposta quando è a priori $\langle P_1, \dots, P_n \rangle$ è

$$I(\langle P_1, \dots, P_n \rangle) = \sum_{i=1}^n -P_i \log_2 P_i$$

(chiamata anche **entropia** a priori)

- ▶ Lancio moneta: $I(1/2, 1/2) = 1$ bit
- ▶ Lancio moneta truccata che da testa al 99%: $I(1/100, 99/100) = 0,08$ bit
- ▶ Lancio moneta che da sempre testa: 0 bit

Scegliere il miglior attributo

- ▶ Un attributo A dividerà i campioni ad un nodo in diversi sottoinsiemi (o nodi figlio) $E_{A = v_1}, \dots, E_{A = v_M}$, dove A ha M valori distinti $\{v_1, \dots, v_M\}$.
 - ▶ Generalmente ogni sottoinsieme $E_{A = v_i}$ avrà campioni con etichette diverse, quindi se andiamo lungo quel ramo avremo bisogno di ulteriori $I(E_{A = v_i})$ bit di informazione.
-
- ▶ **Esempio**
- Nel problema del ristorante, al ramo *Patrons = Full* del nodo radice, abbiamo $c1 = \text{True}$ con 2 campioni e $c2 = \text{False}$ con 4 campioni, quindi

$$I(E_{\text{Patrons}=\text{Full}}) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} = 0.9183 \text{ bits}$$

Scegliere il miglior attributo

- ▶ Indichiamo con $P(A = v_i)$ come la probabilità che un campione segua il ramo $A = v_i$.
- ▶ **Esempio**

Nel problema del ristorante, al nodo radice abbiamo

$$\begin{aligned} P(Patrons = Full) &= \frac{\text{No. of samples where } Patrons = Full}{\text{No. of samples at the root node}} \\ &= \frac{6}{12} = 0.5 \end{aligned}$$

Scegliere il miglior attributo

- ▶ Dopo aver testato l'attributo A, abbiamo bisogno di un **resto** di

$$Remainder(A) = \sum_{i=1}^M P(A = v_i) I(E_{A=v_i})$$

bit per classificare i campioni.

- ▶ Il **guadagno** di informazioni nel test dell'attributo A è la differenza tra il contenuto dell'informazione originale ed il contenuto delle nuove informazioni, cioè

$$\begin{aligned} Gain(A) &= I(R) - Remainder(A) \\ &= I(R) - \sum_{i=1}^M P(A = v_i) I(E_{A=v_i}) \end{aligned}$$

dove $\{E_{A=v_1}, \dots, E_{A=v_M}\}$ sono i nodi figlio di R dopo il test attributo A.

Scegliere il miglior attributo

- ▶ L'idea chiave alla base della funzione SCEGLI-ATTRIBUTO dell'algoritmo consiste nello scegliere l'attributo che dà il **massimo guadagno** (GAIN) di informazioni.

- ▶ Esempio:

Nel problema del ristorante, al nodo radice

$$\begin{aligned} Gain(Patrons) &= 1 - \frac{2}{12}(-\log_2 1) - \frac{4}{12}(-\log_2 1) - \frac{6}{12}\left(-\frac{2}{6}\log_2 \frac{2}{6} - \frac{4}{6}\log_2 \frac{4}{6}\right) \\ &\approx 0.5409 \text{ bits.} \end{aligned}$$

- ▶ Con calcoli simili,

$$Gain(Type)=0$$

confermando che *Patrons* è un attributo migliore di *Type*. Infatti alla radice *Patrons* fornisce il maggior guadagno di informazioni.

Riassumendo

Supponiamo di avere p esempi positivi ed n esempi negativi alla radice

→ $I(\langle p/(p+n), n/(p+n) \rangle)$ bit richiesti per classificare un nuovo esempio

Es., per 12 ristoranti d'esempio, $p = n = 6$, quindi abbiamo bisogno di 1 bit

Un attributo divide gli esempi E nei sottoinsiemi E_i , ognuno dei quali (si spera) richiedano meno informazione per completare la classificazione

Sia E_i caratterizzato da esempi positivi p_i e negativi n_i

→ $I(\langle p_i/(p_i+n_i), n_i/(p_i+n_i) \rangle)$ bit richiesti per classificare un nuovo esempio

→ il numero di bit atteso per classificare un esempio su tutti i branch è

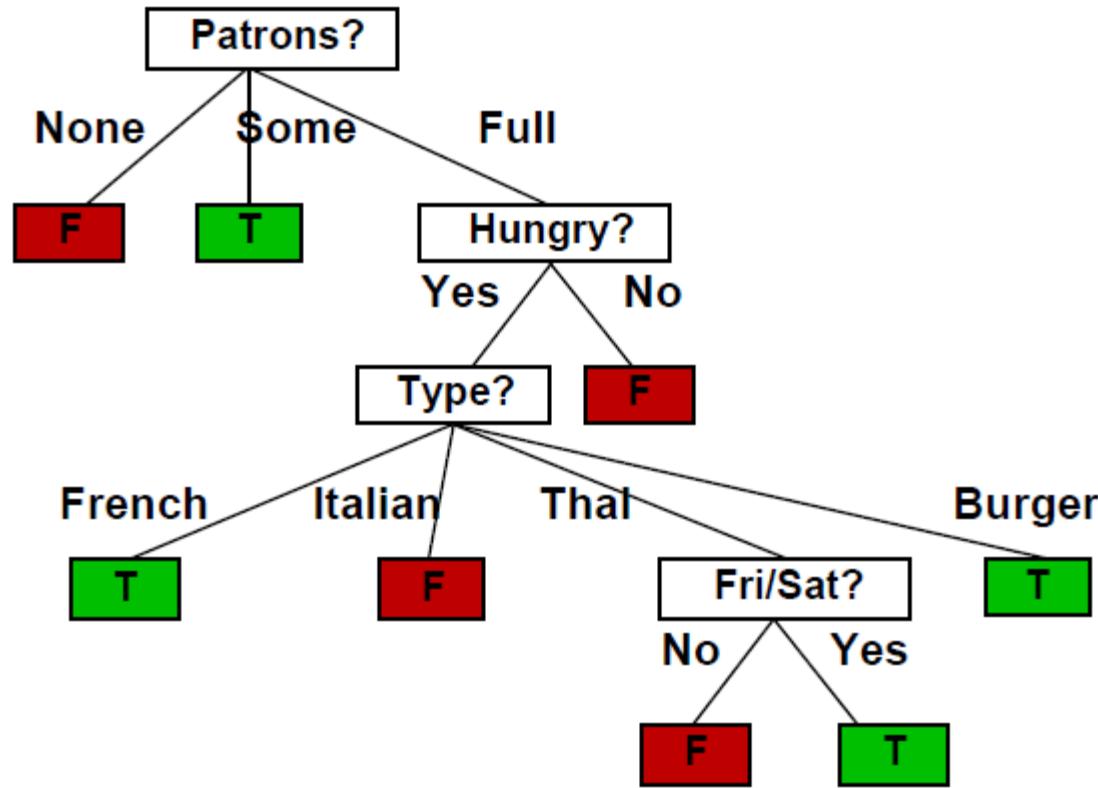
$$\sum_i \frac{p_i+n_i}{p+n} I(\langle p_i/(p_i+n_i), n_i/(p_i+n_i) \rangle)$$

Per *Patrons?*, è 0.459 bit, per *Type* è (ancora) di un 1 bit

→ scegliere l'attributo che minimizza la quantità d'informazione necessaria rimasta

Esempio

Albero di decisione appreso da 12 esempi:



Sostanzialmente più semplice dell'albero visto prima. Uno scarso quantitativo di dati non giustifica la formulazione di un'ipotesi più complessa

Impurità

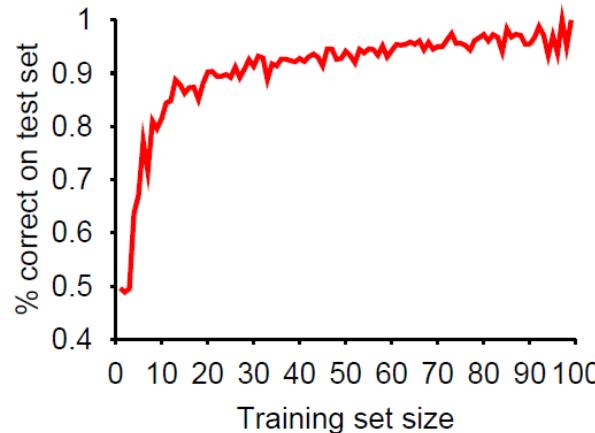
- ▶ Un punto di vista diverso è considerare *I(nodo)* come una misura di **impurità**. Più sono "misti" i campioni in un nodo (cioè proporzioni uguali di tutte le label di classe), maggiore è il valore di impurità. D'altra parte, un nodo omogeneo (cioè ha campioni di una sola classe) avrà impurità zero.
- ▶ Il valore Gain(A) può quindi essere visto come la **quantità di riduzione dell'impurità** se dividiamo secondo A.
- ▶ Ciò offre l'idea intuitiva che possiamo costruire alberi di decisione in modo ricorsivo cercando di ottenere nodi foglia che siano **più puri possibile**.

Misurazione delle prestazioni

Come facciamo a sapere che $h \approx f$? (**Problema d'induzione** di Hume)

- 1) Usare teoremi sulla teoria dell'apprendimento computazionale/statistico
- 2) Provare h su un nuovo insieme di esempi di test
(usando la stessa distribuzione sullo spazio dell'esempio come insieme di training)

Curva d'apprendimento = % corretti sull'insieme di test in funzione delle dimensioni delle training se stesso



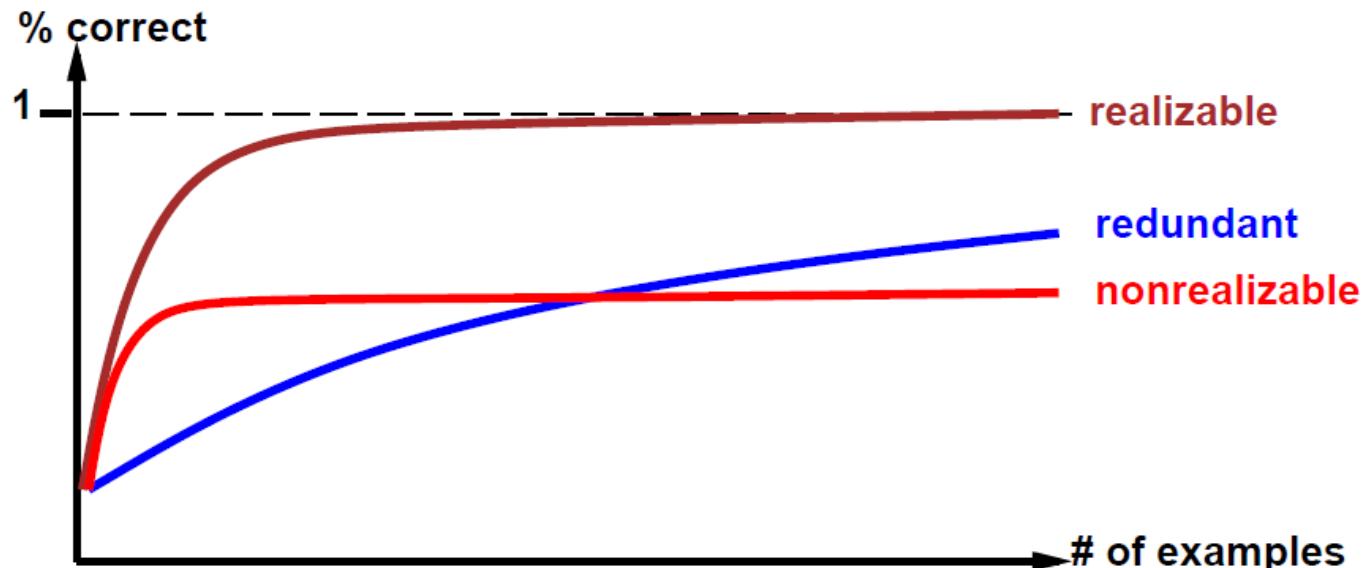
Overfitting: Rischio di utilizzare predici osservabili irrilevanti per generare un'ipotesi che sia d'accordo con tutti gli esempi nel training set

Tree pruning: Termina la ricorsione quando # errori/gain è piccolo

Misurazione delle prestazioni

La curva di apprendimento dipende da:

- realizzabilità (può esprimere la funzione target) vs non realizzabilità
 - la non realizzabilità può essere dovuta ad attributi mancanti o ad una classe d'ipotesi limitata (ad es. funzione lineare con soglia);
- espressività ridondante (ad esempio, molti attributi irrilevanti)



Generalizzazione e Overfitting

- ▶ **Osservazione:** a volte un'ipotesi appresa è più specifica di quanto giustificano gli esperimenti.
- ▶ Parliamo di **overfitting**, se un'ipotesi **h** descrive un errore casuale nel training set (limitato) piuttosto che nella relazione sottostante.
- ▶ L'**underfitting** si verifica quando **h** non è in grado di catturare l'andamento sottostante dei dati.
- ▶ Qualitativamente: l'**overfitting** aumenta con la dimensione dello spazio delle ipotesi e del numero di attributi, ma diminuisce con il numero di esempi.
- ▶ **Idea:** combattere l'overfitting "generalizzando" gli alberi decisionali calcolati da DTL.

Decision Tree Pruning

- ▶ **Idea:** combattere l'overfitting "generalizzando" gli alberi decisionali → sfoltendo i nodi "irrilevanti".
- ▶ Per la potatura dell'albero decisionale, ripetere quanto segue su un albero decisionale appreso:
 - ▶ Trova un nodo di test **terminale *n*** (ha solo foglie come figli)
 - ▶ se il test è irrilevante, cioè ha un basso **information gain**, eliminalo sostituendo ***n*** con un nodo foglia.
- ▶ Domanda: quanto dovrebbe essere grande il guadagno di informazioni per dividere (→ mantenere) un nodo?
- ▶ **Idea:** utilizzare un *test di significatività statistica*.
- ▶ Un risultato ha **rilevanza statistica**, se la probabilità che possano derivare dall'**ipotesi nulla** (cioè l'assunzione che non vi sia un pattern sottostante) è molto bassa (di solito 5%).



Valutare e scegliere
la miglior ipotesi

Indipendente e Identicamente Distribuito

- ▶ **Problema:** vogliamo apprendere un'ipotesi che si adatta meglio ai dati futuri.
- ▶ **Intuizione:** funziona solo se il training-set è “rappresentativo” per il processo sottostante.
- ▶ **Idea:** pensiamo agli esempi (visti e non visti) come una sequenza ed esprimiamo la “rappresentatività” come un'ipotesi di stazionarietà per la distribuzione di probabilità.
- ▶ **Metodo:** ogni esempio – prima di vederlo – è una variabile casuale E_j , il valore osservato $e_j = (x_j, y_j)$ campiona la sua distribuzione.
- ▶ Una sequenza di E_1, \dots, E_n delle variabili casuali è **indipendente e distribuita in modo identico** (IID breve), se è
 - ▶ **indipendente**, cioè $P(E_j | E_{(j-1)}, E_{(j-2)}, \dots) = P(E_j)$ e
 - ▶ **Identicamente distribuito**, cioè $P(E_i) = P(E_j)$ per tutti gli i e j .
- ▶ Una sequenza di lanci di dadi è IID.
- ▶ **Ipotesi di stazionarietà:** assumiamo che l'insieme E di esempi sia IID in futuro.

Tassi di errore e cross-validation

- ▶ **Goal:** vogliamo apprendere un'ipotesi che si adatta meglio ai dati futuri.
- ▶ Dato un problema di apprendimento induttivo $\langle H, f \rangle$, definiamo il **tasso di errore** di un'ipotesi $h \in H$ come la frazione di errori:

$$\frac{\#\{x \in \text{dom}(f) \mid h(x) \neq f(x)\}}{\#(\text{dom}(f))}$$

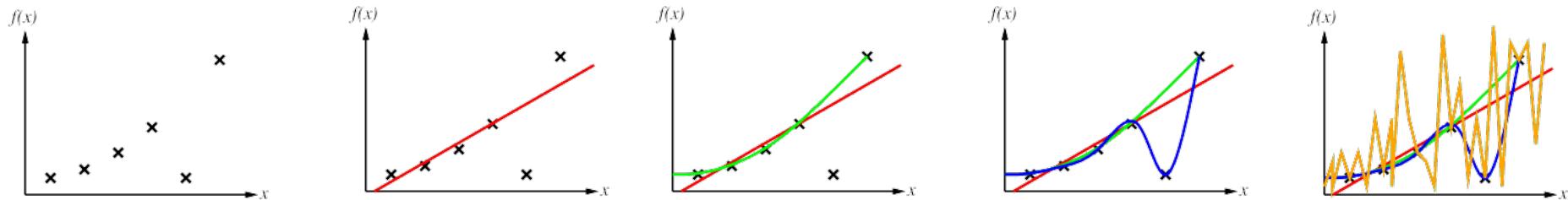
- ▶ Un basso **tasso di errore** sul training set non significa che un'ipotesi si generalizzi bene.
- ▶ Idea: non utilizzare le domande dei compiti a casa durante l'esame.
- ▶ La pratica di suddividere i dati disponibili per l'apprendimento in
 - ▶ un **insieme di addestramento** da cui l'algoritmo di apprendimento produce un'ipotesi h e
 - ▶ un **test set**, che viene utilizzato per valutare h
- ▶ è chiamato **holdout cross validation**. (**non è consentito sbirciare nel set di test**)

Tassi di errore e cross-validation

- ▶ Domanda: Qual è un buon rapporto tra la dimensione del training set e la dimensione del test set?
 - ▶ piccolo **training set** → scarse ipotesi.
 - ▶ piccolo **test set** → scarsa stima dell'accuratezza.
-
- ▶ Nella **k-fold cross validation**, eseguiamo k cicli di apprendimento, ciascuno con $1/k$ dei dati come test set e una media sui k tassi di errore.
 - ▶ Intuizione: ogni esempio ha un duplice compito: per il training e il test.
 - ▶ $k = 5$ e $k = 10$ sono popolari → buona accuratezza con k volte il tempo di calcolo.
 - ▶ Se $k = \#\text{dom}(f)$, la k-fold cross validation è chiamata **leave-one-out cross validation** (LOOCV).

Selezione del modello

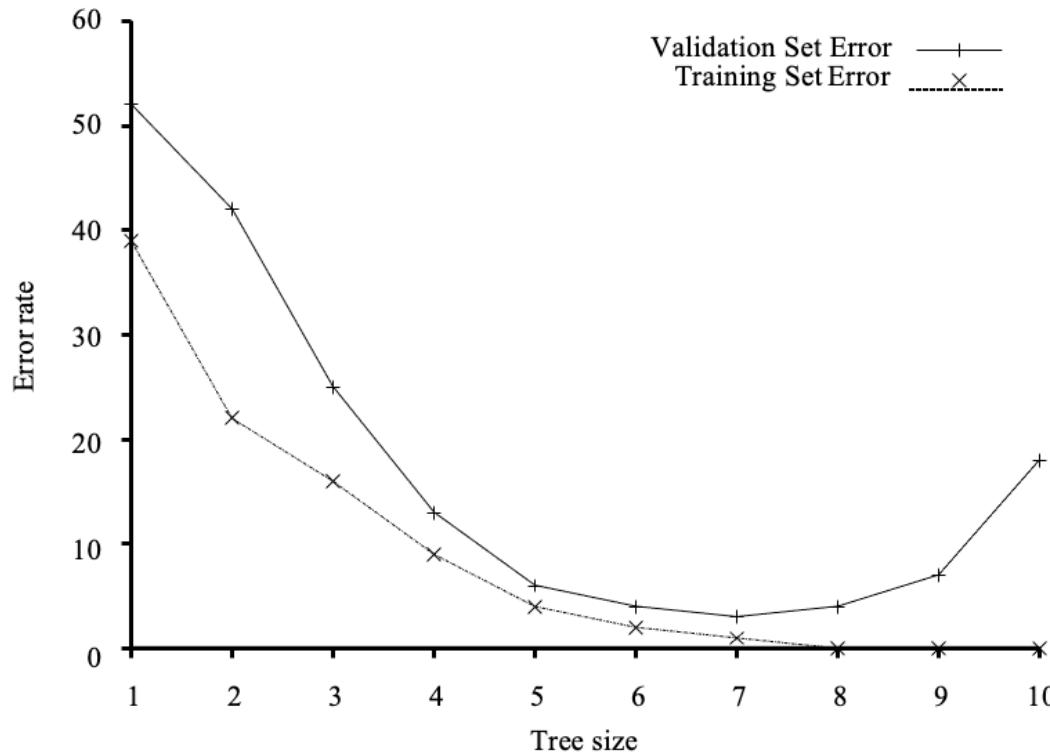
- ▶ Il problema della selezione del modello consiste nel determinare – a partire dai dati – un buon spazio di ipotesi.
- ▶ Qual è il miglior grado polinomiale per adattare i dati



- ▶ Possiamo risolvere il problema di "apprendere f dalle osservazioni" in un processo in due parti:
 - ▶ la **selezione del modello** determina uno spazio di ipotesi H ,
 - ▶ l'**ottimizzazione** risolve il problema di apprendimento induttivo indotto $\langle H, f \rangle$.
- ▶ Idea: risolvi le due parti insieme iterando su "dimensione".
- ▶ Serve una nozione di "dimensione", ad es. numero di nodi in un albero decisionale.
- ▶ Problema concreto: trovare la "dimensione" che meglio bilancia overfitting e underfitting per ottimizzare la precisione del test set.

Tassi di errore su Training/Validation Data

- (Una curva di errore per gli alberi decisionali dei ristoranti).



- Si interrompe quando il tasso di errore del test set converge, scegli l'albero ottimale per la curva di convalida. ([qui un albero con 7 nodi](#))

Da Error–rate a Loss Function

- ▶ Finora abbiamo ridotto al minimo i tassi di errore.
- ▶ **Classificazione dello spam.** È molto peggio classificare ham (mail legittime) come spam che viceversa. (**perdita del messaggio**)
- ▶ Richiamo della razionalità: i decisori dovrebbero **massimizzare l'utilità attesa** (MEU).
- ▶ Quindi: l'apprendimento automatico dovrebbe **massimizzare l'"utilità"**. (non solo ridurre al minimo i **tassi di errore**)
- ▶ L'apprendimento automatico si occupa tradizionalmente di utilità sotto forma di "funzioni di loss".
- ▶ La **loss function** L è definita impostando $L(x, y, \hat{y})$ come la quantità di utilità persa dalla predizione $h(x) = \hat{y}$ invece di $f(x) = y$. Se L è indipendente da x , usiamo spesso $L(y, \hat{y})$.
- ▶ Esempio: $L(\text{spam}, \text{ham}) = 1$, mentre $L(\text{ham}, \text{spam}) = 10$.

Generalizzazione della loss

- ▶ Nota: $L(y, y) = 0$. (nessuna perdita se hai esattamente ragione)
- ▶ Funzioni di loss popolari
 - ▶ absolute value loss $L_1(y, \hat{y}) := |(y - \hat{y})|$
 - ▶ squared error loss $L_2(y, \hat{y}) := (y - \hat{y})^2$
 - ▶ 0/1 loss $L_{0/1}(y, \hat{y}) := 0, \text{ if } y = \hat{y}, \text{ else } 1$
- ▶ Idea: massimizza l'utilità attesa scegliendo l'ipotesi h che riduce al minimo la loss attesa su tutte le coppie $(x, y) \in f$.
- ▶ Sia \mathcal{E} l'insieme di tutti i possibili esempi e $\mathbf{P}(X, Y)$ la distribuzione di probabilità a priori sulle sue componenti, allora la generalization loss per un'ipotesi h rispetto a una loss function L è

$$\text{GenLoss}_L(h) := \sum_{(x,y) \in \mathcal{E}} L(y, h(x)) \cdot \mathbf{P}(x, y)$$

- ▶ e l'ipotesi migliore $h^* := \operatorname{argmin}_{h \in \mathcal{H}} \text{GenLoss}_L(h)$.

Loss Empirica

- ▶ Problema: $P(X, Y)$ è sconosciuto \rightarrow il modello di apprendimento può solo stimare la perdita di generalizzazione:
- ▶ Sia L una funzione di perdita ed E un insieme di esempi con $\#(E) = N$, allora chiamiamo

$$\text{EmpLoss}_{L,E}(h) := \frac{1}{N} \left(\sum_{(x,y) \in E} L(y, h(x)) \right)$$

- ▶ la loss empirica e $\hat{h}^* := \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{EmpLoss}_{L,E}(h)$ la migliore ipotesi stimata.
- ▶ Ci sono quattro ragioni per cui \hat{h}^* può differire da f :
 - ▶ **Realizzabilità**: allora dobbiamo accontentarci di un'approssimazione \hat{h}^* di f .
 - ▶ **Varianza**: diversi sottoinsiemi di f danno diverse \hat{h}^* \rightarrow più esempi.
 - ▶ **Rumore**: se f non è deterministico, allora non possiamo aspettarci risultati perfetti.
 - ▶ **Complessità computazionale**: se H è troppo grande per essere esplorato sistematicamente, usiamo un sottoinsieme e otteniamo un'approssimazione.

Regolarizzazione

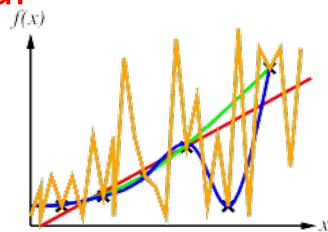
- ▶ **Idea:** utilizzare direttamente la **loss empirica** per risolvere la **selezione del modello**. (**trovare una buona H**)
- ▶ Minimizzare la somma pesata della **loss empirica** e della complessità dell'ipotesi (**per evitare l'overfitting**).
- ▶ Sia $\lambda \in \mathbb{R}$, $h \in H$ ed **E** un insieme di esempi, quindi chiamiamo

$$\text{Cost}_{L,E}(h) := \text{EmpLoss}_{L,E}(h) + \lambda \text{Complexity}(h)$$

- ▶ il **costo totale** di h su **E**.
- ▶ Il processo per trovare un'ipotesi di minimizzazione dei costi totali

$$\hat{h}^* := \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{Cost}_{L,E}(h)$$

- ▶ si chiama **regolarizzazione**; La **complessità** è chiamata **funzione di regolarizzazione o complessità di ipotesi**.
- ▶ (Regolarizzazione per Polinomi).
- ▶ Una buona **funzione di regolarizzazione** per i polinomi è
- ▶ la somma dei quadrati degli esponenti. → stai lontano dalle curve tortuose!



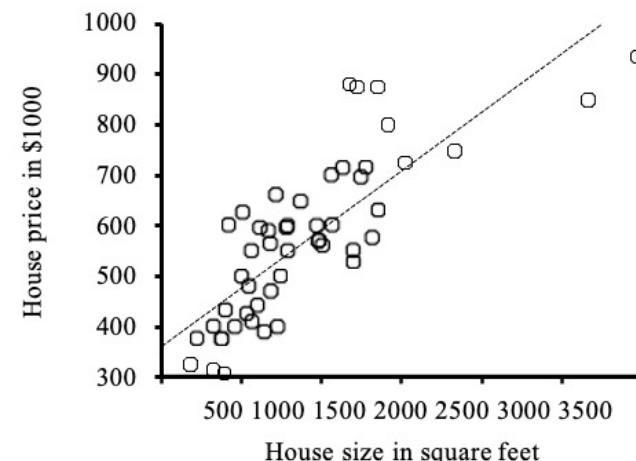
Regressione e classificazione con modelli lineari

Regressione lineare e classificazione

- ▶ Passiamo ad un altro **spazio di ipotesi**: le funzioni lineari su input a valori continui.
- ▶ Chiamiamo un problema di apprendimento induttivo $\langle H, f \rangle$ un problema di **classificazione** se il *co-dominio*(f) è discreto, e un problema di **regressione** se *co-dominio*(f) è continuo, cioè non discreto – di solito a valori reali.

Regressione lineare univariata

- ▶ Una funzione **univariata** è una funzione con un solo argomento.
 - ▶ Ricordiamo: un mapping tra spazi vettoriali è detto lineare se conserva l'addizione vettoriale e la moltiplicazione scalare.
- ▶ Una funzione **univariata**, lineare $f: \mathbb{R} \rightarrow \mathbb{R}$ è della forma
$$f(x) = w_1 x + w_0 \text{ per qualche } w_i \in \mathbb{R}.$$
- ▶ Dato un vettore $w := (w_0, w_1)$, definiamo $h_w(x) := w_1 x + w_0$
- ▶ Dato un insieme di **esempi** $E \subseteq \mathbb{R} \times \mathbb{R}$, il compito di trovare l' h_w che meglio si adatta a E è chiamato **regressione lineare**.
- ▶ Esempio: prezzi delle case rispetto ai feets² nelle case vendute a Berkeley.
- ▶ Ipotesi di funzione lineare che minimizza la perdita (quadrato dell'errore $y = 0,232x + 246$).



Regressione lineare univariata per minimizzazione della loss

- Idea: ridurre al minimo la loss come quadrato dell'errore L_2 su $\{(x_i, y_i) | i \leq N\}$ (utilizzato già da Gauss)

$$\text{Loss}(\mathbf{h}_w) = \sum_{j=1}^N L_2(y_j, \mathbf{h}_w(x_j)) = \sum_{j=1}^N (y_j - \mathbf{h}_w(x_j))^2 = \sum_{j=1}^N (y_j - w_1 x_j + w_0)^2$$

- Compito: trovare $w^* := \underset{w}{\operatorname{argmin}} \text{Loss}(h_w)$
- Ricordiamo: $\sum_{j=1}^N (y_j - w_1 x_j + w_0)^2$ è minimizzato, quando le derivate parziali rispetto ai w_i sono zero, cioè quando

$$\frac{\partial}{\partial w_0} \left(\sum_{j=1}^N (y_j - w_1 x_j + w_0)^2 \right) = 0 \quad \text{and} \quad \frac{\partial}{\partial w_1} \left(\sum_{j=1}^N (y_j - w_1 x_j + w_0)^2 \right) = 0$$

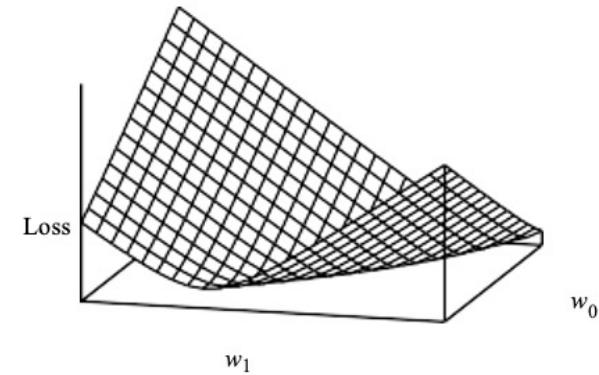
- Queste equazioni hanno una soluzione unica:

$$w_1 = \frac{N(\sum_j x_j y_j) - (\sum_j x_j)(\sum_j y_j)}{N(\sum_j x_j^2) - (\sum_j x_j)^2}$$

$$w_0 = \frac{(\sum_j y_j) - w_1 (\sum_j x_j)}{N}$$

Regressione lineare e classificazione

- ▶ Nota: molte forme di apprendimento implicano la regolazione dei pesi per ridurre al minimo le perdite.
- ▶ Lo **spazio dei pesi** è lo spazio di tutte le possibili combinazioni di pesi. La minimizzazione della perdita in uno spazio di peso è chiamata **adattamento del peso**.
- ▶ Lo **spazio dei pesi** della regressione lineare univariata è $R^2 \rightarrow$ possiamo tracciare graficamente la funzione di perdita su R^2
 - ▶ Nota: è convesso.
 - ▶ La **squared error loss** è **convessa** per qualsiasi problema di regressione lineare \rightarrow non ci sono **minimi locali**.



Discesa del gradiente

- ▶ Se non disponiamo di soluzioni in forma chiusa per ridurre al minimo la loss, dobbiamo cercare.
- ▶ Idea: utilizzare i metodi di ricerca locale (**hill climbing**).
- ▶ L'algoritmo di discesa del gradiente per trovare un minimo di una funzione continua f è l'**hill climbing** nella direzione della discesa più ripida, che può essere calcolata dalle derivate parziali di f .

```
function gradient-descent( $f, \mathbf{w}, \alpha$ ) returns a local minimum of  $f$ 
    inputs: a differentiable function  $f$  and initial weights  $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1)$ .
    loop until  $\mathbf{w}$  converges do
        for each  $\mathbf{w}_i$  do
             $\mathbf{w}_i \leftarrow \mathbf{w}_i - \alpha \frac{\partial}{\partial \mathbf{w}_i} (f(\mathbf{w}))$ 
        end for
    end loop
```

- ▶ Il parametro α è chiamato **learning rate**. Può essere una costante fissa o può decadere man mano che l'apprendimento procede.

Discesa del gradiente per la loss

- ▶ Proviamo a calcolare la **discesa del gradiente** per la **Loss**.
- ▶ Calcolare le derivate parziali per un **esempio** (x, y) :

$$\frac{\partial \text{Loss}(\mathbf{w})}{\partial w_i} = \frac{\partial (y - h_{\mathbf{w}}(x))^2}{\partial w_i} = 2(y - h_{\mathbf{w}}(x)) \frac{\partial (y - w_1 x + w)}{\partial w_i}$$

- ▶ e così abbiamo:

$$\frac{\partial \text{Loss}(\mathbf{w})}{\partial w_0} = -2(y - h_{\mathbf{w}}(x)) \quad \frac{\partial \text{Loss}(\mathbf{w})}{\partial w_1} = -2(y - h_{\mathbf{w}}(x))x$$

- ▶ Collegandolo negli aggiornamenti della discesa del gradiente:

$$w_0 \leftarrow w_0 - \alpha - 2(y - h_{\mathbf{w}}(x)) \quad w_1 \leftarrow w_1 - \alpha - 2(y - h_{\mathbf{w}}(x))x$$

Discesa del gradiente per la loss

- ▶ Analogamente per n esempi di addestramento (x_j, y_j) :

$$\mathbf{w}_0 \leftarrow \mathbf{w}_0 - \alpha \left(\sum_j -2(y_j - h_{\mathbf{w}}(x_j)) \right) \quad \mathbf{w}_1 \leftarrow \mathbf{w}_1 - \alpha \left(\sum_j -2(y_j - h_{\mathbf{w}}(x_n))x_n \right)$$

- ▶ Questi aggiornamenti costituiscono la **regola di apprendimento** della discesa del gradiente batch per la regressione lineare univariata.
- ▶ La convergenza all'unico minimo di perdita globale è garantita (a patto che scegliamo α sufficientemente piccolo), ma potrebbe essere molto lenta.

Regressione lineare multivariata

- ▶ Una **funzione multivariata** o n-aria è una funzione con uno o più argomenti.
 - ▶ Possiamo usarla per la **regressione lineare multivariata**.
- ▶ **Idea:** ogni esempio \vec{x}_j è un vettore di n elementi e lo **spazio delle ipotesi** è l' insieme di funzioni

$$h_{sw}(\vec{x}_j) = w_0 + w_1 x_{j,1} + \dots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

- ▶ **Trucco:** introduciamo $x_{j,0} := 1$ e usiamo la notazione matriciale:

$$h_{sw}(\vec{x}_j) = \vec{w} \cdot \vec{x}_j = \vec{w}^t \vec{x}_j = \sum_i w_i x_{j,i}$$

- ▶ Il miglior vettore di pesi, w^* , **minimizza** la perdita di errore quadratico sugli esempi: $w^* := \underset{w}{\operatorname{argmin}} (\sum_j L_2(y_j)(w \cdot \vec{x}_j))$
- ▶ La **discesa del gradiente** raggiungerà il minimo (unico) della funzione di perdita; l'equazione di aggiornamento per ogni peso w_i

Regressione lineare multivariata

- ▶ Possiamo anche risolvere analiticamente il w^* che **minimizza** la perdita.
- ▶ Sia \vec{y} il vettore di output per gli **esempi** di addestramento e X sia la **matrice di dati**, ovvero la matrice di input con un esempio n-dimensionale per riga.
- ▶ Allora la soluzione

$$w^* = (X^t X)^{-1} X^t \vec{y}$$

minimizza l'errore quadratico.

Regressione lineare multivariata (regolarizzazione)

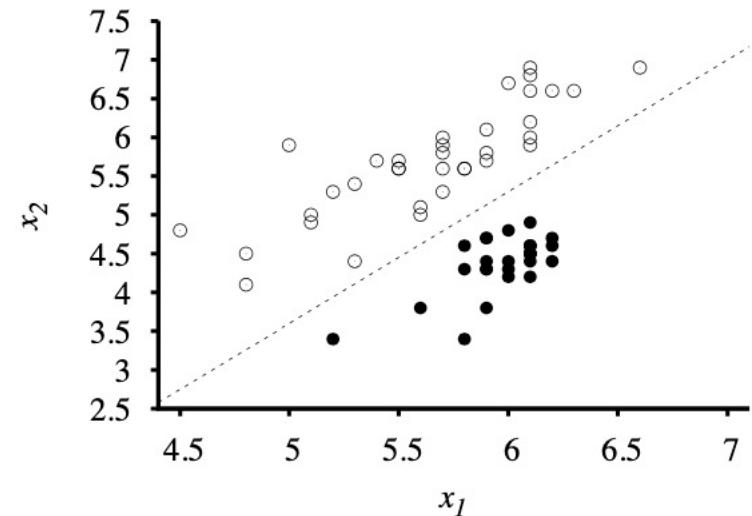
- ▶ **Nota:** la regressione lineare univariata non soffre di **overfit**, ma nel caso multivariato potrebbero esserci "dimensioni ridondanti" che si traducono in un **overfitting**.
- ▶ **Idea:** utilizzare la regolarizzazione con una funzione di complessità basata sui pesi.

$$\text{Complexity}(\mathbf{h}_w) = L_q(\mathbf{w}) = \sum_i |\mathbf{w}_i|^q$$

- ▶ **Avvertenza:** non confonderlo con le **loss function** L_1 e L_2 .
- ▶ **Problema:** quale q dovrebbe essere scelto?
(L_1 e L_2 minimizzano la somma dei valori assoluti / dei quadrati)
- ▶ **Risposta:** dipende dall'applicazione.
- ▶ **Nota:** la regolarizzazione L_1 tende a produrre un **modello sparso**, ovvero imposta molti pesi a 0, dichiarando di fatto gli attributi irrilevanti. Le ipotesi che scartano gli attributi possono essere più facili da comprendere per un essere umano e potrebbero avere meno probabilità di overfit.

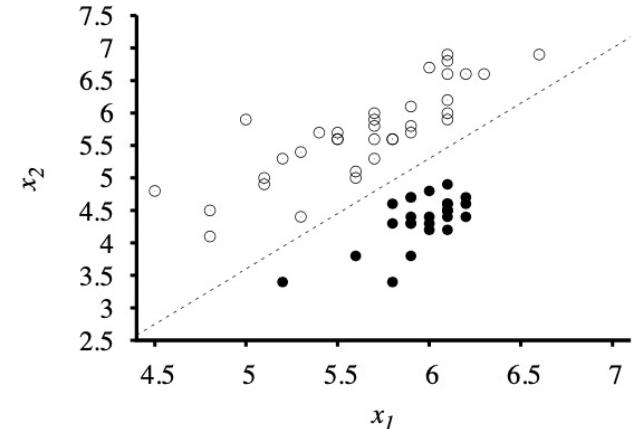
Classificatori lineari con soglia rigida

- ▶ **Idea:** il risultato della regressione lineare può essere utilizzato per la **classificazione**.
- ▶ Esempio (**Verifica del divieto di test nucleari**).
- ▶ Grafici dei parametri dei dati sismici: magnitudo dell'onda di volume x_1 vs. magnitudo dell'onda superficiale x_2 .
- ▶ Bianco: terremoti, nero: esplosioni sotterranee
- ▶ **Inoltre:** h_{w^*} come confine di decisione
 $x_2 = 17x_1 - 4,9.$



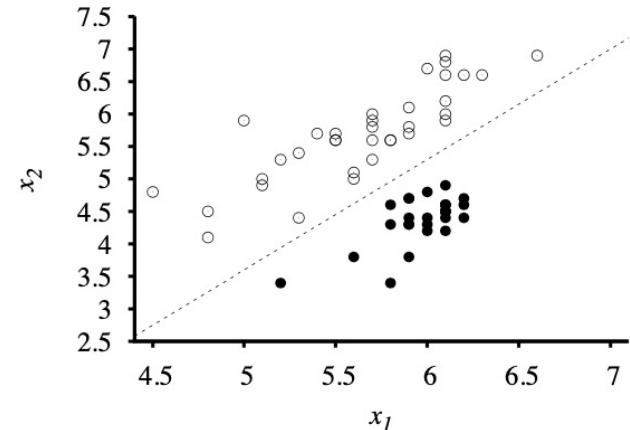
Classificatori lineari con soglia rigida

- ▶ **Idea:** il risultato della regressione lineare può essere utilizzato per la **classificazione**.
- ▶ Esempio (**Verifica del divieto di test nucleari**).
- ▶ Grafici dei parametri dei dati sismici:
 - ▶ magnitudo dell'onda corporea x_1 vs.
 - ▶ magnitudo dell'onda superficiale x_2 .
- ▶ Bianco: terremoti, nero: esplosioni sotterranee
- ▶ Inoltre: h_{w^*} come confine di decisione $x_2 = 17x_1 - 4,9$.
- ▶ Un **confine di decisione** è una linea (o una superficie, in dimensioni superiori) che separa due classi di punti. Un confine di decisione lineare è chiamato **separatore lineare** e i dati che ne ammettono uno sono chiamati **linearmente separabili**.
- ▶ Per l'esempio, il separatore lineare è definito da $-4.9 + 1.7x_1 - x_2 = 0$, le esplosioni sono caratterizzate da $-4.9 + 1.7x_1 - x_2 > 0$, terremoti da $-4.9 + 1.7x_1 - x_2 < 0$.



Classificatori lineari con soglia rigida

- ▶ **Idea:** il risultato della regressione lineare può essere utilizzato per la **classificazione**.
- ▶ Esempio (**Verifica del divieto di test nucleari**).
- ▶ Grafici dei parametri dei dati sismici:
 - ▶ magnitudo dell'onda corporea x_1 vs.
 - ▶ magnitudo dell'onda superficiale x_2 .
- ▶ Bianco: terremoti, nero: esplosioni sotterranee
- ▶ Inoltre: h_{w^*} come confine di decisione $x_2 = 17x_1 - 4,9$.
- ▶ Un **confine di decisione** è una linea (o una superficie) che separa due classi di punti. Un confine di decisione lineare è chiamato **separatore lineare** e i dati che ne ammettono uno sono chiamati **linearmente separabili**.
- ▶ Per l'esempio, il separatore lineare è definito da $-4.9 + 1.7x_1 - x_2 = 0$, le esplosioni sono caratterizzate da $-4.9 + 1.7x_1 - x_2 > 0$, terremoti da $-4.9 + 1.7x_1 - x_2 < 0$.
- ▶ **Trucco:** se introduciamo la coordinata fittizia $x_0 = 1$, allora possiamo scrivere l'**ipotesi di classificazione** come $h_w(\mathbf{x}) = 1$ se $\mathbf{w} \cdot \mathbf{x} > 0$ e 0 altrimenti.



Classificatori lineari con soglia rigida (Regola del percepitrone)

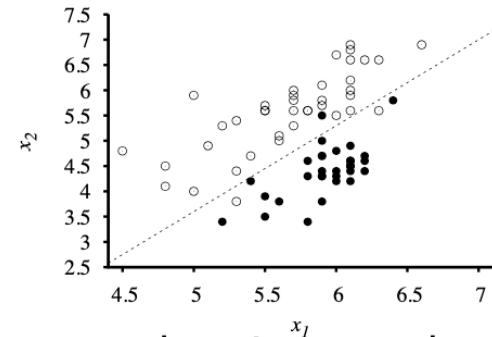
- ▶ Quindi $h_w(x) = 1$ se $w \cdot x > 0$ e 0 altrimenti è ben definito, come scegliere w ?
- ▶ Pensa a $h_w(x) = T(w \cdot x)$, dove $T(z) = 1$, se $z > 0$ e $T(z) = 0$ altrimenti. Chiamiamo T una **funzione di soglia**.
- ▶ Problema: T non è derivabile e $\frac{\partial T}{\partial z} = 0$ è definito →
 - ▶ Nessuna soluzione in forma chiusa impostando $\frac{\partial T}{\partial z} = 0$
 - ▶ Nemmeno i metodi di discesa del gradiente nello spazio dei pesi funzionano.
- ▶ Possiamo apprendere i pesi ripetendo la seguente regola:
- ▶ Dato un **esempio** (x, y) , la **regola di apprendimento del perceptron** è

$$w_i \leftarrow w_i + \alpha \cdot (y - h_w(x)) \cdot x_i$$

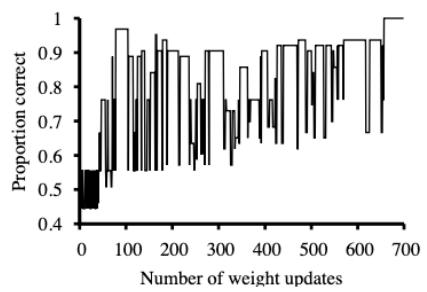
- ▶ poiché stiamo considerando la classificazione 0/1, ci sono tre possibilità:
 1. Se $y = h_w(x)$, allora w_i rimane invariato.
 2. Se $y = 1$ e $h_w(x) = 0$, allora w_i è in/diminuito se x_i è positivo/negativo. (vogliamo rendere $w \cdot x$ più grande in modo che $T(w \cdot x) = 1$)
 3. Se $y = 0$ e $h_w(x) = 1$, allora w_i è decrescente/aumentato se x_i è positivo/negativo. (vogliamo per ridurre $w \cdot x$ in modo che $T(w \cdot x) = 0$)

Classificatori lineari con soglia rigida (Regola del percettrone)

- Curve di apprendimento (plots dell'accuratezza totale del **training set** rispetto al numero di iterazioni) per la regola del percettrone sui dati di terremoti/esplosioni.

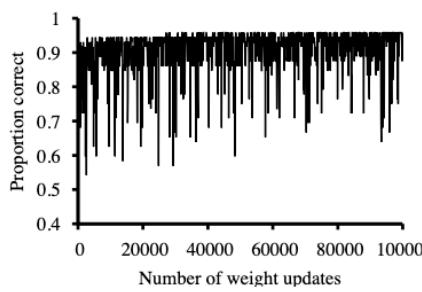


original data



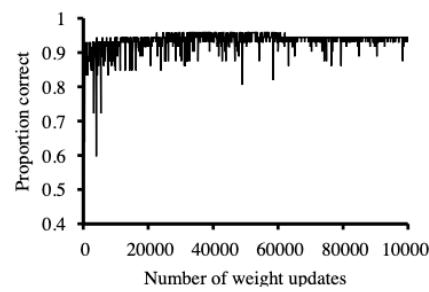
messy convergence
700 iterations

noisy, non-separable data



convergence failure
100,000 iterations

learning rate decay
 $\alpha(t) = 1000/(1000 + t)$

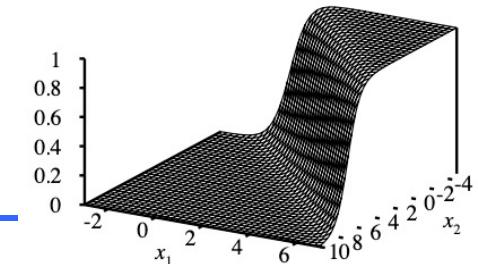
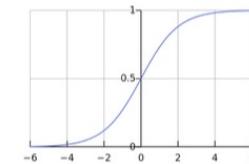


slow convergence
100,000 iterations

- Trovare l'**ipotesi** dell'errore minimo è **NP hard**, ma possibile con il decadimento del **learning rate**.

Classificazione lineare con regressione logistica

- ▶ Finora: passando l'output di una funzione lineare attraverso una **funzione di soglia** T si ottiene un classificatore lineare.
- ▶ Problema: la natura difficile di T porta problemi:
 - ▶ T non è differenziabile né continuo - l'apprendimento tramite la regola del percettrone diventa imprevedibile.
 - ▶ T è “eccessivamente preciso” vicino al confine necessita di giudizi più graduati.
- ▶ Idea: ammorbidire la soglia, approssimarla con una funzione differenziabile. Usiamo la **funzione logistica standard** $l(x) = \frac{1}{1+e^{-x}}$
Quindi abbiamo $h_w(x) = l(w \cdot x) = \frac{1}{1+e^{-(w \cdot x)}}$
- ▶ (**Ipotesi di regressione logistica nello spazio peso**). Grafico di un'ipotesi di regressione logistica per i dati di terremoto/esplosione. Il valore in (w_0, w_1) è la probabilità di appartenere alla classe etichettata 1.
- ▶ Parliamo intuitivamente della **scogliera** nel classificatore.



Regressione logistica

- ▶ Il processo di **adattamento del peso** in $h_w(x) = \frac{1}{1+e^{-(w \cdot x)}}$ viene chiamato **regressione logistica**.
- ▶ Non esiste una soluzione semplice in forma chiusa, ma la discesa del gradiente è una regressione logistica semplice.
- ▶ Poiché le nostre ipotesi hanno un output continuo, utilizziamo la funzione di perdita di errore quadratica L_2 .
- ▶ Per un esempio (x, y) calcoliamo le derivate parziali:

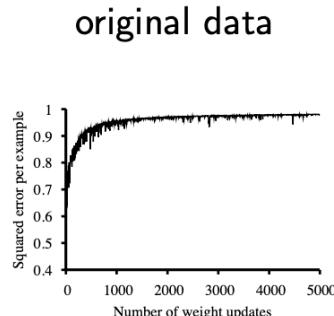
$$\begin{aligned}\frac{\partial}{\partial w_i}(L_2(w)) &= \frac{\partial}{\partial w_i}(y - h_w(x)^2) \\ &= 2 \cdot h_w(x) \cdot \frac{\partial}{\partial w_i}(y - h_w(x)) \\ &= -2 \cdot h_w(x) \cdot l'(w \cdot x) \cdot \frac{\partial}{\partial w_i}(w \cdot x) \\ &= -2 \cdot h_w(x) \cdot l'(w \cdot x) \cdot x_i\end{aligned}\quad (\text{tramite chain rule})$$

Regressione logistica

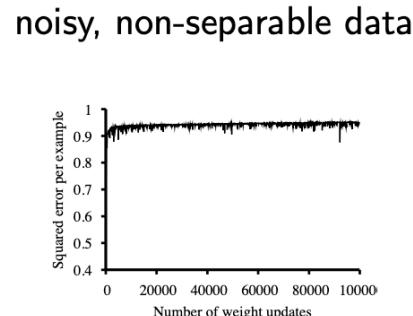
- ▶ La derivata della **funzione logistica** soddisfa $I'(z) = I(z)(1-I(z))$ quindi
 $I'(\mathbf{w} \cdot \mathbf{x}) = I(\mathbf{w} \cdot \mathbf{x})(1 - I(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$
- ▶ La regola per l'**aggiornamento logistico** (aggiornamento del peso per minimizzare la perdita) è

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha \cdot (y - h_{\mathbf{w}}(\mathbf{x})) \cdot h_{\mathbf{w}}(\mathbf{x}) \cdot (1 - h_{\mathbf{w}}(\mathbf{x})) \cdot \mathbf{x}_i$$

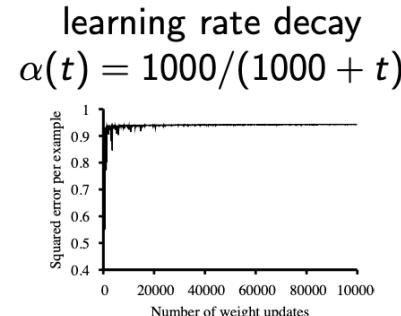
- ▶ (Rifacendo le curve di training).



messy convergence
5000 iterations



convergence failure
100,000 iterations



slow convergence
100,000 iterations

- ▶ Risultato: l'**aggiornamento logistico** sembra funzionare meglio dell'**aggiornamento del percepitrone**.

Modelli non parametrici

Apprendimento basato sulle istanze

- ▶ Apprendimento parametrico vs apprendimento non parametrico
- ▶ L'apprendimento si concentra sull'adattare i parametri di una famiglia ristretta di modelli di probabilità ad un dataset senza restrizioni
- ▶ I metodi di apprendimento parametrico sono spesso semplici ed efficaci, ma semplificano eccessivamente ciò che realmente succede
- ▶ L'apprendimento non parametrico permette alla complessità delle ipotesi di crescere con i dati
- ▶ L'apprendimento basato sulle istanze è non parametrico siccome costruisce delle ipotesi direttamente dai dati di training

Modelli *nearest-neighbor*

- ▶ L'idea chiave: i vicini sono simili
 - sia X un insieme di esempi etichettati (il training set)
 - dato un punto x da classificare, si calcola l'insieme U dei k punti dell'insieme X più vicini ad x secondo una determinata metrica
 - Si calcola la classe C più frequente all'interno dell'insieme U
 - Si assegna x a C
- ▶ Come definire il *vicinato* N
 - ▶ Se è troppo piccolo, non ci sono data point
 - ▶ Se è troppo grande, la densità è la stessa ovunque
 - ▶ Una soluzione è quella di definire N per contenere k *point*, dove k è grande abbastanza da assicurare una stima significativa (di solito tra 5 e 10)

K-NN per una data query x

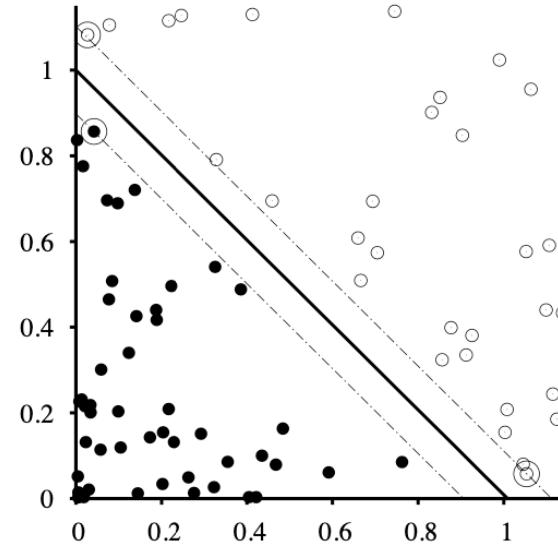
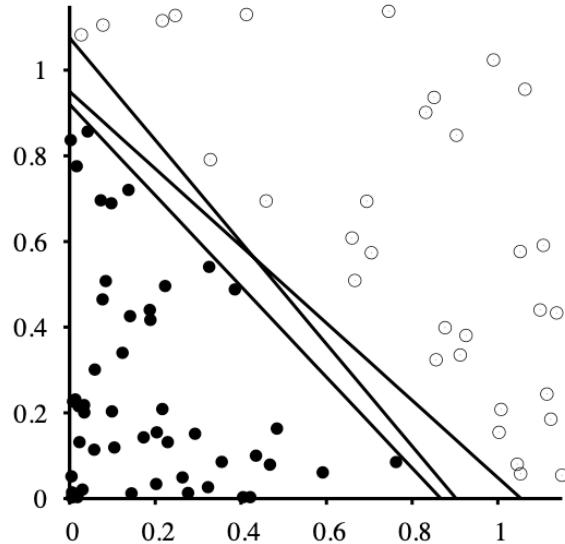
- ▶ Quale data point è più vicino ad x ?
 - ▶ Abbiamo bisogno di una distanza metrica, $D(x_1, x_2)$
 - ▶ La distanza euclidea D_E è la più popolare
 - ▶ Quando ogni dimensione misura qualcosa di diverso è inappropriate usare D_E
 - ▶ È importante standardizzare la scala di ogni dimensione
 - ▶ La distanza di Mahalanobis è una soluzione
 - ▶ Le caratteristiche discrete dovrebbero essere trattate diversamente
 - ▶ Distanza di Hamming
 - ▶ Usare k-NN per predire
- ▶ Alta dimensionalità pone un altro problema
 - ▶ I *nearest neighbors* rappresentano una strada molto lunga!

Support-vector machines

- ▶ Support-vector machines (SVM oppure support-vector networks) sono modelli di apprendimento supervisionato per la classificazione e la regressione.
- ▶ Le SVM
 - ▶ costruiscono un separatore di **margine massimo**, ovvero un confine di decisione con la maggiore distanza possibile dai punti di esempio. Questo li aiuta a **generalizzare** bene.
 - ▶ può incorporare i dati in uno spazio a dimensione superiore, dove è linearmente separabile dal **trucco del kernel**
 - ▶ l'iperpiano di separazione è un'ipersuperficie nei dati originali, danno priorità agli esempi critici (**vettori di supporto**).
(migliore generalizzazione)
- ▶ Uno degli approcci più diffusi per l'apprendimento supervisionato “off-the-shelf”.

Support-vector machines

- ▶ Dato un dataset **E** linearmente separabile, il **separatore di margine massimo** è il **separatore lineare s** che **massimizza il margine**, ovvero la distanza di E da s.
- ▶ Esempio. Tutte le righe a sinistra sono validi separatori lineari:



- ▶ Ci aspettiamo che il separatore del margine massimo a destra si **generalizzi** meglio!
- ▶ **Idea:** ridurre al minimo la **generalized loss** prevista anziché la loss empirica.

Trovare il separatore di margine massimo

- ▶ Prima di vedere come trovare il separatore di margine massimo, . .
- ▶ Abbiamo un training set $\{(x_1, y_1), \dots, (x_n, y_n)\}$ dove
 - ▶ $y_i \in \{-1, 1\}$ (invece di {1,0})
 - ▶ $x_i \in \mathbb{R}^p$ (classificazione multilineare)
- ▶ Vogliamo trovare un iperpiano che separe al massimo i punti con $y_i = -1$ da quelli con $y_i = 1$.
- ▶ Ricordiamo: qualsiasi iperpiano (in particolare qualsiasi separatore lineare) è rappresentato come il set $\{x | (\mathbf{w} \cdot x) + \mathbf{b} = 0\}$, dove
 - ▶ \mathbf{w} è il vettore normale (non necessariamente normalizzato) dell'iperpiano.
 - ▶ Il parametro \mathbf{b} determina l'offset dell'iperpiano dall'origine lungo il vettore normale \mathbf{w} .
- ▶ Idea: utilizzare la discesa del gradiente per cercare lo spazio di tutti \mathbf{w} e \mathbf{b} per massimizzare le combinazioni.

(funziona, ma le SVM seguono un percorso diverso)

Trovare il separatore di margine massimo (caso separabile)

- ▶ Idea: il **margine** è delimitato dai due iperpiani descritti da $(w \cdot x) + b = -1$ (limite inferiore) e $(w \cdot x) + b = 1$ (limite superiore).
La distanza tra loro è $\frac{2}{\|w\|_2}$
- per massimizzare il **margine**, minimizzare $\|w\|_2$ mantenendo x_i fuori dal **margine**.
- ▶ Vincoli: $(w \cdot x_i) + b \geq 1$ per $y_i = 1$ e $(w \cdot x_i) + b \leq -1$ per $y_i = -1$
o semplicemente $y_i(w \cdot x_i - b) \geq 1$ per $1 \leq i \leq n$.
- ▶ Problema di ottimizzazione: minimizzare $\|w\|_2$ mentre $y_i(w \cdot x_i - b) \geq 1$ per $1 \leq i \leq n$.

Trovare il separatore di margine massimo (caso separabile)

- ▶ Dopo un po' di passaggi arriviamo ad una **rappresentazione alternativa**: trova la soluzione ottimale risolvendo l'equazione SVM

$$\underset{\alpha}{\operatorname{argmax}} \left(\sum_j \alpha_j - \frac{1}{2} \left(\sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k) \right) \right)$$

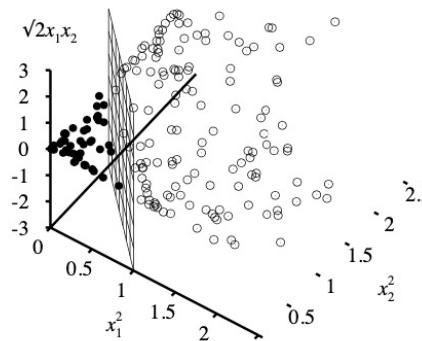
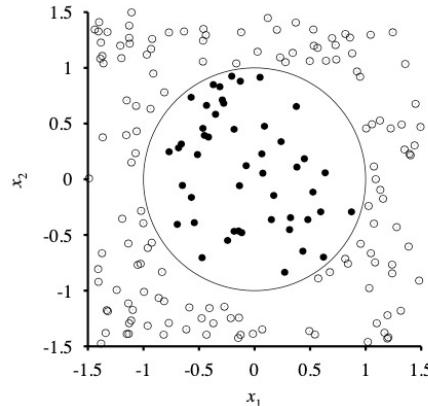
- ▶ sotto i vincoli $\alpha_j \geq 0$ e $\sum_j \alpha_j y_j = 0$
- ▶ **Osservazioni:** Questa equazione ha tre proprietà importanti:
 1. L'espressione è **convessa** → il massimo può essere trovato in modo efficiente.
 2. I dati entrano nell'espressione solo sotto forma di prodotti scalari di coppie di punti → una volta che sono stati calcolati gli α_i ottimali, abbiamo

$$h(\mathbf{x}) = \operatorname{sign} \left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b \right)$$

- 3. I pesi α_j associati a ciascun punto sono zero tranne che in corrispondenza dei **vettori di supporto**, i punti più vicini al **separatore**.
- ▶ Esistono buoni pacchetti software per risolvere tali ottimizzazioni di programmazione quadratica
- ▶ Una volta trovato un vettore ottimale α , utilizziamo $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$.

Support-vector machines (Kernel Trick)

- ▶ **Problema:** cosa succede se i dati **non sono separabili linearmente?**
- ▶ **Idea:** trasforma i dati in uno spazio di dimensioni superiori.
- ▶ (Proiezione di un set di dati non separabile)
a sinistra: il vero limite di decisione è $x_1^2 + x_2^2 \leq 1$



- ▶ **a destra:** mappare in uno spazio di input tridimensionale $\langle x_1^2, x_2^2, \sqrt{2}x_1x_2 \rangle$
→ separabile da un iperpiano.
- ▶ **Risultato:** Mappiamo ogni vettore di input \mathbf{x} su una $F(\mathbf{x})$ con $f_1 = x_1$, $f_2 = x_2$ e $f_3 = \sqrt{2}x_1x_2$.

Support-vector machines (Kernel Trick)

- ▶ Idea: sostituire $x_j \cdot x_j$ con $F(x_j) \cdot F(x_j)$ nell'equazione SVM.
(calcola in uno spazio di dimensione più alta)
- ▶ Spesso possiamo calcolare $F(x_j) \cdot F(x_j)$ senza calcolare F ovunque.
- ▶ Se $F(x) = \langle x_1^2, x_2^2, \sqrt{2}x_1x_2 \rangle$ allora $F(x_j) \cdot F(x_j) = (x_j \cdot x_j)^2$
(abbiamo aggiunto $\sqrt{2}$ in F in modo che funzioni)
- ▶ Chiamiamo la funzione $(x_j \cdot x_j)^2$ una **funzione kernel**. (*ce ne sono altre*)
- ▶ Sia X un insieme non vuoto, a volte indicato come **insieme di indici**.
Una funzione simmetrica $K : X \times X \rightarrow \mathbb{R}$ è chiamata **funzione kernel** su X se e solo se $\sum_{i,j=1}^n c_i c_j K(x_i, x_j) \geq 0$ for any $x_i \in X$, $n \in \mathbb{N}$, and $c_i \in \mathbb{R}$

Support-vector machines (Kernel Trick)

- ▶ In generale: possiamo apprendere i separatori non lineari risolvendo

$$\underset{\alpha}{\operatorname{argmax}} \left(\sum_j \alpha_j - \frac{1}{2} \left(\sum_{j,k} \alpha_j \alpha_k y_j y_k K(\mathbf{x}_j, \mathbf{x}_k) \right) \right)$$

- ▶ dove K è una funzione kernel
- ▶ La funzione $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + \mathbf{x}_j \cdot \mathbf{x}_k)^d$ è una funzione kernel corrispondente a uno spazio delle caratteristiche la cui dimensione è esponenziale in d . Si chiama kernel polinomiale.

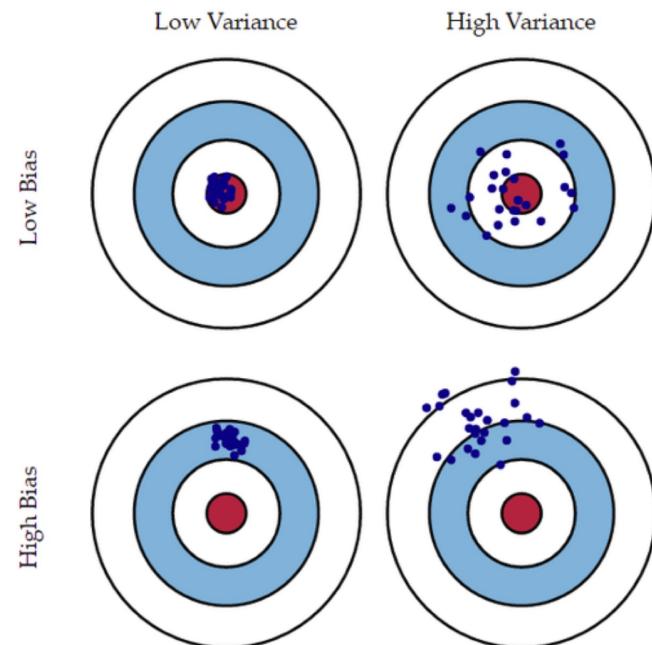
Ensemble Learning

Ensemble Learning

- ▶ Che cos'è l'apprendimento d'insieme (ensemble learning)?
- ▶ L'idea dell'**ensemble learning** è quella di selezionare una raccolta, o insieme, di ipotesi, h_1, h_2, \dots, h_n , e combinare le loro previsioni facendo la media, votazione o attraverso un altro livello di apprendimento automatico. Chiamiamo le singole ipotesi **modelli base** e loro combinazione un **modello ensemble**.
- ▶ Due motivi per usare l' ensemble learning
 - ▶ Il primo è ridurre il **bias** (distorsione).
 - ▶ Il secondo motivo è ridurre la **varianza**.

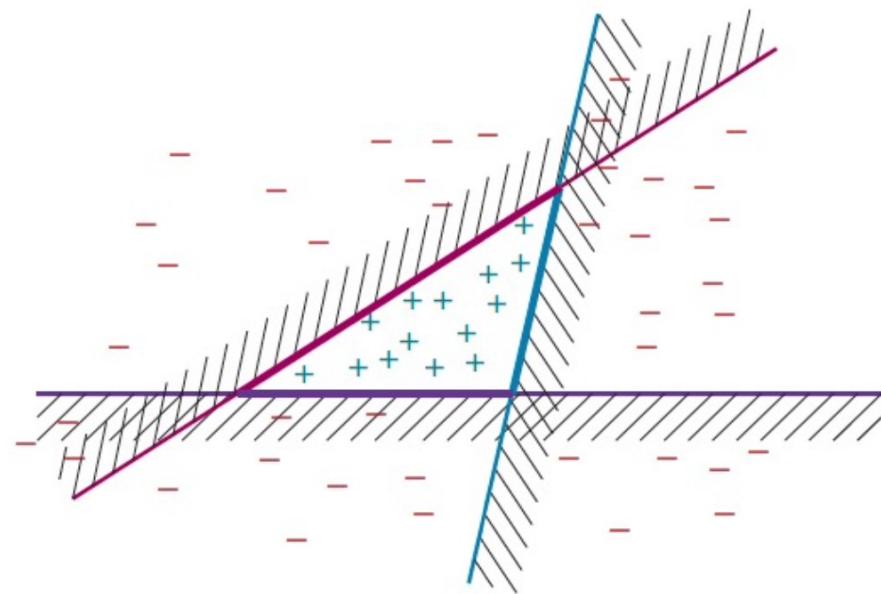
$$\mathbb{E}[(y - t)^2] = \underbrace{(y_* - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\text{Var}(y)}_{\text{variance}} + \underbrace{\text{Var}(t)}_{\text{Bayes error}}$$

▶ Perdita attesa



Ensemble Learning

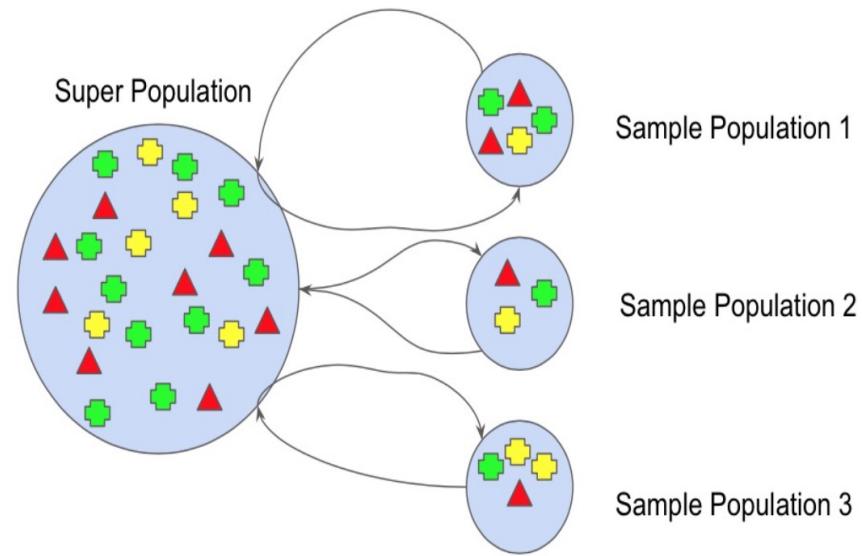
- ▶ Ensemble di classificatori lineari
- ▶ Un ensemble di **tre classificatori lineari** può rappresentare una regione triangolare che non potrebbe essere rappresentata da un **singolo classificatore lineare**.
- ▶ Un **ensemble di n** classificatori lineari consente di **realizzare** più funzioni.



Ensemble Learning

Bagging

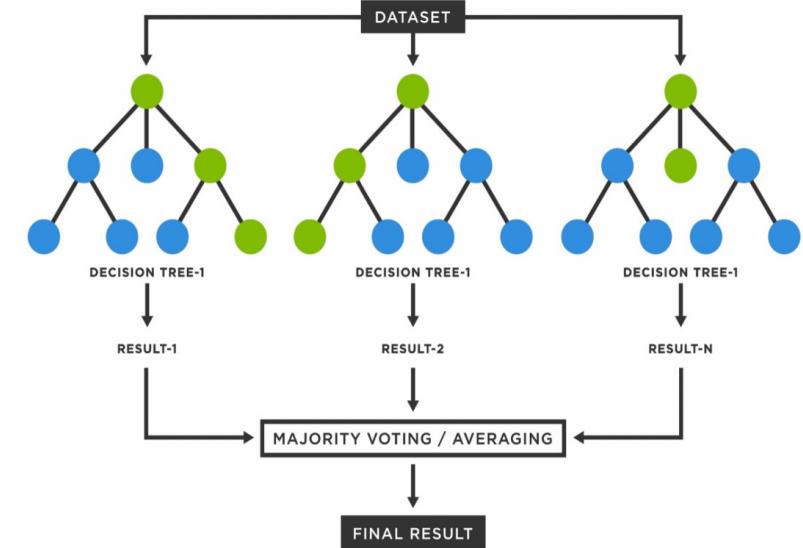
- ▶ Nel bagging, generiamo K distinti training set campionando con sostituzione dal training set originale.
- ▶ Il bagging tende a ridurre la varianza ed è un approccio standard quando i dati sono limitati o quando si ritiene che il modello di base sia in overfitting.
- ▶ Lo si usa per lo più su alberi di decisione



Ensemble Learning

Random forests

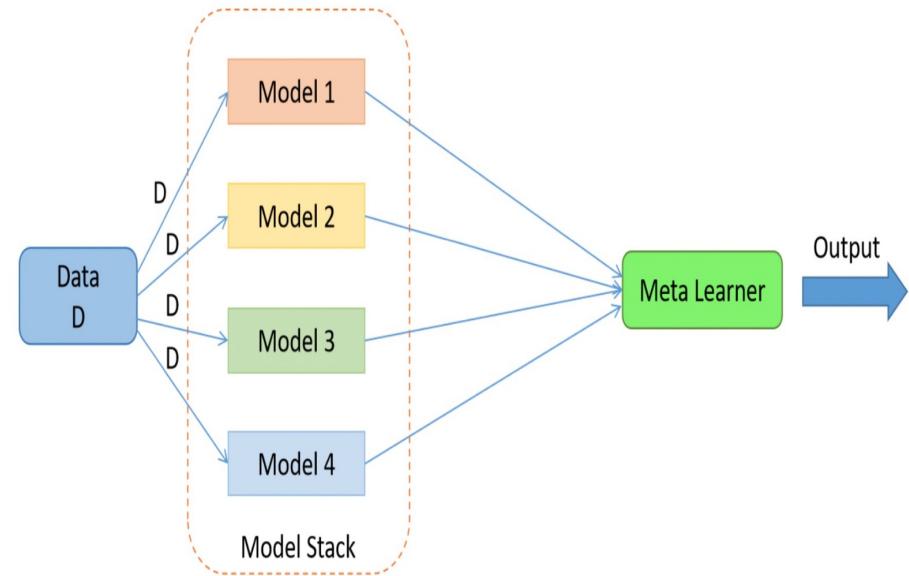
- ▶ Il modello della random forest è una forma di **bagging di alberi decisionali** in cui adottiamo ulteriori passaggi per rendere l'insieme di alberi K più diversificato, per ridurre la **varianza**.
- ▶ Le foreste casuali possono essere utilizzate per la classificazione o la regressione.
- ▶ L'idea chiave è di **variare casualmente le scelte degli attributi** (piuttosto che gli esempi di addestramento).



Ensemble Learning

Stacking

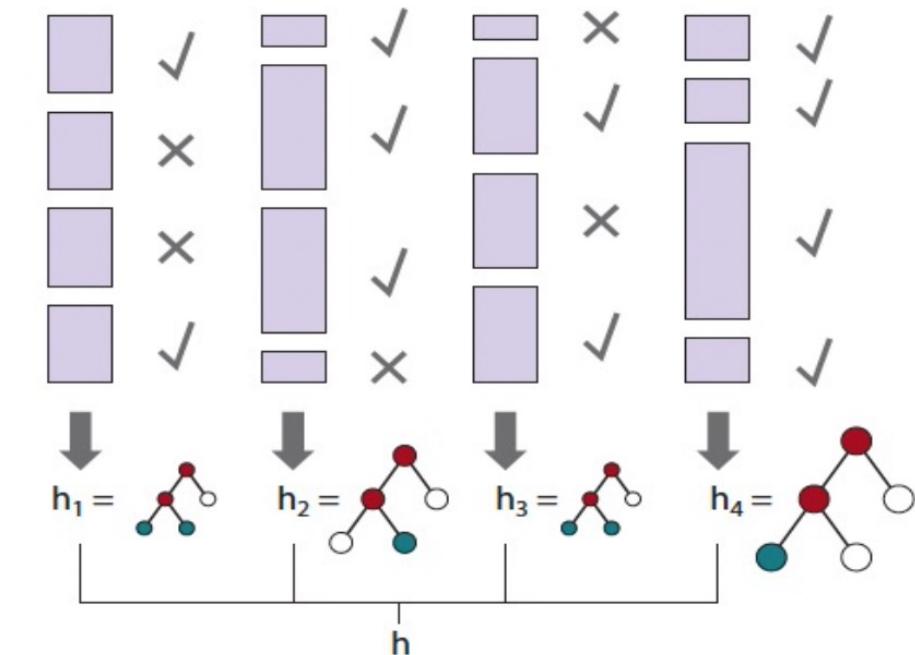
- ▶ Mentre che il bagging combina più modelli di base della stessa classe di modelli addestrati su dati diversi.
- ▶ Stacking (o generalizzazione impilata) **combina più modelli di base da differenti classi di modelli** addestrate sugli stessi dati.
- ▶ Di solito riduce il bias



Ensemble Learning

Boosting

- ▶ Il metodo di ensemble più popolare è chiamato **boosting**.
- ▶ Training set pesato, in cui ogni esempio ha un peso associato $W_j \geq 0$ che descrive quanto l'esempio dovrebbe contare durante il training.
- ▶ Le ipotesi che si sono comportate meglio nel training hanno un peso maggiore nella votazione



Riassunto

1. Se il feedback disponibile fornisce la risposta corretta per ogni input, allora il problema si chiama **apprendimento supervisionato**. Il compito è apprendere una funzione $y = h(x)$.
 2. L'apprendimento di una funzione a valori discreti è chiamato **classificazione**; l'apprendimento di una funzione continua **regressione**.
 3. L'**apprendimento induttivo** implica la ricerca di un'ipotesi che si accorda bene con gli esempi.
 4. Gli **alberi di decisione** possono rappresentare tutte le funzioni booleane. L'euristica del **guadagno di informazioni** fornisce un metodo efficiente per trovare un albero decisionale semplice e coerente.
 5. Le prestazioni di un algoritmo di apprendimento sono misurate dalla **curva di apprendimento**, che mostra l'accuratezza delle predizioni sul test set in funzione della dimensione del set di training. Quando ci sono più modelli tra cui scegliere, la **cross-validation** può essere utilizzata per selezionare un modello che si generalizza bene.
 6. A volte non tutti gli errori sono uguali. Una **loss function** ci dice quanto sia grave ogni errore; l'obiettivo è di minimizzare la loss su un set di validazione.
-

Riassunto

1. La **regressione lineare** è un modello ampiamente utilizzato. I parametri ottimali di un modello di regressione lineare possono essere calcolati esattamente, oppure possono essere trovati mediante la ricerca della **discesa del gradiente** (applicabile anche a modelli che non hanno una soluzione in forma chiusa).
2. Un **classificatore lineare** con una soglia rigida, noto anche come **perceptron**, può essere addestrato mediante una semplice regola di aggiornamento del peso per adattarsi a dati separabili linearmente. In altri casi, la regola non converge.
3. La **regressione logistica** sostituisce la soglia hard del perceptron con una soglia soft definita da una funzione logistica. La **discesa del gradiente** funziona bene anche per dati rumorosi che non sono separabili linearmente.
4. I **modelli non parametrici** utilizzano tutti i dati per fare ogni previsione, invece di cercare di riassumere i dati con pochi parametri. Ad esempio kNN.
5. Le **SVM** trovano separatori lineari con margine massimo per migliorare le prestazioni di generalizzazione del classificatore. I metodi del kernel trasformano implicitamente i dati di input in uno spazio ad alta dimensione in cui può esistere un separatore lineare, anche se i dati originali non sono separabili.
6. I **metodi di ensemble** come il bagging e il boosting spesso hanno prestazioni migliori rispetto ai metodi individuali.