lez1-Agenti e Ambienti

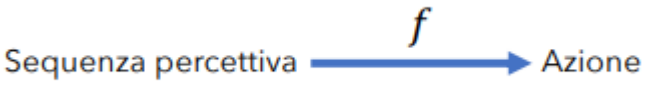
Agenti

Un **agente** è qualsiasi cosa che percepisce il suo ambiente attraverso dei sensori e agisce su di esso mediante degli attuatori

Percezione
Termine usato per indicare i dati che i sensori di un agente percepiscono.

Sequenza percettiva
La storia completa di tutto ciò che l'agente ha percepito nella sua esistenza

Funzione agente
Esprime il comportamento di un agente. Descrive la corrispondenza tra una qualsiasi sequenza percettiva e una specifica azione



Programma agente
la funzione agente di un agente artificiale è implementata da un programma agente. Compito dell'IA: progettare il programma agente.

Def agente razionale:per ogni sequenza di percezioni compie l'azione che massimizza il valore atteso della misura di prestazione, considerando le sue percezioni passate e la sua conoscenza pregressa.

Razionalità
In un dato momento, ciò che è razionale dipende da 4 fattori.

Misura di prestazione
che definisce il criterio del successo

Conoscenza pregressa
dell'ambiente da parte dell'agente

Azioni
che l'agente può compiere

Sequenza percettiva
dell'agente fino all'istante corrente

Agenti autonomi
un agente è autonomo nella misura in cui il suo comportamento dipende dalla sua capacità di acquisire esperienza (non dall' ''aiuto'' del progettista) Un agente il cui comportamento fosse determinato solo dalla sua conoscenza built-in (i.e., pregressa), sarebbe non autonomo e poco flessibile.

Ambienti

Ambiente operativo: un «problema» di cui un agente razionale rappresenta una «soluzione».

Un ambiente operativo può essere specificato mediante una descrizione PEAS.

- Performance (prestazione)
- Environment (ambiente)
- Actuators (attuatori)
- Sensors (sensori)

Es.: Agente autista di taxi

	P	E	A	S
tipo di agente	**misura di prestazione**	**ambiente**	**attuatori**	**sensori**
guidatore di taxi	sicuro, veloce, ligio alla legge, viaggio confortevole, massimizza i profitti, minimizza l'impatto su altri utenti della strada	strada, altri veicoli nel traffico, polizia, pedoni, clienti, tempo atmosferico	sterzo, acceleratore, freni, frecce, clacson, schermo, voce	telecamere, radar, tachimetro, GPS, sensori del motore, accelerometro, microfoni, touchscreen

Proprietà degli ambienti operativi

- **Osservabilità**
  - **Completamente osservabile**
    sensori di un agente gli danno accesso allo stato completo dell'ambiente in ogni momento, Non occorre memorizzare uno stato interno per tenere traccia del mondo
  - **Parzialmente osservabile**
    Sono presenti limiti o inaccuratezze nell'apparato sensoriale,
    Es.: una parte dei dati dell'ambiente non viene rilevata dai sensori.

- **Agente singolo/ multi-agente**
  - **Ambiente multi-agente competitivo:** 2+ agenti che cercano di massimizzare una misura di prestazione, ciascuno a scapito degli altri.
    Esempio: giochi come gli scacchi
  - **Ambiente multi-agente cooperativo:** 2+ agenti che hanno lo stesso obiettivo (massimizzare la misura di prestazione per uno non la minimizza necessariamente per un altro), e comunicano.

- **Predicibilità**

    - **Deterministico:**Se lo stato successivo è completamente determinato dallo stato corrente e dall'azione. Esempio: scacchi.
    - **Stocastico:** Esistono degli elementi di incertezza con associata probabilità. Esempio: guida (non si puo' prevedere esattamente il traffico)
    - **Non deterministico:**Esistono vari risultati possibili di una stessa azione (si possono elencare), ma non in base ad una probabilità definita. Esempio: esiste la probabilità che domani piova.

- **Episodico/Sequenziale**
    - **Episodico:**
        - L'esperienza dell'agente è divisa in episodi atomici.
        - In ogni episodio, l'agente riceve una percezione e poi esegue un'azione .
        - Ogni episodio non dipende dalle azioni intraprese negli episodi precedenti. L'agente non deve «pensare avanti».
    - **Sequenziale:**
        - Ogni decisione può influenzare quelle successive
        - Esempi: la guida e gli scacchi.

- **Statico/Dinamico**
    - **Statico:** l'ambiente non cambia mentre l'agente decide l'azione. Esempio: cruciverba.
    - **Dinamico:** L'ambiente cambia nel tempo, va osservata la contingenza. Non rispondere in tempo corrisponde a scegliere di non fare nulla. Esempio: taxi.
    - **Semidinamico:** Nel tempo l'ambiente non cambia, ma la valutazione della prestazione dell'agente sì. Esempio: scacchi con timer

- **Discreto/continuo**
    - Possono assumere valori discreti o continui: lo stato (es., solo un numero finito di stati), il (modo in cui è gestito il) tempo, le percezioni, le azioni.

- **Noto/Ignoto**
    - Si riferisce non all'ambiente in sé, ma allo stato di conoscenza dell'agente circa le «leggi fisiche» dell'ambiente.
    - L'agente conosce come funziona l'ambiente oppure deve compiere azioni esplorative?
    - Noto diverso da osservabile (esempio: giochi di carte, noto ma parzialmente osservabile in quanto le regole del gioco sono note ma non conosciamo il valore delle carte coperte)

La struttura degli Agenti

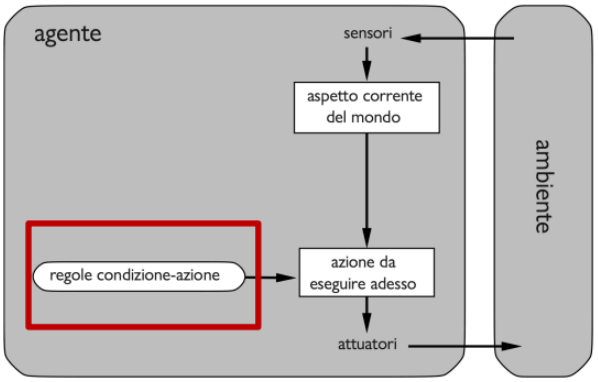
Agente basato su tabella

La scelta dell'azione consiste nell'accedere a una tabella che associa un'azione ad ogni possibile sequenza di percezioni.
Problemi:

- Dimensione: per giocare a scacchi la tabella avrebbe un numero di righe >> numero di atomi nell'universo
- Difficile da costruire
- Nessuna autonomia
- Di difficile aggiornamento, apprendimento complesso.

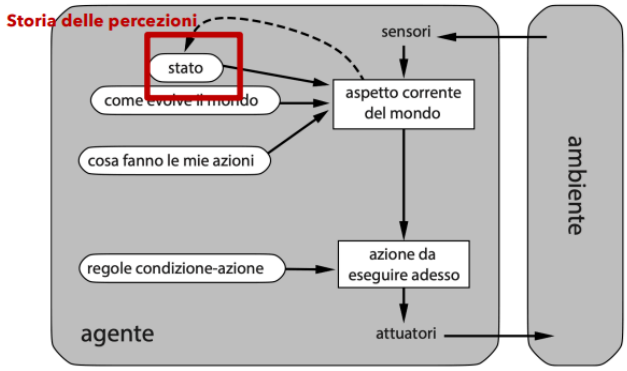
Agenti reattivi semplici

Scelgono le azioni sulla base della percezione corrente, ignorando tutta la storia percettiva precedente. No storia in memoria



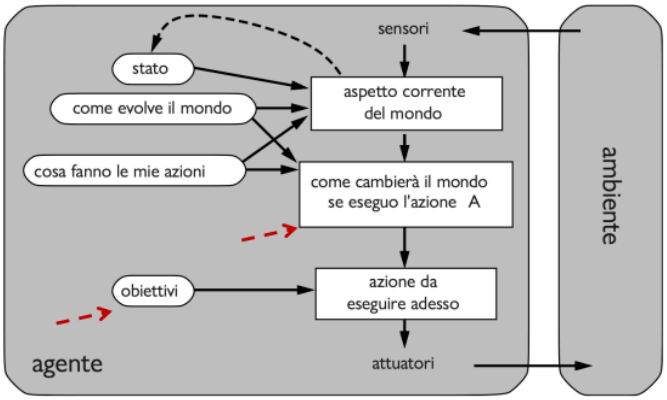
Agenti basati su modello

L'agente mantiene uno stato interno che serve per tenere traccia della parte del mondo che non può vedere all'istante corrente. Lo stato dipende dalla storia delle percezioni.



Agenti con obbiettivo

Oltre che della descrizione dello stato corrente, l'agente ha bisogno di qualche informazione circa il suo obiettivo (goal).



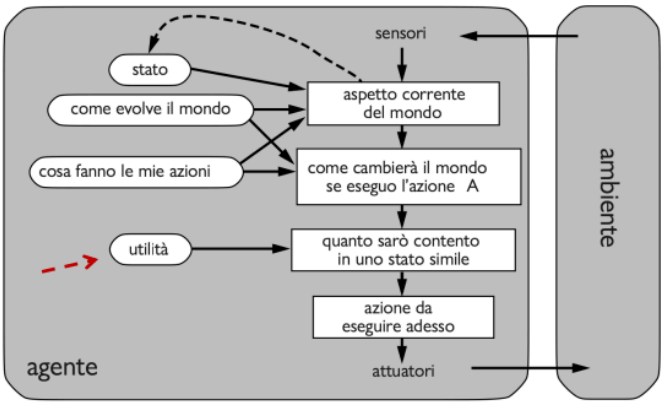
Agenti con valutazione di utilità

Obiettivi alternativi (o più modi per raggiungere l'obiettivo)

- l'agente deve decidere verso quali di questi muoversi.
- necessaria una funzione di utilità (che associa ad uno stato obiettivo un numero reale)
- Taxi: stessa destinazione, diverse utilità per: tempo, economia, sicurezza…. Obiettivi

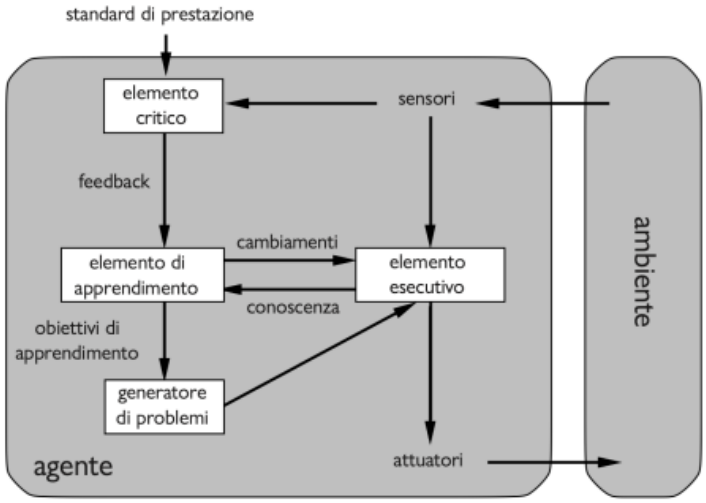
più facilmente raggiungibili di altri

- la funzione di utilità tiene conto anche della probabilità di successo (e/o di ciascun risultato): utilità attesa (o in media)



Agenti che apprendono

Qualsiasi tipo di agente visto finora può essere costruito come agente in grado di apprendere.
Un agente capace di apprendere può essere diviso in quattro componenti astratti.



Elemento di apprendimento (learning element)

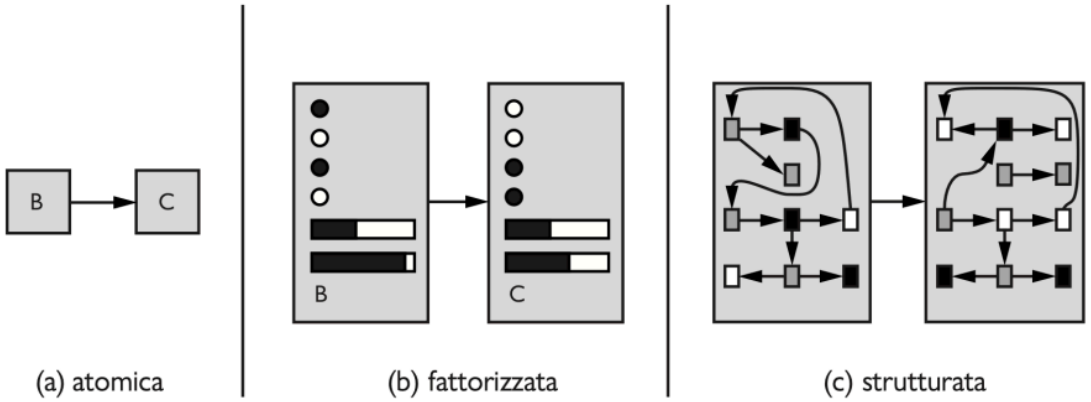
- Produce cambiamenti al programma agente
- Migliora prestazioni, adattando i suoi componenti, apprendendo dall'ambiente

Elemento esecutivo (performance element) Il programma agente (visto sinora per decidere le azioni)

Elemento critico Valuta le prestazioni correnti dell'agente e determina se e come modificare l'elemento esecutivo affinché in futuro si comporti meglio

Generatore di problemi Suggerisce nuove situazioni da esplorare.

TIPI DI RAPPRESENTAZIONI
Come vengono rappresentati dai componenti l'ambiente abitato dall'agente.



Atomica → ogni stato del mondo è indivisibile, non ha struttura interna
Fattorizzata → suddivide ogni stato in un insieme fissato di variabili o attributi, ognuno dei quali può avere un valore.
Strutturata → suddivide ogni stato in un insieme fissato di variabili o attributi, ognuno dei quali può avere un valore, e relazioni tra loro.

Risolvere i problemi con la ricerca

Agenti risolutori di problemi

Quando l'azione giusta da compiere non è subito evidente, l'agente deve guardare in avanti.
Ricerca: processo computazionale condotto da un agente risolutore di problemi.

Algoritmi informati
L'agente è in grado di identificare la distanza dall'obiettivo.

Algoritmi non informati
L'agente non è in grado di identificare la distanza dall'obiettivo.

Processo di risoluzione

1. **Formulazione dell'obiettivo**
2. **Formulazione del problema**
    a. Elaborare un modello astratto della parte di mondo interessata.
    b. Descrizione degli stati rilevanti, e delle azioni per raggiungere l'obiettivo.
3. **Ricerca**
    a. L'agente simula nel suo mondo delle sequenze di azioni, fino a che non trova una sequenza che conduce all'obiettivo (soluzione).
4. **Esecuzione**
    a. Eseguire le azioni specificate nella soluzione

Problemi di ricerca e soluzioni

- **Spazio degli stati:**Un insieme di possibili stati in cui si può trovare l'ambiente.
- **Stato iniziale:**Lo stato in cui si trova l'agente inizialmente.
- **Stati obbiettivo:**Può essere uno solo, o più di uno.
- **Azioni possibili:**L'insieme finito di azioni che possono essere eseguite in un dato stato.
- **Modello di transizione:**Descrive ciò che fa ogni azione. e in che stato passa.
- **Funzione costo:** Esprime il costo di eseguire una certa azione quando si è in un certo stato.

Stato iniziale, Azioni possibili e Modello di transizione definiscono implicitamente lo spazio degli stati.

Cammino
Una sequenza di azioni. Costo di un cammino = somma dei costi di ogni azione (additività).

Soluzione
Un cammino che porta dallo stato iniziale ad uno stato obiettivo.

Soluzione Ottimale
Tra tutte le soluzioni, quella con il costo minimo.

lez-2 Ricerca non informata

Algoritmi di ricerca

Un algoritmo di ricerca riceve in input un problema di ricerca e restituisce in output una soluzione o fallimento. Consideriamo algoritmi che sovrappongono un albero di ricerca al grafo dello spazio degli stati.

Albero di ricerca:

- ogni nodo corrisponde a uno stato nello spazio degli stati
- i rami corrispondono alle azioni
- la radice corrisponde allo stato iniziale del problema

Spazio degli stati: Descrive l'insieme (potenzialmente infinito) degli stati nel mondo e le azioni che consentono transizioni tra essi.

Albero di ricerca:

- Descrive i cammini tra gli stati per raggiungere l'obiettivo.
- Possono esserci più cammini per raggiungere qualsiasi stato.
- Più nodi possono corrispondere allo stesso stato.
- Per ogni nodo, c'e' un solo cammino per tornare alla radice.

Espansione di un nodo
Generiamo un nuovo nodo (figlio, o successore) per ognuno degli stati risultati, ciascuno dei nodi generati avra' il nodo espanso come padre.

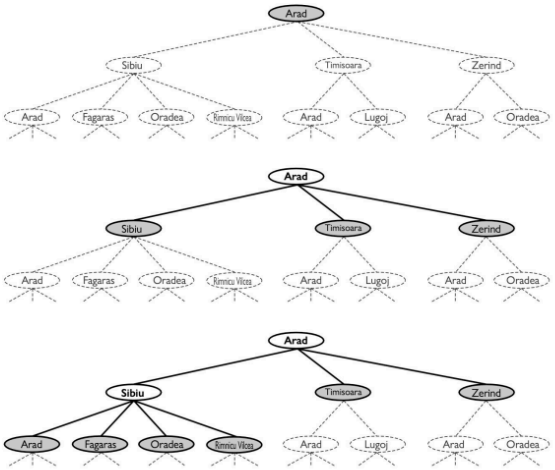
Frontiera
Insieme dei nodi generati ma non ancora espansi.

Separa due regioni del grafo dello spazio degli stati:

- una regione **interna** in cui ogni nodo e' stato espanso
- una regione **esterna** di stati che non sono ancora stati raggiunti

Stati raggiunti
stati del grafo generati anche non espansi.

Esempio di alberi di ricerca per il problema della Romania.



La ricerca va avanti espandendo ogni volta un nodo della frontiera.

Tipi di algoritmi

Il ruolo dell'algoritmo è quello di capire quale nodo della frontiera espandere.

Best-first

- scegliamo un nodo n che corrisponde al valore minimo di una funzione di valutazione $f(n)$
- Restituiamo il nodo se è lo stato obiettivo
- Ogni nodo figlio viene aggiunto alla frontiera se non era già stato raggiunto in precedenza, oppure se è stato raggiunto, ma con un cammino di costo maggiore.
- alla fine, o restituisce fallimento, oppure un nodo che rappresenta un cammino che porta ad un obiettivo.

Strutture dati per la ricerca

Nodo

- n.stato: lo stato a cui corrisponde il nodo
- n.padre: il nodo che lo ha generato
- n.azione: l'azione che ha portato dal padre al nodo n
- n.cammino: il costo dal nodo iniziale a n, g(n)--> sarà il costo

Frontiera

implementata tramite una coda, le operazioni svolte sono:

- Vuota: l'algoritmo termina con fallimento
- PoP: rimuove il nodo in cima e lo restituisce
- Top: restituisce il nodo in cima ma non lo rimuove
- Aggiungi: aggiunge un nodo in coda

Stati raggiunti

Tabella di lookup

- ogni chiave è uno stato
- ogni valore è il nodo per tale stato

Stato ripetuto
stato che corrisponde a più nodi (stesso stato) dello stesso cammino.

Cammino ciclico
Anche se lo spazio degli stati ha un piccolo numero di stati, l'albero di ricerca completo è infinito.

Cammino ridondante
Una via peggiore per arrivare allo stesso stato.

Nella ricerca su grafo si controlla la presenza di cammini ridondanti, nelle ricerche ad albero non controlla la presenza di cammini ridondanti.

Come trattare i cammini ridondanti:

- ricordare tutti gli stati precedentemente raggiunti
  - Adatto per: spazio degli stati in cui ci siano molti cammini ridondanti, casi in cui c'è sufficiente memoria per tenere traccia dei raggiunti
- Non preoccuparti di cammini ridondanti
- controllare la presenza di cicli ma non di cammini ridondanti.
  - Possiamo controllare la presenza di cicli senza aggiungere memoria, usando la catena di puntatori ai padri (all'indietro nell'albero)

Misure di prestazione

1. **Completezza**: l'algoritmo garantisce di trovare una soluzione, se esiste, e di riportare correttamente il fallimento, se non esiste

2. **Ottimalità rispetto al costo**: trova la soluzione con il costo di cammino minimo.

3. **Complessità temporale:** quanto tempo.

4. **Complessità spaziale**: quanta memoria.

Come misurare la complessità
Se il grafo è una struttura esplicita si usa come misura di difficoltà del problema $|V| + |E|$.

misuriamo la complessità in termini di:

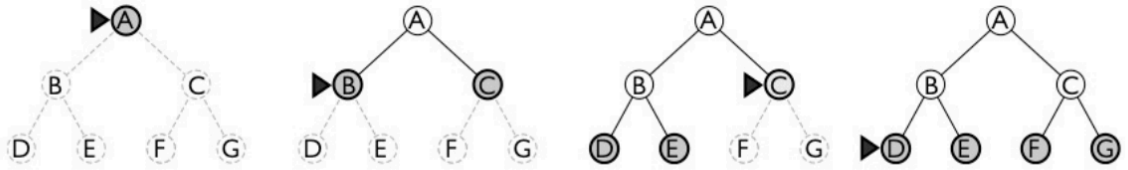
- **depth** (profondità) ***d*** → numero di azioni di una soluzione ottima.
- **massima profondità** ***m*** → numero massimo di azioni in qualsiasi cammino.
- **branching factor** (fattore di ramificazione) b → numero di successori di un nodo.

Strategie di ricerca non informata

I metodi di ricerca non informata hanno accesso soltanto alla definizione del problema. Gli algoritmi costruiscono un albero di ricerca nel tentativo di trovare una soluzione, e differiscono tra loro in base a quale nodo espandono prima.

Ricerca in ampiezza

appropriata quando tutte le azioni hanno lo stesso costo



Si diffonde a «ondate» di profondità uniforme

implementazione 1 → chiamata a Ricerca-best-First, usando come funzione di valutazione *f(n) = depth(n)*

implementazione 2 (efficiente) →

- **Frontiera** → Implementata come coda FIFO
- **Raggiunti** → insieme degli stati raggiunti (implementato come Set e non come lookup table) una volta trovato lo stato non possiamo trovarne uno migliore per raggiungerlo.
- **Test obiettivo** anticipato, controllare se un nodo è soluzione quando viene generato e non quando viene espanso dalla coda.

Pseudo-codice implementazione 2

```
function RICERCA-IN-AMPIEZZA(problema) returns un nodo soluzione o fallimento
    nodo ←NODO(problema.STATOINIZIALE)
    if problema.È-OBIETTIVO(nodo.STATO) then return nodo
    frontiera ← una coda FIFO, con nodo come elemento iniziale
    raggiunti ← {problema.STATOINIZIALE}
    while not VUOTA?(frontiera) do
        nodo ← POP(frontiera)
        for each figlio in ESPANDI(problema, nodo) do
            s ← figlio.STATO
            if problema.È-OBIETTIVO(s) then return figlio
            if s non è in raggiunti then
                aggiungi s a raggiunti
                aggiungi figlio a frontiera
    return fallimento
```

Ricerca in ampiezza
Implementazione 2
(più efficiente)

Analisi

- Completa (trova sempre la soluzione)
- Ottima rispetto al costo per problemi in cui ogni azione ha lo stesso costo.
- Costo temporale e spaziale → #nodi generati in totale per soluzione di profondità = $O(b^d)$. (b nodi successori, d profondità della soluzione)

Esempio: $b = 10$; 1 milione nodi al sec generati; 1 nodo occupa 1000 byte

Piu incisivo!

Profondità	Nodi	Tempo	Memoria
2	110	0,11 ms	107 kilobyte
4	11.100	11 ms	10,6 megabyte
6	10^6	1.1 sec	1 gigabyte
8	10^8	2 min	103 gigabyte
10	10^{10}	3 ore	10 terabyte
12	10^{12}	13 giorni	1 petabyte
14	10^{14}	3,5 anni	1 esabyte

Scala male: solo istanze piccole!

Ricerca a costo uniforme (algoritmo di Dijkstra)

le azioni hanno costi diversi → si diffonde a <<ondate>> di costo uniforme quindi visito tutti i cammini in ordine crescente di costo. Ricerca best-first con funzione di valutazione = costo del cammino dalla radice fino al nodo corrente.

Analisi

Completa

- Esamina sistematicamente tutti i cammini in ordine crescente di costo
- Non entra mai in un cammino infinito (assumendo costi positivi)

Ottima rispetto al costo

- La prima soluzione che trova ha un costo basso almeno come quello degli altri nodi sulla frontiera

Complessità

La complessità in questo caso è caratterizzata da

- C^* → costo della soluzione ottima
- ε → limite inferiore imposto al costo di ogni azione
- limite inferiore $[C^*/\varepsilon]$ → è il numero maggiore di azioni che servono per raggiungere il costo della soluzione ottima (può essere >> d)

Costo in tempo e spazio

Goal-test e profondità $O\left(b^{1+\left\lfloor \frac{C^*}{\epsilon} \right\rfloor}\right)$ arresto posticipato, solo dopo aver generato anche l'ultima frontiera, oltre la del goal.

se ogni azione ha lo stesso costo *simile a ricerca in ampiezza*

$O(b^{1+d})$

Ricerca in profondità

La ricerca raggiunge immediatamente il livello più profondo dell'albero, dove i nodi non hanno successori, quindi "torna indietro" al nodo più profondo che ha ancora successori non espansi. Espande il nodo a profondità maggiore della frontiera.

Frontiera → implementata come coda LIFO (last in first out). In cui vengono inseriti gli elementi espansi seguendo un ordine definito a priori e si va a estrarre, dopo l'ordinamento, l'ultimo nodo messo in coda.

Senza tabella dei raggiunti

Analisi

Per spazi finiti e che sono alberi è completa ed efficiente, per spazi infiniti è incompleta. Può bloccarsi in un ciclo infinito.

PRO: quando è praticabile una ricerca ad albero è molto efficiente in memoria dato che non viene tenuta la tabella dei raggiunti.

Se spazio degli stati finito e struttura ad albero:

- Costo in tempo → $O(b^m)$
- Costo in memoria → O(b*m)
- Backtracking: quando si espande un nodo viene generato solo un successore e si ricorda quale deve essere generato in seguito solo $O(m)$

Ricerca a profondità limitata

Forniamo un limite ***l*** di profondità alla ricerca, per evitare cammini infiniti

```
function RICERCA-PROFONDITÀ-LIMITATA(problema, ℓ) returns un nodo soluzione o fallimento o soglia
    frontiera ← una coda LIFO (stack) con NODO(problema.STATOINIZIALE) come elemento iniziale
    risultato ← fallimento
    while not VUOTA?(frontiera) do
        nodo ← POP(frontiera)
        if problema.È-OBIETTIVO(nodo.STATO) then return nodo
        if PROFONDITÀ(nodo) > ℓ then
            risultato ← soglia
        else if not È-CICLO(nodo) do
            for each figlio in ESPANDI(problema, nodo) do
                aggiungi figlio a frontiera
    return risultato
```

Soglia: valore di output che segnala che potrebbe esistere una soluzione a un livello di profondità maggiore di l

Analisi

Se si sceglie male L, l si sceglie in base al problema e al diametro del grafo.

L'algoritmo non riesce a trovare la soluzione: incompleto. Non ottimale.
Complessità in tempo → $O(b^l)$
Complessità spaziale → O(b*l)

Ricerca ad approfondimento iterativo

Provando la ricerca a profondità limitata con tutti i valori di *l* finché non si trova una soluzione oppure restituisce fallimento.

```
function RICERCA-APPROFONDIMENTO-ITERATIVO(problema) returns un nodo soluzione o fallimento
    for profondità = 0 to ∞ do
        risultato ← RICERCA-PROFONDITÀ-LIMITATA(problema, profondità)
        if risultato ≠ soglia then return risultato
```

Analisi

- Completa su spazi di stati aciclici finiti, o su qualsiasi spazio degli stati finito se controlliamo la presenza di cicli risalendo l'intero cammino
- Ottima per problemi in cui tutte le azioni hanno lo stesso costo
- Complessità temporale: $O(b!)$ se esiste una soluzione, $O(b")$ se non esiste
- Complessità spaziale: $O(b*d)$ se esiste una soluzione, $O(b*m)$ se non esiste

Direzione della ricerca

Un problema ortogonale alla strategia è quello della direzione:

1) **ricerca in avanti o guidata dai dati**: si esplora lo spazio di ricerca dallo stato iniziale allo stato obiettivo;
2) **ricerca all'indietro o guidata dall'obiettivo**: si esplora lo spazio di ricerca a partire da uno stato goal e riconducendosi a sotto-goal fino a trovare uno stato iniziale.

Quale direzio

criterio	in ampiezza	a costo uniforme	in profondità	a profondità limitata	ad approfondimento iterativo	bidirezionale (se applicabile)
completa?	Sì[1]	Sì[1,2]	No	No	Sì[1]	Sì[1,4]
ottima rispetto al costo?	Sì[3]	Sì	No	No	Sì[3]	Sì[3,4]
tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

[1] – se b è finito, e lo spazio degli stati ha una soluzione o è finito

[2] – completa se tutti I costi di azione sono $\geq \varepsilon > 0$

[3] – ottima rispetto al costo se i costi delle azioni sono tutti identici

[4] – se in entrambe le direzioni si usa una ricerca in ampiezza o una ricerca a costo uniforme

ne?
Conviene procedere nella direzione in cui il fattore di diramazione è minore

- Si preferisce ricerca all'indietro quando, e.g: l'obiettivo è chiaramente definito o si possono formulare una serie limitata di ipotesi.
- Si preferisce ricerca in avanti quando, e.g.: gli obiettivi possibili sono molti.

Ricerca bidirezionale

Si procede in entrambe le direzioni fino ad incontrarsi

- Occorre tenere traccia di due frontiere e avere due tabelle di raggiunti (es. best first bidirez.)
- Nodo da espandere: sempre quello con valore minimo della funzione di valutazione

Si espande un solo nodo per volta

Analisi

Complessità tempo: $O(b^{d/2})$
Complessità spazio: $O(b^{d/2})$

Confronto

lez-3 Ricerca Informata

Strategie di ricerca informata o euristica

Ricerca informata → usare la conoscenza specifica del dominio del problema per fornire suggerimenti su dove si potrebbe trovare l'obiettivo.

h(n) → **funzione euristica**, restituisce il costo stimato del cammino meno costoso dallo stato del nodo n a uno stato obiettivo
NB: *i valori della funzione h(n) non sono ottenibili direttamente dalla descrizione del problema.*

Algoritmi di ricerca Informata

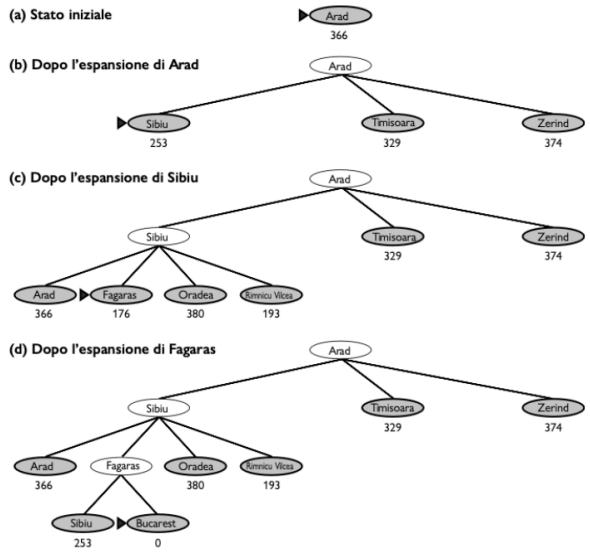
Ricerca best-first greedy

Idea: espandere prima il nodo con il valore più basso di $h(n)$, cioè quello che appare più vicino all'obiettivo.

Analisi:

- completa per spazi finiti ma non per spazi infiniti.
- Non ottima rispetto al costo. (guarda esempio sulle slide)
- Complessità spaziale e temporale O(|V|)

esempio:

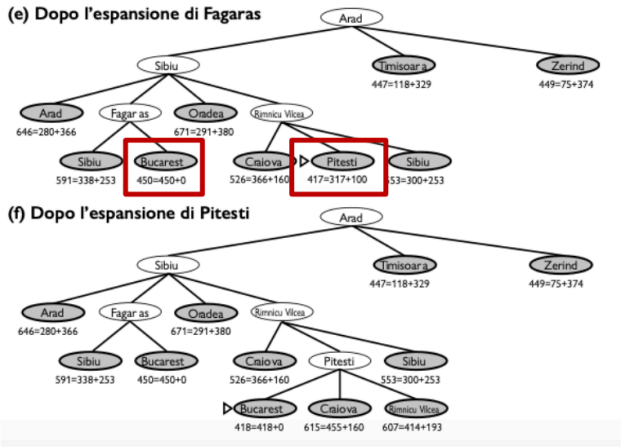


Ricerca A*

idea: RIcerca Best-first che usa la funzione f(n) = g(n) + h(n)
dove:

- g(n): costo del cammino dal nodo iniziale al nodo n
- h(n): costo stimato del cammino più breve da n a uno stato obiettivo

esempio:



Analisi

- Completa, se tutti i costi di azione sono > ε > 0 e lo spazio degli stati ha una soluzione o è finito
- Ottima rispetto al costo se l'euristica $h(n)$ è ammissibile

def. Euristica ammissibile

Un'euristica è ammissibile se non sovrastima mai il costo per raggiungere un obiettivo. Se h*(n) è il costo minimo di un cammino da n fino a un goal, allora h(n) ≤ h*(n).

Dimostrazione per assurdo

Se $h(\cdot)$ è ammissibile allora A* è ottimo rispetto al costo

Dim per assurdo.

A* restituisce un cammino di costo C, ma esiste una soluzione di costo $C^* < C$.

Allora certamente esiste un nodo n sul cammino ottimo che non è stato espanso.

Se avessimo espanso tutti i nodi sul cammino Ottimo avremmo restituito la soluzione ottima

Se $h(\cdot)$ è ammissibile allora A* è ottimo rispetto al costo Dim per assurdo.

Denotiamo con:

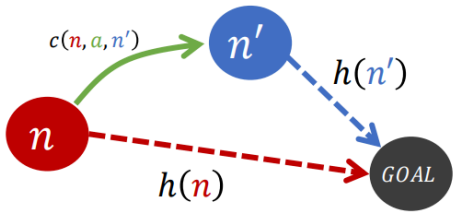
- $g^*(n)$ costo del cammino ottimo *dall'inizio fino ad* n
- $h^*(n)$ costo del cammino ottimo *da* n *fino ad un obiettivo*

- $f(n) > C^*$ Altrimenti n sarebbe stato espanso
- $f(n) = g(n) + h(n)$ Per definizione
- $f(n) = g^*(n) + h(n)$ Siccome n si trova su un cammino ottimo
- $f(n) \leq g^*(n) + h^*(n)$ Siccome $h(n)$ è ammissibile allora $h(n) \leq h^*(n)$
- $f(n) \leq C^*$ Per definizione, $C^* = g^*(n) + h^*(n)$

Quindi A* restituisce solo cammini ottimali rispetto al costo.

Euristica consistente → un'euristica $h(n)$ è consistente se, per ogni nodo n, e per ogni successore n' di n generato da un'azione a, risulta:

$$h(n) \leq c(n, a, n') + h(n')$$



Nota: se h è un'euristica consistente, la funzione di valutazione $f = g + h$ non decresce mai lungo i cammini. In questo caso si dice anche che h è un'euristica *monotona*.

Ad ogni salto avremo h(n) sempre più precisa

Euristica Consistente => Euristica Ammissibile

A* con euristica consistente →

- Ottimo rispetto al costo
- La prima volta che raggiungiamo uno stato sarà su un cammino ottimo

Potatura di sottoalberi: aspetto focale per l'efficienza di A*. Non dobbiamo generare e tenere in memoria sotto-alberi inutili per trovare una soluzione ottima.

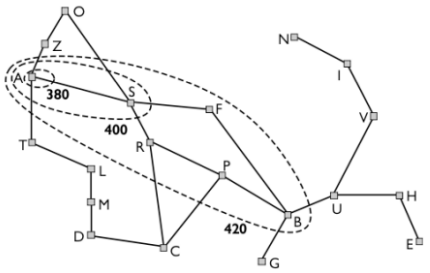
A* con euristica inconsistente →

- Possiamo trovarci con più cammini che raggiungono lo stesso stato
- Se ogni cammino nuovo ha costo inferiore al precedente finiremo con l'avere più nodi corrispondenti a quello stato sulla frontiera

A* con euristica inammissibile →

Può essere **ottima** rispetto al costo in due casi

- Esiste un cammino ottimo rispetto al costo lungo cui $h(n)$ è ammissibile per tutti i nodi sul cammino
- Se la soluzione ottima ha costo C^* e la seconda soluzione migliore ha costo C_2, e se $h(n)$ sovrastima i costi, ma al massimo di una quantità ≤ $C_2 - C^*$



Ricerca A* pesata

$f(n) = g(n) + W * h(n)$, per $W > 1$.

Se la soluzione ottima ha costo C^*, troverà una soluzione di costo compreso tra C^* e $W \times C^*$

Algoritmi di ricerca subottima

- **Ricerca a subottimalità limitata**: cerchiamo una soluzione con garanzia che si trovi entro un fattore costante W del costo ottimo.
    - **es A* pesata.**

- **Ricerca a costo limitato**: cerchiamo una soluzione il cui costo sia inferiore a una costante C.

- **Ricerca a costo illimitato**: accettiamo una soluzione di qualsiasi costo purché riusciamo a trovarla rapidamente.

Ricerca con memoria limitata

La problematica principale della ricerca A* è legato all'uso della memoria.

- La memoria è divisa tra frontiera e raggiunti

Trucco #1: gli stati sono presenti solo in una delle due posizioni (frontiera o raggiunti).
Trucco #2: rimuovere gli stati da raggiunti quando si può dimostrare che non servono più.

Ricerca Beam

Limita i nodi della frontiera ai K nodi con i migliori costi di ***f***.

- Incompleta e subottima, esplora solo un settore dei confini concentrici.

IDA*: Ricerca ad approfondimento iterativo combinata con A*

Viene mantenuto un valore di soglia è sul valore della funzione f = g + h, all'inizio threshold = f(radice).
Per ogni iterazione:

- ricerca esaustiva (in profondità) fino ad un costo f_{limite}
- quando viene trovato un nodo n con costo f(n) > $f(l)$, termina iterazione e ricomincia la successiva con $f(l) = f$(n), se non era obiettivo.

Risulta più lento di A* però lineare per la memoria occupata, perché tengo traccia di un cammino per volta ed effettuo il pruning (potatura dei sottoalberi) sui nodi di costo f(n) > f_{limite}.

Ricerca Best-first ricorsiva (RBFS)

Ricerca in BF con interruzione quando f supera un valore $flimite$.

- $flimite$ = il costo trovato sul miglior cammino alternativo
- se il nodo corrente supera questo limite, la ricorsione torna al cammino alternativo

Durante il ritorno, sostituisce il valore f di ogni nodo lungo il cammino con il miglior f dei suoi nodi figli:

- valore di backup che indica il valore della foglia migliore in un cammino abbandonato (per eventualmente tornarci)

Memory bounded A* (SMA*)

Idea: usare al meglio (tutta) la memoria disponibile

- SMA* procede coma A* fino ad esaurimento della memoria disponibile poi, «dimentica» il nodo peggiore, memorizzando nel nodo del padre il valore del nodo dimenticato
- A parità di f: espande la foglia migliore più recente e cancella la foglia peggiore più vecchia
- Ottimale se il cammino soluzione sta in memoria

Funzioni euristiche

Fattore di ramificazione effettivo

N = numero totale di nodi generato da A*
d = profondità dell'albero generato da A*
$b*$ = fattore di ramificazione che un albero uniforme di profondità d dovrebbe avere per contenere N + 1 nodi.

b* = media dei figli che ha ogni nodo

Più il fattore di ramificazione più tende a 1 più è migliore

Dominazione $\rightarrow$ $h2$ domina $h1$ quando per ogni nodo n, risulta $h2(n)$ >= $h1$ (n)

Generalmente meglio usare un'euristica con valori più alti (**velocizza la ricerca**)

- a patto che sia consistente (quindi ammissibile)
- non richieda troppo tempo di calcolo

Costruzione di euristiche

Problema rilassato
Un problema rilassato è un problema con meno vincoli rispetto all'originale, dove

- il grafo dello spazio degli stati del problema rilassato è un supergrafo di quello dello spazio degli stati del problema originale

Aggiunge archi al grafo dello spazio degli stati

Euristica da problema rilassato

Il costo di una soluzione ottima per un problema rilassato è un euristica ammissibile per il problema originale. Quindi la soluzione ottima di un problema rilassato rappresenta una sottostima del problema originale.

Euristica da sottoproblemi
Il costo della soluzione ottimale di un sottoproblema è una sottostima del costo del problema originale. Quindi è un euristica ammissibile.

Massimizzazione
Se abbiamo un insieme di euristiche ammissibili $h_1,...,h_n$ e nessuna domina le altre possiamo costruire una euristica massimizzata data dalla funzione

$$h(n) = \max\{h_1(n),...,h_n(n)\}$$

Database di patterns
Memorizzare i costi esatti delle soluzioni di ogni possibile istanza di un sottoproblema
in questo modo possiamo avere un euristica ammissibile per ogni stato incontrato durante la ricerca estraendo dal database la configurazione del sottoproblema corrispondente.

Sottoproblemi multipli
Possiamo individuare più sottoproblemi per uno stesso problema di partenza quindi avere più euristiche e scegliere quella massima, ma non possiamo sommarle!
(nel problema dei tasselli: se sposto dei tasselli per risolvere 1-2-3-4, sposterò anche dei tasselli che contribuiscono a risolvere 5-6-7-8)

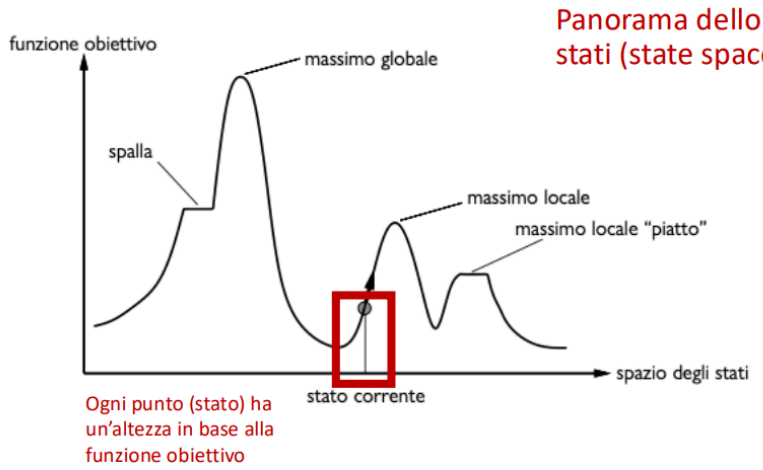
Database di pattern disgiunti
Nel calcolo del costo di una soluzione per un sottoproblema considero solo il costo delle azioni che coinvolgono i tasselli corrispondenti, in questo modo si possono sommare i costi ottenendo una euristica ammissibile.

lez-4 Ricerca Locale

Ricerca Locale

Gli algoritmi di ricerca locale operano a partire da uno stato iniziale e procedono verso gli stati adiacenti, **senza tenere traccia dei cammini e senza tenere traccia degli stati già raggiunti.**
Questi algoritmi **non sono sistematici** cioè potrebbero non esplorare mai una porzione dello spazio degli stati. Però usano **pochissima memoria** e riescono a trovare **soluzioni ragionevoli in spazi degli stati grandi.**

Problemi di Ottimizzazione
Lo scopo è trovare lo stato migliore secondo una funzione obiettivo.



Ricerca Hill climbing

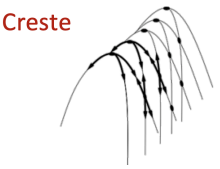
- Ricerca locale greedy.
- L'algoritmo tiene traccia solo dello stato corrente, quindi non del cammino né degli stati raggiunti
- Ad ogni iterazione passa allo stato vicino con valore più alto (nodo vicino)

function HILL-CLIMBING(*problema*) **returns** uno stato che è un massimo locale
corrente ← *problema*.STATOINIZIALE
while *true* **do**
vicino ← un successore di *corrente* di valore più alto
if VALORE(*vicino*) ≤ VALORE(*corrente*) **then return** *corrente*
corrente ← *vicino*

Hill climbing problematiche

Essendo Hill Climbing un algoritmo greedy, prende il nodo successivo senza pensare a dove lo porterà in futuro. Per questo l'algoritmo può arrestarsi su una soluzione che in realtà è solo un massimo/minimo locale oppure ritrovarsi su "pianori".

Esempi



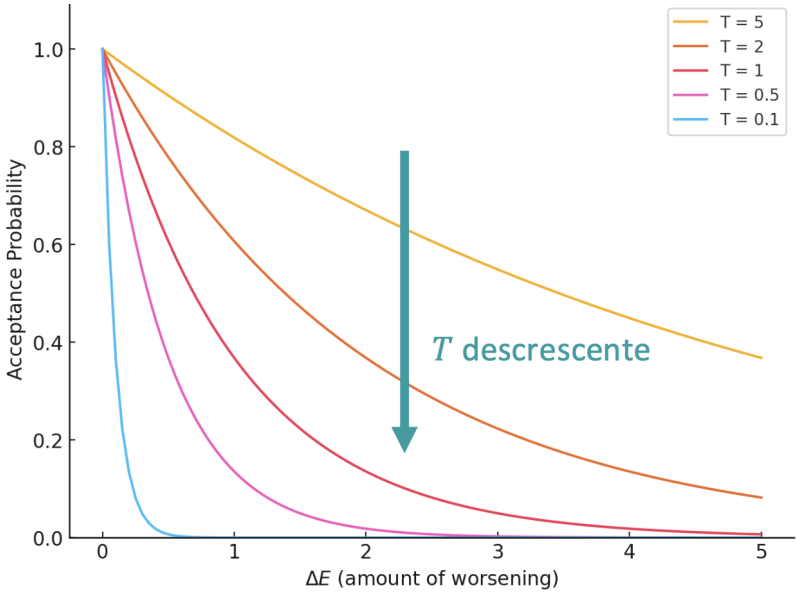
Possibili Soluzioni

- **consentire (un numero limitato di) «mosse laterali» per affrontare plateau.**
  - L'algoritmo può permettere alcuni passi in cui la funzione obiettivo rimane invariata. Questo consente all'algoritmo di esplorare l'area circostante nella speranza di trovare un percorso che lo porti fuori dal plateau e verso un miglioramento. ponendo un numero di mosse limite.
- **Hill climbing Stocastico scegliere a caso tra tutte le mosse che vanno nella direzione della funzione crescente (che puntano verso l'alto)**
  - probabilità di scelta maggiore per mosse che portano più in alto
  - Converge più lentamente, ma può trovare soluzioni migliori
- **condurre una serie di ricerche hill climbing partendo da stati iniziali generati casualmente, fino a che non viene raggiunto un obiettivo.**
  - Completo con probabilità 1
  - prima o poi genera lo stato obiettivo come stato iniziale
  - Numero atteso di riavvii = $1/p$, dove p = probabilità di successo per ogni ricerca hill climbing

Simulated annealing

Struttura simile a quella dell'hill climbing, invece che la mossa migliore, viene scelta una mossa casuale:

- se la mossa migliora la situazione, viene accettata.y
- se la mossa non migliora la situazione viene accettata con probabilità di: $e^{-\frac{\Delta E}{T}}$;



Al crescere della cattiva qualità della mossa decresce, con la temperatura, la probabilità che quella mossa venga scelta. Se T viene fatta decrescere abbastanza lentamente l'algoritmo troverà un massimo globale con probabilità che tende a 1.

```
function simulated-annealing(problema, velocità_raffreddamento) returns uno stato soluzione
        corrente ← problema.StatoIniziale
        for t ← 1 to ∞ do
                T ← velocità_raffreddamento[t]
                if T = 0 then return corrente
                successivo ← un successore scelto a caso di corrente
                ΔE ← Valore(successivo) - Valore(corrente)        Di quanto peggioro
                if ΔE < 0 then corrente ← successivo
                else corrente ← successivo solo con probabilità exp(−ΔE/T)
```

Legenda

- **delta E** è l'indice di peggioramento della soluzione trovata;
- **La temperatura** indica quanti passi abbiamo fatto, più andremo avanti con l'algoritmo più la temperatura scende;
- **la velocità di raffreddamento** indica quanto scende la temperatura per ogni azione presa.

Ricerca Local Beam

Idea: nella ricerca locale, tenere traccia di k stati anziché solo di 1.

- Si inizia con k stati generati casualmente
- Ad ogni passo, si generano i successori di tutti i k stati
    - se uno qualsiasi è obiettivo, l'algoritmo lo restituisce e termina
    - altrimenti, sceglie i k successori migliori dalla lista di tutti i successori e ricomincia
- Diverso dal riavvio casuale hill climbing, questa passa informazioni utili da un thread all'altro abbandonando cammini infruttuosi e spostando le risorse dove si stanno verificando i progressi maggiori

Algoritmi evolutivi

Varianti della ricerca beam stocastica motivati dalla metafora della selezione naturale in biologia.

- Una popolazione di individui → **stati**
- Gli individui più adatti → valore più alto
- producono prole → stati successori

Ricombinazione → processo in cui gli individui più adatti di una generazione si combinano nei nuovi individui della generazione successiva

Caratteristiche

- **Dimensione della popolazione**
- **Rappresentazione di ciascun individuo** → ogni individuo è una stringa su un alfabeto finito
- **Numero di genitori** → es. p = 2 genitori che si combinano per generare la prole

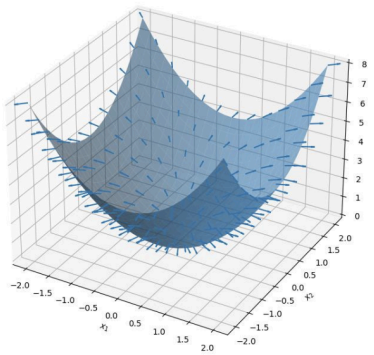
- **Processo di selezione** → selezionarli tra gli individui con probabilità proporzionale al loro punteggio di adattamento **(fitness)**
- **Procedura di ricombinazione** → selezionare un punto di crossover per suddividere ognuna delle stringhe genitori e poi ricombinare le parti per formare i figli
- **Tasso di mutazione** → determina la frequenza con cui la prole presenta mutazioni (genetiche) casuali della loro rappresentazione
- **Formazione della nuova generazione:**
    - Può essere semplicemente la nuova prole
    - Può contenere alcuni genitori con il migliore punteggio dalla generazione precedente **(*elitismo*)** garantisce che l'adattabilità totale non diminuisca
    - Si può eliminare ogni individuo al di sotto di una certa soglia di fitness **(*abbattimento*)**

Ricerca locale in spazi continui

Molti casi reali hanno spazi di ricerca continua: stato descritto da variabili continue $x_1,...,x_n$

Un ambiente organizzato in spazi di stati continui può apparire ostico infatti si avrebbero **fattori di ramificazione** infiniti con gli approcci visti finora, per questo dobbiamo introdurre delle ottimizzazioni:

- **Discretizzare**
    - ogni posizione è limitata a punti fissi in una griglia rettangolare possibile usare algoritmi di ricerca locale nello spazio discreto ottenuto
- **Gradiente empirico**
    - rendere limitato il fattore di ramificazione mediante un campionamento casuale degli stati successori e.g., valuto $f(\mathbf{x} + \delta)$ e verifico se il valore ottenuto è migliore rispetto a quello di $f(\mathbf{x})$.



$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix}$$

il gradiente ∇f fornisce l'entità e la direzione della pendenza più ripida

Ricerca con azioni non deterministiche

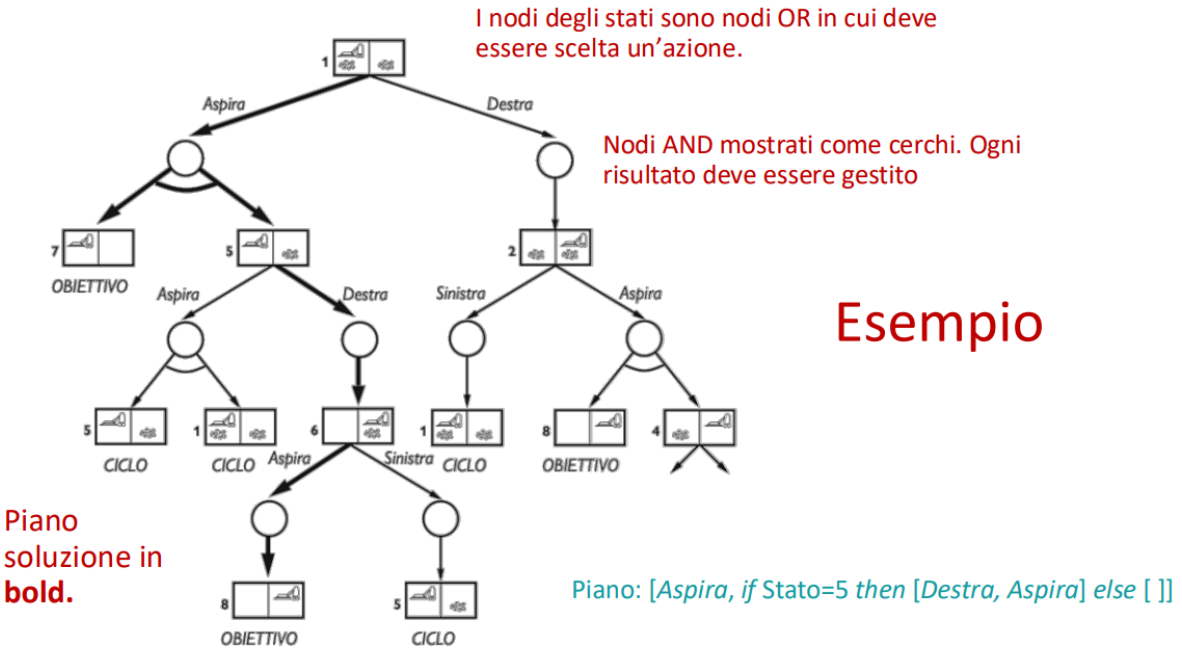
Nei casi reali l'ambiente è parzialmente osservabile e non deterministico le percezioni sono importanti perché restringono gli stati possibili ed informano sull'effetto dell'azione.

- Più che un piano l'agente può elaborare una "strategia", che tiene conto delle diverse eventualità: un piano con contingenza.

determinismo: “sono nello stato $s1$ e se eseguo l’azione a andrò nello stato $s2$”
non determinismo: “Sono nello stato $s1$ o $s3$, e se eseguo l’azione a passerò nello stato $s2$, $s4$ o $s5$”
stato-credenza: un insieme di stati che l’agente ritiene siano possibili

Alberi di ricerca AND - OR

Nodo OR: le scelte dell’agente l’agente deve compiere una scelta di 1 sola azione
Nodo AND: le diverse contingenze più stati possibili, da considerare tutti



P2 - Rappresentazione della conoscenza

Agenti basati su conoscenza
Un agente basato su conoscenza mantiene una base di conoscenza (KB): un insieme di enunciati (formule) espressi in un linguaggio di rappresentazione (PROP, FOL). Questo permette all'agente AI di prendere decisioni in autonomia grazie alla base di conoscenza che ha. La deduzione di di un certo fatto alfa è data dal problema: $KB \models \alpha$ cioè alfa è **conseguenza logica** di KB.

Interpretazione → Assegnare i valori di verità ai letterali

Modello → un'interpretazione che rende vera una formula

Tautologia/Validità → una formula per cui tutte le interpretazioni sono vere

Soddisfacibilità → quando una formula ha almeno un modello

Insoddisfacibilità → quando una formula non ha modelli

Equivalenza logica → due formule si dicono equivalenti se hanno gli stessi modelli

Conseguenza logica → una formula α è conseguenza logica di un insieme di formule KB se e solo se in ogni modello di KB, anche α è vera ($KB \models \alpha$)

NB: $KB \models \alpha \Leftrightarrow KB \Rightarrow \alpha$

Model checking → enumerare tutti i modelli di KB e verificare che lo siano anche di α

Th refutazione → $KB \models \alpha$ sse (KB and ¬α) è insoddisfacibile

Th Herbrand → Se $KB \models \alpha$ allora c'è una dimostrazione che coinvolge solo un sotto-insieme finito della KB proposizionalizzata.

Correttezza di un sistema inferenziale → se $KB \vdash \alpha$ allora $KB \models \alpha$

Completezza di un sistema inferenziale → se $KB \models \alpha$ allora $KB \vdash \alpha$

α $\models$ **β** se e soltanto se **α ⇒ β**

TV-Consegue

Enumera tutte le possibili interpretazioni 2^k, con k = simboli proposizionali

```
function TV-Consegue?(KB, α) // returns true oppure false
        inputs:  KB, la base di conoscenza, una formula della logica proposizionale
                 α, la query, una formula della logica proposizionale
simboli ← una lista dei simboli proposizionali contenuti in KB e α
return TV-Verifica-Tutto(KB, α, simboli, { })
```

```
function TV-Verifica-Tutto(KB, α, simboli, modello) //returns true oppure false
if Vuoto?(simboli) then
        if PL-Vero?(KB, modello) then return PL-Vero?(α, modello)
        else return true // quando KB è false, restituisce sempre true
else do
        P ← Primo(simboli); resto ← Resto(simboli)
        return  TV-Verifica-Tutto(KB, α, resto, modello ← {P = true})
                and
                TV-Verifica-Tutto(KB, α, resto, modello ← {P = false})
```

DPLL

Usa il th di refutazione KB unito not(α) e verifica se è soddisfacibile. Se non lo è hai dimostrato
che α è conseguenza logica di KB. **NB:** dimostra l'insoddisfacibilità.
Th refutazione

```
function DPLL-Soddisfacibile?(s) returns true oppure false
        inputs: s, una formula della logica proposizionale
clausole ← l'insieme di clausole nella rappresentazione CNF di s
simboli ← una lista di tutti i simboli proposizionali in s
return DPLL(clausole, simboli, { })
```

```
function DPLL(clausole, simboli, modello) returns true oppure false
 if ogni clausola in clausole è vera in modello then return true
 if qualche clausola in clausole è falsa in modello then return false
 P, valore ← Trova-Simbolo-Puro(simboli, clausole, modello)
 if P è diverso da null then return DPLL(clausole, simboli – P, modello ← {P = valore})
 P, valore ← Trova-Clausola-Unitaria(clausole, modello)
 if P è diverso da null then return DPLL(clausole, simboli – P, modello ← {P = valore})
 P ← Primo(simboli); resto ← Resto(simboli)
 return          DPLL(clausole, resto, modello ← {P = true}) or
                 DPLL(clausole, resto, modello ← {P = false})
```

WalkSat

NB: Se max-flips = ∞ e l'insieme di clausole è soddisfacibile prima o poi termina. Se è insoddisfacibile non termina.

```
function WalkSAT(clausole, p, max_flips) returns un modello o fallimento
        inputs: clausole, un insieme di clausole della logica proposizionale
        p, la probabilità di effettuare una "camminata casuale", tipicamente intorno a 0,5
        max_flips, numero massimo di inversioni di valore prima di abbandonare

        modello ← un assegnamento casuale di valori di verità ai simboli in clausole

        for i = 1 to max_flips do
          if modello soddisfa clausole then return modello
          clausola ← una clausola, falsa in modello, scelta casualmente nell'insieme clausole
          if Random(0, 1) =< p then inverti il valore in modello di un simbolo scelto
                                    casualmente in clausola
          else    inverti il valore di verità del simbolo in clausole che massimizza il numero
                  di clausole soddisfatte

        return fallimento
```

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutatività di $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutatività di $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associatività di $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associatività di $\vee$
$\neg(\neg\alpha) \equiv \alpha$	eliminazione della doppia negazione
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contrapposizione
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	eliminazione dell'implicazione
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$	eliminazione del bicondizionale
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$	distributività di $\wedge$ su $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)$	distributività di $\vee$ su $\wedge$

(leggi)

Regole di Inferenza

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$ *Modus ponens*

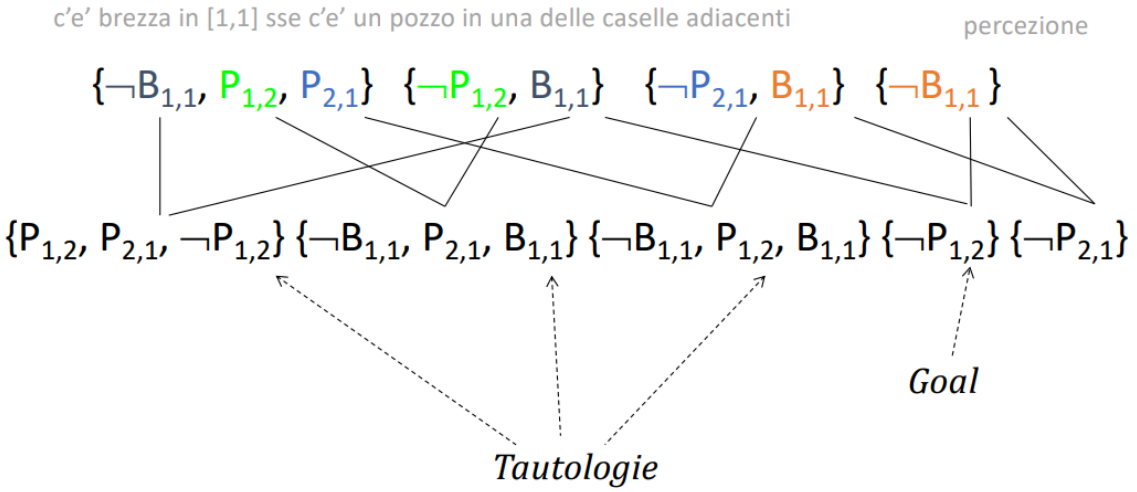
$$\frac{\alpha \wedge \beta}{\alpha}$$ *Eliminazione dell'AND*

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

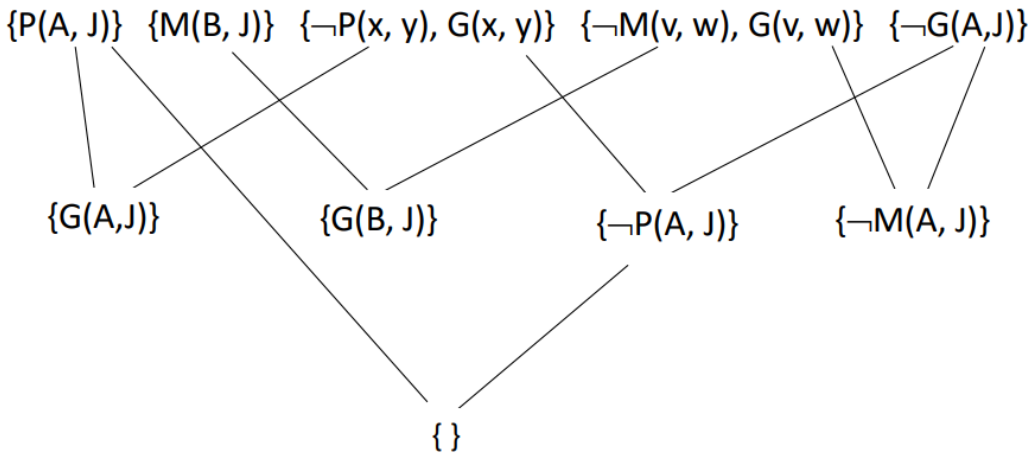
$$\frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Eliminazione e introduzione della doppia implicazione

Grafo di Risoluzione



Esempio di refutazione: il grafo



FOL

Trasformazione in Forma a Clausole

Istanziazione Universale

Istanziazione dell'Universale (∀-eliminazione)

$$\frac{\forall x\, A[x]}{A[x/g]}$$

dove *g* è un termine *ground* e A[*x*/*g*] è il risultato della sostituzione di *g* per *x* in A. possiamo così eliminare il "per ogni", permettendo così all'AI di inferire che per tutti gli elementi vale quella regole

Istanziazione Esistenziale

Istanziazione dell'esistenziale (∃-eliminazione)

$$\frac{\exists x\, A[x]}{A[x/k]}$$

1. se ∃ non compare nell'ambito di ∀, *k* è una costante nuova (*costante di Skolem*)
2. altrimenti va introdotta una funzione (di Skolem) nelle variabili quantificate universalmente

∃x Padre(x, G)	diventa	Padre(K, G)
∀x ∃y Padre(x, y)	diventa	∀x Padre(x, p(x))
	e non	∀x Padre(x, K)

... altrimenti tutti avrebbero lo stesso padre !

Posso mettere un segnaposto quando sostituisco il quantificatore Esiste per dire che esiste un elemento per cui il termine A è valido. Questo segnaposto si chiama costante di skolem.
NB: la precedenza degli operatori logici è data da

$$= > \neg > \wedge > \vee > \Rightarrow, \Leftrightarrow > \forall, \exists$$

UNO: Eliminazione delle implicazioni

$$A \Rightarrow B \quad \text{diventa} \quad \neg A \vee B$$

$$A \Leftrightarrow B \quad \text{diventa} \quad (\neg A \vee B) \wedge (\neg B \vee A)$$

DUE: Negazioni all'interno

$$\neg\neg A \quad \text{diventa} \quad A$$

$$\neg(A \wedge B) \quad \text{diventa} \quad \neg A \vee \neg B \quad \text{(De Morgan)}$$

$$\neg(A \vee B) \quad \text{diventa} \quad \neg A \wedge \neg B \quad \text{(De Morgan)}$$

$$\neg\forall x\, A \quad \text{diventa} \quad \exists x\, \neg A$$

$$\neg\exists x\, A \quad \text{diventa} \quad \forall x\, \neg A$$

TRE: Standardizzazione delle variabili: facciamo in modo che ogni quantificatore usi una variabile diversa

QUATTRO: Skolemizzazione: eliminazione dei quantificatori esistenziali (quando ci sono quantificatori esistenziali nell'ambito di uno universale si introducono le funzioni di Skolem)

CINQUE: Eliminazione quantificatori universali

$(\forall x\ A) \vee B$ diventa $\forall x\ (A \vee B)$

$(\forall x\ A) \wedge B$ diventa $\forall x\ (A \wedge B)$

SEI: Forma normale congiuntiva (congiunzione di disgiunzioni di letterali)

$A \vee (B \wedge C)$ diventa $(A \vee B) \wedge (A \vee C)$

SETTE: Notazione a clausole

$\{Animale(F(x)), Ama(G(x), x)\}$

OTTO: Separazione delle variabili: clausole diverse, variabili diverse

Nota: $\forall x\ (P(x) \wedge Q(x)) \Leftrightarrow \forall x_1\ P(x_1) \wedge \forall x_2\ Q(x_2)$

$\{Animale(F(x)), Ama(G(x), x)\} \rightarrow \{Animale(F(x_1)), Ama(G(x_1), x_1)\}$

$\{\neg Ama(x, F(x)), Ama(G(x), x)\} \rightarrow \{\neg Ama(x_2, F(x_2)), Ama(G(x_2), x_2)\}$

Algoritmo di Unificazione (AIMA)

```
function Unify(x, y, θ =vuoto) returns una sostituzione che rende x e y identici, o fallimento
    if θ = fallimento then return fallimento
    else if x = y then return θ                                  // caso di successo
    else if Variabile?(x) then return Unify-Var(x, y, θ)
    else if Variabile?(y) then return Unify-Var(y, x, θ)
    else if Composta?(x) and Composta?(y) then                   // es. Op(F(A,B)) = F   Args(F(A,B)=(A,B)
        return Unify(Args(x), Args(y), Unify(Op(x), Op(y), θ))
    else if Lista?(x) and Lista?(y) then
        return Unify(Resto(x), Resto(y), Unify(Primo(x), Primo(y), θ))
    else return fallimento
```

Esempio 1: P(A, y, z) e P(x, B, z)

- UNIFY(P(A, y, z), P(x, B, z), { })
- UNIFY((A, y, z), (x, B, z), UNIFY(P, P, { }))
- UNIFY((A, y, z), (x, B, z), { })
- UNIFY((y, z), (B, z), UNIFY(A, x, { }))
- UNIFY((y, z), (B, z), UNIFY(x, A, { }))
- UNIFY((y, z), (B, z), UNIFY-VAR(x, A, { }))
- UNIFY((y, z), (B, z), {x/A})
- UNIFY((z), (z), {y/B, x/A})
- UNIFY(z, z, {y/B, x/A})
- {y/B, x/A}

Esempio 2: P(f(x), x) e P(z, z)

- UNIFY(P(f(x), x), P(z, z), { })
- UNIFY((f(x), x), (z, z),UNIFY(P, P, { }))
- UNIFY((f(x), x), (z, z), { })
- UNIFY((x), (z), UNIFY(f(x), z, { })
- UNIFY((x), (z), UNIFY-VAR(z, f(x), { }))
- UNIFY((x), (z), {z/f(x)})
- UNIFY-VAR(x, z, {z/f(x)})
- UNIFY(x, f(x), {z/f(x)})
- OCCUR-CHECK(x, f(x))
- *fallimento*

if {*x*/*val*} ∈ θ per qualche *val* **then return** Unify(*var*, *val*, θ)

CLAUSOLE DI HORN
Una clausola di Horn è una disgiunzione di letterali che contiene al massimo un letterale positivo(non negato)

Clausole Horn definite: esattamente un letterale positivo

$$\neg P_1 \vee \dots \vee \neg P_k \vee Q$$
$$\neg(P_1 \wedge \dots \wedge P_k) \vee Q$$

$\{Q, \neg P_1, \dots, \neg P_k\}$ $(k \geq 0)$

Se la KB contiene solo clausole Horn definite i meccanismi inferenziali sono molto più semplici, il processo molto più "guidato", senza rinunciare alla completezza.

Meccanismi
I metodi di inferenza usati nei sistemi basati su regole e nella programmazione logica sono di due tipi

Inferenza all'indietro (backward chaining)
Lavora a ritroso per trovare le premesse che supportano la conclusione, data una conclusione si cerca di dimostrare le premesse

Inferenza in avanti (forward chaining)
Inizia con le premesse e procede verso le conclusioni. Si continua fino a quando non si raggiunge un obiettivo o non si possono fare più inferenze.

Programma logico (PL)
I programmi logici sono KB costituiti di clausole Horn definite espressi come fatti e regole.

Regole: implicazioni logiche che descrivono come i fatti sono in relazione tra loro.
A è vero se tutte le B sono vere.

$A :- B_1, B_2, \ldots, B_n.$ regola, con *testa* A, il conseguente

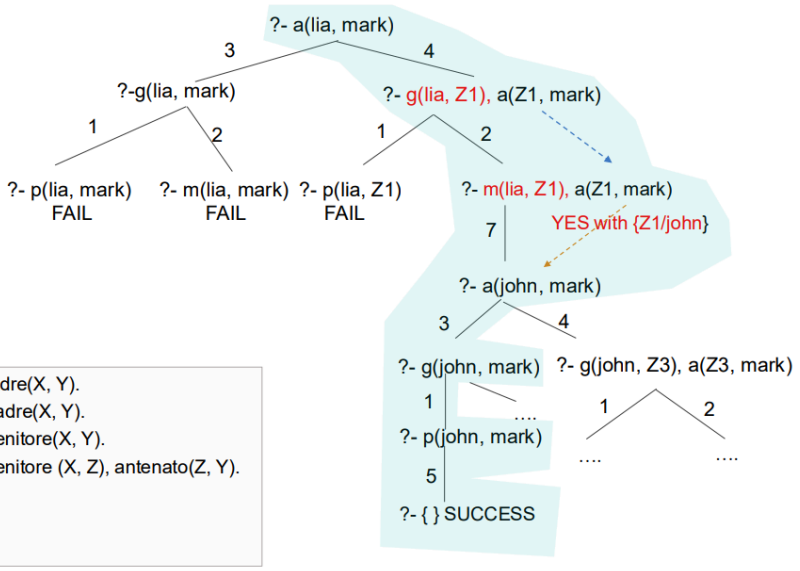
Altre convenzioni: in PL le variabili sono indicate con lettere maiuscole, le costanti con lettere minuscole

Risoluzione Selection Linear Definite Clauses

- la risoluzione SLD è completa per clausole Horn.
- A partire da un programma P e da un goal G si costruisce l'albero di risoluzione

Dato un programma logico P, l'albero SLD per un goal G è definito come segue:

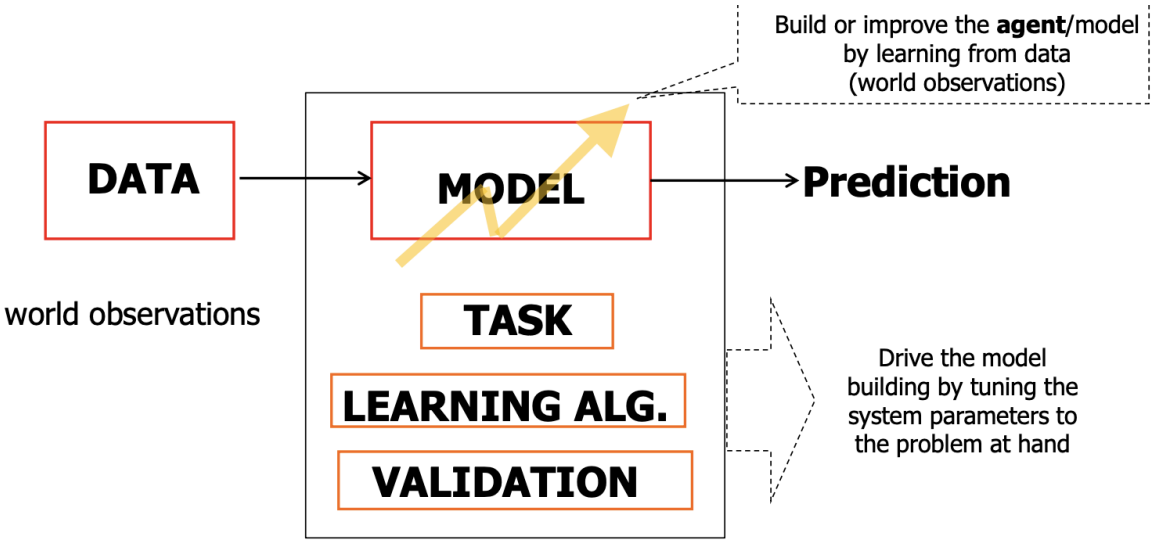
- ogni nodo dell'albero corrisponde a un goal [congiuntivo]
- la radice è $:- G_1, G_2, \ldots, G_k$, il goal di partenza
- sia $:- G_1, G_2, \ldots, G_k$ un nodo dell'albero; il nodo ha tanti discendenti quanti sono i fatti e le regole in P la cui testa è unificabile con G_1
  se $A :- B_1, \ldots, B_k$, e A è unificabile con G_1 con γ = MGU(A, G_1)
  un discendente è il goal $:- (B_1, \ldots, B_k, G_2, \ldots, G_k)\gamma$
- i nodi che sono clausole vuote sono *successi (tutti i sotto-goal possono essere risolti)*
- i nodi che non hanno successori sono *fallimenti (quache sotto-goal non puo' essere risolto)*



1. genitore(X, Y) :– padre(X, Y).
2. genitore(X, Y) :– madre(X, Y).
3. antenato(X, Y) :– genitore(X, Y).
4. antenato(X, Y) :– genitore (X, Z), antenato(Z, Y).
5. padre(john, mark).
6. padre(john, luc).
7. madre(lia, john).

ML - Introduction

ML predictive system



Modello: lo scopo del modello è trovare le relazioni intrinseche tra i dati. Rappresenta la **classe di funzioni** che l'algoritmo può implementare (lineare o no) e come interpretare i dati di input. Le componenti fondamentali di un modello sono:

- vettore delle caratteristiche: input o feature
- parametri **w**: pesi
- funzione target f
- ipotesi h funzione proposta dal modello

ML: Learning as an approximation of an unknown function from examples. Data una funzione sconosciuta f, trova una buona approssimazione di f.

Supervised learning: viene dato un training example <input, output> = (x, d) di dati etichettati.

- Classification: f(x) return the (assumed) correct class for x f(x) is a discrete-valued function {1,2,…,k} classes
- Regression: real continuous output values approximate a real-valued target function, in R or R^k

Unsupervised learning: viene dato un training set senza etichette <input> = <x>

- classifica e organizza su caratteristiche comuni per cercare di fare previsioni su input futuri

Learning Algorithm: basandosi su un set di dati, un task ed un modello ricerchiamo la migliore ipotesi nello spazio delle ipotesi.

ML - concept learning

Concept Learning

Concept Learning → costruire una funzione booleana h(x) attraverso gli esempi di training positivi (TRUE) e negativi (FALSE).

In generale quando dobbiamo inferire una funzione booleana non possiamo mapparle tutte. Infatti abbiamo |N| = # funzioni booleane possibili su n letterali come 2^{2^n}. Quindi con gli esempi di training riusciamo a eliminarne un pò ma rimangono comunque un numero troppo grande da esplorare in totale.

In generale dobbiamo lavorare con uno spazio delle ipotesi più piccolo dell'insieme delle ipotesi totali. In particolare, vedremo:
Regole congiuntive → in uno spazio finito H.
Linear Functions → se lavoriamo su uno spazio continuo.

Il Problema EnjoySport

Il Concept Learning per EnjoySport consiste nel mappare lo Spazio delle Istanze **X** (giorni definiti dai 6 attributi Sky, Temp, Humidity, Wind, Water, Forecast), verso una Target Function **c** che restituisce un valore binario {0,1}.

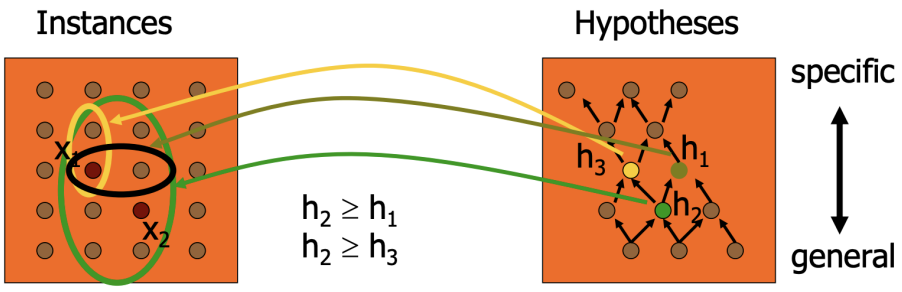
Lo Spazio delle Ipotesi **H** contiene tutte le possibili congiunzioni di letterali, dove ogni attributo può assumere un valore specifico, un simbolo "don't care" ? (massima generalità) o l'insieme vuoto ∅ (massima specificità).

Dato un insieme di Training Examples D composto da coppie di esempi positivi e negativi, l'obiettivo dell'apprendimento è trovare un'ipotesi h all'interno di H tale che h(x) = c(x) per ogni istanza nel dataset.

In termini computazionali, il processo di Learning viene interpretato come una ricerca euristica all'interno dello spazio delle ipotesi, ordinato dal più generale al più specifico.

Possiamo inoltre introdurre un ordine parziale nell'insieme delle ipotesi in generale possiamo dire che h_1 è più generale di (>=) h_2 se

per ogni x di X : [$h_2(x) = 1$ allora $h_1(x) = 1$]



x_1=< Sunny,Warm,High,Strong,Cool,Same>

x_2=< Sunny,Warm,High,Light,Warm,Same>

h_1=< Sunny,?,?,Strong,?,?>

h_2=< Sunny,?,?,?,?,?>

h_3=< Sunny,?,?,?,Cool,?>

Algoritmi per l’addestramento

Find-s

Il suo obiettivo è trovare l'ipotesi più **specifica** possibile all'interno di uno spazio delle ipotesi (che sia coerente con tutti gli esempi positivi forniti nell'addestramento).

1. Initialize h to the most specific hypothesis in H
2. For each **positive** training instance x
   - For each attribute a_i in h
     If the a_i in h is satisfied by x
     then do nothing
     else replace a_i in h by the next more general constraint that is satisfied by x *[e.g. remove from h literals not satisfying x]*
3. Output hypothesis h

1) Inizializzazione: Si imposta l'ipotesi h come la più specifica in H. In un sistema a congiunzioni, questa è solitamente indicata come <Ø, Ø, Ø,Ø,Ø,Ø> .
2) Iterazione sugli esempi:
Per ogni esempio di addestramento x:

- **Se l'esempio è negativo**: L'algoritmo lo ignora completamente.
- **Se l'esempio è positivo (x+)**: Controlla se l'ipotesi attuale h è già soddisfatta da x altrimenti l'algoritmo modifica h sostituendo ogni attributo non soddisfatto con il vincolo immediatamente più generale che lo includa (quindi scorre l’albero degli esempi).

3) Risultato: Al termine degli esempi, l'algoritmo restituisce l'ipotesi h finale.

Esempio

Inizio: h = <Ø, Ø, Ø,Ø,Ø,Ø>

Esempio 1 (Positivo): <Sunny, Warm, Normal, Strong, Warm, Same>.
L'ipotesi diventa l'esempio stesso: h1 = <Sunny, Warm, Normal, Strong, Warm, Same>.

Esempio 2 (Negativo): <Rainy, Cold, High, Strong, Warm, Change>.
Find-S ignora questo esempio e l'ipotesi resta invariata.

Esempio 3 (Positivo): <Sunny, Warm, High, Strong, Warm, Same>.
Qui l'attributo Humid è diverso (Normal in h1 vs High nell'esempio). Find-S generalizza quell'attributo a ?.

Nuova ipotesi: h2 = <Sunny, Warm, ?, Strong, Warm, Same>.

Proprietà

- Lo spazio delle ipotesi è rappresentato come congiunzione di attributi (molto limitativo)
- L'algoritmo darà in output l'ipotesi più specifica, nello spazio H, che è consistente con gli esempi positivi del training set.
- L'ipotesi di output h sarà anche coerente con gli esempi negativi, a condizione che il concetto target sia contenuto in H.

Svantaggi

- Non ci dice se la funzione data converga con il target, non è in grado di determinare se ha trovato l'unica ipotesi coerente con gli esempi di allenamento.
- Non dice quando i dati di allenamento sono incoerenti, in quanto ignora gli esempi di allenamento negativi.

Version spaces
Il Version Space $\mathbf{VS_{H,D}}$, rispetto ad uno spazio delle ipotesi H, e un training set D, è il sottoinsieme delle ipotesi di H consistente con tutti gli esempi di D.

$$\mathrm{VS}_{H,D} = \{h \in H \mid \mathrm{Consistent}(h, D)\}$$

List-Then Eliminate Algorithm

L'algoritmo List-Then-Eliminate è un metodo concettuale per identificare lo Spazio delle Versioni (Version Space), ovvero l'insieme di tutte le ipotesi coerenti con i dati di addestramento forniti.

Versione Space
Insieme delle ipotesi consistenti con i dati di training.

1. *VersionSpace* ← a list containing every hypothesis in H
2. For each training example <x,c(x)>
   remove from *VersionSpace* any hypothesis that is inconsistent with the training example, i.e. h(x) ≠ c(x)
3. Output the list of hypotheses in *VersionSpace*

1) Inizializzazione: Viene creata una lista che contiene inizialmente **ogni possibile ipotesi** presente nello spazio delle ipotesi H.

2) Eliminazione: Per ogni esempio di addestramento <x, c(x)> (sia positivo che negativo), l'algoritmo controlla ogni ipotesi nella lista. Se un'ipotesi h non è coerente con l'esempio (ovvero se la previsione h(x) è diversa dal valore reale c(x)), tale ipotesi viene **rimossa** dalla lista.

3) Risultato: Al termine del processo, l'algoritmo restituisce la lista delle ipotesi rimaste, che costituiscono appunto il *Version Space*.

Algoritmo Candidate Elimination

$G \leftarrow$ maximally general hypotheses in H
S $\leftarrow$ maximally specific hypotheses in H
For each training example d=<x,c(x)>

If d is a positive example
Remove from G any hypothesis that is inconsistent with d (def. of VS)
For each hypothesis s in S that is not consistent with d
[generalize S]

- remove s from S.
- Add to S all minimal generalizations h of s such that
  - h consistent with d
  - Some member of G is more general than h ($\leftarrow$*)
- Remove from S any hypothesis that is more general than another hypothesis in S

Confine S (Specifico): L'insieme delle ipotesi massimamente specifiche coerenti con i dati.
Confine G (Generale): L'insieme delle ipotesi massimamente generali coerenti con i dati.
Proprietà: Ogni ipotesi che si trova "tra" questi due confini ($g \geq h \geq s$) fa parte del Version Space.

L'algoritmo inizializza S con l'ipotesi più specifica (Ø) e G con la più generale (?). Poi elabora ogni esempio d = <x, c(x)>:

- **Se l'esempio è POSITIVO:**
  - Rimuove da G le ipotesi inconsistenti con d.
  - Generalizza S: per ogni s in S inconsistente, lo sostituisce con le sue generalizzazioni minime che siano coerenti con d e più specifiche di qualche elemento in G.
- **Se l'esempio è NEGATIVO:**

- Rimuove da S le ipotesi inconsistenti con d.
- Specializza G: per ogni g in G inconsistente, lo sostituisce con le sue specializzazioni minime che siano coerenti con d e più generali di qualche elemento in S.

Esempio

1) **inizializzazione**
2) **Esempio x1** (Sunny, Warm, Normal, Strong, Warm, Same) [+]:
    a) Poiché è positivo, l'ipotesi in S deve essere generalizzata per coprirlo. Diventa quindi l'esempio stesso: {<Sunny, Warm, Normal, Strong, Warm, Same>}
    b) G rimane invariato perché l'ipotesi generale è già coerente con questo esempio.
3) **Esempio x2** (Sunny, Warm, High, Strong, Warm, Same) [+]**:**
    a) S deve ora coprire sia x1 che x2. L'unica differenza tra i due è l'attributo Humidity (Normal vs High), che viene quindi generalizzato con un "?"
    b) Nuovo S: {<Sunny, Warm, ?, Strong, Warm, Same>}.
4) **Esempio x3** (Rainy, Cold, High, Strong, Warm, Change) [-]:
    a) Questo esempio è negativo, quindi dobbiamo specializzare G affinché escluda x3 ma continui a coprire gli elementi in S
    b) L'algoritmo cerca le specializzazioni minime di <?,?,?,?,?,?> che non siano coerenti con x3. Vengono generate tre ipotesi in G:
        i) <Sunny, ?, ?, ?, ?, ?> (perché in x3 il cielo è Rainy).
        ii) <?, Warm, ?, ?, ?, ?> (perché in x3 la temperatura è Cold).
        iii) <?, ?, ?, ?, ?, Same> (perché in x3 il forecast è Change).

Inductive Bias

Un algoritmo di apprendimento, per generalizzare dai dati osservati a casi nuovi, deve fare alcune assunzioni implicite sul mondo. Per questo viene usato l'inductive bias cioè l'insieme delle preferenze, assunzioni o vincoli che guidano l'apprendimento verso certe ipotesi invece di altre.

Generalizzazione = Dati + Bias

L'apprendimento senza bias è inefficiente perché ci costringe ad esplorare tutto lo spazio delle ipotesi e non ci permette di generalizzare dagli esempi per predire il risultato di una nuova istanza.

- Algoritmo di apprendimento mnemonico (tabella di consultazione): memorizza esempi, classifica x se e solo se corrisponde a un esempio osservato in precedenza.

    - Nessun bias induttivo → nessuna generalizzazione

- Algoritmo Candidate Elimination (CE) nel Version Space (VS).
    - Bias: lo spazio delle ipotesi contiene il concetto target (congiunzione di attributi) |H|= 973 contro 1028
- Find-S
    - Bias: lo spazio delle ipotesi contiene il concetto target e tutte
      le istanze sono istanze negative a meno che il contrario non sia implicato da altre conoscenze (viste come esempi positivi).

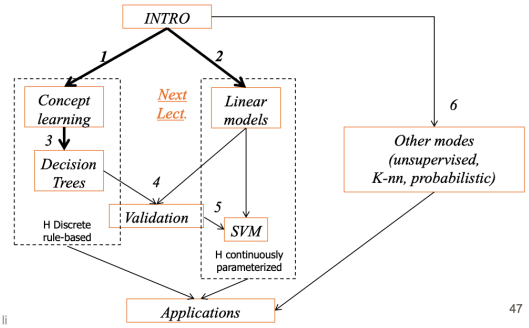
      In altre parole: abbiamo un bias linguistico dovuto all'AND sui letterali più il bias di ricerca dovuto alla preferenza dell'ipotesi più specifica

ML - linear functions

D → dati di allenamento presi di esempio.
H → spazio delle ipotesi, costituisce l'insieme delle funzioni che possono essere realizzate dal sistema di apprendimento.
Alg. di apprendimento → Alg. di ricerca nello spazio delle ipotesi.

L'obbiettivo è quello di approssimare minimizzando l'errore una certa funzione ***f*** che rappresenta i reali valori di output.

Nota bene → **H** non può coincidere con l'insieme di tutte le funzioni possibili e la ricerca essere esaustiva (Bias Induttivo).

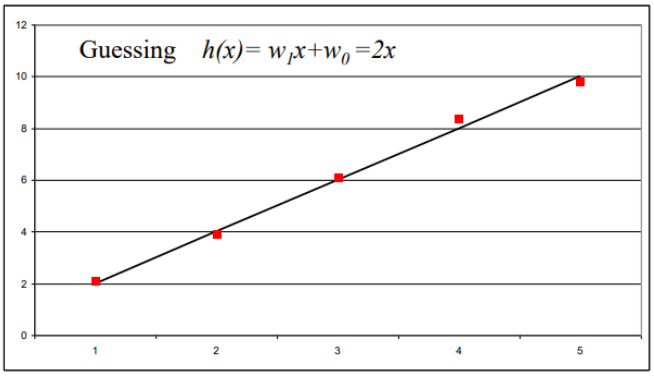


Nel concept learning lo spazio delle ipotesi rappresenta uno spazio discreto, andiamo adesso a vedere come possiamo elaborare un alg. di apprendimento in uno **spazio continuo**, come vedremo di seguito l'obbiettivo del nostro alg. di apprendimento sarà quello di trovare i migliori pesi per approssimare ***f***. Trattandosi i pesi di variabili continue lo spazio delle ipotesi sarà quindi infinito.

Liner Model

Regressione
Processo di stima di una funzione a valori reali sulla base di un insieme finito di campioni. Andiamo ad approssimare la funzione ***f*** con la retta di regressione. Andando a stimare i valori non noti con una funzione $h(x) = w_1x + w_0$ dove i parametri w_1, w_0 rappresentano i pesi con la quale costruiamo la retta che meglio approssima agli esempi noti.

x	*target*
1	2.1
2	3.9
3	6.1
4	8.4
5	9.8
...	...



Regressione Lineare Univariata

Metodo statistico col quale cerchiamo la miglior interpretazione dei parametri liberi assumendo come modello ***h(x) = w_0 + w_1x + loss***.

dove loss rappresenta l'errore (rumore) della nostra retta rispetto ai valori reali di output.

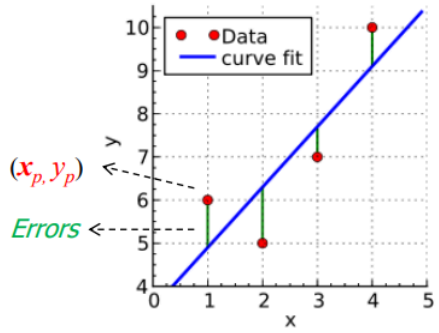
Least Mean Square
Dobbiamo scegliere i parametri w_0,w_1 della nostra h(x) per questo definiamo l'errore di **Least Mean Square** (**loss**) come:

la funzione **E(w)** dove **w** rappresenta il vettore per la stima dell'errore come:

$$Loss(h_{\mathbf{w}}) = E(\boldsymbol{w}) = \sum_{p=1}^{l} (y_p - h_{\mathbf{w}}(x_p))^2 = \sum_{p=1}^{l} (y_p - (w_1 x_p + w_0))^2$$

error — *p runs over patterns/examples*

Nella sommatoria facciamo la sottrazione del valore reale con il valore approssimato dalla funzione ottenendo così la somma di tutte le distanze dei punti dalla retta di regressione



$y = w_1 x + w_0 + noise$

Different blue lines will have different green bars.
Minimizing the green bars (residuals /errors)
is a way to find the best approximation/fitting of the data
(i.e. our $h_w(x)$ or blue line).
*The squares of errors E(**w**) quantify such green bars:*

$$E(w) = \sum_{p=1}^{l} (y_p - h_w(x_p))^2$$

Come risolverlo

Il minimo locale è un punto stazionario dove il gradiente è nullo.

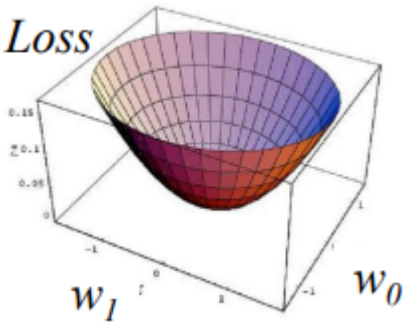
$$\frac{\partial E(\mathbf{w})}{\partial w_i} = 0, \qquad i = 1,\ldots,\dim\_input + 1$$

Per la regressione lineare semplice abbiamo che la funzione di Loss è minimizzata quando le sue derivate parziali rispetto a w_0 e w_1 sono 0:

Search the w such that $\quad \frac{\partial E(\mathbf{w})}{\partial w_0} = 0 \qquad \frac{\partial E(\mathbf{w})}{\partial w_1} = 0$

Se la funzione di perdita é convessa abbiamo la seguente soluzione diretta poiché non abbiamo nessun minimo locale.

Per w_1 e w_0 abbiamo che:



$$w_1 = \frac{Cov[x,y]}{Var[x]}\ ,\ w_0 = \frac{\sum_{p=1}^{l} y_p - w_1 \cdot \sum_{p=1}^{l} x_p}{l}$$

Calcolo delle derivate parziali di w_0 e w_1.

Hence we omit p for x

Basic rules:

$$\frac{\partial}{\partial w}k = 0,\ \frac{\partial}{\partial w}w = 1,\ \frac{\partial}{\partial w}w^2 = 2w$$

$$\frac{\partial (f(w))^2}{\partial w} = 2f(w)\frac{\partial (f(w))}{\partial w}$$

Der. sum = sum of der.

We will call $(y-h(x))$ "delta"

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial (y - h_{\mathbf{w}}(x))^2}{\partial w_i} =$$

$$= 2(y - h_{\mathbf{w}}(x))\frac{\partial (y - h_{\mathbf{w}}(x))}{\partial w_i} = 2(y - h_{\mathbf{w}}(x))\frac{\partial (y - (w_1 x + w_0))}{\partial w_i}$$

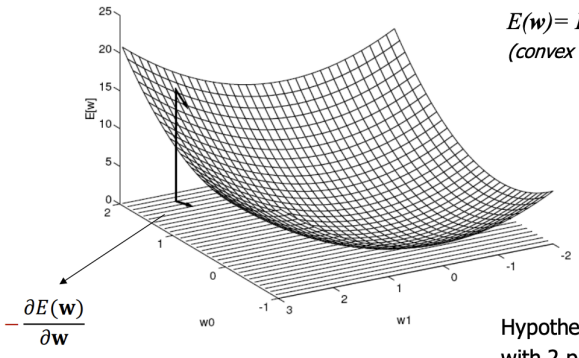
$$\frac{\partial E(\mathbf{w})}{\partial w_0} = -2(y - h_{\mathbf{w}}(x)) \qquad \frac{\partial E(\mathbf{w})}{\partial w_1} = -2(y - h_{\mathbf{w}}(x)) \cdot x$$

Hill Climbing Iterativo

Se la funzione è continua e differenziabile. Il minimo o massimo si può cercare utilizzando il gradiente, che restituisce la direzione di massima pendenza nel punto.

Hill climbing iterativo → x_{new} = x + **eta** * gradiente di f (Noi lo applichiamo al vettore dei pesi **w**)

eta = lunghezza del passo (**learning rate**).



Gradient of E(w)
Our "compass" to find the minimum

Parabolic for the
$E(\mathbf{w}) = E([w_0, w_1]^T)$
(convex quadratic functior

Hypothesis space
with 2 parameters
w_0, w_1

Gradiente discendente

Il gradiente ci indica la direzione di salita possiamo spostarci verso il minimo con discesa del gradiente Δw = -delta E(w). La ricerca locale inizia con il vettore che contiene i valori dei pesi, viene modificato iterativamente per diminuire fino a minimizzare l'errore della funzione.

Delta rule: Δw_i = eta * sommatoria[(Target - Output)^2] * la derivata parziale rispetto a w_i

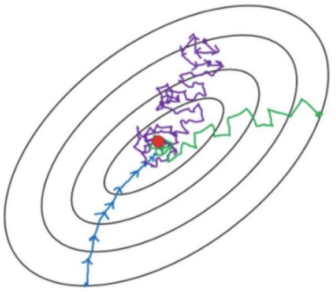
Questa formula ci indica quanto e come correggere il tiro per una certa w_i l'assegnazione del nuovo valore $w_{i\text{-new}} = w_{i\text{-old}} + \Delta w$

discussioni finali
L'approccio della discesa del gradiente è un metodo di ricerca locale semplice ed efficace per la soluzione LMS.

- Permette di esplorare uno spazio di ipotesi infinito!
- Può essere facilmente applicato a H continuo e a funzioni di perdita differenziabili: NON SOLO ai modelli lineari!!!! (ad esempio, reti neurali e modelli di deep learning)
- Efficiente? Sono possibili molti miglioramenti, ad esempio i metodi di Newton Gradiente coniugato, …

Estendiamo questo metodo per tutti gli ***l*** patterns di training dati al modello:

Batch algorithm → Linea blu
Calcoliamo la sommatoria dell'errore quadratico su l pattern diversi e otteniamo il gradiente, dopo i dati di training sommati abbiamo una "epoca" e aggiorniamo i w_i . Puó essere molto lento.



On-line algorithm → Linea verde e viola
Calcoliamo il gradiente su un pattern p e poi aggiorniamo subito w_i.
Solitamente é piú veloce del Batch.

Notazione:

Pattern	x_1	x_2	x_i	x_n
Pat 1	$x_{1,1}$	$x_{1,2}$		$x_{1,n}$
…				
Pat *p*	$x_{p,1}$	$x_{p,2}$	$x_{p,i}$	$x_{p,n}$
…				

$\boldsymbol{X}$ is a matrix $\boldsymbol{l \times n}$

$\boldsymbol{l}$ rows, $\boldsymbol{n}$ columns
(features, variables, attributes)

$p=1..l, \quad i=1..n$

We often need to omit some indices when the context is clear, e.g.:

- Each row, generic $\boldsymbol{x}$ (vector - bold), a raw in the table: (input) example, pattern, instance, sample, …, input vector, ...
- x_i or x_j (scalar): component *i* or *j* (given a pattern, i.e. omitting *p*)
- $\boldsymbol{x}_p$ (or $\boldsymbol{x}_i$) (vector – bold) *p-th* (or *i-th*) raw in the table = pattern *p* (or *i)*
- $x_{p,i}$ (scalar) also as $(\boldsymbol{x}_p)_i$: component *i* of the pattern *p*
  (or we use $x_{p,j}$ for the component *j, etc.*)

Input multidimensionali
L'output sarà quindi dato dalla formula:

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w_0 + \sum_{i=1}^{n} w_ix_i$$

$$\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{w}$$

W_0 viene anche chiamato intercept, threshold, bias.

$$\boldsymbol{x}^T = [1, x_1, x_2, \ldots, x_n]$$
$$\boldsymbol{w}^T = [w_0, w_1, w_2, \ldots, w_n]$$

$$h(\boldsymbol{x}_p) = \boldsymbol{x}_p{}^T\boldsymbol{w} = \sum_{i=0}^{n} x_{p,i} w_i$$

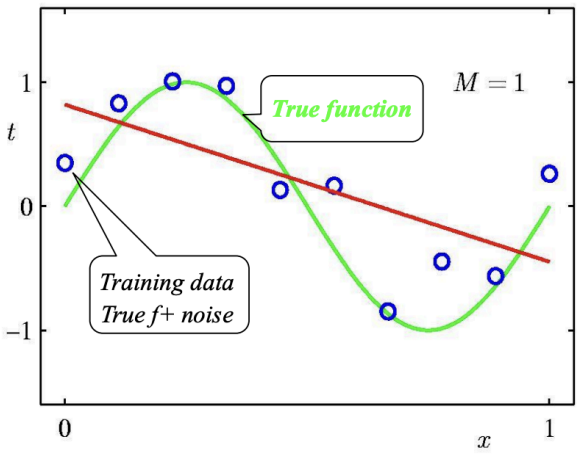
Dobbiamo quindi minimizzare la funzione:

$$E(\mathbf{w}) = \sum_{p=1}^{l} (y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})^2 = \quad ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2$$

La nuova formula per calcolare la direzione in uno spazio n-dimensionale è

$$\Delta\boldsymbol{w} = -\nabla E(\boldsymbol{w}) = -\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \begin{bmatrix} -\frac{\partial E(\boldsymbol{w})}{\partial w_0} \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_1} \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_2} \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_j} \\ \cdots \\ -\frac{\partial E(\boldsymbol{w})}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \Delta w_0 \\ \Delta w_1 \\ \Delta w_2 \\ \Delta w_j \\ \cdots \\ \Delta w_n \end{bmatrix}$$

Estendiamo i modelli lineari per funzioni non lineari
Possiamo trasformare la nostra funzione $h_w(x)$ in una funzione polinomiale in **x**, mantenendo la linearità per i parametri liberi **w**.



Too poor solution

Per rappresentare una funzione non lineare abbiamo bisogno di una trasformazione.

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

polynomial regression

Dove la **M** è il numero di pesi moltiplicati per le trasformazioni dell'input

Linear Basis Expansion (LBE)

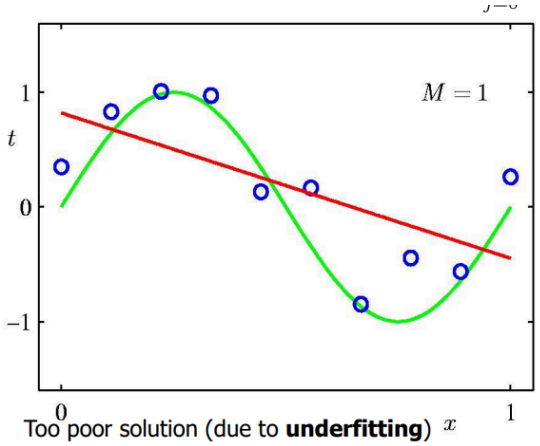
$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{k=0}^{K} w_k \phi_k(\boldsymbol{x})$$

Manteniamo i parametri liberi lineari mantenendo la somma tra di loro, andiamo a **trasformare** la x in input attraverso la funzione ϕ_k che può essere una qualsiasi funzione non lineare (es. log(x), x^n).

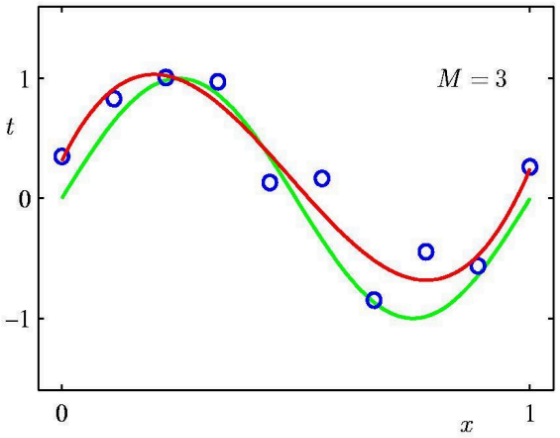
Basis Expansion criticism

1) Dobbiamo scegliere quale ϕ utilizzare
2) Dobbiamo controllare la complessità del modello.

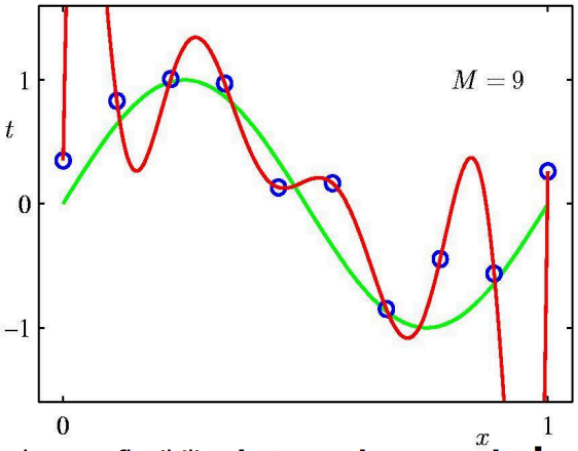
esempi



Too poor solution (due to **underfitting**)



More flexibility is useful !!!



Much more flexibility, **but may be excessive!**

E(**w**)= 0 on training data!!! But error on test set ?

Too complex model: fit the noise!

Poor representation of the (green) true function, due to **overfitting**!

Underfitting: la funzione non definisce correttamente o in maniera troppo approssimativa l'andamento degli esempi.
Overfitting: anche se la funzione $E(h_w)$ è uguale a 0 avremo comunque una funzione che non rappresenta correttamente nuovi input.

Regolarizzazione
La regolarizzazione è la capacità dell'algoritmo di penalizzare l'overfitting pur mantenendo una buona flessibilità della funzione. La più semplice spiegazione è anche la più corretta (vecchio saggio).

Regolarizzazione by Ridge Regression / Tikhonov regularization

def. E' possibile aggiungere vincoli alla somma dei valori di w_j favorendo modelli "sparsi", ad esempio con meno termini dovuti ai pesi wj = 0 (o vicino a 0).

Error data term + Regularization/penalty term

$$Loss(h_w) = \sum_{p=1}^{l} (y_p - h_w(\boldsymbol{x}_p))^2 + \lambda \| \boldsymbol{w} \|^2 \rightarrow \sum_i w_i^2$$

$$\boldsymbol{w}_{new} = \boldsymbol{w} + eta * \Delta \boldsymbol{w} - 2\lambda \boldsymbol{w}$$

Il parametro lambda è la manopola che decide l'importanza che diamo al penalty term.
Low lambda → overfitting
High lambda → underfitting

Se scegliamo un lambda gigantesco, stiamo obbligando l'algoritmo a concentrarsi quasi solo sul mantenere il modello semplice, al punto da ignorare gli errori commessi sui dati reali; questo genera una curva troppo rigida che non impara il vero andamento, cadendo nell'**underfitting**.

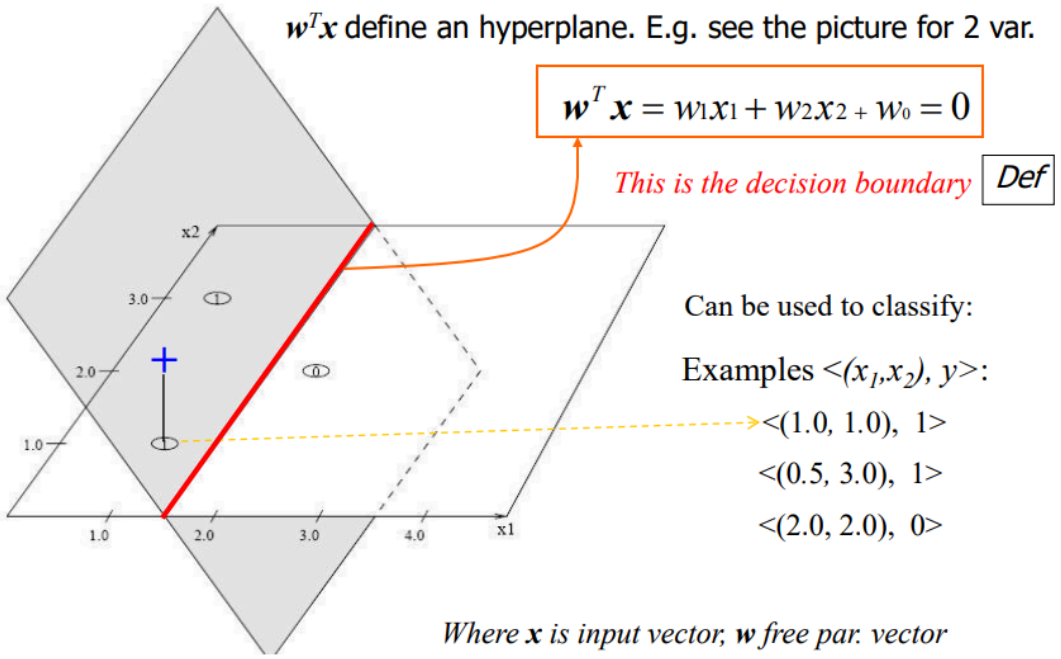
Il cuore del problema è proprio il **trade-off**: dobbiamo calibrare attentamente la manopola di lambda per trovare il bilanciamento perfetto tra un modello che segue bene i dati ma resta abbastanza flessibile da fare ottime previsioni su esempi futuri mai visti.

Limitazioni di **Fixed Basis Functions**

- Il limite delle funzioni fisse: Se usiamo metodi come i polinomi su tanti input, il numero di combinazioni possibili esplode, creando troppe nuove variabili.
- La maledizione della dimensionalità: Avendo troppe dimensioni, i dati si "disperdono" nello spazio matematico.
- Per far imparare il modello senza che vada in overfitting, servirebbe una quantità enorme (esponenziale) di dati.
- Il problema della scelta a priori: Le funzioni di base classiche vengono decise "alla cieca" dall'umano, prima ancora che il modello veda i dati.

Classificazione

Gli stessi modelli (utilizzati per la regressione) possono essere utilizzati per la classificazione: Creare modelli che classificano gli input in output positivo o negativo. Usiamo un iperpiano $w^T * x$ che rappresenterà la nostra linea di confine che divide la zona positiva da quella negativa. Sfruttiamo questi modelli per decidere se un punto x appartiene alla zona positiva o negativa dell'iperpiano. Quindi vogliamo impostare il vettore w in modo tale da ottenere una buona precisione di classificazione.



Quindi possiamo definire una funzione di classificazione del tipo:

$$h(x) = \begin{cases} 1 & \text{if } w^T x + w_0 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

[0,1] output range

Questa funzione può essere espressa anche:

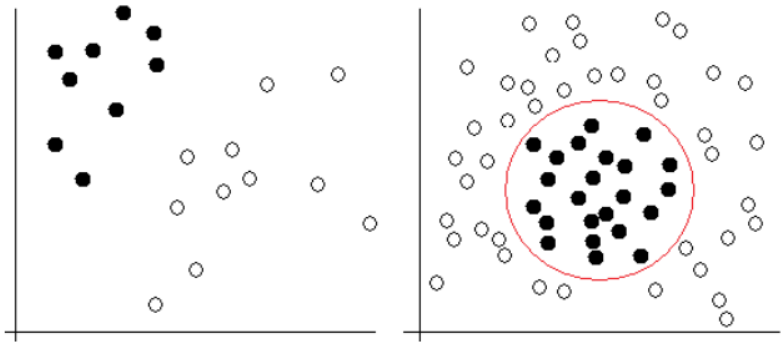
$$h(x) = sign(w^T x + w_0)$$

[-1,+1] output range

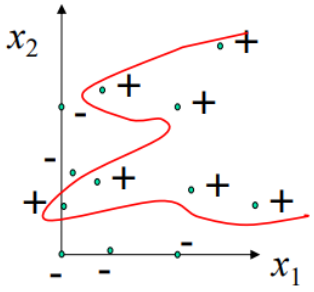
$$h(x_p) = sign(x_p{}^T w) = sign\left(\sum_{i=0}^{n} x_{p,i}\, w_i\right)$$

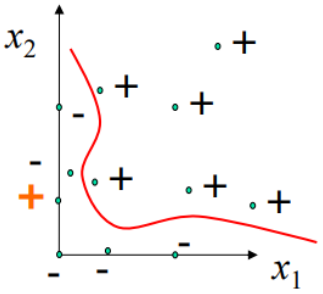
Limitazioni

In geometria, due insiemi di punti in un grafico bidimensionale sono linearmente separabili quando i due insiemi di punti possono essere completamente separati da una singola retta. In generale, due gruppi sono linearmente separabili in uno spazio n-dimensionale se possono essere separati da un iperpiano (n-1)-dimensionale.



Anche in questo caso si possono applicare anche l'espansione in base lineare e la regolarizzazione di Tikhonov.



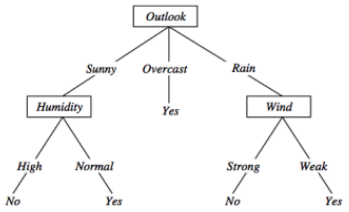


ML -decision tree

Decision Tree: approccio alternativo nel ML

ID3 → Algoritmo che costruisce a partire da un training set un decision tree.

ID3(*X, T, Attrs*) *X*: training examples
T: target attribute (e.g. *PlayTennis*),
Attrs: other attributes, initially all attributes



Create *Root* node
If all X's are +, *return* *Root* with class +
If all X's are –, *return* *Root* with class –
If *Attrs* is empty *return* *Root* with class most common value of *T* in *X*
else
A ← best attribute; decision attribute for Root ← *A*
For each possible value v_i of *A*:
- add a new branch below *Root*, for test $A = v_i$
- X_i ← *subset* of *X* with $A = v_i$
- *If* X_i is empty *then* add a new leaf with class the most common value of *T* in *X*
else add the subtree generated by **ID3**(X_i, *T*, *Attrs* – {*A*})

return *Root*

Note: restricted to subset of examples X_i *and to other attributes*

Best Attribute
La scelta del miglior attributo per la costruzione di un decision tree si basa sul guadagno di informazione dato dai valori che può assumere l'attributo, questo viene misurato attraverso l'entropia.

def. Entropia
Data una collezione di esempi **S**, una proporzione di esempi positivi $\mathbf{p_+}$ e una proporzione di esempi negativi $\mathbf{p_-}$. L'entropia è la misura dell'impurità di un certo insieme di dati.

p_+ = #esempi positivi di Sv / #esempi Sv
p_- = #esempi negativi di Sv / #esempi Sv

$$Entropy(S) = – p_+ * log_2 (p_+) – p_- * log_2 (p_-)$$

L'**Information Gain** la riduzione di entropia *attesa* causata dal partizionamento degli esempi sulla base degli attributi può essere calcolata come segue:

Def $$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|Sv|}{|S|} Entropy(Sv)$$

A → Attributo
Values(A) → valori possibili per A
Sv → sottoinsieme di tutti gli esempi S nei quali A ha un valore *v*

- *Maggiore* è l'Information Gain e più l'algoritmo riesce a classificare i dati di training.
- I valori più omogenei come [14+, 0–] o [0+,7-] permettono una *classificazione chiara*!

- Scegliamo quindi l'attributo **A** che massimizza la funzione di *Gain*

esempi:
Entropy([0+, 6-]) = 0
Entropy([2+, 2-]) = 1

GainRatio
Misura alternativa per il guadagno di informazione, tiene in considerazione anche della ramificazione del ramo. (può voler dire maggior ramificazione = entropia bassa ma poca informazione)

Def $$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

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

S_i sono gli insiemi ottenuti partizionando in base al valore v_i di A, fino a c valori.
SplitInformation misura l'entropia di S rispetto ai valori di A. Quanto più uniformemente dispersi sono i dati, tanto maggiore è il valore.

Bias Induttivo

Bias di ricerca → bias indotto nella strategia di ricerca
- ID3 ricerca uno **spazio completo di ipotesi**, garantito dalla struttura dati ad albero.
- la strategia di **ricerca di ID3 è incompleta**

Bias di linguaggio → bias indotto nella rappresentazione delle informazioni,
- Candidate Elimination ricerca in uno **spazio di ipotesi incompleto**, perché le ipotesi sono delle disgiunzioni del tipo <sunny, hot, high> quindi per natura non sono in grado di esprimere tutto lo spazio delle ipotesi.
- la strategia **di ricerca nel Candidate Elimination è completa.**

Tra i due Bias è preferito quello di ricerca.

Overfitting → alberi che si "adattano troppo" agli esempi di training crescendo di dimensione.

h ∈ H, spazio delle ipotesi overfitta i dati se esiste *h'* ∈ H che si comporta peggio sul Training Set e meglio su dati mai visti prima.

$error_D$ (h) < $error_D$ (h') and $error_X$ (h') < $error_X$ (h)

Mitigare l'overfitting

1. **early stop**
2. post-pruning

Possiamo dividere il Training Set in due parti:

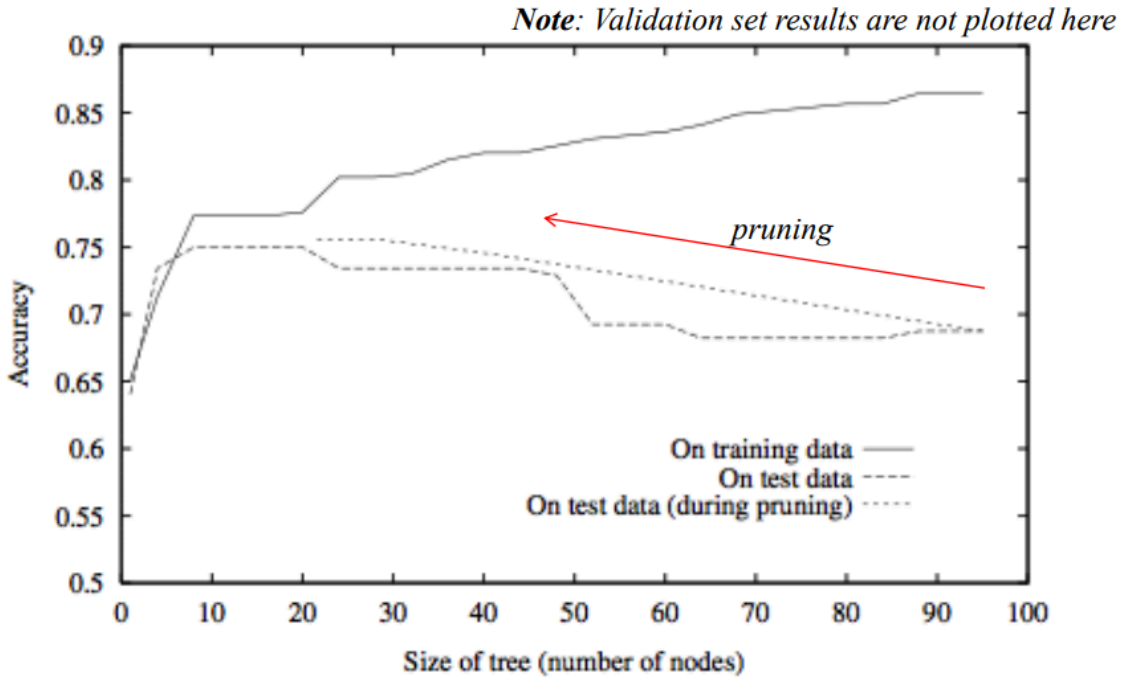
training set: dati per addestrare il modello

validation set: dati per valutare l'addestramento su dati mai visti

Potatura a errore ridotto

Ogni nodo é un candidato per la potatura. **La potatura** consiste nella rimozione di un sottoalbero radicato in un **nodo**, quest'ultimo **diventa una foglia e viene assegnata la classificazione più comune.**

- I nodi vengono rimossi solo se l'albero risultante non ha prestazioni peggiori sul set di validazione.
- I nodi vengono eliminati in modo iterativo. Ad ogni iterazione viene eliminato il nodo la cui rimozione aumenta la precisione del set di validazione.
- La potatura si interrompe quando nessuna potatura aumenta la precisione.



Regole Post potatura

1) Creazione dell'albero di decisione a partire dal Training Set fino a che i dati di training fittano al loro meglio. (Overfitting intenzionale)
2) Conversione dell'albero in un insieme equivalente di regole, più leggibile per gli umani.
    a) Ogni path corrisponde a una regola.
    b) Ogni nodo lungo un path corrisponde a una pre-condizione.
3) Potatura (Generalizzazione) di ogni regola rimuovendo quelle pre-condizioni la cui rimozione migliora l'accuratezza sia sul validation set, sia sul training set.
4) Ordinamento delle regole in ordine di precisione stimata e considerare in sequenza quando si classificano nuove istanze.

Attributi con valori continui → Si imposta un valore di soglia di modo da avere una ramificazione discreta.

Valori di attributi mancanti → vengono scelti dei valori per la loro distribuzione statistica nel Training example.

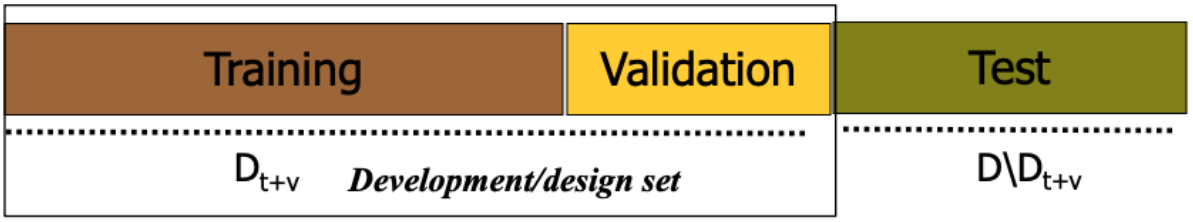
ML - Validation

Validazione

La fase della validazione ha due obiettivi principali:

1. selezione di un modello, seleziona il miglior modello (e.g. polynomial order, lambda of ridge regression, …).
2. valutazione di un modello, valutare le performance quindi l'accuratezza su nuovi dati.

Durante la fase di validazione vedo quanto un modello è in grado di generalizzare su nuovi esempi mai visti prima grazie alla funzione $h_{w,lambda}$.



Se i dati sono sufficienti una possibile proporzione tra i dati è 50% TR, 25% VL, 25% TS insiemi disgiunti.

Training Set → i dati per l'addestramento, usati per fittare i pesi

Validation Set → i dati per la validazione, usati per la selezione del miglior modello (iper-parametri, espansione di basi…).

Test Set → i dati per il test, usati per la stima del rischio.

Grid Search - un semplice meta algoritmo

Questo algoritmo ha lo scopo di selezionare la combinazione ideale degli iperparametri (cioè quelle configurazioni che non vengono apprese dal modello, come avviene per i pesi w, ma selezionate a priori come il grado del polinomio o il coefficiente di regolarizzazione lambda)

- Separate TR (training), VL (validation) and TS (test) sets
- Search best $h_{w,\lambda}()$ changing the model hyper-parameters λ [e.g. the polynomial order, the lambda for ridge regression]:
  - For each different values of λ (external *grid search* loop)
    - Search best $h_{w,\lambda}()$ that minimize error/empirical loss (fitting the TR set) finding the best w parameters (internal *training* loop)

    where best = minimum error on TR set [e.g. $argmin_{\mathbf{w}}\ Loss(\mathbf{w})\ in\ L_2$]
  - Select the best best $h_{w,\lambda}()$: where best = minimum error on the VL set
- (Optional: Now it is also possible to fit $h_{w,\lambda}(x)$ on TR+VL with best λ model)
- Evaluate the final $h_{w,\lambda}(x)$ on the TS

Ottenendo in questo modo una tabella che rappresenta tutte le possibili combinazioni tra gli iperparametri (colonne e righe della tabella). Nelle celle della tabella avremo i risultati delle valutazioni dei modelli.

Hyper-param.	Lambda 0.1	Lambda 0.01	Lambda 0.001
Degree 1	Res1	Res4	Res7
Degree 2	Res2	Res5	Res8
Degree 4	Res3	Res6	Res9

Come operazione opzionale dopo la fase di training e validazione possiamo fittare i pesi w sui dati sia di training

che di validazione così da ottenere un vettore w più preciso.

Infine come ultimo passaggio sottoponiamo il modello vincitore alla fase di test per avere il reale valore delle capacità di generalizzazione del modello.

K-fold cross validation

Invece della classica suddivisione in TR, VL, TS (hold out) possiamo suddividere il dataset D in k sottoinsiemi mutuamente esclusivi: d_1, d_2, d_3, …, d_k

Addestrare l'algoritmo di apprendimento su $D \backslash D_i$ e testarlo su D_i
Riassumere calcolando la media di tutti i risultati su D_i (diagonale)

NOTA: Questa tecnica può essere utilizzata sia per il dataset di validazione che per il dataset di test, utilizza tutti i dati per l'addestramento e la validazione o il test

Statistical Learning Theory

$$R = \int L(d, h(\boldsymbol{x}))dP(\boldsymbol{x}, d)$$

R è la funzione che ci restituisce il rischio che vogliamo minimizzare
L(d, h(x)) → errore empirico calcolato (es. $(d - h(x))^2$)

dP(x, d) è la probabilità che questo esempio si presenti nella vita reale, questa probabilità ovviamente è impossibile da avere perché presupporrebbe di avere l'intero insieme di casi possibili. Per questo usiamo una funzione più semplice per calcolare l'errore empirico sui casi noti.

$$R_{emp} = \frac{1}{l}\sum_{p=1}^{l}(d_p - h(\boldsymbol{x}_p))^2$$

Empirical Risk Minimization (erm) Inductive Principle

VC-Bound

$$R \leq \underbrace{R_{emp}}_{} + \underbrace{\varepsilon(1/l, VC, 1/\delta)}_{\text{VC-confidence}}$$

ε → epsilon o *VC-condidence*, funzione qualsiasi che prende in input *l, VC* e *δ*

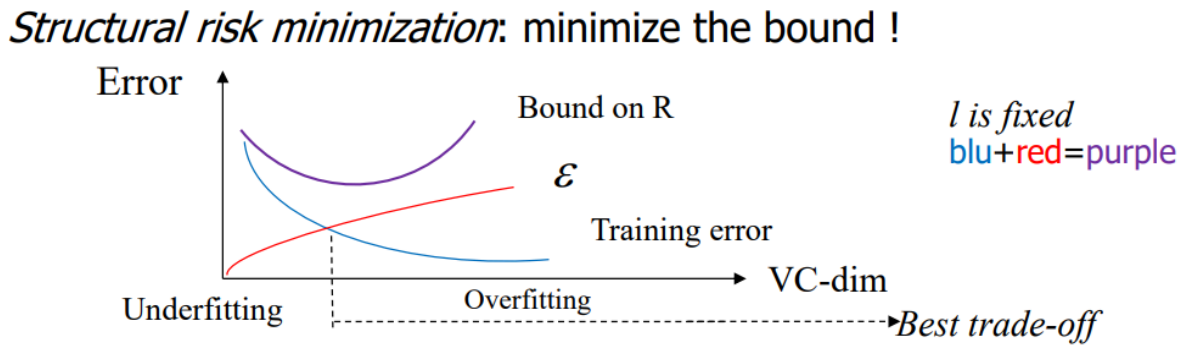
VC → VC-dimension, complessità del modello, numero di parametri, grado del polinomio, λ ecc…
δ → è la “confidence”, probabilità con la quale il “legame regge” (es: un basso delta 0.01 mantiene una probabilità del legame dello 0.99)
L → grandezza del dataset, + dati abbiamo minore è la penalizzazione per un modello complesso

la parte destra della disequazione rappresenta un limite superiore del rischio atteso R, se minimizziamo la parte destra minimizziamo anche il rischio atteso.

Nota: un aumento della VC-dimension corrisponde a un aumento del rischio empirico, che cresce insieme alla complessità del modello.

Dalla formula si nota che:
Se aumento la complessità del modello, minimizzo l’errore calcolato da R_{emp} ma aumento la penalizzazione epsilon modello (per la complessità).
Se uso un modello più “rigido”: aumento R_{emp} ma diminuisco la penalizzazione epsilon.

Structural risk minimization: trovare il miglior compromesso tra complessità ed errore.



1. VC-dim (Vapnik-Chervonenkis Dimension)

- **Cos'è:** È un **singolo numero** che misura la **complessità o flessibilità** intrinseca del tuo modello.
- **Cosa indica:** Misura la capacità del modello di "imparare a memoria" i dati. Più questo numero è alto, più il modello è snodato, potente e complesso.
- **Esempio pratico:** In un modello lineare semplice, la VC-dim è strettamente legata al numero di parametri (i pesi w). In un polinomio, è legata al grado del polinomio. Nelle Support Vector Machine (SVM), è inversamente proporzionale al margine: un margine ampio significa una VC-dim bassa (modello semplice).

2. VC-confidence (La Penalità)

- **Cos'è:** Rappresenta il tuo "margine di *sfiducia*".
- **Cosa indica:** Ci dice di quanto **non** dobbiamo fidarci dell'errore misurato sul Training Set (R_emp). È letteralmente una penalità che viene aggiunta all'errore di addestramento.

- **Da cos'è composta:** La VC-confidence è una funzione che dipende da tre cose: epsilon(1/l, VC, 1/delta).
    - aumentando lambda (mitighi l'overfitting) diminuisce la vc confidence
    - Aumenta se la **VC-dim** è alta.
    - Diminuisce se il numero di dati **l** è alto.
    - Dipende da **delta**, che è il livello di certezza probabilistica che vogliamo avere.

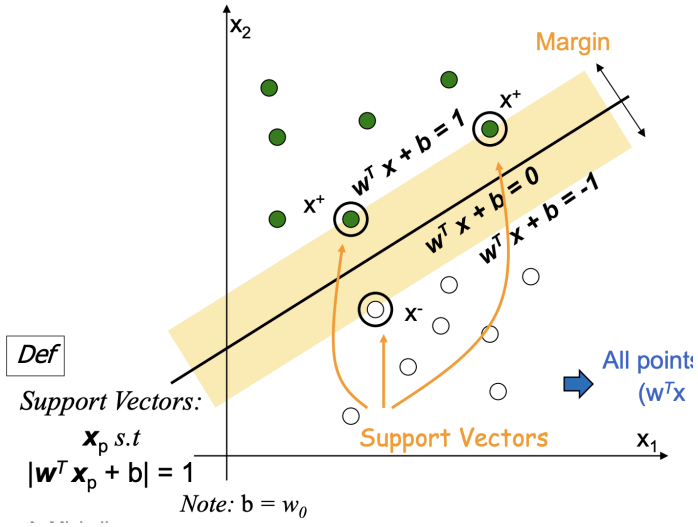
ML - SVM

Support Vector Machines

Le **SVM** (*Support Vector Machines*) sono algoritmi di machine learning supervisionato usati soprattutto per la classificazione e la regressione.

Immagina di avere punti di due classi: blu e rossi. In un piano cartesiano. L'obiettivo è tracciare una linea che separi le due classi. La SVM cerca quella che lascia il **margine massimo** tra le classi. Introducendo il bias induttivo:

- se ho un ampio margine → la classificazione è robusta
- se ho un margine sottile → la classificazione è più fragile



Margine
è il doppio della distanza tra l'iperpiano separatore e il dato più vicino

Support Vectors
Sono i punti più vicini al confine, questi punti determinano la posizione dell'iperpiano.

Max margin optimization problem

Problema di addestramento: trovare (w,b) (dove b è W_0) in modo che tutti i punti siano classificati correttamente e il margine sia massimizzato

La SVM cerca un iperpiano: $w^t x+b=0$ che separi le classi massimizzando il margine. La distanza del margine risulta proporzionale a:

2/||w|| quindi per massimizzare il margine dobbiamo cercare di minimizzare la **||w||**.

def. La VC-dim della SVM ha un valore inverso al margine, ovvero diminuisce con margini elevati.

def. L'iperpiano ottimo è quello che massimizza il margine e classifica correttamente gli esempi di training.

Definizione del problema primale

Training problem (primal form) :

minimize $\| \boldsymbol{w} \|^2/2$ (i.e. $\boldsymbol{w}^T\boldsymbol{w}$) — *Objective function*

such that $(\boldsymbol{w}^T\boldsymbol{x}_p + b)\, y_p \geq 1$ for all $p = 1..l$ — *Constraints*

Definizione del problema duale

$$h(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x} + b) = sign(\sum_{p=1}^{l} \alpha_p y_p \boldsymbol{x}_p^T\boldsymbol{x} + b) = sign(\sum_{p \in SV} \alpha_p y_p \boldsymbol{x}_p^T\boldsymbol{x} + b)$$

gli alfa diversi da zero sono support vectors. L'iperpiano dipende soltanto dai support vectors.

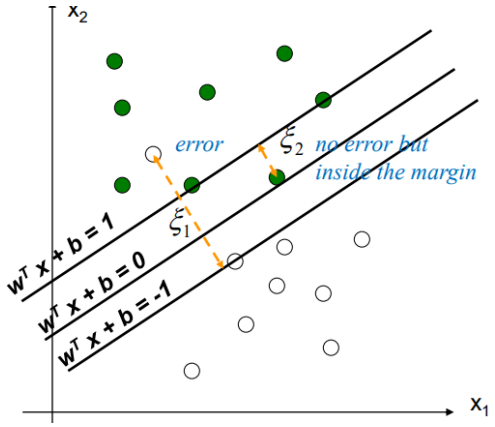
Soft margin

Si permette un margine di errore per avere un noise-tolerance e avere un margine più grande. Ammettiamo alcuni errori con le slack-variables ξ.

Primal training problem:

minimize $\| \boldsymbol{w} \|^2/2 + C \cdot \Sigma_p \xi_p$

such that $(\boldsymbol{w}\boldsymbol{x}_p + b)\, y_p \geq 1 - \xi_p$ and $\xi_p \geq 0$ for all p



C: è un iper parametro definito dall'utente rappresenta il costo di ogni errore

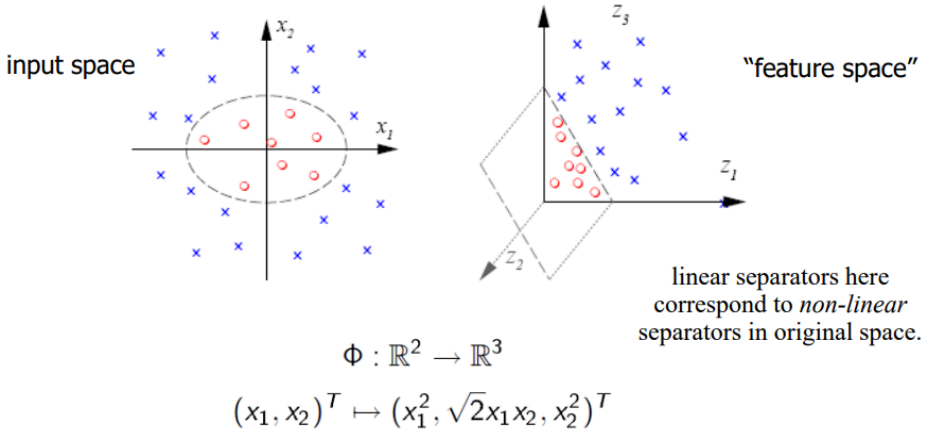
- **C** basso sono consentiti molti errori TR possibile underfitting (modello semplice)
- **C** alto non sono consentiti errori TR piccolo margine possibile → overfitting (modello complesso o flessibile)

ξp si possono vedere come la distanza tra il vettore ed il punto che stiamo analizzando

Kernel

L'idea: Quando dobbiamo separare delle classi di punti non separabili linearmente possiamo applicare la linear basis expansion in maniera efficiente via kernel.

Mappiamo i punti dati nello spazio di input in un feature space ad alta dimensionalità, dove possono essere linearmente separabili.



La complessità dipende dal numero di free-variables w nella formula:

$$h_{\mathbf{w}}(\mathbf{x}) = sign(\sum_k w_k \phi_k(\mathbf{x}) \ )$$

Kernel Trick
Ci permette di portare i dati in uno spazio ad alta dimensionalità.
Tramite i kernel (aumentando la dimensionalità) aumenta enormemente la flessibilità del modello e potrebbe causare overfitting. Tuttavia, nelle SVM la complessità del classificatore è controllata dalla massimizzazione del margine (equivalentemente dalla minimizzazione di $||w||^2$), quindi il modello può rimanere semplice e generalizzare bene anche in feature spaces molto grandi.

$$h(\mathbf{x}) = sign(\sum_{p \in SV} \alpha_p y_p \boxed{K(\mathbf{x}_p, \mathbf{x})})$$

La funzione del kernel $K(x_p,x)$ permette di calcolarlo in maniera efficiente.

Le SVM possono overfittare a seconda degli iper-parametri scelti: *C, kernel function e kernel parameters*.

- con kernel troppo flessibili
- con parametri scelti male
- con pochi dati

ML - KNN, unsupervide learning

K-Nearest Neighbors - Supervised Learning

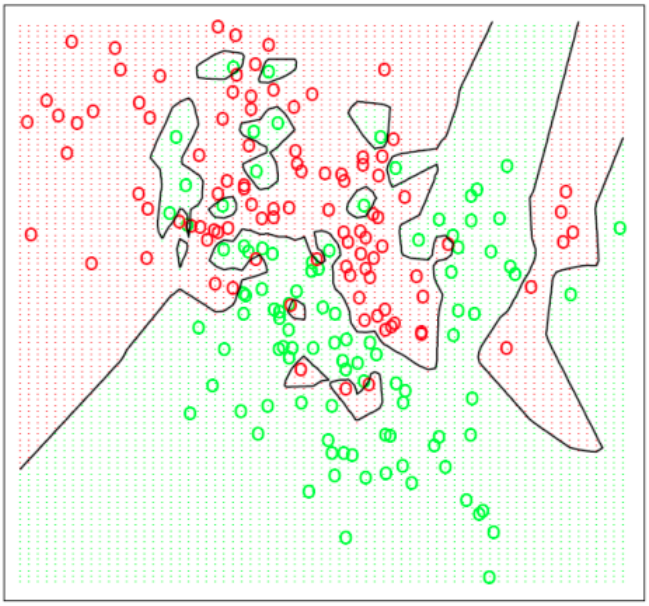
I modelli precedenti che abbiamo visto sono ***modelli parametrici***, ossia che sono modelli che si comportano in modi diversi in base ad alcuni parametri modificabili. Esistono anche i modelli ***non parametrici***, cioè modelli non caratterizzabili attraverso dei parametri. **K-NN** fa parte del **Supervised Learning**.

1-Nearest Neighbor (1NN)

Questo algoritmo non impara, ma sfrutta tutti i valori del training set. Abbiamo i dati nella forma $<x_i,y_i>$, dato un input x di dimensione n, dobbiamo trovare l'esempio di training "più vicino" ovvero quello che minimizza la distanza euclidea:

$$d(\boldsymbol{x}, \boldsymbol{x}_p) = \sqrt{\sum_{t=1}^{n} (x_t - x_{p,t})^2} = ||\boldsymbol{x} - \boldsymbol{x}_p||$$

Ottenendo una classificazione dei dati estremamente flessibile come nell'immagine d'esempio:



K-Nearest Neighbor

Si classifica in base alla media dei K punti più vicini:

$$avg_k(\boldsymbol{x}) = 1/k \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$$

Per la classificazione possiamo usare una formula del tipo:

$$h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad avg_k(\boldsymbol{x}) > 0.5 \\ 0 & \_\_\_\_\_otherwise \end{cases} \quad for\ y_i = \{0,1\}\ (targets)$$

Per la regressione possiamo usare direttamente il risultato della media: avg_k.

Limitazioni
Costo computazionale elevato (temporale) perché per ogni nuovo input bisogna calcolare le distanze dall'esempio a tutti i vettori memorizzati. Inoltre è anche costoso dal punto di vista spaziale dato che tutti i dati sono memorizzati.

K-means - Unsupervised Learning

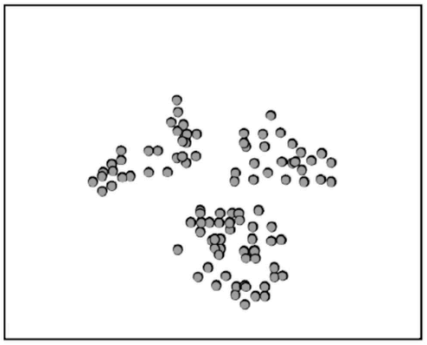
K-means fa parte del **Unsupervised Learning**. Il **K-means** è l'algoritmo più semplice e più comunemente usato che utilizza un criterio di **errore al quadrato**. L'algoritmo **K-means** è popolare perché è facile da implementare ed è **generalmente efficiente**.

Clustering

tecnica di analisi dei dati che consiste nel raggruppare insiemi di oggetti in modo che gli oggetti nello stesso gruppo (***cluster***) siano più simili tra loro rispetto a quelli in altri gruppi. È una tecnica di apprendimento **non supervisionato**, il che significa che non richiede etichette o supervisione durante il processo di addestramento.

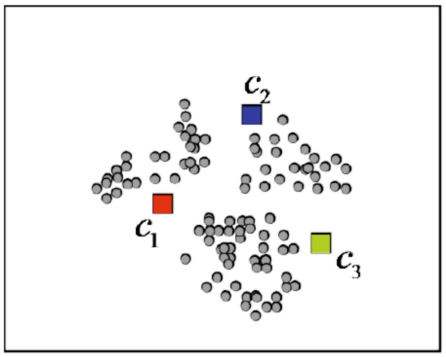
Funzionamento

Si inizia rappresentando tutti i punti nello spazio.



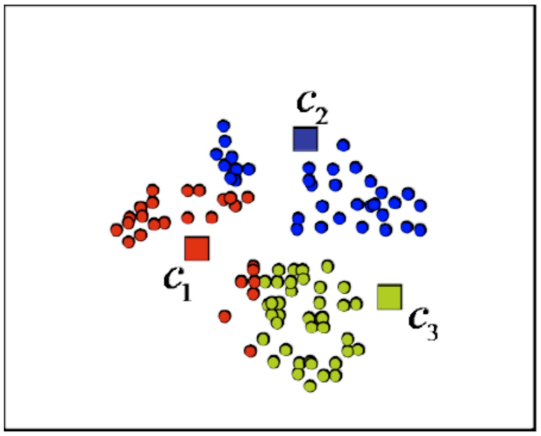
Vengono scelti i **k centri** di **cluster** (***centroide***) che coincidano con K **pattern** scelti casualmente o **K punti** definiti anch'essi casualmente all'interno dell'iper volume contenente il pattern set.

- I **centroidi** verranno nominati $c_1,...,c_k$

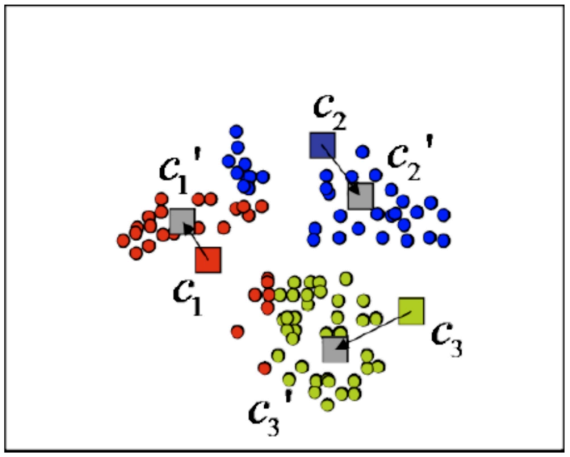


Assegnare ogni punto al centroidi più vicino. Per ogni x calcoliamo:

$$i^*(\mathbf{x}) = \arg\min_i \left\|\mathbf{x} - c_i\right\|^2 \text{ ----> } Norm\ 2$$



Viene ricalcolato il centroide utilizzando i nuovi punti.

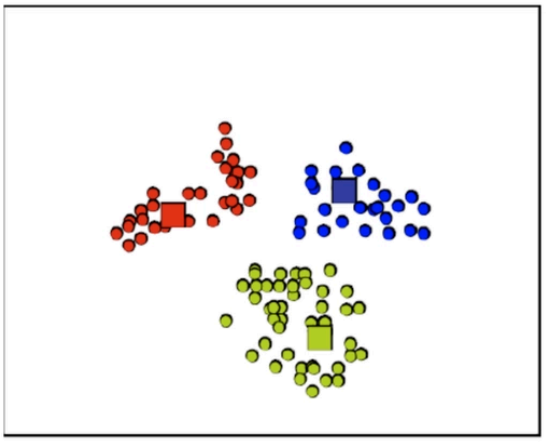


$$c_i = \frac{1}{|cluster_i|} \sum_{j:x_j \in cluster_i} \mathbf{x}_j \qquad \Lambda$$

Vector — *Number of cluster members* — *Vectors*

Se non viene soddisfatto un **criterio di convergenza**, andare al passaggio 2

- criteri come nessuna o minima riassegnazione di schemi a nuovi centri di cluster.
- una minima diminuzione dell'errore quadratico.



Limitazioni

- Il numero di cluster da trovare deve essere fornito (questo porta a fare trial and error per trovare il K che fitta meglio).
- I minimi locali della **Loss** rendono il metodo dipendente dall'inizializzazione, si esegue più volte da diverse inizializzazioni casuali (ci sono anche dei metodi che inizializzano con un'euristica).
- K-means può funzionare bene per cluster compatti, ma non consente di proiettare i dati in uno spazio di dimensione minore.

Reti Neurali

Vengono utilizzate sia nel Supervised che nel Unsupervised Learning. Sono simili alla visione del Linear Threshold Unit, una rete neurale è una rete di modelli *non lineari* con una capacità di approssimazione universale e capacità predittive. Gli strati interni sono "nascosti" e ogni unità è non lineare, fornendo alla rete neurale la capacità di estrarre (imparando) una nuova rappresentazione dei dati.

Abbiamo una espansione della base non lineare in w, e le phi sono imparate automaticamente, ma purtroppo questo scaturisce in un problema di ottimizzazione non lineare.

Deep Learning

Il Deep Learning è quel campo di ricerca dell'apprendimento automatico (machine learning) e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso.

In altre parole, si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa.