

Appunti di Introduzione all’Intelligenza Artificiale Unipi - Parte 1

Raffaele Apetino

Marzo 2020

Contents

1 Premessa	2
2 Introduzione	2
2.1 Test di Turing	2
3 Agenti Intelligenti	3
3.1 Caratteristiche	3
3.2 Agenti Razionali	4
3.3 Agenti Autonomi	4
4 Ambienti	4
4.1 Descrizione PEAS dei problemi	4
4.2 Proprietà dell’ambiente	5
4.3 Simulatore di ambienti	5
5 Tipi di Agente	6
5.1 Struttura generale di un agente	6
5.2 Agente basato su tabella	6
5.3 Agenti reattivi semplici	6
5.4 Agenti basati su modello	7
5.5 Agenti con obiettivo	7
5.6 Agenti con valutazione di utilità	8
5.7 Agenti che apprendono	8
6 Agenti risolutori di problemi	9
6.1 Formulazione di un problema	9
6.2 Algoritmi di ricerca	9
6.3 Il problema dell’itinerario	9
6.3.1 Formulazione del problema dell’itinerario	10
6.4 Il problema dell’aspirapolvere	10
6.4.1 Formulazione del problema dell’aspirapolvere	10
7 Ricerca della soluzione (non informata)	11
7.1 Strutture dati per gli algoritmi di ricerca	11
7.2 Ricerca ad Albero	11
7.3 Strategie non informate VS Strategie informate "euristiche"	12
7.4 Valutazione di una strategia	12
7.5 Problemi cammini ciclici e ridondanze	12

7.6	Ricerca in ampiezza - BF	13	
7.6.1	BF-Albero	13	
7.6.2	BF-Grafo	13	
7.6.3	Analisi complessità BF	13	
7.7	Ricerca in profondità - DF	14	
7.7.1	Analisi complessità DF-Albero	14	
7.7.2	Analisi complessità DF-Grafo	14	
7.8	Ricerca in profondità ricorsiva - DF con backtracking	15	
7.9	Ricerca in profondità limitata - DL	15	
7.9.1	Analisi complessità DL	15	
7.10	Approfondimento Iterativo - ID	15	
7.10.1	Analisi complessità ID	15	
7.11	Ricerca Bidirezionale - Bidir.	16	
7.11.1	Analisi complessità Bidir.	16	
7.12	Ricerca di costo uniforme - UC	16	
7.12.1	UC-Albero	17	
7.12.2	UC-Grafo	17	
7.12.3	Analisi complessità UC	17	
7.13	Confronto finale delle strategie	18	
8	Ricerca Euristică	18	
8.1	Algoritmo Best-First - BFH	18	
8.2	Algoritmo Greedy-Best-First - GBF	19	
8.3	Algoritmo A	19	
8.3.1	Completezza Algoritmo A	19	
8.3.2	Dimostrazione Completezza di A	20	
8.4	Algoritmo A^*	20	
8.4.1	Esempio A^* sul problema dell'itinerario	20	
8.4.2	Osservazioni su A^*	20	
8.4.3	Ottimalità di A^*	21	
8.4.4	Euristica Consistente (o monotona)	21	
8.4.5	Dimostrazione ottimalità di A^*	21	
8.4.6	Vantaggi di A^*	21	
8.5	Costruire le euristiche di A^*	22	
8.6	Valutare Algoritmi di ricerca euristica	22	
8.7	Inventare euristiche	23	
8.8	Algoritmi evoluti basati su A^*	23	
8.8.1	Beam Search	23	
8.8.2	A^* con approfondimento iterativo - <i>IDA</i> *	23	
8.8.3	Best-First ricorsivo - RBFS	24	
8.8.4	A^* con memoria limitata - <i>SMA</i> *	24	
9	Ricerca Locale	25	
9.1	Spazio degli stati	25	
9.2	Algoritmo Hill-Climbing	25	
9.2.1	Problemi e Miglioramenti per Hill-Climbing	26	
9.3	Algoritmo Tempra Simulata (Simulated Annealing)	26	
9.4	Algoritmo Local Beam	27	
9.4.1	Local Beam Search Stocastica	27	
9.5	Algoritmi Genetici - GA	27	
9.6	Gradiente (Hill-Climbing iterativo)	28	

10 Oltre la ricerca classica	29
10.1 Problema dell'aspirapolvere non deterministico	29
10.2 Alberi AND-OR	29

1 Premessa

Questi appunti sono stati controllati più volte, ma come ben si sa:

Sa chi sa che nulla sa, e chi sa che nulla sa ne sa più di chi ne sa.

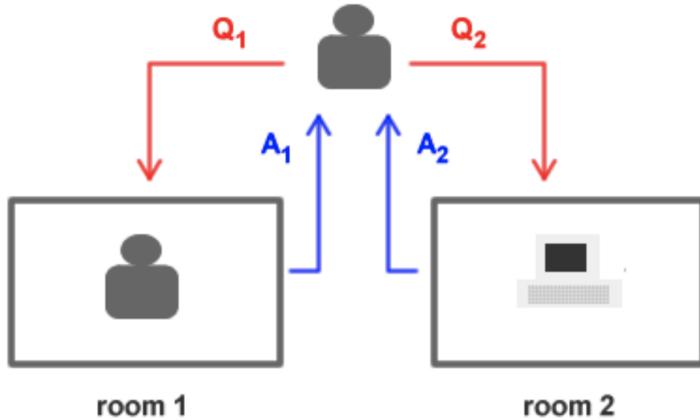
Quindi invito a guardare per bene le slide del corso e studiare su quelle. Consiglio di sfruttare questi appunti per un ripasso veloce. Per segnalare errori mandatemi una mail a r.apetino at studenti.unipi.it
∨ apriete una Issue su GitHub ∨ fate una Pull Request ∨ cercatemi su Telegram.

2 Introduzione

L'intelligenza artificiale si occupa della comprensione e riproduzione del comportamento intelligente. L'approccio psicologico (psicologia cognitiva) ha come obiettivo la comprensione dell'intelligenza umana e quindi risolvere i problemi con gli stessi processi usati dall'uomo. L'approccio informatico è quello di costruire entità dotate di razionalità, cioè si occupa dell'automazione del comportamento intelligente. Quest'ultimo viene eseguito attraverso la meccanizzazione del ragionamento e la comprensione mediante modelli computazionali della psicologia e del comportamento degli uomini.

C'è però una domanda molto importante riguardo a cosa sia l'intelligenza: capacità di ragionamento? Buon senso? Capacità sociali e di comunicazione? Capacità di comprendere e provare emozioni?

2.1 Test di Turing



In due stanze separate ci sono una persona ed un computer, fuori da queste stanze con le porte chiuse, c'è una seconda persona che fa domande ad entrambi. Il test di Turing si basa su dopo quanto tempo l'uomo esterno riesce a capire chi è uomo e chi macchina.

Da qui ci viene una domanda fondamentale, dobbiamo dotare i computer di senso comune? (esistono già dei progetti nominati CYC e OpenMind) Una definizione di senso comune possiamo darla, è la capacità di un uomo di poter riconoscere in modo immediato ricorrendo all'uso della ragione naturale. Quindi possiamo anche dare una definizione di intelligenza: è la qualità mentale che consiste nell'abilità di apprendere dall'esperienza, di adattarsi a nuove situazioni, comprendere e gestire concetti astratti, utilizzare la conoscenza per agire sul proprio ambiente.

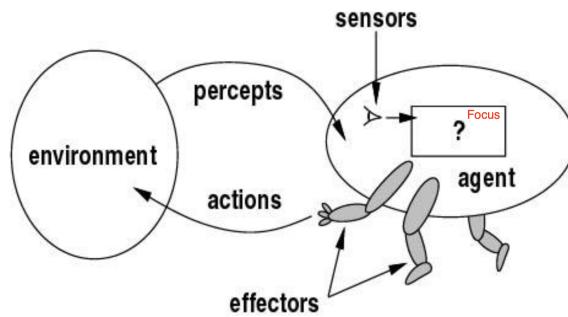
L'intelligenza artificiale è pericolosa? Andrebbe regolata? Le macchine dovrebbero avere un etica? Ma su quale etica dovrebbero basarsi? L'intelligenza collettiva può essere estratta o inferita dai dati?

3 Agenti Intelligenti

L'approccio moderno dell'IA si basa sulla costruzione di agenti intelligenti e sulla creazione del programma agente da parte del programmatore. La visione ad agenti ci offre un quadro di riferimento e una prospettiva diversa dall'analisi dei sistemi software. Il nostro primo obiettivo è realizzare agenti per la risoluzione di problemi vista come ricerca in uno spazio di stati.

3.1 Caratteristiche

Un agente è qualcosa in grado di percepire l'ambiente attraverso i sensori e agire su quell'ambiente sfruttando gli attuatori.



Agenti Intelligenti:

- Sono situati: ricevono percezioni da un ambiente (tramite input dei sensori), agiscono sull'ambiente mediante azioni (sfruttando gli attuatori).
La sequenza percettiva è la storia completa delle percezioni. La scelta dell'azione è funzione unicamente della sequenza percettiva
- Gli agenti hanno abilità sociale: sono capaci di comunicare, collaborare, difendersi da altri agenti.
- Gli agenti hanno credenze, obiettivi, intenzioni...
- Gli agenti sono embodied: hanno un corpo, fino a considerare i meccanismi delle emozioni.

In termini matematici diciamo che il comportamento dell'agente è descritto dalla funzione agente. La funzione agente è una descrizione matematica che definisce l'azione da compiere per ogni sequenza percettiva, il programma agente è una implementazione concreta di questa funzione. Il nostro compito è proprio quello di progettare il programma agente.

3.2 Agenti Razionali

Un agente razionale interagisce con il suo ambiente in maniera efficace cioè "fa la cosa giusta". Serve quindi un criterio di valutazione oggettivo dell'effetto delle azioni dell'agente¹, come ad esempio il costo minimo di un cammino per arrivare alla soluzione. La razionalità è relativa alla misura delle prestazioni, alle conoscenze pregresse dell'ambiente e alle capacità dell'agente.

Agente Razionale: Per ogni sequenza di percezioni compie l'azione che massimizza il valore atteso della misura delle prestazioni, considerando le sue percezioni passate e la sua conoscenza pregressa.

Raramente tutta la conoscenza sull'ambiente può essere fornita "a priori", l'agente razionale deve essere in grado di modificare il proprio comportamento con l'esperienza (percezioni passate oppure percezioni che è in grado di apprendere in futuro).

3.3 Agenti Autonomi

Modificare il proprio comportamento con l'esperienza implica la creazione di agenti autonomi:

Un agente è autonomo nella misura in cui il suo comportamento dipende dalla sua esperienza.

Un agente il cui comportamento fosse determinato solo dalla sua conoscenza built-in, sarebbe non autonomo e poco flessibile.

4 Ambienti

Definire un problema per un agente significa caratterizzare l'ambiente in cui l'agente opera.

4.1 Descrizione PEAS dei problemi

- Performance (prestazione, obiettivo)
- Environment (ambiente, dove attua le sue azioni)
- Actuators (attuatori, meccanismi con cui agisco sull'ambiente)
- Sensors (sensori, percezioni)

Prestazione	Ambiente	Attuatori	Sensori
Arrivare alla destinazione, sicuro, veloce, ligio alla legge, viaggio confortevole, minimo consumo di benzina, profitti massimi	Strada, altri veicoli, pedoni, clienti	Sterzo, acceleratore, freni, frecce, clacson, schermo di interfaccia o sintesi vocale	Telecamere, sensori a infrarossi e sonar, tachimetro, GPS, contachilometri, acelerometro, sensori sullo stato del motore, tastiera o microfono

Figure 1: Esempio descrizione PEAS di un agente guidatore di taxi

¹vedremo che una azione ha come conseguenza la creazione di un nuovo stato nell'ambiente

4.2 Proprietà dell'ambiente

- L'ambiente è osservato dall'agente che ne apprende le sue caratteristiche.
 - Completamente osservabile: conoscenza completa dell'ambiente, non c'è bisogno di mantenere uno stato del mondo esterno.
 - Parzialmente osservabile: sono presenti limiti o inaccuratezze che riguardano la conoscenza del mondo.
 - Non osservabile: se l'agente non ha sensori
- Il mondo può cambiare anche per eventi, non necessariamente per azioni di agenti.
 - Agente singolo.
 - Multi-agente: può essere a sua volta competitivo oppure cooperativo quindi con lo stesso obiettivo (comunicano).
- Si possono predire i cambiamenti del mondo.
 - Deterministico: se lo stato successivo è completamente determinato dallo stato corrente e dall'azione.
 - Stocastico: esistono elementi di incertezza con associata probabilità.
 - Non deterministico: non si può sapere come evolve il mondo quindi si tiene traccia di più stati possibili risultanti da una azione eseguita.
- L'ambiente si può caratterizzare anche come
 - Episodico: l'esperienza dell'agente è divisa in episodi atomici indipendenti.
 - Sequenziale: ogni decisione influenza le successive.
- L'ambiente può cambiare nel corso del tempo.
 - Statico: il mondo non cambia mentre l'agente è fermo e sta decidendo l'azione.
 - Dinamico: il mondo cambia nel tempo, tardare equivale a non agire.
 - Semi-dinamico: l'ambiente non cambia ma la valutazione dell'agente si.
- L'ambiente è caratterizzato da valori.
 - Discreto: i valori del mondo sono limitati (ad esempio gli stati possono essere di numero finito)
 - Continuo: i valori del mondo possono essere infiniti (ad esempio il tempo può essere infinito)
- Lo stato di conoscenza dell'agente può essere:
 - Noto: conosco l'ambiente, questo non significa che sia osservabile (ad esempio in un gioco di carte, le carte sono note ma se sono coperte non sono osservabili!)
 - Ignoto: devo compiere azioni esplorative per conoscerlo tutto.

Gli ambienti reali sono parzialmente osservabili, stocastici, sequenziali, dinamici, continui, multi-agente, ignoti.

4.3 Simulatore di ambienti

E' uno strumento software che si occupa di generare stimoli per gli agenti, raccogliere le azioni di risposta, aggiornare lo stato dell'ambiente e valutare le prestazioni dell'agente.

5 Tipi di Agente

5.1 Struttura generale di un agente

Struttura Agente = Architettura² + Programma

Funzione Agente³ = Ag : Percezioni → Azioni

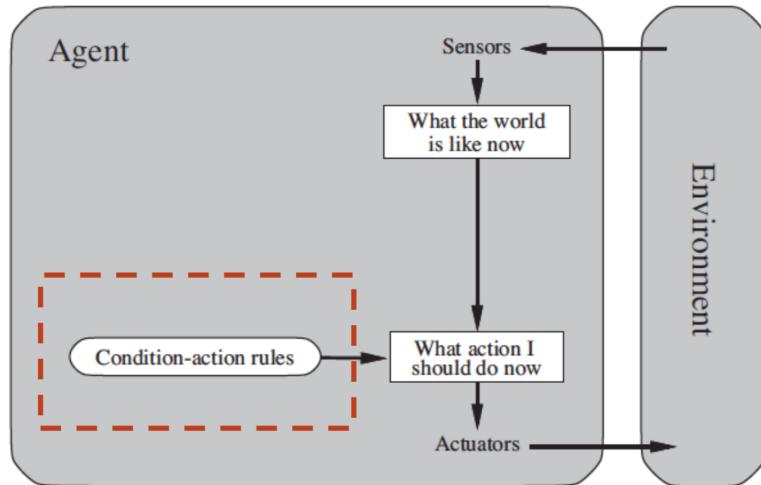
Il programma dell'agente implementa la funzione Ag.

5.2 Agente basato su tabella

La scelta dell'azione è un accesso a una tabella che associa una azione ad ogni possibile sequenza di percezioni. Ci sono ovvi problemi:

1. Dimensione: per giocare a scacchi la tabella ha un numero di righe molto maggiore di 10^{80} perciò la situazione è ingestibile.
2. Difficile da costruire.
3. Nessuna autonomia.
4. Difficile da aggiornare, quindi l'"apprendimento" diventa complesso.

5.3 Agenti reattivi semplici

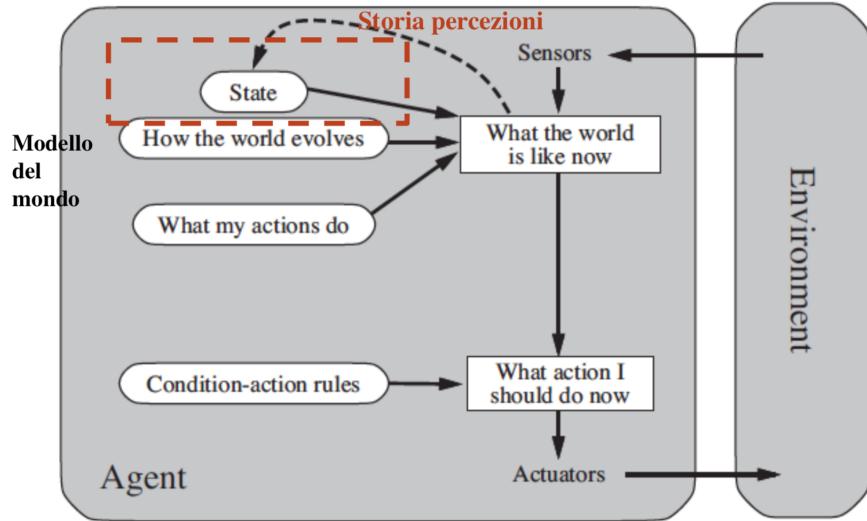


Sono presenti delle regole if-then costruite a priori che mi dicono quale azione fare in base allo stato e alle regole in quel dato istante.

²è la parte hardware dove gira il nostro programma agente

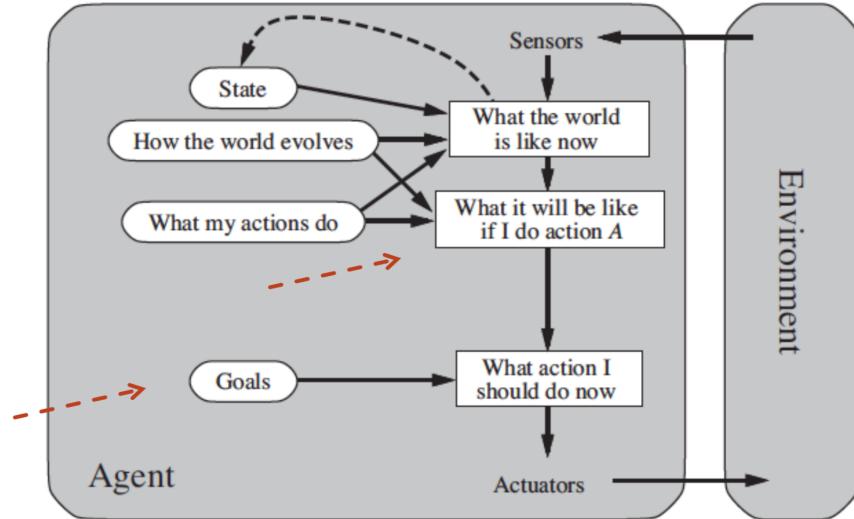
³La funzione agente definisce l'azione da compiere per ogni sequenza percettiva

5.4 Agenti basati su modello



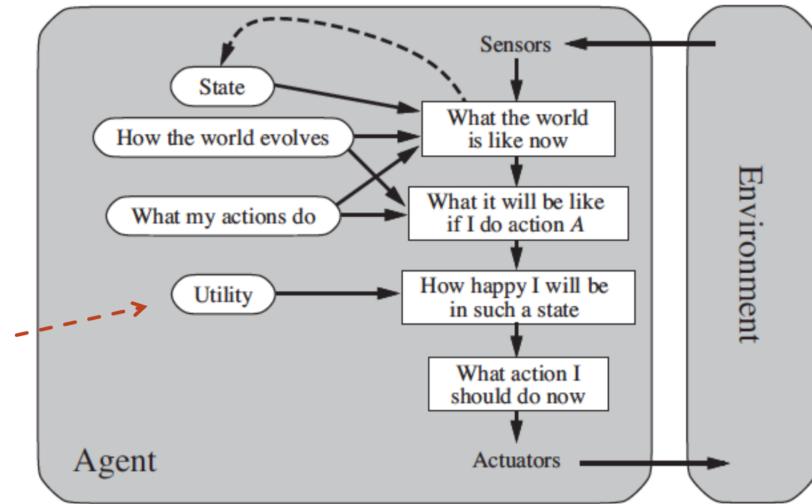
E' presente un modello del mondo che comprende lo stato aggiornato con la storia delle percezioni. Sono ancora presenti le regole if-then.

5.5 Agenti con obiettivo



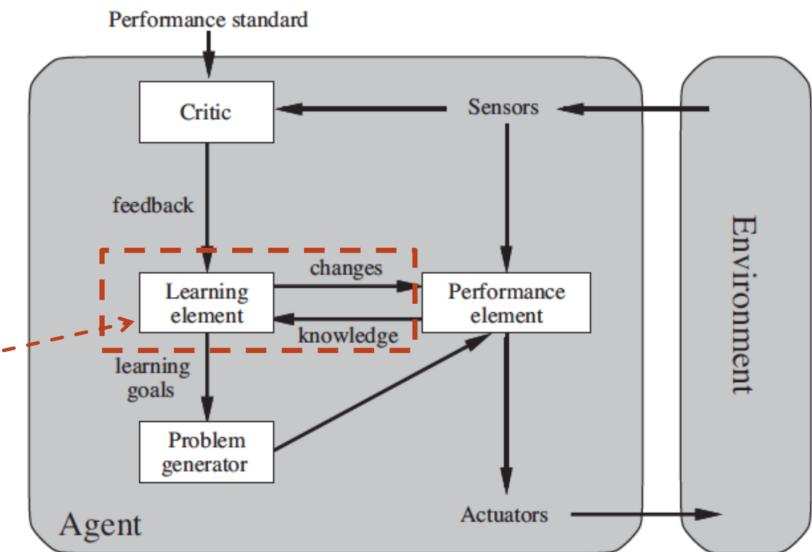
Sono agenti guidati da un obiettivo nella scelta dell'azione (viene fornito un goal esplicito, ad esempio raggiungere una città). L'azione migliore dipende da quale obiettivo bisogna raggiungere e pianificano le proprie azioni in base al goal. L'agente si preoccupa di capire come sarà il mondo dopo aver eseguito una azione.

5.6 Agenti con valutazione di utilità



Ci sono obiettivi alternativi magari più facilmente raggiungibili. L'agente deve decidere verso quali di questi muoversi. E' necessaria una funzione di utilità che associa ad uno stato un numero reale. La funzione di utilità tiene conto anche della probabilità di successo e di utilità attesa.

5.7 Agenti che apprendono



E' presente una componente di apprendimento che produce cambiamenti al programma agente, migliora le prestazioni adattando i suoi componenti, apprendendo dall'ambiente. L'elemento esecutivo è il programma agente, l'elemento critico osserva e dà feedback sul comportamento. Infine è presente un generatore di problemi, suggerisce nuove situazioni da esplorare.

6 Agenti risolutori di problemi

Adottano il paradigma della risoluzione di problemi come ricerca in uno spazio di stati. Sono agenti con modello che adottano una rappresentazione atomica dello stato, hanno un obiettivo e pianificano l'intera sequenza di mosse prima di agire.

Processo di risoluzione di un problema:

1. Determinare un obiettivo (un insieme di stati tali che l'obiettivo è soddisfatto)
2. Formulare un problema (rappresentazione degli stati e delle azioni)
3. Determinare soluzione mediante ricerca
4. Esecuzione soluzione

Assumiamo un ambiente statico, osservabile, discreto e deterministico.

6.1 Formulazione di un problema

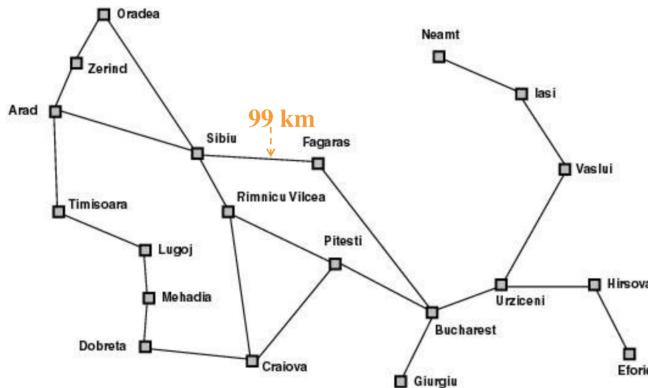
Un problema può essere definito formalmente mediante 5 componenti:

1. Stato iniziale
 2. Azioni possibili nello stato ($Azioni(stato)$)
 3. Modello di transizione: $Risultato(stato, azione) = nuovo_stato$
 4. Test obiettivo: insieme di stati obiettivo ($GoalTest(stato) = \{true, false\}$)
 5. Costo del cammino: somma dei costi delle azioni, costo di passo definito come $c(s, a, s')$
- 1, 2 e 3 definiscono implicitamente lo spazio degli stati.

6.2 Algoritmi di ricerca

Il processo che cerca una sequenza di azioni che raggiunge l'obiettivo è detto ricerca. Gli algoritmi di ricerca prendono in input un problema e restituiscono un cammino soluzione. La misura delle prestazioni è definita come: Costo totale = costo della ricerca + costo del cammino soluzione. Valuteremo gli algoritmi riguardo la ricerca cercando di ottimizzare il cammino soluzione.

6.3 Il problema dell'itinerario



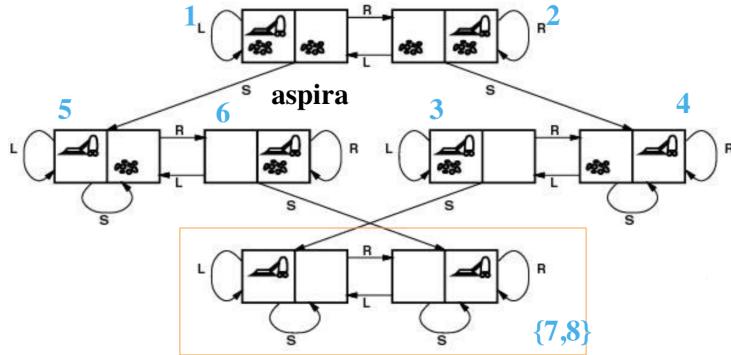
Il problema dell'itinerario riguarda la ricerca del percorso più breve da una città di partenza a una città di arrivo.

6.3.1 Formulazione del problema dell'itinerario

1. Stati: le città
2. Stato iniziale: la città da cui si parte ($In(Arad)$)
3. Azioni: spostarsi su una città vicina collegata ($Azioni(In(Arad)) = \{Go(Sibiu), Go(Zerind), \dots\}$)
4. Modello di transizione ($Risultato(In(Arad), Go(Sibiu)) = In(Sibiu)$)
5. Test obiettivo $\{In(Bucarest)\}$
6. Costo del cammino: somma delle lunghezze delle strade

Lo spazio degli stati coincide con la rete (grafo) di collegamenti tra città.

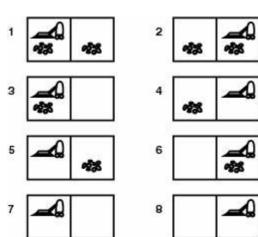
6.4 Il problema dell'aspirapolvere



Il problema dell'aspirapolvere riguarda la pulizia di due stanze adiacenti con il minimo sforzo.

6.4.1 Formulazione del problema dell'aspirapolvere

1. Stati: sono determinati sia dalla posizione dell'aspirapolvere che dalla posizione dello sporco



2. Stato iniziale: qualsiasi stato può essere uno stato iniziale
3. Percezioni: Sporco - Non sporco
4. Azioni: Sinistra (L) - Destra (R) - Aspira (S)
5. Modello di transizione: Aspira(stanza) = stanza pulita, Destra = si sposta nella stanza a destra, Sinistra = si sposta nella stanza a sinistra
6. Test obiettivo: rimuovere lo sporco (stati 7 o 8)
7. Costo del cammino: ogni azione ha costo 1

7 Ricerca della soluzione (non informata)

Si tratta di generare un albero di ricerca sovrapposto allo spazio degli stati (generato da possibili sequenze di azioni).

7.1 Strutture dati per gli algoritmi di ricerca

Gli algoritmi di ricerca hanno bisogno di strutture dati per tenere traccia di come l'albero di ricerca si sta formando.

Un nodo n è una struttura dati con quattro componenti:

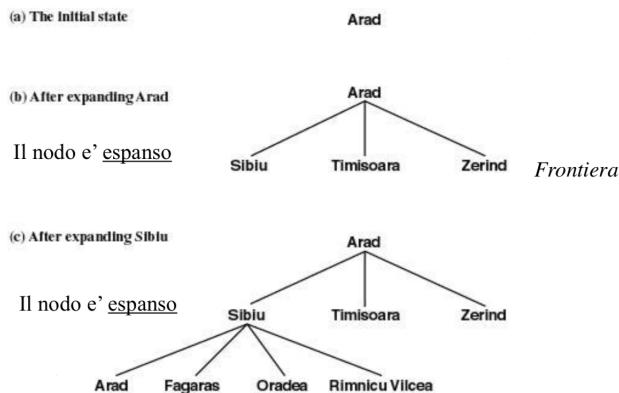
1. Uno stato: n.estado
2. Il nodo padre: n.padre
3. L'azione effettuata sul padre per generarlo: n.azione
4. Il costo del cammino a partire dal nodo iniziale: n.costocammino definito come:

$$g(n) = n.padre.costocammino + \text{costo dell'ultimo passo}$$

La frontiera è lista dei nodi in attesa di essere espansi (le foglie dell'albero di ricerca). Essa può essere implementata come una coda FIFO (viene estratto l'elemento più vecchio della coda), LIFO (viene estratto quello più recentemente inserito) o con priorità (viene estratto quello con priorità più alta). Sulla frontiera sono definite le seguenti operazioni:

- *Vuota(coda)* // mi dice se la coda è vuota
- *Pop(coda)* // estrae il primo elemento dalla coda in base alla strategia utilizzata
- *Inserisci(elemento, coda)* // inserisce un elemento della coda

7.2 Ricerca ad Albero



Nella ricerca ad albero non controlliamo se i nodi (stati) siano già stati esplorati (questo controllo lo vedremo successivamente sui grafi). Come prima operazione inizializzo la frontiera con lo stato iniziale (Arad), ad ogni ciclo controllo se la frontiera è vuota, se lo è FAIL altrimenti scelgo un nodo della frontiera e lo rimuovo. Se il nodo rimosso è contenuto negli stati obiettivo allora ho trovato la soluzione, altrimenti espando il nodo e aggiungo i successori alla frontiera. Come si può notare dall'immagine la ricerca ad albero può portare alla creazione di loop.

Il problema quindi è quale tra i nodi della frontiera scelgo?

7.3 Strategie non informate VS Strategie informate "euristiche"

Strategie non informate:

- Ricerca in ampiezza (BF)
- Ricerca in profondità (DF)
- Ricerca di costo uniforme (UC)
- Ricerca in profondità limitata (DL)
- Ricerca con approfondimento iterativo (ID)

Le strategie informate "euristiche" le vedremo dopo, fanno uso di informazioni riguardo alla distanza stimata dalla soluzione.

7.4 Valutazione di una strategia

Dobbiamo definire un criterio per capire quale algoritmo di ricerca è migliore di un altro.

- Completezza: l'algoritmo garantisce di trovare una soluzione quando esiste?
- Ottimalità (ammissibilità): trova la soluzione migliore con costo minore
- Complessità in tempo: tempo richiesto per trovare la soluzione
- Complessità in spazio: memoria richiesta per trovare la soluzione

Per descrivere la complessità di ogni algoritmo sfrutteremo le seguenti definizioni che valgono sia per i grafi che per gli alberi:

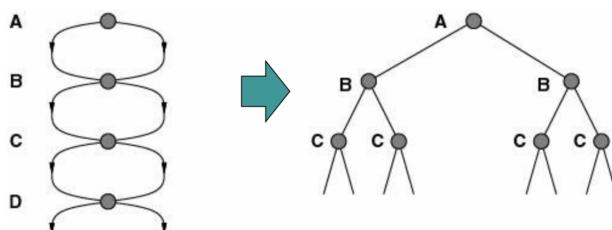
b = fattore di ramificazione (branching, numero di nodi successori)

d = profondità del nodo obiettivo più superficiale (depth)

m = lunghezza massima dei cammini nello spazio degli stati (max)

7.5 Problemi cammini ciclici e ridondanze

I cammini ciclici rendono gli alberi di ricerca infiniti. Su spazi di stati a grafo si generano più volte gli stessi nodi (o meglio nodi con stesso stato) nella ricerca, anche in assenza di cicli.



Ricordare gli stati già visitati occupa spazio ma ci consente di evitare di visitarli di nuovo. Ci sono tre soluzioni:

1. Non tornare nello stato da cui si proviene, si elimina il padre dai nodi successori (ma non evita cammini ridondanti).
2. Per evitare di creare cammini con cicli si controlla che i successori non siano antenati del nodo corrente.
3. Per non generare nodi con stati già visitati/esplorati teniamo in memoria ogni nodo visitato con complessità in spazio di $O(\text{StatiPossibili})$.

7.6 Ricerca in ampiezza - BF

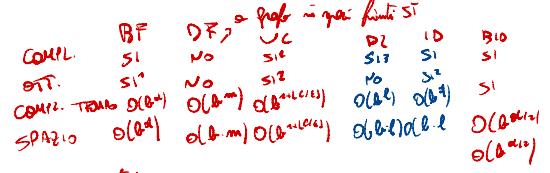
Esplorare il grafo dello spazio degli stati a livelli progressivi di stessa profondità. La frontiera è implementata con una coda che inserisce alla fine (FIFO).

7.6.1 BF-Albero

```

function Ricerca-Aampiezza-A (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    if problema.Test-Obiettivo(nodo.Stato) then return Soluzione(nodo)
    frontiera = una coda FIFO con nodo come unico elemento
    loop do
        if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera)
        for each azione in problema.Azioni(nodo.Stato) do
            espansione {
                figlio = Nodo-Figlio(problema, nodo, azione) [costruttore: vedi AIMA]
                if Problema.TestObiettivo(figlio.Stato) then return Soluzione(figlio)
                frontiera = Inserisci(figlio, frontiera) /* frontiera gestita come coda FIFO
            end

```



Nota che in questa versione gli stati sono goal-tested al momento in cui sono generati. → più efficiente, si ferma appena trova goal prima di espandere

7.6.2 BF-Grafo

```

function Ricerca-Aampiezza-g (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    if problema.Test-Obiettivo(nodo.Stato) then return Soluzione(nodo)
    frontiera = una coda FIFO con nodo come unico elemento
    esplorati = insieme vuoto
    loop do
        if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera); aggiungi nodo.Stato a esplorati
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            if figlio.Stato non è in esplorati e non è in frontiera then
                if Problema.TestObiettivo(figlio.Stato) then return Soluzione(figlio)
                frontiera = Inserisci(figlio, frontiera) /* in coda

```

Aggiunge in verde per gestire gli stati ripetuti

Nota che in questa versione gli stati sono goal-tested al momento in cui sono generati. → più efficiente, si ferma appena trova goal prima di espandere

22 MAX

$$8+8+3+3+5+(3) \approx \\ (2) \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ \text{VNU RE} \quad \text{OPTIMAL} \quad \text{PROB.}$$

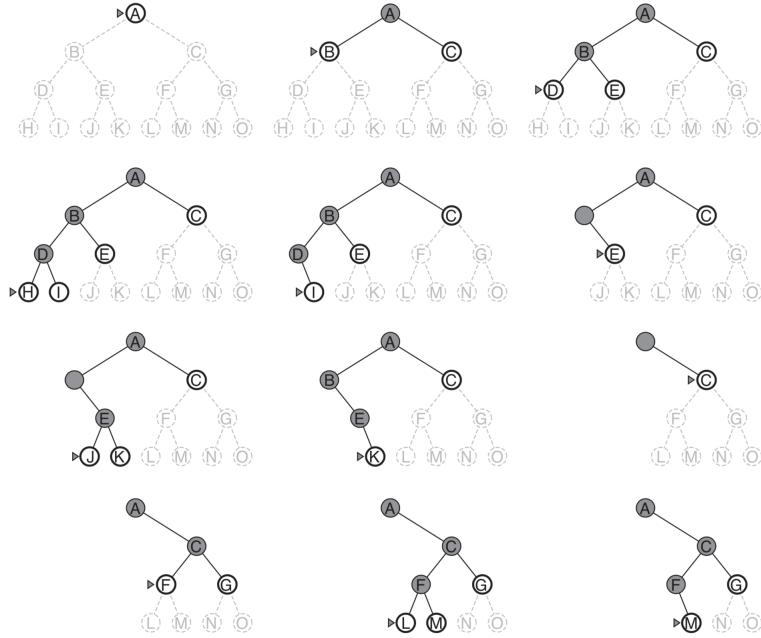
7.6.3 Analisi complessità BF

- Strategia Completa: SI
- Strategia Ottimale: SI se gli operatori hanno tutti lo stesso costo k ⁴
- Complessità Tempo: $O(b^d)$ (b figli per ogni nodo all'altezza d)
- Complessità Spazio: $O(b^d)$ (occupa un sacco di memoria)

⁴cioè $g(n) = k * \text{depth}(n)$ dove $g(n)$ è il costo del cammino per arrivare a n

7.7 Ricerca in profondità - DF

La ricerca Depth-First esplora il grafo dello spazio degli stati arrivando in profondità⁵ per ogni nodo corrente sulla frontiera. La frontiera è implementata con una coda che inserisce i successori in testa alla lista (LIFO). Cancella rami già completamente esplorati ma tiene in memoria tutti i successori del path corrente, occupando così in memoria solo b^*m .



7.7.1 Analisi complessità DF-Albero

- Strategia Completa: NO, si possono creare dei loop
- Strategia Ottimale: NO
- Complessità Tempo: $O(b^m)$ (che può essere maggiore di $O(b^d)$)
- Complessità Spazio: $O(b * m)$ (drastico risparmio di memoria)

7.7.2 Analisi complessità DF-Grafo

In caso di DF con visita su grafo si perdono i vantaggi della memoria: la memoria torna ad essere $O(b^d)$ ma così DF diventa completa su spazi degli stati finiti (al caso pessimo estende tutti i nodi) resta comunque non completa su spazi infiniti.

⁵fino a che non ha più successori

7.8 Ricerca in profondità ricorsiva - DF con backtracking

Ancora più efficiente in occupazione di memoria perché mantiene in memoria solo il cammino corrente (solo m nodi al caso pessimo). L'algoritmo è realizzato con "backtracking" che non necessita di tenere in memoria b nodi per ogni livello, ma salva lo stato su uno stack a cui torna in caso di fallimento per fare altri tentativi.

```
function Ricerca-DF-ricorsiva(nodo, problema)
    returns soluzione oppure fallimento
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    else
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            risultato = Ricerca-DF-ricorsiva(figlio, problema)
            if risultato ≠ fallimento then return risultato
        return fallimento
```

7.9 Ricerca in profondità limitata - DL

Il fallimento della ricerca DF in uno spazio di stati infinito viene mitigato in parte se si procede in profondità fino ad un certo livello predefinito l .

7.9.1 Analisi complessità DL

- Strategia Completa: SI solo per problemi in cui si conosce un limite superiore per la profondità della soluzione cioè se $l > d$. Se scegliamo $l < d$ la ricerca risulta incompleta.
- Strategia Ottimale: NO
- Complessità Tempo: $O(b^l)$
- Complessità Spazio: $O(b * l)$

7.10 Approfondimento Iterativo - ID

Ad ogni iterazione aumento il limite della ricerca in profondità limitata (DL) e rincomincio dalla radice.

7.10.1 Analisi complessità ID

- Strategia Completa: SI
- Strategia Ottimale: SI se gli operatori hanno tutti lo stesso costo
- Complessità Tempo: $O(b^d)$
- Complessità Spazio: $O(b * d)$

Miglior compromesso tra BF e DF, ma i nodi dell'ultimo livello sono generati una volta, quello del penultimo due, ..., quelli del primo d volte.

7.11 Ricerca Bidirezionale - Bidir.

Un problema che possiamo valutare è la direzione della ricerca.

- Ricerca in avanti: ricerca guidata dai dati, si esplora lo spazio di ricerca dallo stato iniziale allo stato obiettivo
- Ricerca all'indietro: ricerca 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.

In quale direzione conviene procedere? Convien procedere nella direzione in cui il fattore di diramazione è minore. Procediamo in avanti quando gli obiettivi sono molti e abbiamo una serie di dati da cui partire. Procediamo all'indietro quando l'obiettivo è chiaramente definito oppure i dati del problema non sono noti e la loro acquisizione può essere guidata dall'obiettivo.

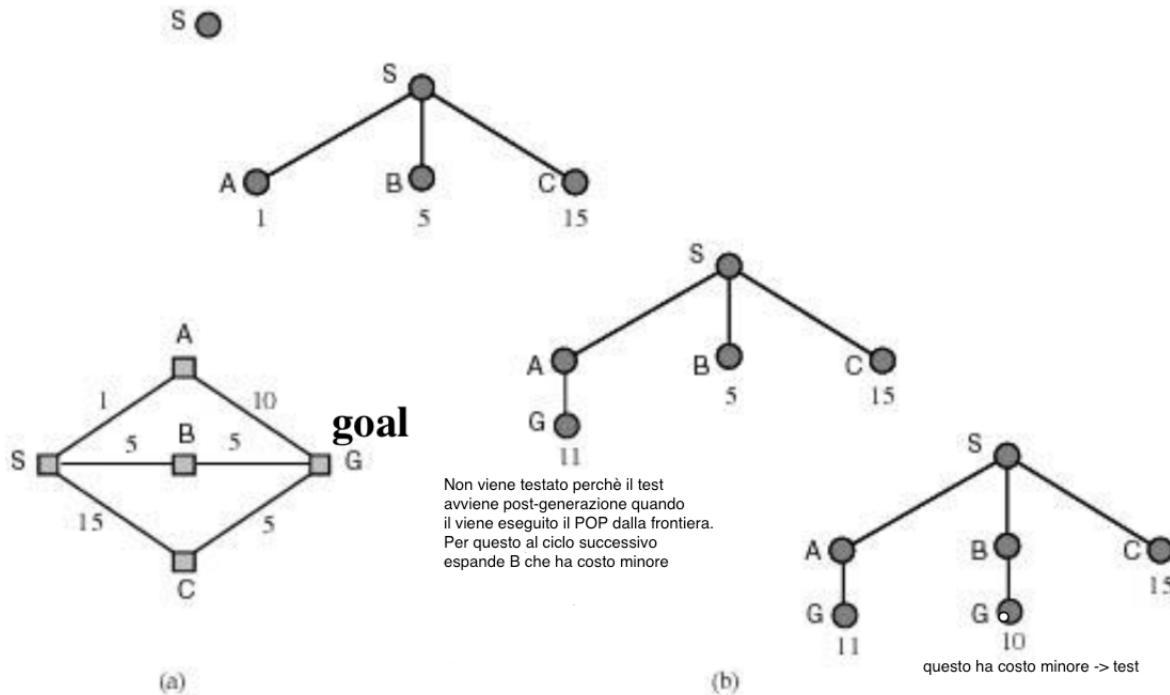
Nella ricerca bidirezionale si procede nelle due direzioni fino ad incontrarsi.

7.11.1 Analisi complessità Bidir.

- Strategia Completa: SI (se il branching è finito e si usa la ricerca BF)
- Strategia Ottimale: SI (se i costi sono tutti uguali e si usa la ricerca BF)
- Complessità Tempo: $O(b^{d/2})$
- Complessità Spazio: $O(b^{d/2})$

7.12 Ricerca di costo uniforme - UC

E' una generalizzazione della ricerca in ampiezza dove i costi di ogni operatore sono diversi. Si sceglie il nodo di costo minore sulla frontiera (costo $g(n)$) e si espande. La frontiera è implementata da una coda ordinata per costo cammino crescente (cioè per primi i nodi di costo g minore). I nodi vengono goal-testati quando vengono scelti per l'espansione e non quando sono generati!



7.12.1 UC-Albero

```

function Ricerca-UC-A (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    frontiera = una coda con priorità con nodo come unico elemento
    loop do
        if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera)
        if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            frontiera = Inserisci(figlio, frontiera) /* in coda con priorità
        end

```

7.12.2 UC-Grafo

```

function Ricerca-UC-G (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    frontiera = una coda con priorità con nodo come unico elemento
    esplorati = insieme vuoto
    loop do
        if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera);
        if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
        aggiungi nodo.Stato a esplorati
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            if figlio.Stato non è in esplorati e non è in frontiera then
                frontiera = Inserisci(figlio, frontiera) /* in coda con priorità
            else if figlio.Stato è in frontiera con Costo-cammino più alto then
                sostituisci quel nodo frontiera con figlio

```

g(n)

7.12.3 Analisi complessità UC

- Strategia Completa: SI se il costo degli archi x sia tale che $x \geq \alpha > 0$ ⁶
- Strategia Ottimale: SI se il costo degli archi x sia tale che $x \geq \alpha > 0$
- Complessità Tempo: $O(b^{1+\lfloor C^*/\alpha \rfloor})$
- Complessità Spazio: $O(b^{1+\lfloor C^*/\alpha \rfloor})$

Assumendo C^* come costo della soluzione ottima e $\lfloor C^*/\alpha \rfloor$ come numero di mosse al caso peggiore, arrotondato per difetto.

⁶dove α è una costante relativamente piccola, questo ci evita che UC non termini a causa di un cammino con infiniti passi di costo 0

7.13 Confronto finale delle strategie

	BF	UC	DF	DL	ID	Bidir.
Completa	SI	SI (-)	NO	SI (+)	SI	SI
Ottimale	SI (*)	SI (-)	NO	NO	SI (*)	SI
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\alpha \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\alpha \rfloor})$	$O(b * m)$	$O(b * l)$	$O(b * d)$	$O(b^{d/2})$

(*) se gli operatori hanno tutti lo stesso costo

(-) per costo degli archi x tale che $x \geq \alpha > 0$

(+) per problemi per cui si conosce un limite alla profondità della soluzione (se $d < l$)

8 Ricerca Euristica

La ricerca esaustiva non è praticabile in problemi di complessità esponenziale, come ad esempio gli scacchi che portano ad una complessità di 10^{120} . Possiamo invece usare la conoscenza del problema e l'esperienza per riconoscere cammini più promettenti, evitando di generarne di inutili e costosi. La conoscenza euristica non evita la ricerca ma la riduce e sotto certe condizioni può essere completa e ottimale.

Per indirizzare la ricerca utilizziamo una funzione di valutazione $f(n)$ che ci indica la qualità dello stato. La funzione di valutazione f include h , detta funzione di valutazione euristica.

Funzione di valutazione: $f(n) = g(n) + h(n)$ dove:

- $h : n \rightarrow \mathbb{R}$ (la funzione si applica al nodo n , ma dipende solo dallo stato - $n.\text{Stato}$)
- $g(n)$ è il costo del cammino

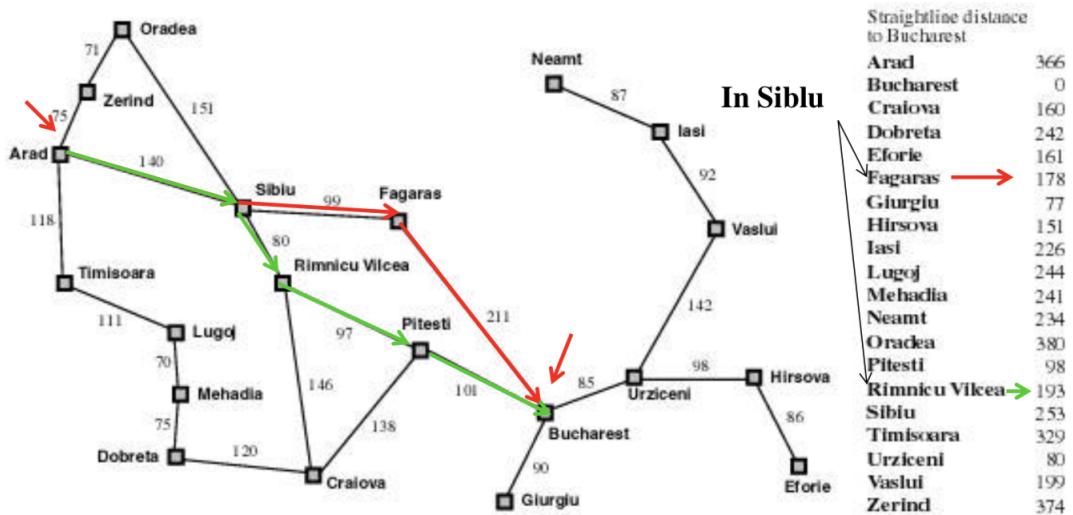
Esempi di euristica h possono essere la distanza in linea d'aria tra due città, numero di caselle fuori posto nel gioco dell'otto, numero di pezzi sulla scacchiera...

8.1 Algoritmo Best-First - BFH

Best-First-Heuristic si basa sullo stesso algoritmo di Uniform Cost (UC) ma con uso della funzione di valutazione f per la coda con priorità. La scelta di $f(n)$ quindi determina la strategia di ricerca. Ad ogni passo si sceglie il nodo sulla frontiera per cui il valore della f è migliore (migliore può assumere significati diversi in base al problema, ad esempio nel problema dell'itinerario significa "minore"). La $h(n)$ contenuta nella funzione di valutazione ha come valore il costo stimato del percorso sul cammino dal nodo n al nodo goal.

8.2 Algoritmo Greedy-Best-First - GBF

È un caso speciale dell'algoritmo Best-First in cui la funzione di valutazione è uguale alla funzione di valutazione euristica, cioè $f(n) = h(n)$. In pratica l'algoritmo cerca di espandere il nodo più vicino al goal secondo l'euristica.



Ci troviamo $\text{In}(\text{Arad})$ e aggiungiamo alla frontiera tutti i nodi adiacenti, la coda è ordinata secondo l'euristica della distanza in linea d'aria perché $f(n) = h(n)$. Da $\text{In}(\text{Arad})$ quindi andiamo $\text{Go}(\text{Sibiu})$, e ci troviamo $\text{In}(\text{Sibiu})$, vengono aggiunti i nodi adiacenti alla coda e per primo troviamo Faragás, $\text{Go}(\text{Faragás})$ e poi $\text{Go}(\text{Bucharest})$ perché ha distanza (ovviamente) 0 dal goal ed è primo nella coda. Notiamo però che l'algoritmo Greedy-Best-First ha trovato un cammino ma non l'ottimo, che sarebbe quello in verde.

8.3 Algoritmo A

Un algoritmo A è un algoritmo Best-First⁷ con una valutazione dello stato del tipo:

$$f(n) = g(n) + h(n) \text{ dove:}$$

- $h(n) \geq 0$
- $h(\text{goal})=0$
- $g(n)$ è il costo del cammino per arrivare a n
- $h(n)$ è una stima del costo per raggiungere il goal da un nodo n

L'algoritmo A nel caso in cui $h(n) = 0$ ottengo $f(n) = g(n) \rightarrow$ Uniform Cost.

Nel caso in cui $g(n) = 0$ ottengo $f(n) = h(n) \rightarrow$ Greedy-Best-First.

8.3.1 Completezza Algoritmo A

Teorema: L'algoritmo A è completo se $g(n) \geq d(n) * \alpha$ dove $d(n)$ è la profondità al nodo n e $\alpha > 0$ è il costo minimo dell'arco. Questo ci garantisce che non ci siano situazioni del tipo



⁷la frontiera è ordinata secondo il valore di f

8.3.2 Dimostrazione Completezza di A

Sia $n_1, n_2, \dots, n_i, \dots, n_k$ un cammino soluzione.

n_i è un nodo sulla frontiera di un cammino soluzione quindi prima o poi sarà espanso.

Se non verrà trovata una soluzione prima, il nodo n_i sarà espanso e i nodi del figlio aggiunti alla frontiera, tra questi anche il successore sul cammino soluzione. Il ragionamento si ripete fino a dimostrare che anche il nodo goal sarà selezionato per l'espansione.

8.4 Algoritmo A^*

L'algoritmo è identico a UC ad eccezione del fatto che A^* utilizza $f(n) = g(n) + h(n)$ per ordinare la coda. (dove h e g hanno le proprietà dell'algoritmo A e h è euristica ammissibile → def in seguito)
Definiamo una funzione di valutazione ideale detta anche oracolo, che idealizza la funzione di valutazione perfetta (cammino minimo).

Funzione di valutazione ideale: $f^*(n) = g^*(n) + h^*(n)$ dove

- $f^*(n)$ è il costo del cammino minimo da radice a goal passando per n
- $g^*(n)$ costo del cammino minimo da radice a n
- $h^*(n)$ costo del cammino minimo da n a goal

Normalmente $g(n) \geq g^*(n)$ perché il costo del cammino è \geq del costo del cammino migliore. Mentre $h(n)$ è una stima di $h^*(n)$, si può andare in sottostima o sovrastima dalla distanza della soluzione.

Una euristica è ammissibile se $\forall n. h(n) \leq h^*(n)$ cioè h è sottostima (lower-bound condition).

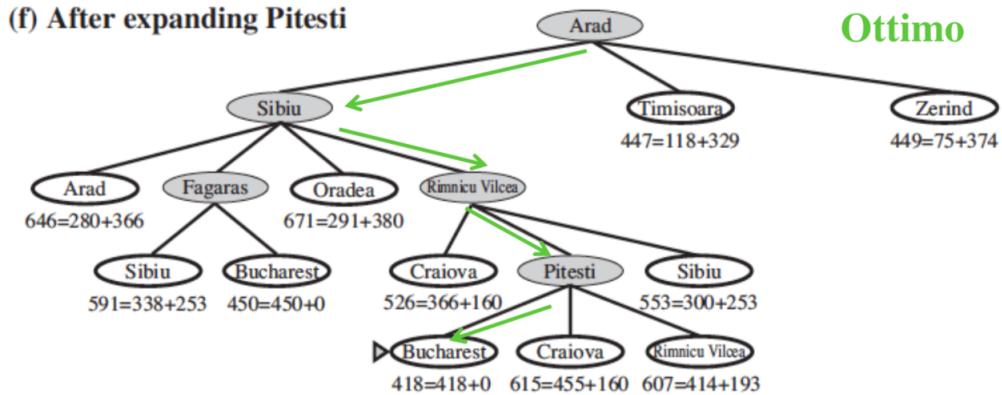
Possiamo quindi ora dare una definizione di un algoritmo A^* :

Un algoritmo A^* è un algoritmo A in cui h è una funzione euristica ammissibile. (Gli algoritmi A^* sono ottimali).

(quindi anche BreadthFirst con passi a costo costante e UC sono ottimali perché sono A^* con $h(n)=0$)

8.4.1 Esempio A^* sul problema dell'itinerario

Ho come euristica la distanza in linea d'aria, quindi è una sottostima ⇒ Euristica Ammissibile.



8.4.2 Osservazioni su A^*

La componente $g(n)$ invece fa sì che si abbandonino cammini che vanno troppo in profondità.

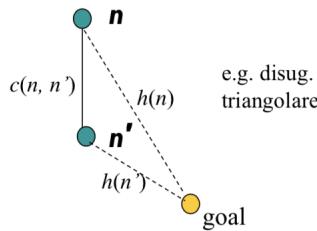
Una euristica sottostima può farci fare del lavoro inutile ma non ci fa perdere il cammino migliore, una sovrastima invece può farci perdere la soluzione ottimale a causa di un taglio che però in realtà poteva essere buono.

8.4.3 Ottimalità di A^*

Come abbiamo già detto gli algoritmi A^* sono ottimali.

- nel caso di ricerca su albero l'uso di una euristica ammissibile è sufficiente a garantire l'ammissibilità e quindi l'ottimalità di A^* .
- nel caso di ricerca su grafo abbiamo bisogno di definire una proprietà chiamata consistenza (o monotonicità).

8.4.4 Euristiche Consistenti (o monotone)



Una euristica è consistente se:

- $h(\text{goal}) = 0$
- $\forall n. h(n) \leq c(n, \text{azione}, n') + h(n')$ dove n' è successore di n e $h(n')$ consistenza locale
ne segue che $f(n) \leq f(n')$

Se $h(n)$ è consistente la $f(n)$ non decresce mai lungo i cammini, da cui il termine monotona.

Una euristica monotona è ammissibile (ci sono però euristiche ammissibili non monotone ma sono molto rare). Le euristiche monotone garantiscono che la soluzione meno costosa venga trovata per prima e quindi sono ottimali anche nel caso di ricerca su grafo.

8.4.5 Dimostrazione ottimalità di A^*

Se $h(n)$ è consistente, i valori di $f(n)$ lungo un cammino sono crescenti.

$$\begin{aligned} h(n) &\leq c(n, a, n') + h(n') \\ g(n) + h(n) &\leq g(n) + c(n, a, n') + h(n') // \text{sommo } g(n) \text{ da entrambe le parti} \\ g(n) + h(n) &\leq g(n') + h(n') // \text{perché } g(n) + c(n, a, n') = g(n') \\ f(n) &\leq f(n') \end{aligned}$$

Ogni volta che A^* seleziona un nodo n per l'espansione, il cammino ottimo a tale nodo è stato trovato.

8.4.6 Vantaggi di A^*

- A^* espande tutti i nodi con $f(n) < C^*$
- A^* espande alcuni i nodi con $f(n) = C^*$
- A^* non espande alcun nodo con $f(n) > C^*$
- Assumendo C^* come costo della soluzione ottima.

Pruning: alcuni nodi non vengono espansi a causa delle regole sopra scritte risparmiando così memoria e rimanendo ottimali. Quindi una $h(n)$ opportuna fa tagliare molto.

Cercheremo quindi una $h(n)$ il più alta possibile tra le ammissibili poiché se molto bassa i nodi restano sempre minori di C^* e li espando tutti.

A^* è completo (discende dalla completezza di A), A^* con euristica monotona è ottimale. Il problema quindi è quale euristica utilizzare? Lo spazio in memoria è ancora molto grande $O(b^{d+1})$ (il costo in tempo dipende dall'euristica utilizzata).

8.5 Costruire le euristiche di A^*

A parità di ammissibilità una euristica può essere più efficiente di un'altra nel trovare il cammino soluzione migliore. Questo dipende dal grado di informazione posseduto dall'euristica.

$$\begin{aligned} h(n) &= 0 // \text{grado di informazione minimo} \\ h^*(n) & // \text{massimo grado di informazione (oracolo)} \end{aligned}$$

In generale per le euristiche ammissibili $0 \leq h(n) \leq h^*(n)$.

Più informata significa più efficiente:

Se $h_1 \leq h_2$, i nodi espansi da A^* con h_2 sono un sottoinsieme di quelli espansi da A^* con h_1 . Cioè A^* con h_2 è almeno efficiente quanto A^* con h_1 . (In questo caso si dice che h_2 domina su h_1)

Una euristica più informata (accurata) riduce lo spazio di ricerca ma è tipicamente più costosa da calcolare (più informata \rightarrow taglia di più).

8.6 Valutare Algoritmi di ricerca euristica

Un modo per caratterizzare la qualità di una euristica è il fattore di diramazione effettivo b^* .

Se:

- N : numero di nodi generati da A^*
- d : profondità della soluzione

allora b^* è il fattore di diramazione effettivo che avrebbe un albero uniforme di profondità d per ottenere un albero con $N + 1$ nodi.

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Ad esempio, se A^* trovasse una soluzione alla profondità 5, usando 52 nodi, il fattore di diramazione effettivo b^* sarebbe di 1.92.

Sperimentalmente una buona euristica ha un b^* abbastanza vicino a 1 (< 1.5) in modo tale da rendere problemi con dimensionalità alta, risolvibili in un tempo computazionale ragionevole.

Possiamo vedere come cambia in base ad alcuni dati:

- con $b^* = 2$ otteniamo a profondità $d = 6$, $N = 127$
- con $b^* = 1.7$ otteniamo a profondità $d = 6$, $N = 57$

Possiamo notare come migliorando di poco l'euristica allo stesso livello abbiamo espanso la metà dei nodi. Quindi migliorando l'euristica a parità di nodi espansi riesco a raggiungere una profondità doppia.

8.7 Inventare euristiche

- Rilassamento del problema: rimuovo dei vincoli (versioni semplificate). Ad esempio nel gioco dell'otto rimuovo i vincoli di mosse solo nelle caselle adiacenti libere (distanza Manhattan dalla sua corretta posizione).
- Massimizzazione euristiche: se ho k euristiche ammissibili senza che alcuna sia migliore dell'altra allora ogni volta conviene prendere il massimo dei loro valori: $h(n) = \max\{h_1(n), h_2(n), \dots, h_k(n)\}$. Se tutte le h_i sono ammissibili allora anche h lo è, e domina su tutte le altre.
- (Sottoproblemi) Database di pattern: ho dei database in cui sono memorizzati dei pattern di sottoproblemi con relativo costo e soluzione. Nel caso di database di pattern normali la somma delle euristiche di questi sottoproblemi non è accurata perché potrebbe essere che due soluzioni tra due sottoproblemi interferiscono l'una con l'altra. Nel caso invece di database di pattern disgiunti è consentita la somma dei costi.
- Combinazione lineare: si esegue una combinazione lineare di euristiche diverse: $h(n) = c_1h_1(n) + c_2h_2(n) + \dots + c_kh_k(n)$
- Apprendere dall'esperienza: faccio girare il programma raccogliendo dati in forma di coppie (stato, h^*) e uso i dati per capire come predire la $h(n)$ con algoritmi di apprendimento induttivo.

8.8 Algoritmi evoluti basati su A^*

Sono algoritmi che si preoccupano di occupare meno memoria⁸.

8.8.1 Beam Search

La Beam Search salva in frontiera ad ogni passo solo i k nodi più promettenti dove k è l'ampiezza del raggio (in pratica nella frontiera abbiamo i k migliori di tutta la storia dell'algoritmo), ovviamente l'algoritmo non è più completo così facendo.

8.8.2 A^* con approfondimento iterativo - IDA^*

IDA^* combina A^* con ID, ad ogni iterazione si ricerca in profondità con un limite (cut-off) dato dal valore della funzione $f(n)$ e non dalla profondità. Il valore f -limit viene aumentato ad ogni iterazione fino a trovare la soluzione (come in ID si aumenta di un livello alla volta). Ma di quanto aumento?

- Nel caso di costo fisso di ogni azione il limite viene aumentato di questo costo.
- Nel caso in cui i costi siano variabili il limite viene aumentato del costo minimo oppure ad ogni passo scelgo il valore minimo delle $f(n)$ scartate perché in quanto superavano il limite all'iterazione precedente.

IDA^* è completo e ottimale se:

- le azioni hanno costo costante k e l' f -limit viene aumentato di k
- le azioni hanno costo variabile e l'incremento di f -limit è \leq del costo minimo degli archi
- il nuovo f -limit è il minimo valore $f(n)$ dei nodi generati ed esclusi all'iterazione precedente

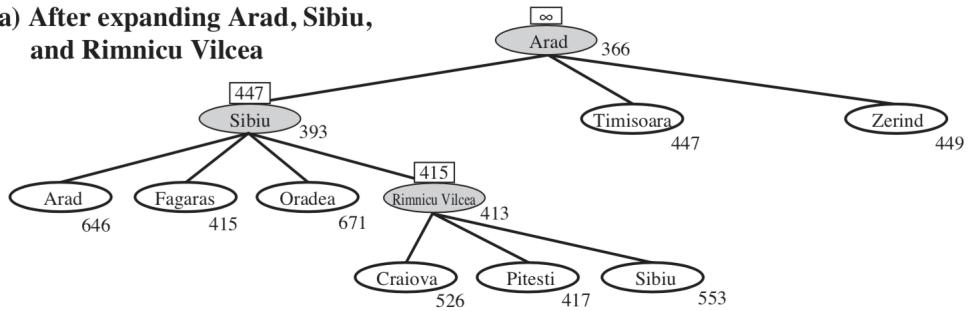
IDA^* occupa $O(b * d)$ memoria.

⁸Le limitazioni di memoria possono rendere un problema intrattabile dal punto di vista computazionale

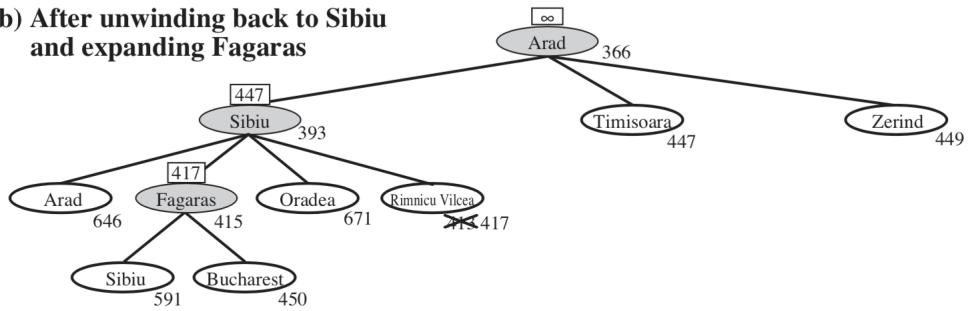
8.8.3 Best-First ricorsivo - RBFS

L'algoritmo è strutturalmente simile a Depth-First Ricorsivo ma cerca di usare meno memoria facendo del lavoro in più, in pratica cerca di imitare Best-First ma usando spazio lineare. Ad ogni livello tengo traccia del migliore percorso alternativo sfruttando la variabile f-limit confrontata con l'f-value⁹ del cammino. In caso di fallimento interrompe l'esplorazione quando trova un nodo meno promettente secondo f(n). Nel tornare indietro si ricorda del migliore nodo che ha trovato nel sottoalbero esplorato, per poterci eventualmente tornare. L'occupazione di memoria è lineare nella profondità della soluzione ottima $O(d)$.

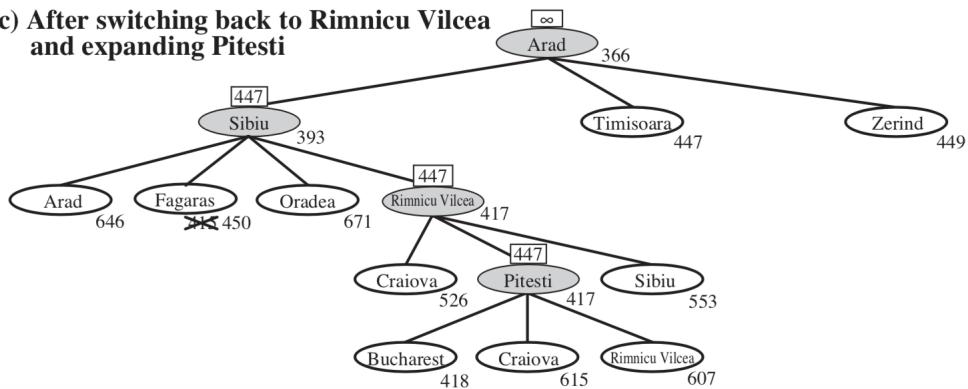
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



(c) After switching back to Rimnicu Vilcea and expanding Pitesti



8.8.4 A^* con memoria limitata - SMA*

SMA^* procede come A^* fino ad esaurimento della memoria disponibile, a quel punto dimentica il nodo peggiore dopo aver aggiornato il valore del padre. A parità di f(n) si dimentica il nodo peggiore più vecchio e tiene il più recente. L'algoritmo è ottimale se il cammino soluzione sta in memoria.

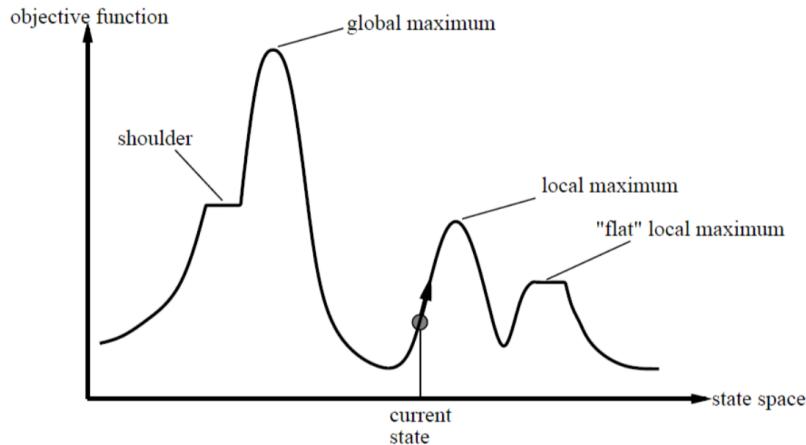
⁹valore della funzione f(n)

9 Ricerca Locale

Gli algoritmi di ricerca locale sono adatti per problemi in cui la sequenza delle azioni non è importante, quello che conta è lo stato di goal. Essi non sono sistematici, tengono traccia solo del nodo corrente e si spostano su nodi adiacenti. Non tengono traccia dei cammini poiché non servono in uscita, questo li rende efficienti in memoria e possono trovare soluzioni ragionevoli anche in spazi molto grandi e infiniti (continui).

9.1 Spazio degli stati

Lo stato migliore viene valutato da una funzione obiettivo f .



Uno stato ha una altezza che corrisponde al valore della funzione obiettivo (funzione costo euristico). Un algoritmo provoca movimento sulla superficie e il suo compito è quello di trovare il minimo o il massimo della funzione in base alla richiesta del problema (se stiamo cercando un costo cerchiamo il minimo).

9.2 Algoritmo Hill-Climbing

E' un algoritmo di ricerca locale greedy. Vengono generati i successori e valutati, viene scelto un nodo che migliora la valutazione dello stato attuale (non si tiene traccia degli altri nodi). Quale nodo scelgo?

- il migliore: Hill-Climbing a salita rapida.
- uno a caso tra quelli che migliorano: Hill-Climbing stocastico.
- il primo trovato: Hill-Climbing con prima scelta.

Se non ci sono stati successori migliori l'algoritmo termina. Non c'è frontiera a cui ritornare, si tiene in memoria solo lo stato corrente.

```

function Hill-climbing (problema)
    returns uno stato che è un massimo locale
    nodo-corrente = CreaNodo(problema.Stato-iniziale)
    loop do
        vicino = il successore di nodo-corrente di valore più alto
        if vicino.Valore  $\leq$  nodo-corrente.Valore then
            return nodo-corrente.Stato // interrompe la ricerca
        nodo-corrente = vicino
        // (altrimenti, se vicino e' migliore, continua)
    
```

Figure 2: Codice algoritmo Hill-Climbing

9.2.1 Problemi e Miglioramenti per Hill-Climbing

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". Possiamo però eseguire dei miglioramenti:

- Consentire un numero limitato di mosse laterali, cioè l'algoritmo si modifica fermandosi quando $nodoVicino.Valore < nodoCorrente.Valore$ invece che \leq .
- Si sfrutta Hill-Climbing stocastico scegliendo a caso tra le mosse in salita. L'algoritmo converge più lentamente ma a volte trova soluzioni migliori.
- Si sfrutta Hill-Climbing con prima scelta, può generare mosse a caso fino a trovarne una migliore dello stato corrente, diventa più efficace quando i successori sono molti.
- Si sfrutta Hill-Climbing con rinvio casuale (random restart), cioè si riparte da un punto scelto a caso. Se la probabilità di successo è p saranno necessarie in media $1/p$ ripartenze per trovare la soluzione. Questo algoritmo è tendenzialmente completo.

9.3 Algoritmo Tempra Simulata (Simulated Annealing)

Hill-Climbing non esegue mai mosse in discesa, quindi viene facile capire che è garantita l'incompletezza. L'algoritmo di Tempra Simulata combina Hill-Climbing con una scelta stocastica ben studiata, ad ogni passo si sceglie un successore a caso:

- se migliora lo stato viene espanso (migliora lo stato significa che la funzione di valutazione è maggiore, cioè $[\Delta E = f(n') - f(n)] \geq 0$)
- altrimenti se peggiora lo stato (caso $[\Delta E = f(n') - f(n)] < 0$) scelgo il nodo n' con probabilità $p = e^{\Delta E/T}$ dove T è un parametro che nel progredire dell'algoritmo decresce, facendo così decresce anche p rendendo improbabili le mosse peggiorative.

Il valore per cui T decresce è dato in input come parametrc¹⁰, se T diminuisce lentamente si raggiunge un ottimo globale con probabilità tendente ad 1.

¹⁰I valori di T iniziali determinati sperimentalmente sono tali che $p = e^{\Delta E/T}$ sia all'incirca 0,5

9.4 Algoritmo Local Beam

Tenere un solo nodo in memoria può sembrare una soluzione estrema al problema dello spazio... Local Beam la versione locale della Beam Search, si tengono in memoria K stati anziché uno solo. Si parte con K stati generati randomicamente e ad ogni passo si generano tutti i successori dei K stati:

- Se si trova un goal ci si ferma
- Altrimenti si prosegue con i K migliori tra quelli della lista completa

È diverso dalla Beam Search normale perché tengo in memoria solo i K migliori del passo corrente e non di tutta la storia. Possiamo notare come se $K=1$ abbiamo Hill-Climbing con prima scelta mentre se $K=\infty$ ho Hill-Climbing a salita rapida.

9.4.1 Local Beam Search Stocastica

Si introduce un elemento di casualità. Al posto di scegliere i migliori K successori dalla lista completa, si scelgono i K successori in modo randomico ma con probabilità maggiore di prendere i migliori.

9.5 Algoritmi Genetici - GA

Gli algoritmi genetici sono varianti della Beam Search Stocastica in cui gli stati successori sono ottenuti combinando due stati genitore. Chiamiamo fitness il valore della funzione obiettivo.

Abbiamo una popolazione iniziale, cioè un insieme di K stati iniziali, generati casualmente. Ogni individuo (stato) è rappresentato da una stringa definita su un alfabeto finito. Ogni individuo viene valutato da una funzione di fitness, si selezionano gli individui per gli accoppiamenti sulla base di una probabilità proporzionale alla fitness. Le coppie scelte¹¹ danno vita ad una "generazione" successiva con la combinazione di due metodi:

- Crossover (combinando materiale genetico): per ogni coppia viene scelto un punto di crossing over e ogni stato viene diviso in due parti ottenendo in totale 4 parti. Vengono poi generati due figli scambiandosi i pezzi ottenuti.
- Con un meccanismo casuale aggiuntivo di mutazione genetica: viene infine effettuata una mutazione casuale che dà luogo alla prossima generazione

La popolazione ottenuta dovrebbe esser migliore.

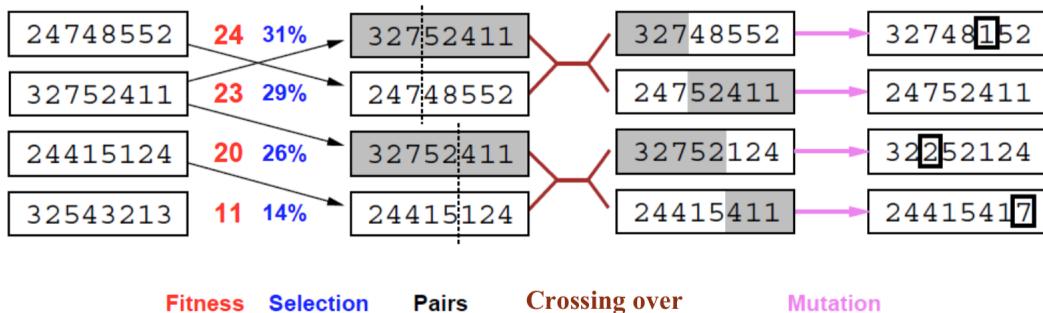


Figure 3: Esempio crossover + mutazione casuale

Gli algoritmi genetici combinano la tendenza a salire della Beam Search Stocastica e la possibilità di interscambio di informazioni tra thread paralleli. Però il punto critico di questi algoritmi è la rappresentazione del problema in stringhe.

¹¹Nel problema delle regine scegliamo le coppie che non si "attaccano"

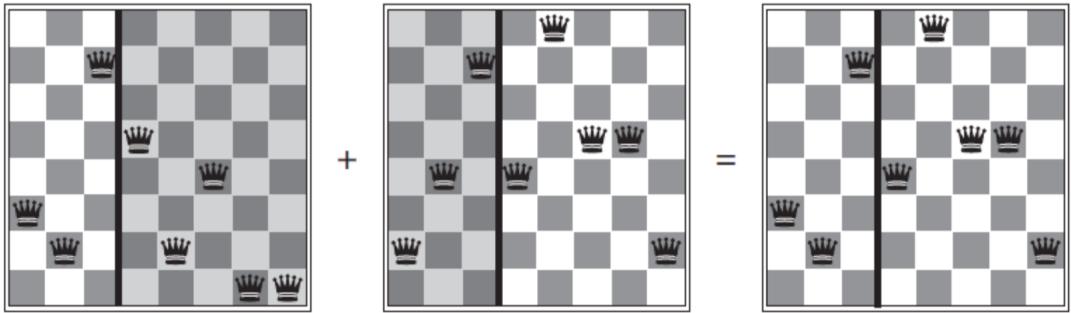


Figure 4: Esempio sul problema delle 8 regine

9.6 Gradiente (Hill-Climbing iterativo)

Molti casi nel mondo reale hanno spazi di ricerca infiniti. Lo stato viene descritto da variabili continue x_1, x_2, \dots, x_n rappresentate anche da un vettore \mathbf{x} (ad esempio nello spazio 3D abbiamo $\mathbf{x} = (x_1, x_2, x_3)$). Avere più dimensioni può sembrare ostico, ma abbiamo molti strumenti matematici che ci permettono di semplificare.

Se la funzione è continua e differenziabile, il minimo e il massimo può essere cercato utilizzando il gradiente, che restituisce la direzione di massima pendenza nel punto.

Data f obiettivo:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)$$

dove ∂ è la derivata parziale e ∇ è inteso come calcolo vettoriale. In alcuni casi possiamo trovare il massimo risolvendo l'equazione $\nabla f = 0$ (calcolo la derivata, vedo dove si annulla e ottengo i punti di min/max), ma in altri casi non è possibile risolverla in forma chiusa. Definiamo Hill-Climbing iterativo come

$$x_{new} = x_{old} + \eta \nabla f(x)$$

dove η è lo step-size (di quanto aumentare).

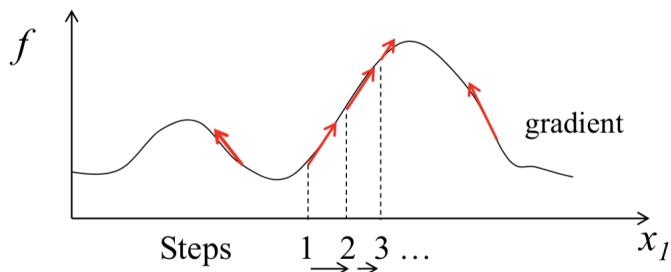


Figure 5: Esempio con una dimensione x_1 , gli spostamenti sono guidati dal gradiente

10 Oltre la ricerca classica

Gli agenti risolutori di problemi "classici", visti fino ad ora, assumono ambienti completamente osservabili e deterministici. Questo gli permette di esplorare offline lo spazio degli stati in ricerca di un goal e restituendo una sequenza di azioni che può essere eseguito senza imprevisti per raggiungere l'obiettivo. Per avvicinarci alla realtà dobbiamo riconsiderare il nostro ambiente valutando azioni non deterministiche e ambienti parzialmente osservabili.

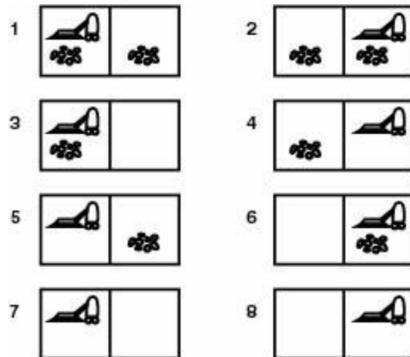
In un ambiente parzialmente osservabile e non deterministico le percezioni sono importanti perché restringono gli stati possibili e informano sull'effetto dell'azione eseguita. L'agente che si trova in questo ambiente può elaborare una strategia che tiene conto delle differenti eventualità: un piano di contingenza.

10.1 Problema dell'aspirapolvere non deterministico

Analizziamo il comportamento dell'aspirapolvere:

- Se aspira in una stanza sporca → la pulisce, ma talvolta pulisce anche la stanza adiacente.
- Se aspira in una stanza pulita a volte rilascia dello sporco.

Il modello di transizione restituisce un insieme di stati ma l'agente non sa in quale si troverà. Il piano di contingenza sarà un piano (un albero) condizionale con probabili cicli.



Analizzando lo spazio degli stati, Risultato(Aspira,1)¹² mi può portare negli stati 5 o 7. Per rappresentare il problema possiamo usare gli alberi di ricerca AND-OR.

10.2 Alberi AND-OR

- Nodi OR → scelte dell'agente.
- Nodi AND → le diverse eventualità da considerare tutte (ambiente non deterministico!).

Una soluzione a un problema di ricerca AND-OR è un albero che ha un nodo obiettivo in ogni foglia, specifica un'unica azione nei nodi OR e include tutti gli archi uscenti da nodi AND che visita (possiamo notare che sono le caratteristiche della soluzione in foto sottostante).

¹²esegue l'azione di aspirare mentre si trova nello stato 1

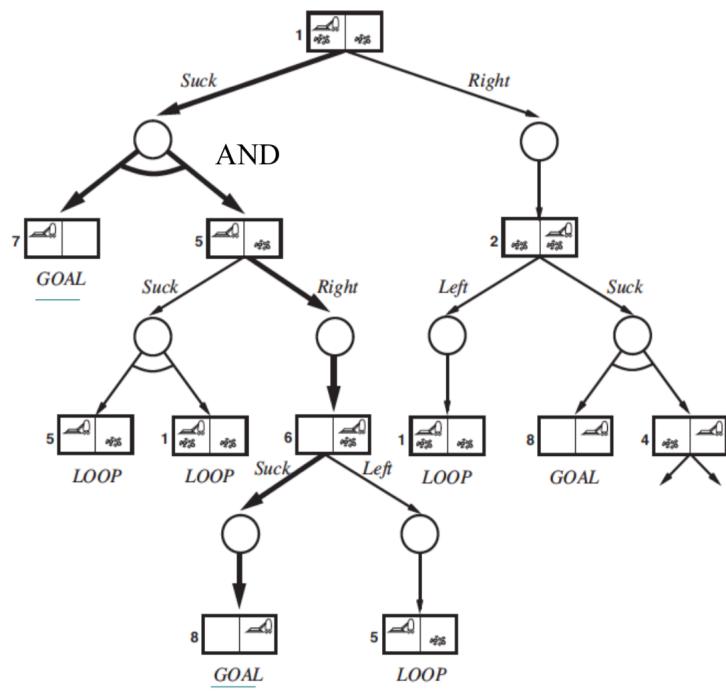


Figure 6: Esempio albero AND-OR problema dell’aspirapolvere. La soluzione è in grassetto.

Appunti di Introduzione all’Intelligenza Artificiale Unipi - 2

Parte

Raffaele Apetino

Marzo 2020

Contents

1 Giochi con avversario	2
1.1 Ciclo Pianifica-Agisci-Percepisci	2
1.2 Definizione spazio degli stati	2
1.3 Game Tree	2
1.4 Algoritmo MIN-MAX	3
1.4.1 Come si calcola il valore MIN-MAX?	3
1.4.2 Costo MIN-MAX	4
1.5 Algoritmo MIN-MAX euristico (con orizzonte)	4
1.5.1 Come si calcola il valore MINIMAX euristico (H-MINIMAX)?	5
1.5.2 Esempio H-MIN-MAX in Tic-Tac-Toe	5
1.6 Problemi con MIN-MAX	5
1.7 Ottimizzazione MIN-MAX	5
1.8 Potatura ALFA-BETA	6
1.8.1 Costo potatura ALFA-BETA	7
1.8.2 Ottimizzazioni ALFA-BETA	7
1.9 Giochi Multiplayer	7
1.10 Giochi Complessi	7
2 Problemi di soddisfacimento di vincoli (CSP)	8
2.1 Formulazione di problemi CSP	8
2.1.1 Problema della colorazione di una mappa	8
2.1.2 Problema 8 regine	9
2.2 Strategie per problemi CSP	9
2.3 Ricerca per problemi CSP	9
2.4 Ricerca con backtracking (BT) a profondità limitata	9
2.5 Codice Algoritmo backtracking	10
2.5.1 Select-Unassigned-Variable()	10
2.5.2 Order-Domain-Values()	10
2.5.3 Inference()	10
2.5.4 Backtrack()	11
3 Agenti Logici KB	12
3.1 Approccio dichiarativo VS Approccio procedurale	12
3.2 Tell-Ask	12
3.3 Base di conoscenza o Base di dati?	12
3.4 Formalismi per la rappresentazione della conoscenza	13

4 Agenti Logici: Calcolo proposizionale	13
4.1 Sintassi (regole)	13
4.2 Semantica (significato)	13
4.3 Conseguenza logica	13
4.4 Il mondo del Wumpus	14
4.4.1 Esempio dal mondo del Wumpus	15
4.5 Formule logiche, Validità e Soddisfacibilità	15
4.6 Inferenza per calcolo proposizionale (PROP)	16
4.7 Algoritmo TT-Entails (Model Checking)	16
4.7.1 Esempio TT-Entails	16
4.8 Algoritmo DPLL (Alg. Soddisfacibilità)	16
4.8.1 Forma a clausole	16
4.8.2 Esempio DPLL	17
4.8.3 Miglioramenti DPLL	17
4.9 Algoritmo WALK-SAT	17
4.9.1 Esempio WALK-SAT	18
4.9.2 Analisi WALK-SAT	18
4.9.3 Problemi SAT difficili	18
4.10 Inferenza come deduzione	19
4.10.1 Alcune regole di inferenza	19
4.11 Regola di risoluzione	19
4.11.1 Regola di risoluzione in generale per PROP	19
4.11.2 Esempio grafo di risoluzione	20
4.11.3 Refutazione	20
4.11.4 Osservazioni	21
5 Agenti Logici: Logica del prim'ordine	21
5.1 Il mondo dei blocchi	21
5.2 Concettualizzazioni	22
5.3 FOL	22
5.3.1 Predicati	22
5.3.2 Termini	22
5.3.3 Formule	22
5.3.4 Interpretazione	22
5.4 Sematica Composizionale	23
5.4.1 Semantica Standard VS Semantica Database	23
5.5 Interazione con la KB tramite FOL	23
5.6 Regola di inferenza per \forall	23
5.7 Regola di inferenza per \exists	24
5.8 Riduzione a inferenza proposizionale	24
5.8.1 Teorema di Herbrand	24
5.9 Verso un metodo di risoluzione per il FOL	24
5.9.1 Forma a clausole	24
5.9.2 Esempio di trasformazione (passo passo)	25
5.10 Sostituzione	25
5.11 Unificazione	26
5.11.1 Algoritmo di unificazione	26
5.11.2 Esempi	26
5.12 Metodo di risoluzione per FOL	27
5.12.1 Esempio metodo di risoluzione	27
5.12.2 Problemi	28
5.12.3 Completezza del metodo di risoluzione	28

5.13	Refutazione	28
5.13.1	Esempio di Refutazione	28
5.14	Risoluzione efficiente per FOL	29
5.14.1	Strategie di cancellazione	30
5.14.2	Strategie di restrizione	30
5.14.3	Strategie di ordinamento	31
5.15	Sistemi a regole	32
5.15.1	Regole in avanti e indietro	32
5.16	Programmazione logica (Backward Chaining)	32
5.16.1	Risoluzione SLD	33
5.16.2	Alberi di risoluzione SLD	33
5.16.3	Esempio di albero SLD	33
5.17	Sistemi a regole in avanti (Forward Chaining - FOL_FC_Ask)	34
5.17.1	Esempio di concatenazione in avanti	34
5.17.2	Analisi di FOL_FC_Ask	35

1 Giochi con avversario

Il modello base a cui ci siamo affidati fin'ora è realizzato su ambienti osservabili, deterministici e con utente singolo. Nella prima parte del corso abbiamo visto che esistono gli ambienti multi-agente, in cui un agente deve tenere conto anche delle azioni degli altri agenti che lo circondano. I giochi con avversario¹ si basano su ambienti deterministici multi-agente competitivi, in particolare su un ambiente reso strategico a causa della presenza di un avversario.

Vedremo i problemi di soddisfacimento dei vincoli (CSP) in cui lo stato ha una struttura fattorizzata. Vedremo che nei sistemi basati su conoscenza lo stato è una "base di conoscenza" (KB) a cui rivolgere domande sul mondo rappresentato in un linguaggio espressivo come il PROP (calcolo proposizionale) o FOL (logica del primo ordine).

1.1 Ciclo Pianifica-Agisci-Percepisci

Ci troviamo nel caso in cui abbiamo due agenti che agiscono a turno, si pianifica considerando le possibili risposte dell'avversario e le risposte alla possibile sua risposta e così via. Una volta decisa la mossa migliore da fare, si agisce, si percepisce la mossa dell'avversario e infine si ri-pianifica la mossa. La decisione ottima teorica è definita come la mossa migliore in un gioco con uno spazio di ricerca completamente esplorabile (per trovarla utilizzeremo l'algoritmo MIN-MAX). E' possibile che ci troveremo in situazioni in cui a causa della complessità non sarà possibile eseguire una esplorazione esaustiva dello spazio. Sfrutteremo, inoltre, tecniche di ottimizzazione della ricerca (algoritmo ALFA-BETA).

1.2 Definizione spazio degli stati

Un gioco può essere definito formalmente come una sorta di problema di ricerca con i seguenti elementi:

- Stati: configurazioni del gioco
- Stato iniziale: configurazione iniziale del gioco
- Player(s): a chi tocca eseguire l'azione nello stato s
- Actions(s): mosse legali in s
- Result(s,a): stato risultante dopo aver eseguito la mossa a nello stato s (modello di transizione)
- Terminal-Test(s): determina la fine del gioco, controlla che s sia lo stato terminale (e quindi il gioco finito)
- Utility(s,p): funzione di utilità (chiamata anche funzione obiettivo o funzione pay-off) che restituisce un valore numerico che valuta gli stati terminali del gioco per p (ad esempio somma di punteggi ecc...)

1.3 Game Tree

Lo stato iniziale insieme alle funzioni Actions e Result definiscono l'albero di gioco (Game Tree). È un albero dove ogni nodo è uno stato del gioco e i rami definiscono una mossa. Ogni foglia definisce uno stato terminale del gioco con associato di un valore di utilità definito dalla funzione Utility. Il prossimo esempio che vedremo è il gioco del Tris, i due giocatori si alternano tra MAX che posiziona le X e MIN che posiziona i cerchi, fino a che si arriva ad una delle foglie dell'albero che rappresentano un nodo terminale.

¹due giocatori, turni alterni, a somma zero (se uno vince, l'altro perde)

1.4 Algoritmo MIN-MAX

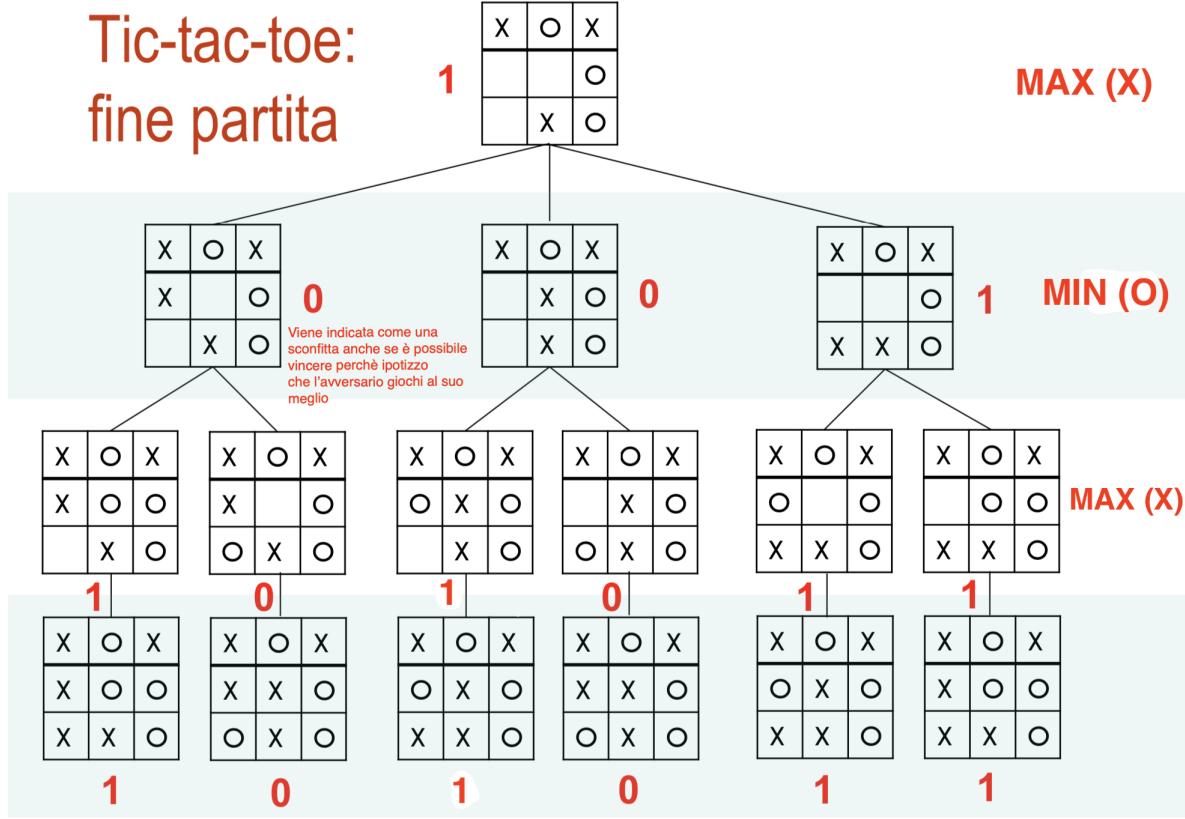


Figure 1: Albero parziale costruito dall'algoritmo MIN-MAX

L'albero creato da MIN-MAX è molto simile ad un albero AND-OR, in cui i nodi MAX hanno il ruolo di OR e MIN quello di AND.

1.4.1 Come si calcola il valore MIN-MAX?

Dato un albero di gioco, la strategia ottima è determinata dal valore "MINIMAX" di ogni nodo:

$$\text{MINIMAX}(s) =$$

- if (Terminal-Test(s) == true) Utility(s,MAX)
- if (Player(s) == MAX) $\max_{a \in \text{Actions}(s)}(\text{MINIMAX}(\text{Result}(s,a)))$
- if (Player(s) == MIN) $\min_{a \in \text{Actions}(s)}(\text{MINIMAX}(\text{Result}(s,a)))$

Ovviamente il valore MINIMAX di uno stato terminale è la sua utilità. Altrimenti, per ogni possibile azione (e quindi ramo), MAX preferisce spostarsi sui nodi in cui il valore MINIMAX è massimo, MIN su quelli di valore minimo.

L'albero di gioco conviene esplorarlo in profondità, perché in ampiezza occuperei troppa memoria, ma soprattutto a noi interessa trovare uno dei tanti percorsi di vittoria.

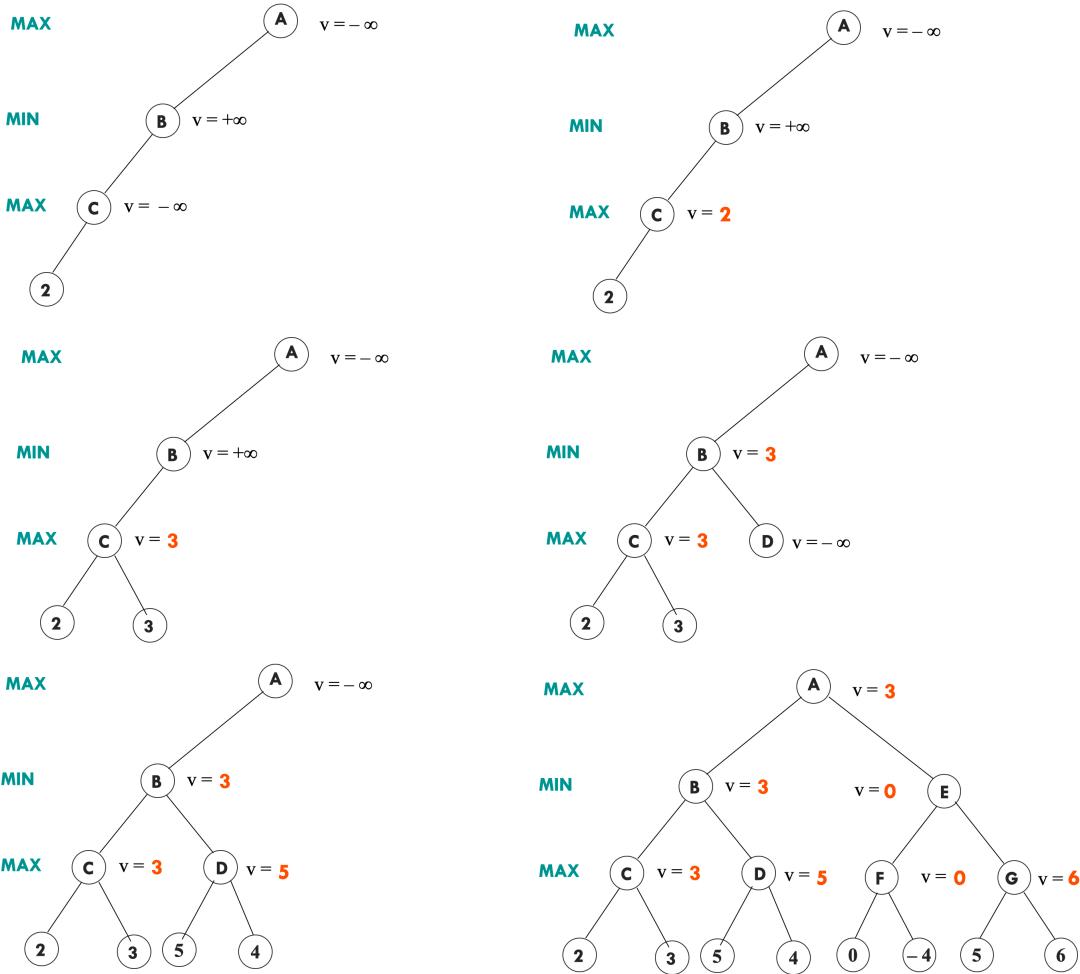


Figure 2: Algoritmo MIN-MAX in azione

1.4.2 Costo MIN-MAX

- Tempo: $O(b^m)$
- Spazio: $O(m)$

In giochi complessi come gli scacchi abbiamo un albero di ricerca molto grande 35^{100} (35 mosse in media, 100 mosse in media per ogni partita quindi 50 mosse per player), purtroppo anche mantenendo solo i nodi distinti, il grafo di ricerca rimane ampio: (10^{40}) nodi! È quindi improponibile una esplorazione sistematica, se non per giochi veramente semplici. Troviamo la necessità di far uso di euristiche per stimare il valore di uno stato del gioco.

1.5 Algoritmo MIN-MAX euristico (con orizzonte)

Nei casi più complessi quindi occorre usare una funzione di valutazione euristica dello stato $\text{Eval}(s)$. La strategia che applicheremo sarà quella di espandere l'albero di ricerca un certo numero di "d" livelli, si valutano gli stati ottenuti e si propaga indietro il risultato con la regola del MAX e MIN. Quindi al posto di eseguire un Terminal-Test eseguiremo un CutOff-Test che ci dirà quando eseguire la funzione di valutazione.

1.5.1 Come si calcola il valore MINIMAX euristico (H-MINIMAX)?

$H\text{-MINIMAX}(s,d) =$

- if ($\text{CutOff-Test}(s,d) == \text{true}$) $\text{Eval}(s)$
- if ($\text{Player}(s) == \text{MAX}$) $\max_{a \in \text{Actions}(s)} H - \text{MINIMAX}(\text{Result}(s,a), d+1)$
- if ($\text{Player}(s) == \text{MIN}$) $\min_{a \in \text{Actions}(s)} H - \text{MINIMAX}(\text{Result}(s,a), d+1)$

1.5.2 Esempio H-MIN-MAX in Tic-Tac-Toe

La funzione di valutazione $\text{Eval}(s)$ è una stima dell'utilità attesa a partire da una certa posizione nel gioco. La "qualità" della funzione è fondamentale: deve essere consistente con la vera utilità se applicata a stati terminali del gioco (deve mantenere lo stesso ordinamento dei nodi), deve essere efficiente da calcolare (immaginiamo dei giochi a tempo) ma deve anche essere fortemente correlata alle probabilità di vittoria.

Usiamo come funzione di valutazione la differenza tra le $X(s)$ righe aperte per X e $O(s)$ righe aperte per O. La nostra euristica sarà: $\text{Eval}(s) = X(s) - O(s)$

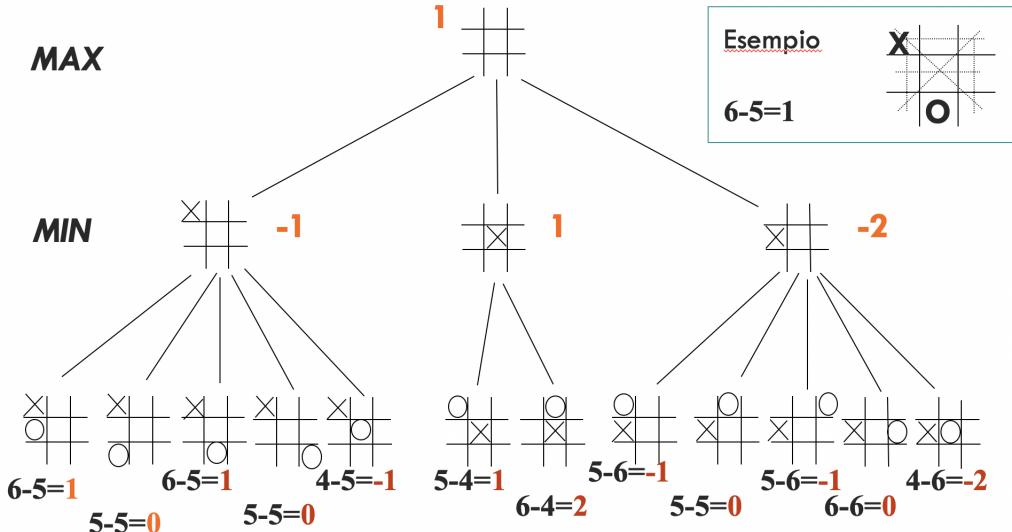


Figure 3: Algoritmo MIN-MAX euristico in azione

1.6 Problemi con MIN-MAX

- Stati non quiescenti: l'esplorazione fino ad un certo livello può mostrare una situazione molto vantaggiosa, ma alla mossa successiva la situazione può peggiorare esponenzialmente. La soluzione da applicare è la funzione di valutazione a stati quiescenti, cioè stati in cui $\text{Eval}(s)$ non è soggetto a mutamenti repentini (questo implica la ricerca di quiescenza).
- Effetto orizzonte: può succedere che vengano privilegiate mosse divisorie che hanno il solo effetto di spingere il problema oltre l'orizzonte. Una sorta di serie di mosse che non fanno altro che "scappare" dal problema il più possibile, magari ritrovandosi in un loop infinito.

1.7 Ottimizzazione MIN-MAX

E' necessario esplorare ogni cammino? NO! si può sfruttare un metodo per dimezzare la ricerca pur mantenendo una decisione della prossima mossa corretta (Potatura ALFA-BETA).

1.8 Potatura ALFA-BETA

Il problema principale della ricerca MIN-MAX è il numero di stati che cresce esponenzialmente in relazione alla profondità dell'albero. Purtroppo non possiamo eliminare l'esponente, ma possiamo dimezzarlo.

ALFA-BETA è una tecnica di potatura per ridurre l'esplorazione dello spazio di ricerca in algoritmi MIN-MAX. L'idea è che si vada avanti in profondità fino al livello desiderato, si propaghino indietro i valori e a questo punto si decida se si può abbandonare l'esplorazione nel sotto-albero. Sfruttiamo due valori "MaxValue" (α) e "MinValue" (β) rispettivamente inizializzati a $\alpha = -\infty$ e $\beta = +\infty$. Rappresentano rispettivamente la migliore alternativa per MAX e per MIN fino a quel momento.

I tagli del sottoalbero avvengono quando nel propagare indietro troviamo:

- $v \geq \beta$ per i nodi MAX (taglio β) β miglior alternativa per minimo
- $v \leq \alpha$ per i nodi MIN (taglio α) α migliore alternativa per massimo

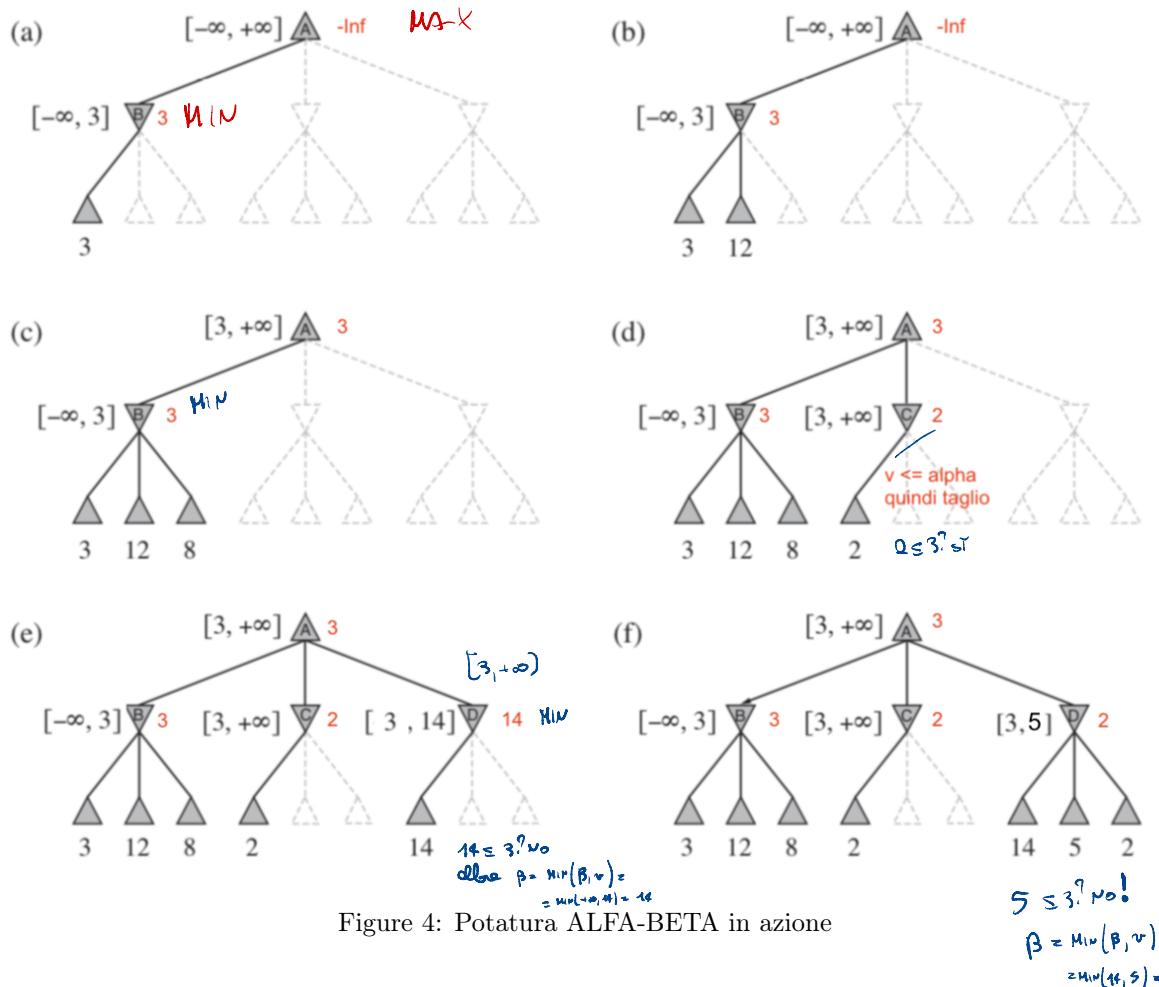


Figure 4: Potatura ALFA-BETA in azione

1.8.1 Costo potatura ALFA-BETA

- Tempo: $O(b^{m/2})$ quindi ALFA-BETA può arrivare a profondità doppia rispetto a MIN-MAX (best case)
- Spazio: $O(m)$

1.8.2 Ottimizzazioni ALFA-BETA

Quale potrebbe essere l'ordinamento ottimale?

- Ordinamento dinamico: usando un approfondimento iterativo si possono scoprire informazioni utili per l'ordinamento delle mosse da usare in una successiva iterazione (mosse killer). Possiamo inoltre tenere in memoria una tabella delle trasposizioni, cioè per ogni stato incontrato si memorizza la sua valutazione.
- Potatura in avanti: si potrebbe anche eseguire una potatura in avanti, cioè esplorare solo alcune mosse ritenute promettenti e tagliare le altre (tagli probabilistici basati su esperienza).
- Database di mosse: possiamo inoltre sfruttare un database di mosse di apertura e chiusura. Nelle prime fasi ci sono poche mosse sensate e ben studiate, inutile esplorarle tutte, mentre per le fasi finali il computer può esplorare off-line in maniera esaustiva e ricordarsi le migliori chiusure.

1.9 Giochi Multiplayer

Molti giochi permettono di far partecipare più di due giocatori. Dobbiamo modificare il singolo valore di ogni nodo con un vettore (ad esempio in foto vengono valutati allo stesso tempo 3 valori, uno per ogni giocatore).

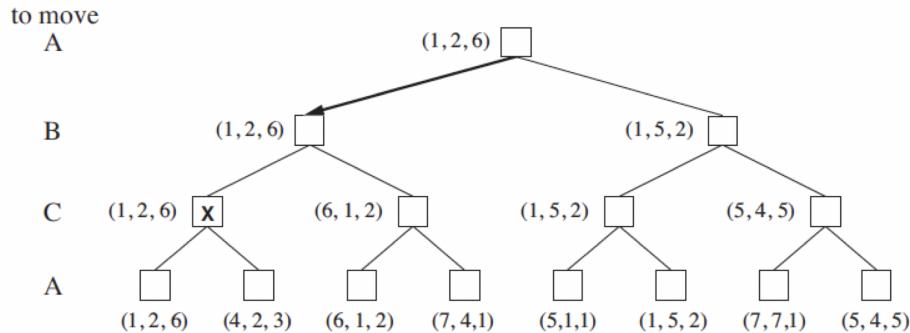


Figure 5: Giochi Multiplayer

1.10 Giochi Complessi

- Giochi stocastici: i giochi in cui è prevista una variabile aleatoria (lancio di dadi oppure la distribuzione di carte)
- Giochi parzialmente osservabili: le mosse sono deterministiche ma non si conoscono gli effetti delle mosse dell'avversario. (Battaglia navale)

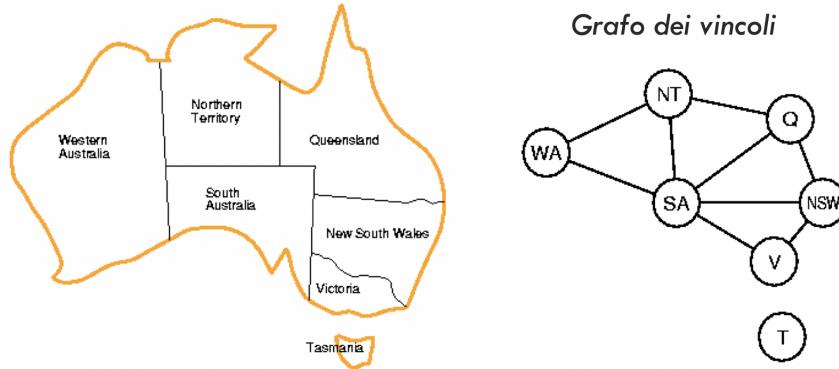
2 Problemi di soddisfacimento di vincoli (CSP)

Sono problemi con una struttura particolare che necessitano algoritmi di ricerca specializzati, si sfrutta una rappresentazione fattorizzata dello stato rendendo esplicita la sua struttura (o almeno in parte). Un problema è risolto quando ogni variabile ha un assegnamento che soddisfa tutti i vincoli. Esistono euristiche generali che si applicano a questi problemi e consentono la risoluzione anche se il problema ha dimensioni significative.

2.1 Formulazione di problemi CSP

- Problema: descritto da tre componenti
 1. X insieme di variabili
 2. D insieme di domini (ogni dominio D_i è a sua volta un insieme di valori possibili per la variabile X_i)
 3. C insieme di vincoli (ogni vincolo C_i è una coppia [variabili,relazione])
- Stato: un assegnamento parziale o completo di valori a variabili (ad esempio $X_i = v_i, X_j = v_j \dots$)
- Stato iniziale: {}
- Azioni: assegnamento di un valore ad una variabile
- Soluzione: un assegnamento completo della formula, dove tutte le variabili hanno un valore consistente, cioè dove tutti i vincoli sono soddisfatti

2.1.1 Problema della colorazione di una mappa



Problema: Si tratta di colorare i diversi paesi sulla mappa con tre colori in modo che paesi adiacenti abbiano colori diversi.

1. $X = \{WA, NT, SA, Q, NSW, V, T\}$
2. $D_{WA} = D_{NT} = D_{SA} = D_Q = D_{NSW} = D_V = D_T = \{\text{red, green, blue}\}$
3. $C = \{WA \neq NT, WA \neq SA, NT \neq Q, NT \neq SA, SA \neq Q, SA \neq NSW, SA \neq V, NSW \neq V, NSW \neq Q\}$

2.1.2 Problema 8 regine

Problema: Si tratta di posizionare le 8 regine sulla scacchiera senza che nessuna sia in grado di mangiarne un'altra.

1. $X = \{Q_1, \dots, Q_8\}$ una regina per colonna della scacchiera
2. $D_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$ numero di riga
3. C = vincoli di non attacco

2.2 Strategie per problemi CSP

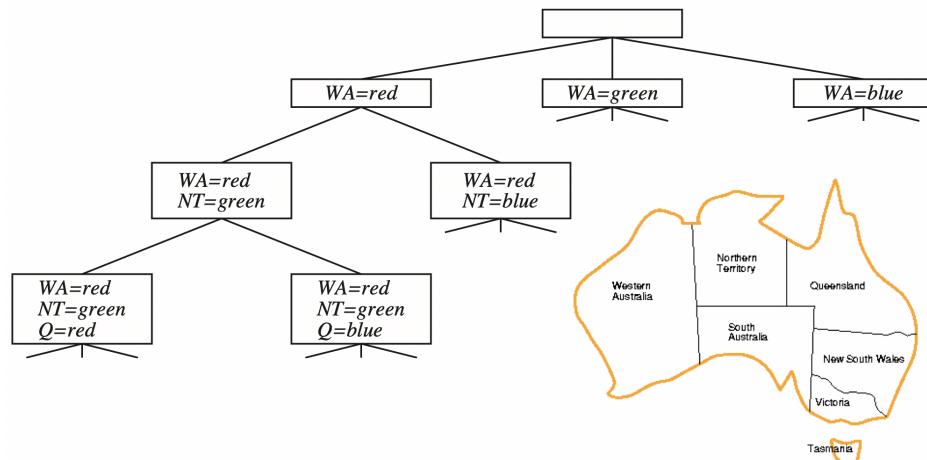
Fin'ora potevamo solo ricercare la soluzione nel grafo degli stati, invece adesso possiamo usare delle euristiche specializzate per questa classe di problemi. Possiamo fare delle inferenze che ci portano a restringere i domini e quindi a limitare la ricerca, in più possiamo eseguire backtracking intelligente.

2.3 Ricerca per problemi CSP

Ad ogni passo si assegna una variabile, la massima profondità della ricerca è fissata dal numero di variabili n. L'ampiezza dello spazio di ricerca invece è definito da $|D_1| * |D_2| * \dots * |D_n|$ (dove $|D_i|$ è il numero di elementi del dominio di X_i). Il fattore di diramazione è pari a $n * d$ al primo passo cioè posso scegliere tra d valori da assegnare a n variabili, al secondo passo $(n - 1) * d$ e così via... si genera così un albero con $n! * d^n$ foglie anche se esistono al massimo d^n assegnamenti completi differenti. Possiamo ridurre drasticamente lo spazio di ricerca sfruttando il fatto che il goal-test è commutativo (non è importante l'ordine con cui scelgo le variabili, quindi ad ogni livello posso scegliere una sola variabile a cui assegnare un valore). Con questa restrizione otteniamo d^n foglie come volevamo.

2.4 Ricerca con backtracking (BT) a profondità limitata

Si esegue un controllo anticipato sulla violazione dei vincoli, perché è inutile andare avanti fino alla fine e poi controllare la correttezza. La ricerca è limitata in profondità dal numero di variabili quindi il metodo è completo. L'algoritmo sceglie ripetutamente una variabile senza assegnamento e a turno prova tutti i valori del dominio di quella variabile fino a trovare una soluzione. Se viene trovata una inconsistenza l'algoritmo si ferma ed esegue il backtracking facendo provare alle chiamate precedenti un nuovo valore.



2.5 Codice Algoritmo backtracking

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)
function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp) Quale variabile scegliere?
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do Quali valori scegliere?
        if value is consistent with assignment then controllo anticipato
            add {var = value} to assignment
            inferences  $\leftarrow$  INFERENCE(csp, var, value) Qual è l'influenza di un assegnamento sulle altre variabili? come restringe i domini?
            if inferences  $\neq$  failure then
                add inferences to assignment
                result  $\leftarrow$  BACKTRACK(assignment, csp) Come evitare di ripetere i fallimenti?
                if result  $\neq$  failure then
                    return result
                remove {var = value} and inferences from assignment
            return failure
```

Le parti evidenziate sono i punti dove posso usare euristiche!

2.5.1 Select-Unassigned-Variable()

Scelta delle variabili: qual è la prossima variabile da scegliere?

- MRV (Minimum Remaining Values): scegliere la variabile che ha meno valori legali (residui) da poter assegnare, cioè la variabile più vincolata. Si scoprono prima i fallimenti (fail first)!
- Eristica del grado: scegliere la variabile coinvolta in più vincoli con le altre variabili (la variabile più vincolante o di grado maggiore). (Usata a parità di MRV)

2.5.2 Order-Domain-Values()

Scelta dei valori: una volta scelta la variabile, come scegliere il valore da assegnare?

- Valore meno vincolante: scegliamo il valore che esclude meno valori per le altre variabili direttamente collegate alla variabile scelta. Meglio valutare prima un assegnamento che ha più probabilità di successo, se volessimo tutte le soluzioni l'ordine non sarebbe importante.

2.5.3 Inference()

Propagazione di vincoli: qual è l'influenza di un assegnamento sulle altre variabili?

- Verifica in avanti (Forward Checking o FC): assegnato un valore ad una variabile, si possono eliminare i valori incompatibili per le altre variabili direttamente collegate da vincoli.
- Consistenza di nodo e arco: si restringono i valori dei domini delle variabili tenendo conto dei vincoli unari e binari su tutto il grafo (se una variabile ad un certo punto ha il dominio ridotto all'insieme vuoto, si esegue immediatamente backtracking. Altrimenti si itera finché tutti i nodi ed archi sono consistenti).

2.5.4 Backtrack()

Quando la ricerca arriva ad un assegnamento che viola i vincoli, come posso evitare di ripetere in futuro lo stesso errore?

- Backtracking Cronologico: viene cambiato il valore dell'ultima variabile assegnata, continuando a fallire fino al soddisfacimento dei vincoli.
- Backtracking Intelligente: si considerano alternative solo per le variabili che hanno causato il fallimento (X_i, X_j, \dots), cioè un "insieme dei conflitti". Ad esempio, nel problema delle regine, si parte con tutte le variabili assegnate (tutte le regine sulla scacchiera) e ad ogni passo si modifica l'assegnamento ad una variabile per cui un vincolo è violato (si muove una regina minacciata su una colonna). Questo è un algoritmo di riparazione euristica. Un'altra euristica potrebbe consistere nello scegliere un nuovo valore che crea meno conflitti (euristica dei conflitti minimi).

3 Agenti Logici KB

Vogliamo migliorare le capacità razionali dei nostri agenti dotandoli di rappresentazioni di mondi più complessi. Il mondo è tipicamente complesso, ci serve una rappresentazione parziale e incompleta (una astrazione) del mondo, utile agli scopi dell'agente. Per descrivere ambienti parzialmente osservabili e complessi ci servono linguaggi di rappresentazione della conoscenza più espressivi, ma soprattutto con capacità inferenziali. Gli agenti basati su conoscenza sono dotati di una KB (knowledge base) con conoscenza espressa in maniera esplicita e dichiarativa. La conoscenza può essere codificata a mano, ma anche estratta dai testi o appresa dall'esperienza.

3.1 Approccio dichiarativo VS Approccio procedurale

L'approccio dichiarativo consiste nella creazione di una KB che racchiude tutta la conoscenza necessaria a decidere l'azione da compiere in forma dichiarativa. L'alternativa (approccio procedurale) è scrivere un programma che implementa il processo decisionale, una volta per tutte.

Un agente KB è più flessibile: più semplice acquisire conoscenza in modo incrementale e modificare il comportamento con l'esperienza!

3.2 Tell-Ask

Un agente basato su conoscenza mantiene una base di conoscenza (KB), cioè un insieme di enunciati espressi in un linguaggio di rappresentazione. L'agente interagisce con la KB mediante una interfaccia funzionale definita come Tell-Ask:

- Tell: per aggiungere nuovi enunciati a KB
- Ask: per interrogare la KB
- Retract: per eliminare enunciati

Gli enunciati nella KB rappresentano le credenze dell'agente, le risposte X dell'agente quindi devono essere tali che X è una conseguenza (discende necessariamente) della KB.

Il nostro obiettivo sarà capire quando, avendo una base di conoscenza KB contenente una rappresentazione dei fatti che si ritengono veri, un certo fatto α è vero di conseguenza ($KB \models \alpha$, cioè α è conseguenza logica della KB).

3.3 Base di conoscenza o Base di dati?

Qual è la differenza tra una KB e un database?

- Base di conoscenza: una rappresentazione esplicita dello stato, parziale e compatta, espressa in un linguaggio simbolico, che contiene fatti di tipo specifico e fatti di tipo generale o anche regole. Quello che caratterizza una base di conoscenza è la capacità inferenziale, cioè la capacità di derivare nuovi fatti da quelli memorizzati esplicitamente.
- Base di dati: solo fatti specifici e posso solo interrogarla, non ho modo di inferire nuovi fatti.

Ma il problema fondamentale della rappresentazione della conoscenza sta nel trovare il giusto compromesso tra: espressività del linguaggio di rappresentazione e complessità del meccanismo inferenziale. Sfortunatamente più il linguaggio è espressivo, meno efficiente è il meccanismo inferenziale.

3.4 Formalismi per la rappresentazione della conoscenza

Un formalismo per la rappresentazione della conoscenza ha tre componenti:

1. Una sintassi: cioè un linguaggio composto da un vocabolario e regole per la formazione delle frasi (enunciati).
2. Una semantica: stabilisce una corrispondenza tra gli enunciati e fatti del mondo. Se un agente ha un enunciato X nella sua KB, crede che il fatto corrispondente sia vero nel mondo.
3. Un meccanismo inferenziale: (codificato o meno tramite regole di inferenza come nella logica) che ci consente di inferire nuovi fatti.

A questo punto, qual è la complessità computazionale di sapere se α è conseguenza logica della KB nei vari linguaggi logici? Quali sono gli algoritmi di decisione e le strategie di ottimizzazione? I linguaggi logici, calcolo proposizionale (PROP) e logica dei predicati (FOL), sono adatti per la rappresentazione della conoscenza?

4 Agenti Logici: Calcolo proposizionale

4.1 Sintassi (regole)

La sintassi definisce quali sono le frasi legittime (ben formate) del linguaggio.

- simbolo $\rightarrow P \mid Q \mid R \dots$
- formula $\rightarrow formulaAtomica \mid formulaComplessa$
- formulaAtomica $\rightarrow True \mid False \mid simbolo$
- formulaComplessa $\rightarrow \neg formula \mid formula \wedge formula \mid formula \vee formula \mid formula \Rightarrow formula \mid formula \Leftrightarrow formula$

Possiamo omettere le parentesi assumendo questa precedenza tra gli operatori:

$\neg > \wedge > \vee > \Rightarrow, \Leftrightarrow$

4.2 Semantica (significato)

La semantica definisce il significato di un enunciato, cioè se esso è vero o falso rispetto ad una interpretazione. Una interpretazione definisce un valore di verità per tutti i simboli proposizionali. Il significato di una frase è determinato dal significato dei suoi componenti, a partire dai simboli proposizionali.

4.3 Conseguenza logica

Come abbiamo visto precedentemente, il problema è capire quando, data una base di conoscenza KB contenente una rappresentazione dei fatti che si ritengono veri, un certo fatto α è vero di conseguenza. ($KB \models \alpha$, cioè α è conseguenza logica della KB).

Una formula α è conseguenza logica di un insieme di formule KB se e solo se in ogni modello² di KB anche α è vera.

Indicheremo con $M(\alpha)$ i modelli di α , cioè l'insieme delle interpretazioni che rendono α vera, e con $M(KB)$ i modelli dell'insieme di formule in KB otteniamo:

$$KB \models \alpha \Leftrightarrow M(KB) \subseteq M(\alpha)$$

²Modello: un'attribuzione di un significato a tutti gli enunciati (le formule) del linguaggio che le rende tutte vere

4.4 Il mondo del Wumpus

Il mondo del Wumpus è una caverna fatta di stanze connesse tra di loro. Il Wumpus mangia chiunque entri nella stanza in cui si trova. Il Wumpus può essere ucciso dall'agente, che ha solo una freccia a disposizione. In alcune stanze sono presenti dei pozzi: se l'agente entra in una di queste stanze cade nel pozzo e muore. In una delle stanze si trova l'oro e l'obiettivo dell'agente è di trovarlo e tornare a casa sano e salvo. L'agente non conosce l'ambiente, né la sua locazione. Solo all'inizio sa dove si trova (in [1,1]). In questo problema gli ambienti sono generati a caso e alcuni di questi non sono risolubili (ma sappiamo che lo stato (1,1) è safe): l'agente in alcune situazioni deve decidere se rischiare di essere ucciso pur di trovare l'oro e quando conviene tornare a casa a mani vuote.

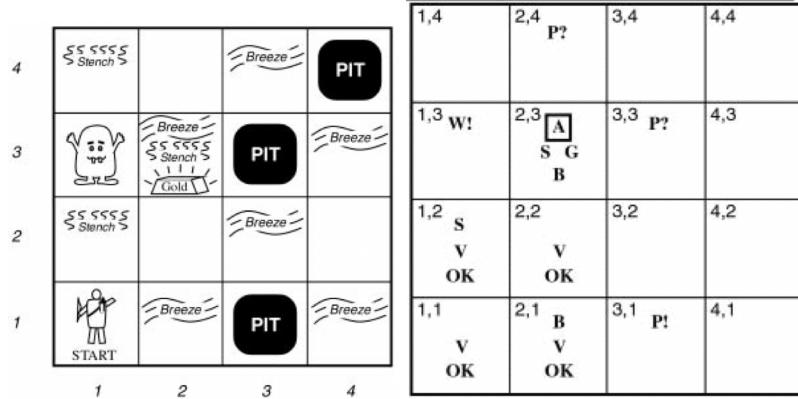


Figure 6: Wumpus' world

- Misura delle prestazioni:
 - +1000 se trova l'oro, torna in [1,1] e esce
 - -1000 se muore
 - -1 per ogni azione
 - -10 se usa la freccia
- Percezioni:
 - **puzzo** nelle caselle adiacenti al Wumpus
 - **brezza** nelle caselle adiacenti alle buche
 - **luccichio** nelle caselle con l'oro
 - **bump** se sbatte in un muro
 - **urlo** se il Wumpus viene ucciso
 - L'agente non percepisce la sua locazione
- Azioni:
 - avanti
 - destra
 - sinistra
 - afferra oggetto
 - scaglia la freccia
 - esci

Le percezioni sono una quintupla [puzzo, brezza, luccichio, bump, urlo]. ([none, none, none, none, none] se non percepisce niente).

4.4.1 Esempio dal mondo del Wumpus

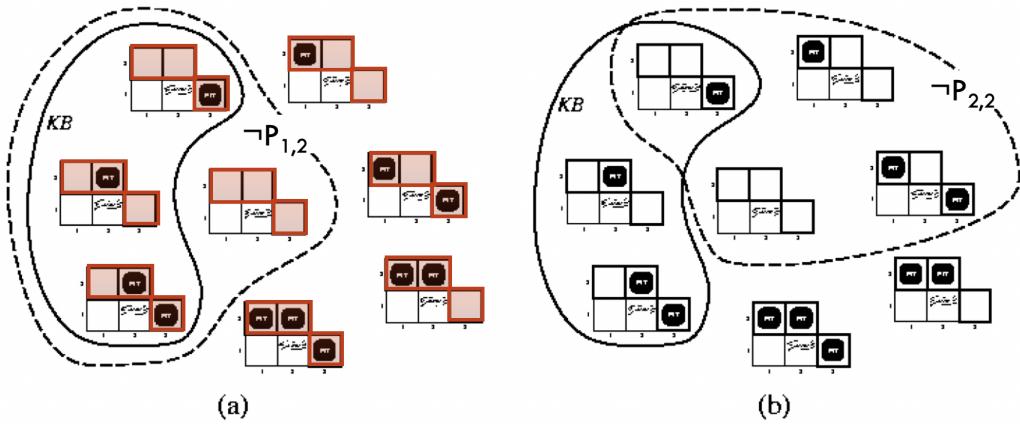
$KB = \{B_{2,1}, \neg B_{1,1}, + \text{le regole del WumpusWorld}\}$

Vogliamo stabilire l'assenza di pozzi in [1,2] e in [2,2]

$KB \models \neg P_{1,2}?$

$KB \models \neg P_{2,2}?$

Ci sono otto possibili interpretazioni o mondi, considerando solo l'esistenza di pozzi nelle 3 caselle $P_{1,2}$, $P_{2,2}$, $P_{3,1}$ (perchè non posso muovermi in diagonale). Guardando le immagini sottostanti, a sinistra prendiamo tutti i mondi in cui non c'è pozzo in [1,2] e notiamo che facendo così includo tutti i mondi della mia KB, quindi $KB \vdash \neg P_{1,2}$. A destra prendo i mondi dove non c'è pozzo in [2,2] e vediamo che questi non includono tutti i mondi della mia KB, quindi $KB \not\vdash \neg P_{2,2}$.



4.5 Formule logiche, Validità e Soddisfacibilità

$$\begin{aligned}
 (\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{cominutativity of } \wedge \\
 (\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{cominutativity of } \vee \\
 ((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge \\
 ((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee \\
 \neg(\neg \alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
 (\alpha \Rightarrow \beta) &\equiv (\neg \beta \Rightarrow \neg \alpha) \quad \text{contraposition} \\
 (\alpha \Rightarrow \beta) &\equiv (\neg \alpha \vee \beta) \quad \text{implication elimination} \\
 (\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
 \neg(\alpha \wedge \beta) &\equiv (\neg \alpha \vee \neg \beta) \quad \text{de Morgan} \\
 \neg(\alpha \vee \beta) &\equiv (\neg \alpha \wedge \neg \beta) \quad \text{de Morgan} \\
 (\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee \\
 (\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
 \end{aligned}$$

Figure 7: Equivalenze Logiche (leggi)

A valida sse è vera in tutte le interpretazioni (detta anche tautologia).

A soddisfacibile sse esiste una interpretazione in cui A è vera.

A è valida sse $\neg A$ è insoddisfacibile.

4.6 Inferenza per calcolo proposizionale (PROP)

- Model checking: una forma di inferenza che fa riferimento alla definizione di conseguenza logica (si enumerano i possibili modelli e si controllano tramite le tabelle di verità)
- Algoritmi per la soddisfabilità: problemi riconducibili ad altri problemi.
 $KB \models A$ sse $KB \wedge \neg A$ è insoddisfacibile.

4.7 Algoritmo TT-Entails (Model Checking)

Come facciamo a sapere se $KB \models \alpha$? L'algoritmo TT-Entails enumera tutte le possibili interpretazioni di KB (k simboli, 2^k possibili interpretazioni). Per ciascuna interpretazione:

- Se non soddisfa $KB \rightarrow$ OK (non ci interessa, andiamo avanti)
- Se soddisfa $KB \rightarrow$ si controlla che soddisfi anche α .
 Se si trova anche solo una interpretazione che soddisfa KB e non α la risposta è NO.

4.7.1 Esempio TT-Entails

```
(\(\neg A \vee B) \wedge (A \vee C) \models (B \vee C) ?           Symbols = [A, B, C]
  □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [A, B, C], \{ \ }) 
    □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [B, C], \{ A=t \ })
      □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [C], \{ A=t, B=t \ })
        □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [ ], \{ A=t, B=t, C=t \ }) OK
        □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [ ], \{ A=t, B=t, C=f \ }) OK
      □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [C], \{ A=t, B=f \ })
        □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [ ], \{ A=t, B=f, C=t \ }) OK
        □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [ ], \{ A=t, B=f, C=f \ }) OK
    □ TT-CHECK-ALL((\(\neg A \vee B) \wedge (A \vee C), (B \vee C), [B, C], [A=f] ) ...
```

Solo alla fine, dopo avere provato tutti i possibili assegnamenti, possiamo rispondere se Vero o Falso.

4.8 Algoritmo DPLL (Alg. Soddisfabilità)

4.8.1 Forma a clausole

Gli algoritmi per la soddisfabilità usano KB in forma a clausole.

Ad esempio $\{A, B\} \ \{\neg B, C, D\} \ \{\neg A, F\}$ è una forma normale congiuntiva cioè una congiunzione di disgiunzioni letterali. Il suo significato quindi è:

$(A \vee B) \wedge (\neg B \vee C \vee D) \wedge (\neg A \vee F)$, la cosa interessante è che non è restrittiva: è sempre possibile ottenerla con trasformazioni che preservano l'equivalenza logica.

I passi da seguire per ottenere la forma a clausole sono:

1. Eliminazione della \Leftrightarrow : $(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Eliminazione dell' \Rightarrow : $(A \Rightarrow B) \equiv (\neg A \vee B)$
3. Negazioni all'interno:
 - $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ (de Morgan)
 - $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
4. Distribuzione di \vee su \wedge : $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$

L'algoritmo DPLL sfrutta la soddisficiabilità di $KB \models A$ se $KB \wedge \neg A = \{\}$. Parte da una KB in forma a clausole, avviene una enumerazione in profondità di tutte le possibili interpretazioni alla ricerca di un modello. Tre miglioramenti rispetto a TT-Entails:

1. Terminazione anticipata: Si può decidere sulla verità di una clausola anche con interpretazioni parziali, basta che un letterale sia vero. Ad esempio se A è vero lo sono anche $\{A, B\}$ e $\{A, C\}$ indipendentemente dai valori di B e C . (ricordiamo che sono in $\vee!$). Se anche una sola clausola è falsa l'interpretazione non può essere un modello dell'insieme di clausole.
2. Simboli puri: un simbolo puro è un simbolo che appare con lo stesso segno in tutte le clausole. Ad esempio $\{A, \neg B\} \{\neg B, \neg C\} \{C, A\}$ dove A e B sono puri. Nel determinare se un simbolo è puro se ne possono trascurare le occorrenze in clausole già rese vere cioè i simboli puri possono essere assegnati a True se il letterale è positivo, False se negativo. Facendo così non si eliminano modelli utili, se le clausole hanno un modello continuano ad averlo dopo questo assegnamento. L'assegnamento è obbligato.
3. Clausole unitarie: una clausola con un solo letterale non assegnato, ad esempio $\{B\}$ è unitaria ma anche $\{B, \neg C\}$ è unitaria quando $C = \text{True}$. Conviene assegnare prima valori ai letterali in clausole unitarie. L'assegnamento è univoco (True se positivo, False se negativo).

4.8.2 Esempio DPLL

$$KB = (\{\neg B_{1,1}, P_{1,2}, P_{2,1}\} \{\neg P_{1,2}, B_{1,1}\} \{\neg P_{2,1}, B_{1,1}\} \{\neg B_{1,1}\}) \models \{\neg P_{1,2}\}?$$

Aggiungiamo $\{P_{1,2}\}$ (perché sarebbe $\neg(\neg P_{1,2})$) e vediamo se l'insieme è insoddisfacibile:

$$\text{DPLL}(\{\neg B_{1,1}, P_{1,2}, P_{2,1}\} \{\neg P_{1,2}, B_{1,1}\} \{\neg P_{2,1}, B_{1,1}\} \{\neg B_{1,1}\} \{P_{1,2}\})$$

Notiamo che $\{P_{1,2}\}$ è unitaria quindi assegnamo $P_{1,2} = \text{True}$.

La prima clausola $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}$ e $\{P_{1,2}\}$ sono soddisfatte.

La seconda clausola $\{\neg P_{1,2}, B_{1,1}\}$ diventa unitaria, $B_{1,1} = \text{True}$.

A questo punto $\{\neg P_{1,2}, B_{1,1}\}$ e $\{\neg P_{2,1}, B_{1,1}\}$ sono soddisfatte, ma $\{\neg B_{1,1}\}$ no (perché prima abbiamo assegnato $B_{1,1} = \text{True}$, quindi $\neg B_{1,1} = \text{False}$).

FAIL!

Non esistono modelli, quindi possiamo dire che $\neg P_{1,2}$ è conseguenza logica della KB.

4.8.3 Miglioramenti DPLL

DPLL è completo e termina sempre. Alcuni miglioramenti: analisi di componenti (se le variabili possono essere suddivise in sotto-insiemi disgiunti, cioè senza simboli in comune), ordinamento di variabili e valori (scegliere la variabile che compare in più clausole), backtracking intelligente e altre ottimizzazioni...

4.9 Algoritmo WALK-SAT

E' un algoritmo di ricerca locale, l'obiettivo è un assegnamento che soddisfa tutte le clausole (un modello) partendo da un assegnamento casuale. Ad ogni passo si cambia il valore di una proposizione (flip). Gli stati sono valutati contando il numero di clausole non soddisfatte (meno sono meglio è). Ci sono molti minimi locali, per non incapparci serve introdurre perturbazioni casuali, come succedeva con Hill Climbing con riavvio casuale o Simulated Annealing. WALK-SAT è uno degli algoritmi più semplici ed efficaci.

WALK-SAT ad ogni passo:

- Sceglie a caso una clausola non ancora soddisfatta
- Sceglie un simbolo da modificare (flip) scegliendo con probabilità p (di solito 0,5) tra una delle due:
 - Passo casuale: un simbolo a caso
 - Passo di ottimizzazione: sceglie quello che rende più clausole soddisfatte
- Si arrende dopo un certo numero di flip predefinito (variabile max-flips)

4.9.1 Esempio WALK-SAT

Come euristica usiamo il numero di clausole soddisfatte (valore da massimizzare).

Rosso: passo casuale

Blu: passo di ottimizzazione

(1){ $\neg B_{1,1}, P_{1,2}, P_{2,1}$ } (2){ $\neg P_{1,2}, B_{1,1}$ } (3){ $\neg P_{2,1}, B_{1,1}$ } (4){ $\neg B_{1,1}$ }
 $[B_{1,1} = F, P_{1,2} = T, P_{2,1} = T]$ clausola 2, 3 F; scelgo clausola 2; a caso eseguo flip $B_{1,1}$
 $[B_{1,1} = T, P_{1,2} = T, P_{2,1} = T]$ clausola 4 F; scelgo 4; ottimizzazione: flip $B_{1,1}$ (unica scelta)
 $[B_{1,1} = F, P_{1,2} = T, P_{2,1} = T]$ clausole 2, 3 F; scelgo clausola 2; a caso eseguo flip $P_{1,2}$
 $[B_{1,1} = F, P_{1,2} = F, P_{2,1} = T]$ clausola 3 F; scelgo 3; ottimizzazione: flip $P_{2,1}$
 $[B_{1,1} = F, P_{1,2} = F, P_{2,1} = F]$ modello

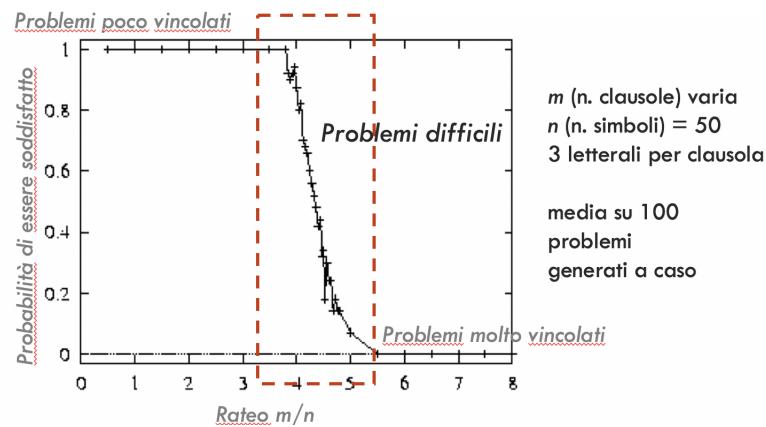
4.9.2 Analisi WALK-SAT

Se $\text{max-flips} = \infty$ e l'insieme di clausole è soddisfacibile prima o poi termina. Va bene per cercare un modello, sapendo che esiste, ma se è insoddisfacibile non termina. Non può essere usato per verificare l'insoddisfacibilità, il problema quindi è decidibile ma l'algoritmo non è completo.

4.9.3 Problemi SAT difficili

Se un problema ha molte soluzioni (problema sotto-vincolato) è più probabile che WALK-SAT ne trovi una in tempi brevi. Ma dato che SAT è un problema NP-Completo, alcune istanze richiedono tempo esponenziale per la risoluzione. Un esempio è che nella seguente istanza esistono 16 soluzioni su 32 assegnamenti possibili, un assegnamento ha il 50% di probabilità di essere giusto: in circa 2 passi random si indovina.

$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$
quello che conta è il rapporto m/n , dove m è il numero di clausole (vincoli) e n il numero di simboli.
In questo esempio $5/5=1$, più è grande il rapporto, più vincolato è il problema.



4.10 Inferenza come deduzione

Un altro modo per decidere se $\text{KB} \models A$ è usare un meccanismo di deduzione, scriviamo $\text{KB} \vdash A$ (A è deducibile da KB). La deduzione avviene specificando delle regole di inferenza.

- Correttezza: Se $\text{KB} \vdash A$ allora $\text{KB} \models A$. Tutto ciò che è derivabile è conseguenza logica.
- Completezza: Se $\text{KB} \models A$ allora $\text{KB} \vdash A$. Tutto ciò che è conseguenza logica è ottenibile tramite il meccanismo di inferenza.

Nota: questa nozione di completezza (detta anche completezza semantica) si riferisce al teorema di completezza di Gödel. Il teorema di incompletezza di Gödel³ non asserisce che non esistono sistemi deduttivi completi per il FOL, ma che una logica abbastanza potente da assiomatizzare l'aritmetica non è in grado di dimostrare tutte le affermazioni vere sui numeri naturali.

4.10.1 Alcune regole di inferenza

$$\begin{array}{ll} \text{Modus Ponens (eliminazione dell'implicazione)} & \frac{\alpha \Rightarrow \beta \quad \alpha}{\beta} \\ \text{Eliminazione dell'AND} & \frac{\alpha \wedge \beta}{\alpha} \\ \text{Eliminazione doppia implicazione} & \frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \\ \text{Introduzione doppia implicazione} & \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta} \end{array}$$

Come decidere ad ogni passo qual è la regola di inferenza da applicare? E a quali premesse applicarla? Nasce quindi un problema di esplorazione di uno spazio di stati.

Una procedura di dimostrazione definisce:

- Direzione della ricerca: nella dimostrazione di teoremi conviene procedere all'indietro. Se si vuole dimostrare $A \wedge B$ si cerca di dimostrare A e poi B , se si vuole dimostrare $A \Rightarrow B$, si assume A e si cerca di dimostrare B .
- Strategia di ricerca:
 - Completezza: Le regole della deduzione naturale sono un insieme di regole di inferenza completo, se l'algoritmo di ricerca è completo siamo a posto!
 - Efficienza: La complessità è alta, è un problema decidibile ma NP-Completo.

4.11 Regola di risoluzione

Utilizzeremo un'unica regola: la regola di risoluzione (presuppone la forma a clausole).

$$\frac{\{P, Q\} \quad \{\neg P, R\}}{\{Q, R\}} \quad \frac{P \vee Q \quad \neg P \vee R}{Q \vee R}$$

4.11.1 Regola di risoluzione in generale per PROP

$$\frac{\{I_1, I_2, \dots, I_i, \dots, I_k\} \{M_1, M_2, \dots, M_j, \dots, M_n\}}{\{I_1, I_2, \dots, I_{i-1}, I_{i+1}, \dots, I_k, M_1, M_2, \dots, M_{j-1}, M_{j+1}, \dots, M_n\}}$$

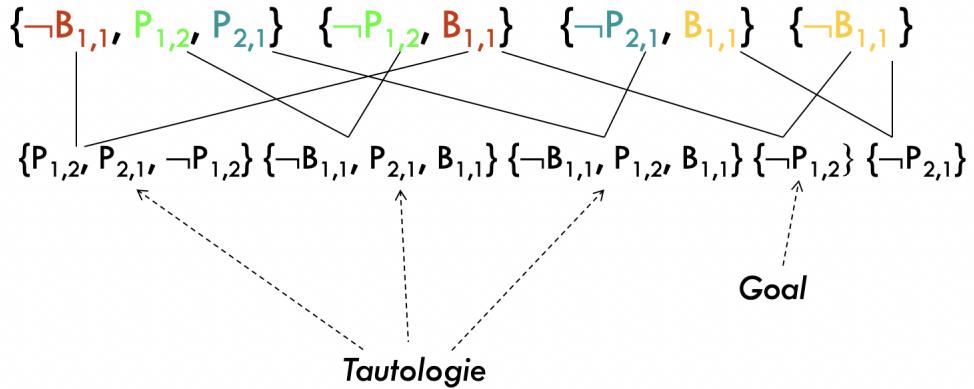
"I" e "M" sono letterali, simboli di proposizione positivi o negativi, la cosa fondamentale da sapere è che I_i e M_j sono simboli uguali ma di segno opposto.

³In ogni formalizzazione coerente della matematica che sia sufficientemente potente da definire la struttura dei numeri naturali dotati delle operazioni di somma e prodotto, è possibile costruire una proposizione sintatticamente corretta che non può essere né dimostrata né confutata all'interno dello stesso sistema.

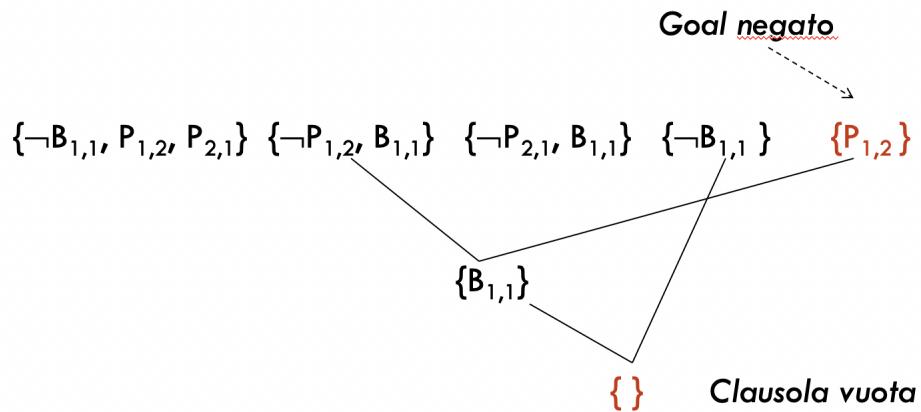
Un caso particolare avviene quando ci troviamo in questa situazione:

$$\frac{\{P\} \quad \{\neg P\}}{\{\}} \text{ clausola vuota} \rightarrow \text{contraddizione}$$

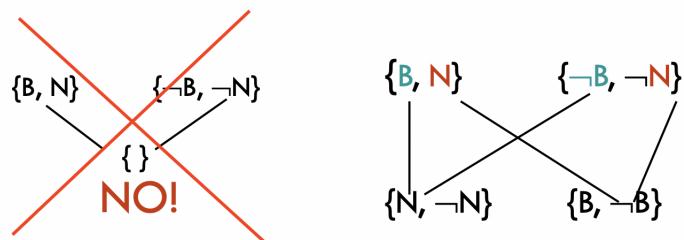
4.11.2 Esempio grafo di risoluzione



4.11.3 Refutazione



ATTENZIONE:



4.11.4 Osservazioni

Abbiamo una singola regola di inferenza, ma è sufficiente? Siamo sicuri che applicando la regola in tutti i modi possibili riesco a dedurre α quando è conseguenza logica? Vale la completezza? Non sempre.

Per fortuna il teorema di refutazione ci offre un modo alternativo di procedere: se voglio sapere se $KB \models \alpha$, aggiungo $\neg\alpha$ a KB e controllo che l'insieme ottenuto sia insoddisfacibile. La correttezza della regola ci dice anche che se da tale insieme derivo la clausola vuota allora in effetti l'insieme è insoddisfacibile. Invece il teorema di risoluzione ci garantisce che se KB è insoddisfacibile allora la clausola vuota sono sempre in grado di trovarla con applicazioni della regola di risoluzione $KB \vdash_{RES} \{\}$. Procedere per refutazione ci garantisce la completezza (naturalmente se la procedura applica la regola in maniera sistematica).

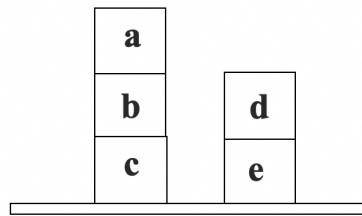
5 Agenti Logici: Logica del prim'ordine

Nella logica dei predicati abbiamo assunzioni che comprendono gli oggetti, le proprietà e le relazioni.

- Gli oggetti: un libro, un evento, una persona... Possono essere identificati relativamente ad altri oggetti o con simboli, oppure mediante funzioni: "la madre di Pietro". L'insieme degli oggetti rilevanti costituiscono il dominio del discorso. (Il dominio potrebbe essere infinito).
- Le proprietà: "la madre di Pietro è simpatica"
- Le relazioni tra gli oggetti: "Pietro è amico di Paolo"

5.1 Il mondo dei blocchi

- Dominio: {a, b, c, d, e} (blocchi)
- Le funzioni: si individuano le funzioni rilevanti che servono per identificare oggetti. Ad esempio la funzione "Hat(x)": dato un blocco "x" identifica il blocco ad esso superiore.
- Le relazioni: si individuano le relazioni interessanti.
 - On = { $\langle a, b \rangle, \langle b, c \rangle, \langle d, e \rangle$ } //a è su b, b è su c, d è su e
 - Clear = {a, d} //non hanno nulla sopra di loro
 - Table = {c, e} //sono poggiati sul tavolo
 - Block = {a, b, c, d, e} //sono blocchi



5.2 Concettualizzazioni

Otteniamo così la concettualizzazione del problema definita come

$\langle \{a, b, c, d, e\}, \{\text{Hat}\}, \{\text{On}, \text{Clear}, \text{Table}, \text{Block}\} \rangle$

Le concettualizzazioni possibili sono infinite, un aspetto importante è il livello di astrazione giusto per gli scopi della rappresentazione. Se fosse rilevante il colore o la grandezza dei blocchi dovremmo introdurre prediciati anche per questi aspetti.

5.3 FOL

5.3.1 Predicati

- Connuttivo $\rightarrow \wedge | \vee | \neg | \Rightarrow | \Leftrightarrow | \Leftarrow$
- Quantificatore $\rightarrow \forall | \exists$
- Variabile $\rightarrow x | y | \dots$
- Costante $\rightarrow A | B | \text{Mario} | 2 | \dots$
- Funzione $\rightarrow \text{Hat} | + | - | \dots$
- Predicato $\rightarrow \text{On} | \text{Clear} | > | < | \dots$

L'arietà è il numero di "parametri" della funzione o del predicato.

5.3.2 Termini

- Termine \rightarrow Costante | Variabile | Funzione(Termine,...)

5.3.3 Formule

- Formula-Atomica \rightarrow True | False | Termine=Termine | Predicato(Termine,...)
- Formula \rightarrow Formula-Atomica | Formula Connuttivo Formula | Quantificatore Variabile Formula | \neg Formula

Di solito le variabili sono usate nell'ambito di quantificatori. In tal caso le occorrenze si dicono legate. Se non sono legate, si dicono libere.

$\text{Mela}(x) \Rightarrow \text{Rossa}(x)$ x è libera in entrambe le occorrenze

$\forall x \text{ Mela}(x) \Rightarrow \text{Rossa}(x)$ x è legata

$\text{Mela}(x) \Rightarrow \exists x \text{ Rossa}(x)$ la prima x è libera, la seconda legata

Formula chiusa: una formula che non contiene occorrenze di variabili libere (altrimenti è detta aperta).

Formula ground: una formula che non contiene variabili.

5.3.4 Interpretazione

La semantica dichiarativa consiste nello stabilire una corrispondenza tra i termini del linguaggio e gli oggetti del mondo.

Una interpretazione "I" stabilisce una corrispondenza precisa tra elementi atomici del linguaggio ed elementi della concettualizzazione. "I" interpreta:

- i simboli di costante come elementi del dominio
- i simboli di funzione come funzioni da n-uple di $D \rightarrow D$
- i simboli di predicato come insiemi di n-uple

5.4 Sematica Composizionale

Il significato di un termine o di una formula composta è determinato in funzione del significato dei suoi componenti.

- Semantica \forall : $\forall x A(x)$ è vera se per ciascun elemento del dominio A è vera. Se il dominio è finito equivale a un grosso \wedge . Tipicamente, siccome difficilmente una proprietà è universale, \forall si usa quasi sempre insieme a \Rightarrow . (Es: $\forall x \text{ Persona}(x) \Rightarrow \text{Mortale}(x)$).
- Semantica \exists : $\exists x A(x)$ è vera se esiste almeno un elemento del dominio per cui A è vera. Se il dominio è finito equivale a un grosso \vee . Tipicamente \exists si usa con \wedge (Es: $\exists x \text{ Persona}(x) \wedge \text{Speciale}(x)$).

$$\begin{array}{ll}
 \forall x \neg P(x) \equiv \neg \exists x P(x) & \neg P \wedge \neg Q \equiv \neg(P \vee Q) \\
 \neg \forall x P(x) \equiv \exists x \neg P(x) & \neg(P \wedge Q) \equiv \neg P \vee \neg Q \\
 \forall x P(x) \equiv \neg \exists x \neg P(x) & P \wedge Q \equiv \neg(\neg P \vee \neg Q) \\
 \neg \forall x \neg P(x) \equiv \exists x P(x) & P \vee Q \equiv \neg(\neg P \wedge \neg Q)
 \end{array}$$

Figure 8: Relazione tra \forall ed \exists

5.4.1 Semantica Standard VS Semantica Database

- Standard: ci permette di inferire.
- Database: abbiamo simboli distinti, oggetti distinti, tutto ciò di cui non si sa che è vero è falso, infine esistono solo gli oggetti di cui si parla.

5.5 Interazione con la KB tramite FOL

- Asserzioni: vengono aggiunte le informazioni alla KB, esempio TELL(KB, King(John)) oppure TELL(KB, $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$)
- Conseguenze logiche: ASK(KB, Person(John)) risponde si, se $\text{KB} \models \text{Person}(John)$. Un altro esempio è ASK(KB, $\exists x \text{ Person}(x)$) risponde con una lista di sostituzioni o legami: $\{\{x/John\}$
 $\{x/George\} \dots\}$

Il metodo di risoluzione per FOL si basa sulla trasformazione in forma a clausole e sull'unificazione. Ci sono anche altri metodi particolari, chiamati sistemi a regole, che vedremo alla fine: il "Backward chaining" insieme alla programmazione logica e il "Forward chaining" con basi di dati deduttive.

5.6 Regola di inferenza per \forall

Istanziazione dell'Universale (eliminazione del \forall)
$$\frac{\forall x A[x]}{A[g]}$$
 dove g è un termine ground (ovvero un termine che non contiene variabili) e $A[g]$ è il risultato della sostituzione di g al posto di x in A .
Esempio: $\forall x \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ si ottiene $\text{King}(John) \wedge \text{Greedy}(John) \Rightarrow \text{Evil}(John)$

5.7 Regola di inferenza per \exists

Istanziamento dell'esistenziale (eliminazione del \exists)
$$\frac{\exists x A[x]}{A[k]}$$

 dove \exists non compare nell'ambito di \forall e k è una costante nuova (costante di Skolem)
 Esempio: $\exists x Padre(x, G)$ diventa $Padre(k, G)$.

Nel caso di variabili quantificate universalmente va introdotta una funzione (di Skolem)
 Esempio: $\forall x \exists y Padre(x, y)$ diventa $\forall x Padre(x, p(x))$

5.8 Riduzione a inferenza proposizionale

Per semplificare il simbolo di quantificazione potremmo sfruttare un meccanismo di proposizionalizzazione, cioè creare tante istanze delle formule quantificate universalmente quanti sono gli oggetti menzionati ed eliminare i quantificatori esistenziali "skolemizzando". A questo punto possiamo trattare la KB come proposizionale e applicare gli algoritmi visti. Però c'è un ovvio problema: le costanti sono in numero finito ma se ci sono funzioni il cui numero di istanze da creare è infinito, come si procede? In aiuto viene il Teorema di Herbrand.

5.8.1 Teorema di Herbrand

Se $KB \models A$ allora c'è una dimostrazione che coinvolge solo un sotto-insieme finito della KB proposizionalizzata. Si può procedere in modo incrementale, iniziando col creare le istanze con le costanti, creare poi quelle con un solo livello di annidamento $Padre(John)$, $Madre(John)$ ed infine quelle con due livelli di annidamento $Padre(Padre(John))$, $Padre(Madre(John))$, ecc... Se $KB \not\models A$ il processo non termina (problema semidecidibile).

5.9 Verso un metodo di risoluzione per il FOL

Per trovare un metodo di risoluzione, come abbiamo fatto per il PROP, dobbiamo estendere al FOL la trasformazione in forma a clausole e dobbiamo introdurre il concetto di unificazione.

5.9.1 Forma a clausole

Una clausola è un insieme di letterali, che rappresenta la loro disgiunzione.

- Clausola $\rightarrow \{\text{Letterale}, \dots, \text{Letterale}\}$
- Letterale $\rightarrow \text{Formula-Atomica} \mid \neg \text{Formula-Atomica}$

Una KB è un insieme di clausole.

Teorema: per ogni formula chiusa α del FOL è possibile trovare in maniera effettiva un insieme di clausole $Forma_a_Clausole(\alpha)$ che è soddisfacibile se solo se α lo era (o insoddisfacibile se solo se α lo era).

5.9.2 Esempio di trasformazione (passo passo)

Esempio di trasformazione in dettaglio per la frase:

“Tutti coloro che amano tutti gli animali sono amati da qualcuno”

La nostra formula di partenza è: $\forall x(\forall y \text{Animale}(y) \Rightarrow \text{Ama}(x, y)) \Rightarrow (\exists y \text{Ama}(y, x))$

1. Eliminazione delle implicazioni ($\Rightarrow, \Leftrightarrow$):

- $A \Rightarrow B$ diventa $\neg A \vee B$
- $A \Leftrightarrow B$ diventa $(\neg A \vee B) \wedge (\neg B \vee A)$

$$\begin{aligned} \forall x(\forall y \text{Animale}(y) \Rightarrow \text{Ama}(x, y)) &\Rightarrow (\exists y \text{Ama}(y, x)) \\ \forall x(\forall y \text{Animale}(y) \Rightarrow \text{Ama}(x, y)) &\vee (\exists y \text{Ama}(y, x)) \\ \forall x(\forall y \neg \text{Animale}(y) \vee \text{Ama}(x, y)) &\vee (\exists y \text{Ama}(y, x)) \end{aligned}$$

2. Negazioni all'interno:

- $\neg\neg A$ diventa A
- $\neg(A \wedge B)$ diventa $\neg A \vee \neg B$
- $\neg(A \vee B)$ diventa $\neg A \wedge \neg B$
- $\neg\forall x A$ diventa $\exists x \neg A$
- $\neg\exists x A$ diventa $\forall x \neg A$

$$\begin{aligned} \forall x \neg(\forall y \neg \text{Animale}(y) \vee \text{Ama}(x, y)) &\vee (\exists y \text{Ama}(y, x)) \\ \forall x(\exists y \neg(\neg \text{Animale}(y) \vee \text{Ama}(x, y))) &\vee (\exists y \text{Ama}(y, x)) \\ \forall x(\exists y \text{Animale}(y) \wedge \neg \text{Ama}(x, y)) &\vee (\exists y \text{Ama}(y, x)) \end{aligned}$$

3. Standardizzazione delle variabili: ogni quantificatore una variabile diversa

$$\forall x(\exists y \text{Animale}(y) \wedge \neg \text{Ama}(x, y)) \vee (\exists z \text{Ama}(z, x))$$

4. Skolemizzazione: eliminazione dei quantificatori esistenziali.

In questo caso essendo che i due quantificatori esistenziali sono nell'ambito di uno universale dobbiamo introdurre due funzioni di Skolem.

$$\forall x(\text{Animale}(F(x)) \wedge \neg \text{Ama}(x, F(x))) \vee (\text{Ama}(G(x), x))$$

5. Eliminazione quantificatori universali: possiamo portarli tutti davanti e poi eliminarli usando la convenzione che le variabili libere sono quantificate universalmente.

$$(\text{Animale}(F(x)) \wedge \neg \text{Ama}(x, F(x))) \vee (\text{Ama}(G(x), x))$$

6. Forma normale congiuntiva (congiunzione di disgiunzioni di letterali):

$$(\text{Animale}(F(x)) \vee (\text{Ama}(G(x), x))) \wedge (\neg \text{Ama}(x, F(x)) \vee (\text{Ama}(G(x), x)))$$

7. Notazione a clausole

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

8. Separazione delle variabili: clausole diverse, variabili diverse

$$\{\text{Animale}(F(x_1)), (\text{Ama}(G(x_1), x_1))\} \{\neg \text{Ama}(x_2, F(x_2)), (\text{Ama}(G(x_2), x_2))\}$$

5.10 Sostituzione

Possiamo eseguire la sostituzione in un insieme finito di associazioni tra variabili e termini, in cui ogni variabile compare una sola volta sulla sinistra. Ad esempio $\{x_1/A, x_2/f(x_3), x_3/B\}$, significa che x_1 va sostituita con A , x_2 va sostituito con $f(x_3)$ e x_3 con B (nota: sulla sinistra sono solo variabili!)

Sia σ una sostituzione e A un'espressione: $A\sigma$ è l'istanza generata dalla sostituzione (delle variabili con le corrispondenti espressioni). Possiamo scriverlo anche come $\text{Subst}(\sigma, A)$.

Esempio:

$$\text{Subst}(\{x/A, y/f(B), z/W\}, P(x, x, y, v)) = P(A, A, f(B), v)$$

$$\text{Subst}(\{x/g(y), y/z, z/f(x)\}, Q(x, y, z)) = Q(g(y), z, f(x))$$

5.11 Unificazione

L'unificazione è una operazione che ci permette di determinare se due espressioni possono essere rese identiche mediante una sostituzione di termini a variabili. Il risultato è la sostituzione che rende le due espressioni identiche detta unificatore, oppure restituisce FAIL se le espressioni non sono unificabili.

Esempio: $P(A, y, z)$ e $P(x, B, z)$ sono unificabili con $\tau = \{x/A, y/B, z/C\}$

possiamo notare come τ sia un unificatore, ma non l'unico, un altro è $\sigma = \{x/A, y/B\}$.

Possiamo infine notare come σ è più generale di τ , cioè istanzia "meno" variabili. Noi però vorremmo l'unificatore più generale di tutti (MostGeneralUnifier - MGU) e abbiamo un teorema che dice che l'unificatore più generale è unico, a parte i nomi delle variabili (l'ordine non conta).

5.11.1 Algoritmo di unificazione

L'algoritmo di unificazione prende in input due espressioni p, q e restituisce un MGU Θ se esiste. $\text{UNIFY}(p, q) = \Theta$ tale che $\text{SUBST}(\Theta, p) = \text{SUBST}(\Theta, q)$, altrimenti FAIL. L'algoritmo esplora in parallelo le due espressioni e costruisce l'unificatore strada facendo. Appena trova espressioni non unificabili fallisce. Una causa di fallimento sono sostituzioni del tipo $x=f(x)$, cioè sostituzioni in cui x occorre già all'interno dell'espressione. Questo controllo si chiama Occurr-Check() e in tal caso restituisce FAIL (è un controllo di complessità quadratica).

5.11.2 Esempi

Esempio 1

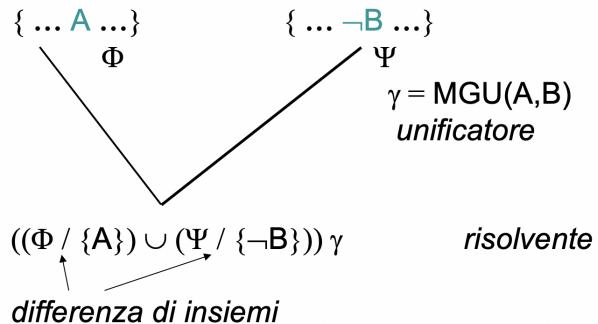
- $\text{UNIFY}(P(A, y, z), P(x, B, z), \{\})$
- $\text{UNIFY}((A, y, z), (x, B, z), \text{UNIFY}(P, P, \{\}))$
- $\text{UNIFY}((A, y, z), (x, B, z), \{ \})$
- $\text{UNIFY}((y, z), (B, z), \text{UNIFY}(A, x, \{\}))$
- $\text{UNIFY}((y, z), (B, z), \text{UNIFY}(x, A, \{\}))$
- $\text{UNIFY}((y, z), (B,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

Esempio 2

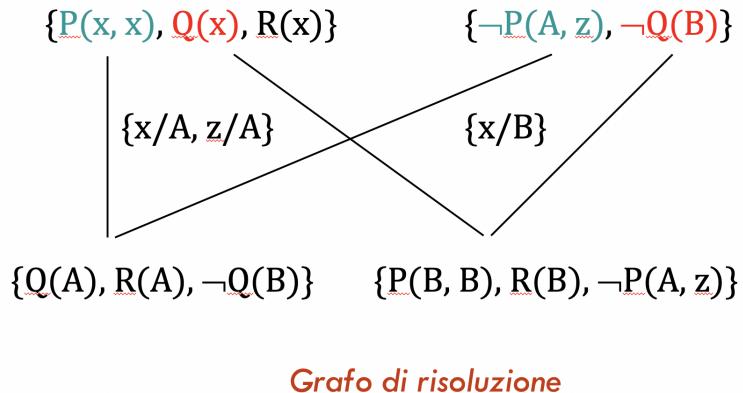
- UNIFY($P(f(x), x)$, $P(z, z)$, {})
- UNIFY(($f(x), x$), (z, z), UNIFY(P, P , {}))
- UNIFY(($f(x), x$), (z, z), {})
- UNIFY((x, z), UNIFY($f(x), z$, {}))
- UNIFY((x, z), UNIFY($z, f(x)$, {}))
- UNIFY((x, z), $\{z/f(x)\}$)
- UNIFY-VAR($x, z, \{z/f(x)\}$)
- UNIFY($x, f(x), \{z/f(x)\}$)
- OCCUR-CHECK($x, f(x)$)
- FAIL

5.12 Metodo di risoluzione per FOL

Siamo ora in grado di definire in generale la regola di risoluzione per FOL.

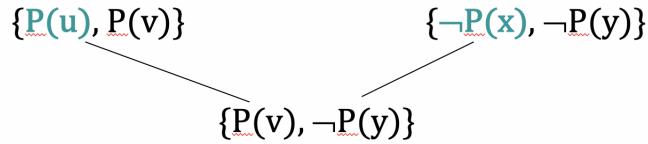


5.12.1 Esempio metodo di risoluzione



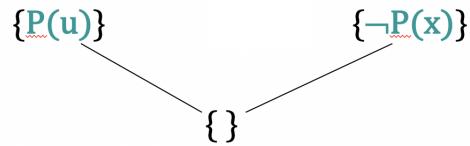
5.12.2 Problemi

Le seguenti clausole dovrebbero produrre la clausola vuota invece...



Se un sottoinsieme dei letterali di una clausola può essere unificato allora la clausola ottenuta dopo tale unificazione si dice fattore della clausola originaria.

Il metodo di risoluzione va applicato ai fattori delle clausole:



5.12.3 Completezza del metodo di risoluzione

La deduzione per risoluzione è corretta se $\Gamma \vdash_{RES} A$ allora $\Gamma \models A$

La deduzione per risoluzione non è completa perché può essere che $\Gamma \models A$ ma non $\Gamma \vdash_{RES} A$

Un esempio è $\{ \} \models \{P, \neg P\}$ ma non è vero che $\{ \} \vdash_{RES} \{P, \neg P\}$

5.13 Refutazione

Come per PROP, il teorema di refutazione ci suggerisce un metodo alternativo completo. Il teorema di refutazione dice che $\Gamma \cup \{\neg A\}$ è insoddisfacibile sse $\Gamma \models A$.

Possiamo dire quindi che Γ è insoddisfacibile sse $\Gamma \vdash_{RES} \{ \}$.

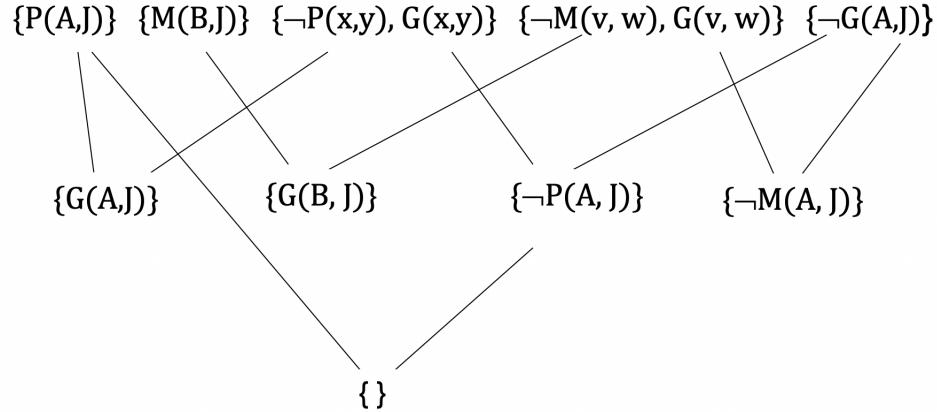
Abbiamo quindi un metodo meccanizzabile, corretto e completo: basta aggiungere il negato della formula da dimostrare e provare a generare la clausola vuota.

5.13.1 Esempio di Refutazione

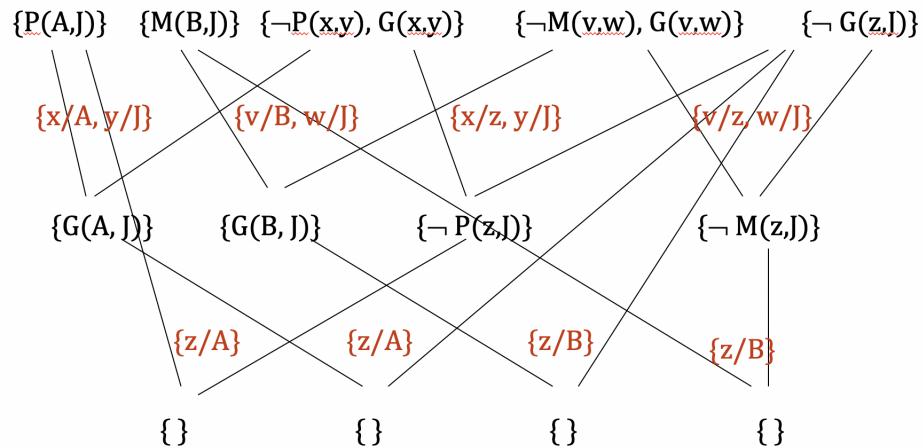
La nostra KB è formata da:

- $\{P(A, J)\}$ A è padre di J
- $\{M(B, J)\}$ B è madre di J
- $\{\neg P(x, y), G(x, y)\}$ padre implica genitore
- $\{\neg M(v, w), G(v, w)\}$ madre implica genitore

Il nostro goal è sapere se A è genitore di J, in forma a clausole se $\{(A, J)\}$ appartiene alla KB. Per refutazione possiamo quindi aggiungere alla KB la negazione del goal $\{\neg G(A, J)\}$ e proviamo a dedurre la clausola vuota.



E per domande del tipo "chi sono i genitori di J?"
Si cerca di dimostrare che $\exists z G(z, J) \rightarrow \{\neg G(z, J)\}$ e la risposta sono tutti i possibili legami per z che consentono di ottenere la clausola vuota.



Le risposte sono: A, B

5.14 Risoluzione efficiente per FOL

Come si può rendere più efficiente il metodo di risoluzione per FOL? Sfrutteremo tecniche per esplorare in maniera efficiente il grafo di risoluzione, probabilmente senza perdere completezza. Distingueremo tra:

- Strategie di cancellazione: ci sono clausole che posso eliminare?
- Strategie di restrizione: posso usare ad ogni passo solo alcune clausole?
- Strategie di ordinamento: posso 'risolvere' i letterali in un ordine specifico?

5.14.1 Strategie di cancellazione

Si tratta di rimuovere dalla KB (ai fini della dimostrazione) certe clausole che non potranno mai essere utili nel processo di risoluzione.

- Clausole con letterali puri: sono quelle clausole che non hanno il loro negato nella KB, ad esempio in $\{\neg P, \neg Q, R\} \{\neg P, S\} \{\neg Q, S\} \{P\} \{Q\} \{\neg R\}$ la seconda e terza clausola contengono il letterale puro S. Le clausole con letterali puri non potranno mai essere risolte con altre clausole per ottenere $\{\}$, tanto vale eliminarle che non si perdono soluzioni.

- Tautologie: la tautologie vengono rappresentate da quelle clausole che contengono due letterali identici e complementari. Ad esempio: $\{P(A), \neg P(A), \dots\}$ oppure $\{P(x), Q(y), \neg Q(y)\}$. La loro rimozione non influenza la soddisfacibilità.

NOTA: Le tautologie possono essere generate, quindi è un controllo da fare ad ogni passo.

- Clausole sussunte⁴: dovremo eliminare le clausole implicate. In generale: α sussume β se solo se $\exists \sigma$ tale che $\alpha\sigma \subseteq \beta$ cioè se un'istanza di α con la sostituzione σ è un sottoinsieme di β .

Ad esempio: $\{P(x), Q(y)\}$ sussume $\{P(A), Q(v), R(w)\}$ infatti $\{P(x), Q(y)\} \{x/A, y/v\} = \{P(A), Q(v)\} \subseteq \{P(A), Q(v), R(w)\}$.

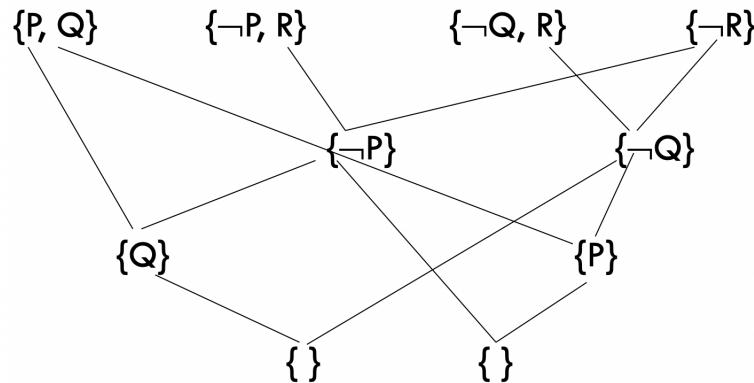
Concludendo, β può essere ricavata da α , quindi β può essere eliminata senza perdere soluzioni.

NOTA: Le clausole sussunte possono essere generate, quindi è un altro controllo da fare ad ogni passo.

5.14.2 Strategie di restrizione

Ad ogni passo si sceglie tra un sottoinsieme delle possibili clausole e si applica l'algoritmo su di esse. Tra le strategie di restrizione possibili:

- Risoluzione unitaria: almeno una delle clausole è unitaria (contiene un solo letterale)



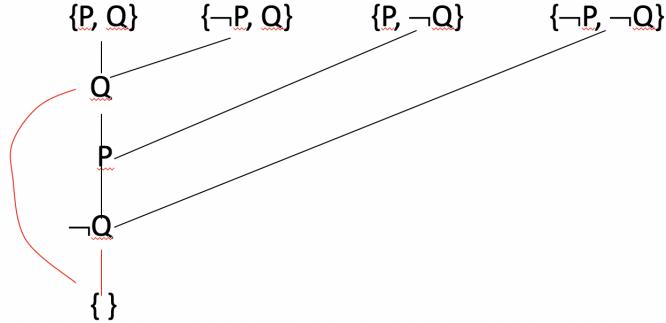
Facile da implementare, converge rapidamente ma la strategia non è completa.

Esempio: $\{P, Q\} \{\neg P, Q\} \{P, \neg Q\} \{\neg P, \neg Q\} \vdash_{RES} \{\}$, ma non otteniamo lo stesso risultato con la risoluzione a clausole unitarie.

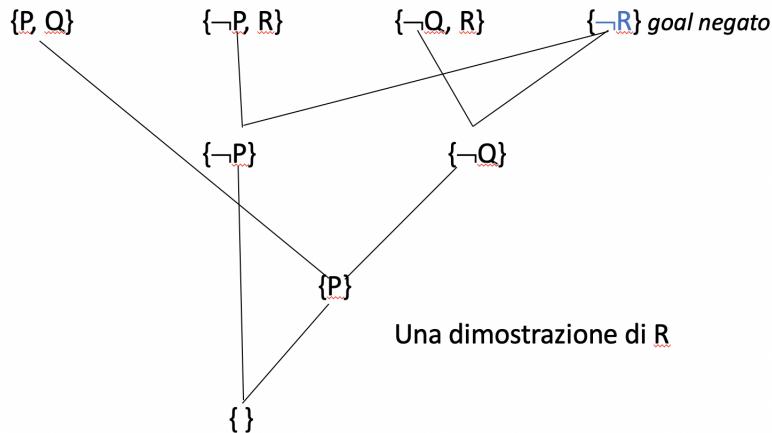
La strategia però è completa per clausole Horn, cioè quelle clausole con al massimo un letterale positivo (ma possono avere più letterali nella stessa clausola), nell'esempio di prima $\{P, Q\}$ non è una clausola Horn!

⁴sussume significa “è più generale di” o “implica”

- Risoluzione lineare: sfruttiamo l'ultima clausola generata con una clausola da input (dalla KB iniziale) oppure una clausola antenata. La risoluzione è completa se applicata insieme ad una refutazione.

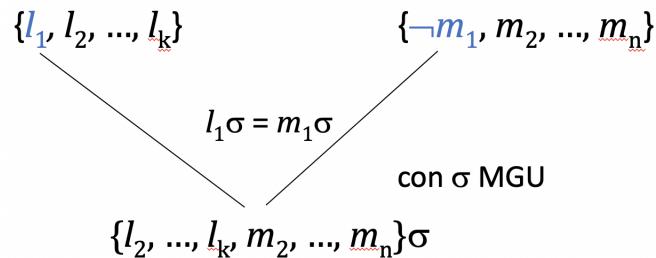


- Risoluzione guidata dal goal: sfruttiamo un insieme di supporto, cioè un sotto-insieme della KB responsabile dell'insoddisfacibilità. Tipicamente, assumendo la KB iniziale consistente, si sceglie come insieme di supporto iniziale il negato della clausola goal (è come procedere all'indietro dal goal). Come si può notare, la strategia è completa per la refutazione.



5.14.3 Strategie di ordinamento

Ogni clausola è un insieme ordinato di letterali e si possono unificare solo i primi letterali delle clausole, l'ordinamento deve essere rispettato nel risolvente. La risoluzione ordinata è completa per clausole Horn.



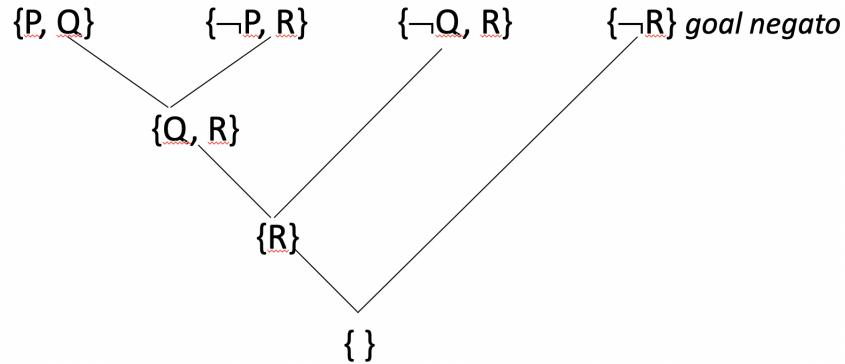


Figure 9: risoluzione ordinata esempio

5.15 Sistemi a regole

Le clausole Horn definite sono quelle clausole con esattamente un letterale positivo⁵. Possono essere riscritte come fatti e regole, cioè

$\neg P_1 \vee \dots \vee \neg P_k \vee Q$ viene riscritta come

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

$(P_1 \wedge \dots \wedge P_k) \Rightarrow Q$ con Q chiamato fatto.

Se la KB contiene solo clausole Horn definite, i meccanismi inferenziali sono molto più semplici senza rinunciare alla completezza. Ovviamente è un sistema restrittivo, non può coincidere con FOL.

5.15.1 Regole in avanti e indietro

- Concatenazione all'indietro (Backward Chaining): approccio guidato dall'obiettivo. Le regole sono applicate alla rovescia. (Programmazione Logica - PROLOG)
- Concatenazione in avanti (Forward Chaining): approccio guidato dai dati. Le regole sono applicate nel senso "antecedente-conseguente" (Basi di dati deduttive)

5.16 Programmazione logica (Backward Chaining)

I programmi logici sono KB costituiti da clausole Horn definite, vengono espressi come fatti e regole, con una sintassi alternativa:

A fatto
 A :- B_1, B_2, \dots, B_n regola, con testa (la conseguenza) A

Come convenzione, in programmazione logica, le variabili sono indicate con lettere maiuscole mentre le costanti con lettere minuscole.

Esempio: Se

$B_1 \wedge B_2 \wedge \dots \wedge B_n$ è il goal,
 $\neg(B_1 \wedge B_2 \wedge \dots \wedge B_n) \vee False$ è il goal negato, ovvero
 $(B_1 \wedge B_2 \wedge \dots \wedge B_n) \Rightarrow False$ che viene scritto
 $\neg B_1, \neg B_2, \dots, \neg B_n$ omettendo il conseguente.

⁵Clausole Horn: possono avere più letterali ma al massimo uno solo positivo
 Clausole Horn Definite: possono avere più letterali ma hanno esattamente un letterale positivo

Nei programmi logici abbiamo due tipi di interpretazione per le regole:

1. Interpretazione dichiarativa: $A \vdash B_1, B_2, \dots, B_n$ quindi A è vero se sono veri B_1, B_2, \dots, B_n (in accordo al significato logico dell'implicazione).
2. Interpretazione procedurale: la testa può essere vista come una chiamata di procedura e il corpo come una serie di procedure da eseguire in sequenza.

5.16.1 Risoluzione SLD

La risoluzione SLD (Selection Linear Definite-clauses) è una strategia ordinata, basata su un insieme di supporto formato dalla clausola goal ed è lineare in input. La risoluzione SLD è completa per clausole Horn.

5.16.2 Alberi di risoluzione SLD

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

- ogni foglia dell'albero corrisponde a un goal
- la radice è $\vdash G_1, G_2, \dots, G_k$ che è il nostro goal
- sia $\vdash G_1, G_2, \dots, G_k$ 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 $A \vdash B_1, \dots, B_k$ e A è unificabile con G_1 e $\gamma = \text{MGU}(A, G_1)$
un discendente è il goal $\vdash (B_1, \dots, B_k, G_2, \dots, G_k)\gamma$
- i nodi che sono clausole vuote sono successi
- i nodi che non hanno successori sono fallimenti

5.16.3 Esempio di albero SLD

Abbiamo le seguenti regole:

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(gio, mark).
6. Padre(gio, luc).
7. Madre(lia, gio).
8. \vdash Antenato(lia, mark). (goal negato, quindi radice del nostro albero)

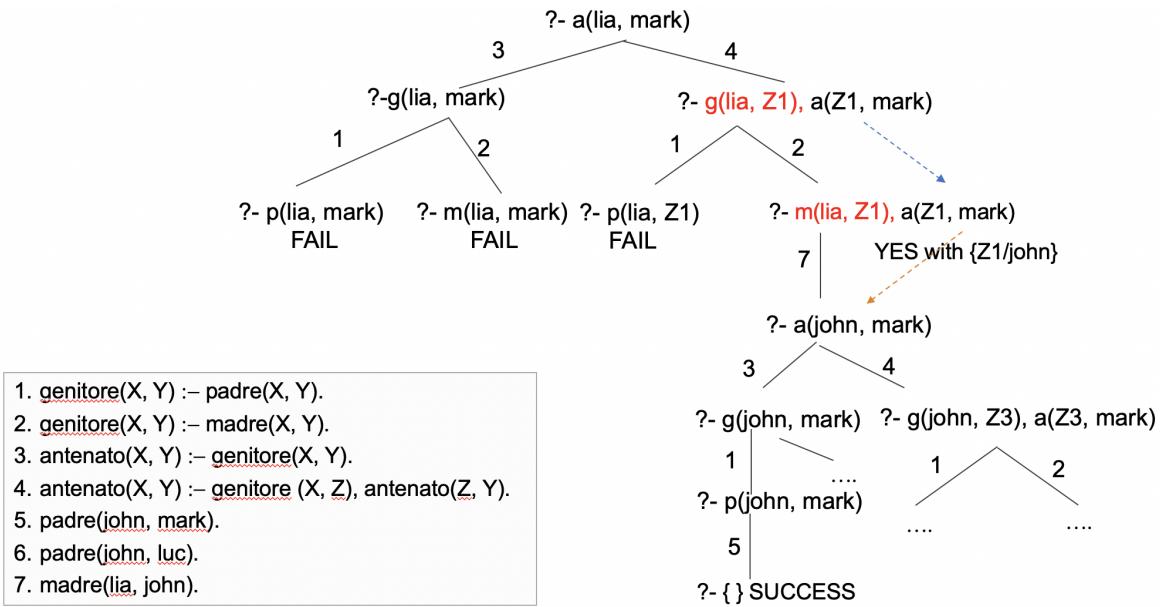


Figure 10: Albero SLD per il goal `antenato(lia,mark)`

La strategia è completa per clausole Horn definite. Se $P \cup \{\neg G\}$ è insoddisfacibile, allora una delle foglie deve essere la clausola vuota (successo).

A seconda di come l'albero viene visitato si potrebbe anche non trovare la clausola vuota, poiché potremmo trovare un cammino infinito e non trovare mai la soluzione. La strategia di ricerca può essere responsabile dell'incompletezza! In PROLOG, il più famoso linguaggio di programmazione logica, la visita dell'albero di risoluzione avviene con una ricerca in profondità, con backtracking in caso di fallimento. La strategia di PROLOG non è completa, le regole vengono applicate nell'ordine in cui sono immesse, infine PROLOG omette l'Occur-Check per motivi di efficienza.

5.17 Sistemi a regole in avanti (Forward Chaining - FOL_FC_Ask)

$$\frac{p'_1, p'_2, \dots, p'_n \quad (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{(q)\Theta}$$

Conosciamo la regola del Modus Ponens generalizzato:

con $\Theta = \text{MGU}(p'_i, p_i)$, per ogni i

ma in realtà per applicare la regola dovremo: istanziare gli universali, istanziare le regole e poi applicare il Modus Ponens classico.

Supponiamo di avere nella KB: `King(John)`, `Greedy(y)` e `King(x) \wedge Greedy(x) \Rightarrow Evil(x)`.

Con $\Theta = \{x/John, y/John\}$ si ottiene:

`King(John), Greedy(John) e King(John) \wedge Greedy(John) \Rightarrow Evil(John)`.

Quindi la conclusione della regola è `Evil(John)`.

5.17.1 Esempio di concatenazione in avanti

È un crimine per un Americano vendere armi a una nazione ostile. Il paese Nono, un nemico dell'America, ha dei missili, e tutti i missili gli sono stati venduti dal colonnello West, un Americano.

Dimostrare che West è un criminale. Formalizzazione:

- $American(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Ostile(z) \Rightarrow Criminale(x)$
- $\exists x Possiede(Nono, x) \wedge Missile(x)$ lo istanziamo con $Possiede(Nono, M_1) \wedge Missile(M_1)$
- $Missile(x) \wedge Possiede(Nono, x) \Rightarrow Vende(West, x, Nono)$
- $Missile(x) \Rightarrow Arma(x)$
- $Nemico(x, America) \Rightarrow Ostile(x)$
- $American(West)$
- $Nemico(Nono, America)$

Un semplice processo inferenziale chiamato FOL_FC_Ask, applica ripetutamente il Modus Ponens generalizzato per ottenere nuovi fatti fino a che si dimostra quello che si desidera o nessun fatto nuovo può essere aggiunto. E' una strategia di ricerca sistematica in ampiezza, in questo esempio non ci sono funzioni e il processo converge: siamo nelle condizioni di un database DATALOG (cioè basi di dati deduttive). Come si procede:

1. $Possiede(Nono, M_1) \wedge Missile(M_1)$
2. $Missile(x) \wedge Possiede(Nono, x) \Rightarrow Vende(West, x, Nono)$ è soddisfatta con $\{x/M_1\}$ e viene aggiunto $Vende(West, M_1, Nono)$
3. $Missile(x) \Rightarrow Arma(x)$ è soddisfatta con $\{x/M_1\}$ e viene aggiunto $Arma(M_1)$
4. sappiamo che $Nemico(Nono, America)$ quindi $Nemico(x, America) \Rightarrow Ostile(x)$ viene soddisfatta con $\{x/Nono\}$ e viene aggiunto $Ostile(Nono)$
5. per finire: $American(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Ostile(z) \Rightarrow Criminale(x)$ è soddisfatta con la sostituzione $\{x/West, y/M_1, z/Nono\}$ e $Criminale(West)$ viene aggiunto.

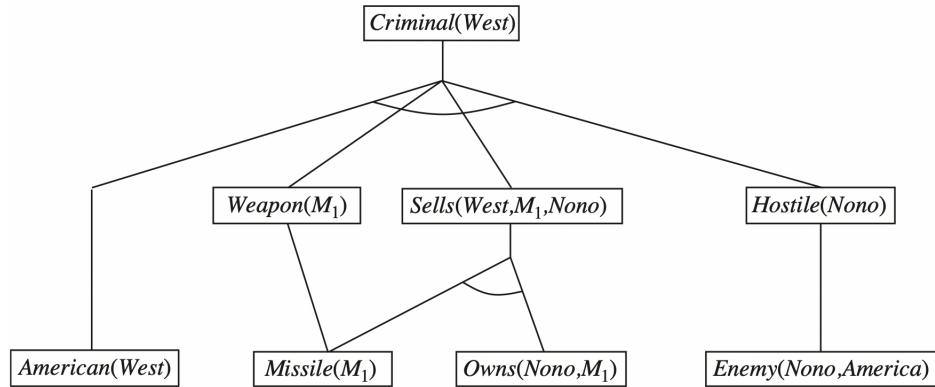


Figure 11: Processo di dimostrazione in avanti

5.17.2 Analisi di FOL_FC_Ask

Il processo è corretto perché il Modus Ponens generalizzato è corretto. E' completo per KB di clausole Horn definite. Diventa completo e convergente per calcolo proposizionale e per KB di tipo DATALOG (senza funzioni) perché la chiusura deduttiva è un insieme finito. Completo anche con funzioni, ma il processo potrebbe non terminare (il problema diventa semidecidibile).

Appunti di Introduzione all’Intelligenza Artificiale Unipi - 3

Parte

Raffaele Apetino

Marzo 2020

Contents

1	Introduzione - Machine Learning	2
1.1	Un esempio concreto	2
1.2	Quando usare il machine learning?	2
1.3	Perché usare il machine learning?	2
2	Overview di un sistema ML	3
2.0.1	Riconoscimento caratteri	3
2.1	TASK: Supervised Learning	4
2.2	TASK: UNsupervised Learning	4
2.3	MODEL	5
2.3.1	Esempi di Modelli	5
2.4	LEARNING ALGORITHM	5
2.5	VALIDATION - Generalizzazione	6
3	Concept Learning	6
3.0.1	Esempio: Apprendimento Funzione Booleana	7
3.1	Regole Congiuntive	7
3.2	Problema: Enjoy Sport	8
3.3	Rappresentare le Ipotesi	8
3.4	TASK del Concept Learning	8
3.5	Apprendimento Induttivo	9
3.6	Qual è il numero di istanze, concetti e ipotesi di un problema?	9
3.7	Da ordine generale a ordine specifico	9
3.8	Algoritmo Find-S (Specific)	10
3.8.1	Esempio su Enjoy Sport	11
3.8.2	Proprietà di Find-S	11
3.8.3	Aspetti negativi di Find-S	11
3.9	Version Spaces	11
3.10	Algoritmo List-Then-Eliminate	12
3.10.1	Esempio di Version Space	12
3.11	Algoritmo Candidate Elimination	13
3.11.1	Esempio di Candidate Elimination	14

4 Inductive Bias	15
4.1 Sistema di apprendimento Unbiased	15
4.1.1 Il learning Unbiased è utile?	15
4.2 Definizione Formale di Inductive Bias	15
4.3 Sistemi Induttivi e Deduttivi equivalenti	16
4.4 Tre (algoritmi) sistemi di apprendimento con Bias differenti	16
5 Modelli Lineari	17
5.1 Esempio di regressione	17
5.2 Modello di Regressione Lineare Semplice (Univariata)	17
5.3 Costruzione della funzione via LMS	18
5.3.1 Gradiente discendente	20
5.4 Modello di Regressione Lineare Multivariato	21
5.4.1 Notazione dei dati	21
5.5 Hyperplane	22
5.6 Gradiente Discendente Algoritmo	22
5.7 Vantaggi dei modelli lineari	23
5.8 Limitazioni	23
5.8.1 Esempio	23
5.9 Linear Basis Expansion	24
5.9.1 Esempi	24
5.9.2 Tradeoff per la complessità	25
5.10 Regolarizzazione - Ridge Regression	26
5.11 Limitazione delle funzioni a base fissa	27
5.12 Classificazione con il modello lineare	28
5.12.1 Visione geometrica dell'iperpiano	28
5.12.2 Classificazione attraverso Decision Boundary lineare	29
5.12.3 Esempio Terremoti/Esplosioni Nucleari	29
5.12.4 Esempio Email Spam	29
5.13 Il problema di apprendimento (per classificatori lineari)	30
5.13.1 Δw come regola della correzione dell'errore	30
5.14 Classificazione dei pattern	30
5.15 Congiunzioni nel caso di Modello Lineare	31
5.15.1 Limitazioni	31
5.16 Altri sistemi di apprendimento per la classificazione	32
5.17 Conclusioni sui Modelli Lineari	32
6 Alberi di decisione	33
6.1 Problema del PlayTennis	33
6.2 (Alg. ID3) Induzione Top-Down sugli alberi di decisione	34
6.3 Entropia: Come scegliere il miglior Attributo	34
6.4 Information Gain	35
6.4.1 Esempio	36
6.5 Problemi con Information Gain	36
6.6 Gain Ratio	37
6.7 Considerazioni sulla ricerca nello Spazio delle Ipotesi (in DT Learning)	37
6.8 Inductive Bias in DT Learning	37
6.9 Problemi con gli alberi di decisione	38
6.9.1 Evitare l'Overfitting	39
6.9.2 Potatura con errore ridotto	39
6.9.3 Regola della post-potatura	39
6.10 Problema: Valori continui degli attributi	40

6.11 Problema: Dati di training incompleti	40
6.12 Problema: attributi con costi differenti	40
6.13 Visione Geometrica (Decision Boundaries dei DT)	41
6.14 Conclusioni sugli alberi di decisione	41
7 Validation	41
7.1 Obiettivi Validazione	42
7.2 Holdout Cross-Validation	42
7.3 Meta-Algoritmo per l'utilizzo del Data Set	42
7.3.1 Esempio	43
7.3.2 Esempio perché separare VL e TS	43
7.4 K-fold cross validation	44
7.5 Esempio di Model Selection e Assessment	44
7.6 Comportamento tipico di un algoritmo di apprendimento	45
7.7 Statistical Learning Theory	46
7.8 Teoria di Vapnik-Chervonenkis + SLT	46
7.9 Structural Risk Minimization	47
7.10 Conclusioni	47
8 Support Vector Machine	47
8.0.1 Obiettivi utilizzo SVM	47
8.1 Maximum Margin Classifier (controllo complessità)	48
8.2 Rappresentazione canonica dell'iperpiano e Support Vector	49
8.3 Verso l'ottimizzazione del margine	50
8.4 Problema di ottimizzazione quadratico	50
8.5 Nuovo classificatore $h(x)$ (Problema Duale)	50
8.6 Margine Soft	51
8.7 Mapping per Spazi Dimensionali Ampi	52
8.7.1 Esempi di Kernel	53
8.8 Riassunto procedimento SVM con Kernel Fun.	53
8.9 Utilizzare bene SVM	53
9 K-Nearest Neighbors	54
9.1 1-Nearest Neighbor	54
9.2 K-NN	55
9.3 Considerazioni su K-NN	56
9.4 Distance Based Methods	56
10 Unsupervised Learning (ripasso)	56
10.1 K-means	56
10.2 Limitazioni K-means	57
10.3 Preprocessing dei dati	58
11 Altri Task del Machine Learning	58
12 Altri Modelli del Machine Learning	59
12.1 Reti Neurali	59
12.2 Deep Learning	59

1 Introduzione - Machine Learning

Il problema dell'apprendimento è uno dei problemi principali dell'intelligenza sia artificiale che biologica. L'apprendimento automatico (Machine Learning) è emerso come un'area di ricerca che combina gli obiettivi della creazione di macchine in grado di apprendere e strumenti statistici/adattivi. Perché le macchine dovrebbero imparare da sole? Poiché c'è una crescente necessità di analizzare dati empirici difficili da gestire con la normale programmazione (paradigma DATA-DRIVEN). Il compito dell'apprendimento automatico è la creazione di sistemi intelligenti adattivi, in grado di analizzare grandi quantità di dati.

1.1 Un esempio concreto

Esempi concreti di applicazione di ML possono essere la classificazione delle email spam oppure il riconoscimento di caratteri scritti a mano, dei visi o del parlato. Non ci sono regole o conoscenza pregressa per trovare la soluzione, ma diventa più facile avendo una fonte di esperienza per apprendere (dati di cui conosciamo già il risultato). Ad esempio nel riconoscimento facciale si combinano reti neurali e altri approcci di ML partendo da milioni di immagini facciali appartenenti a diversi individui.

1.2 Quando usare il machine learning?

Il machine learning è una grande opportunità ma deve essere controllato. Usiamo l'apprendimento automatico quando:

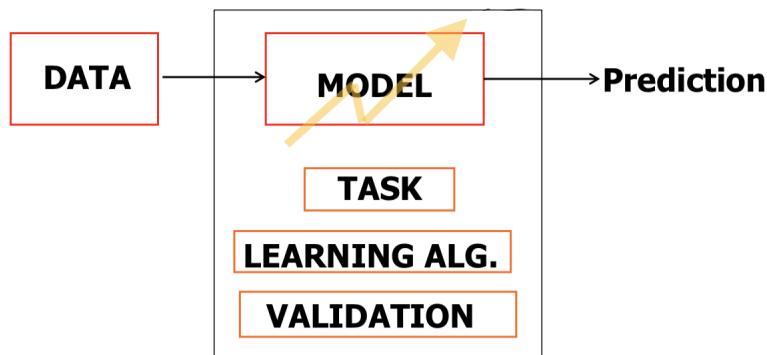
- non abbiamo conoscenza per spiegare il fenomeno oppure è difficile formalizzarlo
- abbiamo dati incerti, rumorosi o incompleti, che ostacolano la formalizzazione di soluzioni
- ci troviamo in ambienti dinamici non conosciuti in precedenza

Abbiamo però anche dei requisiti per poter applicare ML ai nostri problemi. È necessaria una fonte di esperienza formativa, ma spesso è difficile raccogliere molti dati rappresentativi (basti pensare alla raccolta di dati riguardanti una malattia rara). Dobbiamo anche considerare una tolleranza sulla precisione dei dati (se accettiamo di dare pochi dati in input, o poco significativi, dobbiamo anche tollerare una certa percentuale di errore, che però in alcuni casi non può essere accettata).

1.3 Perché usare il machine learning?

È una opportunità per conoscere nuovi paradigmi informatici con un approccio differente dalla programmazione standard, algoritmica e dall'IA classica. Serve per trovare soluzioni approssimate a problemi molto difficili. Non è una metodologia approssimativa, è un approccio rigoroso per trovare funzioni approssimative per affrontare problemi complessi.

2 Overview di un sistema ML

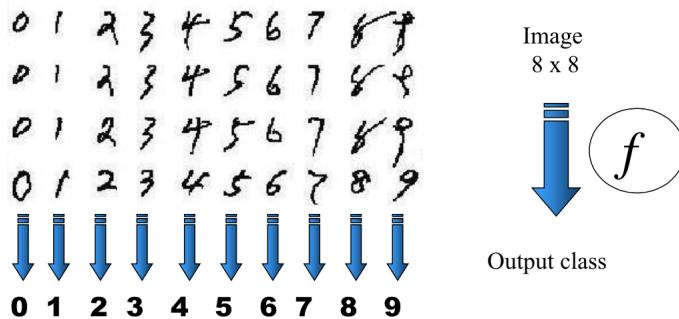


- DATA: i dati li ricaviamo dalle osservazioni del mondo
- MODEL: i dati passano attraverso un modello. Il modello non è fissato: ha dei parametri liberi, modificabili per fornire le predizioni in modo che siano quelle che vorremmo che fossero fornite.
 - TASK: Supervised o Unsupervised Learning
 - LEARNING ALGORITHM: algoritmo che cerca la soluzione in uno spazio di stati
 - VALIDATION: valutazione statistica del modello (quanto riesce ad essere accurato sui dati futuri).
- PREDICTION: è l'output restituito dal modello

"Apprendere" viene visto come una approssimazione di una funzione sconosciuta ricavata dagli esempi.

2.0.1 Riconoscimento caratteri

Un esempio "pilota" che possiamo fare è il riconoscimento di caratteri scritti a mano. Il problema è racchiuso nel riconoscere una funzione che passa per un insieme di punti. Come input abbiamo una collezione di immagini di caratteri scritti a mano. Il problema è costruire un modello che riceve in input queste matrici di 8x8 pixel e come output restituire la cifra riconosciuta.



Non sappiamo quale sia la funzione, perché in quel caso la scriveremmo direttamente in un algoritmo. Cerchiamo quindi di approssimare una funzione facendo restituire come output una classificazione della matrice di pixel. Il problema di classificazione è dato dalla formalizzazione della soluzione esatta, poiché ci possiamo trovare con dati rumorosi o ambigui. Risulta però facile raccogliere collezioni

di esempi etichettati, cioè esempi con soluzioni associate.

Generalizzando il problema possiamo usare la tecnica del Supervised Learning. In particolare abbiamo in input uno spazio contentente "x" dati etichettati, dobbiamo costruire una funzione generale a partire dagli esempi e come output dare una categoria o valori reali.

2.1 TASK: Supervised Learning

Il Supervised Learning è una tecnica di apprendimento automatico che mira a istruire un sistema informatico, in modo da consentirgli di elaborare automaticamente previsioni sui valori di uscita di un sistema rispetto ad un input sulla base di una serie di esempi ideali, costituiti da coppie di input e di output, che gli vengono inizialmente forniti.

Quindi vengono dati esempi di training nella forma di coppie $\langle \text{input}, \text{output} \rangle = \langle x, d \rangle$ (esempi etichettati) per la creazione di una funzione sconosciuta f . Definiamo il valore target come il valore " d " che vogliamo ottenere come output (e che ci viene dato tramite gli esempi).

Bisogna trovare una buona approssimazione di f , cioè una ipotesi h che può essere usata per predire l'output sui dati sconosciuti x' .

L'obiettivo è dare in output una etichetta numerica o la classificazione del dato.

- Classificazione: la funzione a valori discreti $f(x)$ restituisce la presunta classe corretta per x .
- Regressione: consiste nell'approssimare a valori reali la funzione target.

Entrambi sono compiti di approssimazione di funzione.

2.2 TASK: UNsupervised Learning

L'apprendimento non supervisionato è una tecnica di apprendimento automatico che consiste nel fornire al sistema informatico una serie di input (esperienza del sistema) che egli riclassificherà ed organizzerà sulla base di caratteristiche comuni per cercare di effettuare ragionamenti e previsioni sugli input successivi. Abbiamo quindi un TR set che è un insieme di dati non etichettati, il compito è quello di raggruppare i dati in insiemi consistenti. I principali algoritmi sono:

- Clustering: raggruppamento di elementi omogenei in un insieme di dati (cluster) identificando un centroide.
- Dimensionality reduction / Visualization / Preprocessing
- Modeling the data density

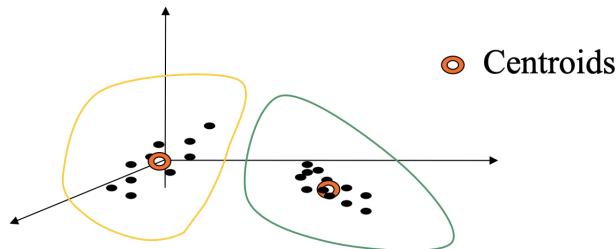


Figure 1: Esempio di clustering

2.3 MODEL

Il compito è quello di catturare e descrivere le relazioni tra i dati sulla base del TASK. Il modello definisce la classe delle funzioni che la macchina può implementare (cioè definisce lo spazio delle ipotesi H).

Concetti utili:

- Training examples: fanno parte del Supervised Learning ed è un insieme di esempi nella forma $(x, f(x))$ dove x è un vettore di caratteristiche e $f(x)$ è il valore obiettivo.
- Target function: la funzione obiettivo f il più possibile corretta
- Hypothesis: è una funzione h che è ritenuta di essere simile a f . La funzione h è data in un linguaggio capace di esprimere le relazioni tra i dati.
- Hypotheses space: è lo spazio di tutte le ipotesi che possono essere output dell'algoritmo. È dove cerchiamo la funzione h .

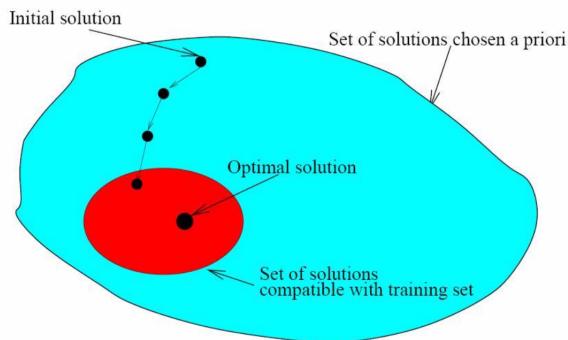
I linguaggi per le ipotesi h devono saper esprimere modelli di ML: ad esempio la logica del primo ordine, equazioni numeriche oppure calcolo delle probabilità.

2.3.1 Esempi di Modelli

- Modelli Lineari: la rappresentazione dello spazio delle ipotesi H definisce uno spazio continuo parametrizzato di potenziali ipotesi. Come parametro abbiamo w , ed ogni assegnamento di w è una ipotesi differente. (es: $h_w(x) = w_1x + w_0$)
- Regole simboliche: lo spazio delle ipotesi è basato su rappresentazioni discrete, sono possibili differenti regole come ad esempio: if $(x_1=0 \text{ and } x_2=1)$ then $h(x)=1$ else $h(x)=0$
- Modelli probabilistici: stimare $p(x,y)$
- Approcci basati sull'istanza: confronta le nuove istanze del problema con le istanze osservate durante l'addestramento, che sono state memorizzate. Si chiama basato sull'istanza perché costruisce ipotesi direttamente dalle istanze di addestramento stesse.

2.4 LEARNING ALGORITHM

Gli algoritmi di apprendimento si basano su DATA, TASK e MODEL. L'euristica consiste nella ricerca delle migliori ipotesi nello spazio delle ipotesi H , cioè la miglior approssimazione della funzione obiettivo f sconosciuta (ad esempio i parametri liberi del modello vengono adattati al compito da svolgere: il miglior w nel modello lineare, la miglior regola per regole simboliche, ecc...). H potrebbe non coincidere con l'insieme di tutte le possibili funzioni e la ricerca può non risultare esaustiva, per questo bisogna fare assunzioni (vedremo in seguito il ruolo dell'Inductive Bias).



2.5 VALIDATION - Generalizzazione

L'apprendimento può essere quindi definito come la ricerca di una buona funzione, in uno spazio di funzioni restituito da dati conosciuti. "Buona funzione" considerando l'errore di generalizzazione (o capacità di generalizzazione) cioè quanto accuratamente il modello prevede nuovi campioni di dati. La generalizzazione è cruciale in ML → strumenti corretti di ML.

- Fase di apprendimento (training, fitting): costruire il modello dai dati di training conosciuti (Training Set).
- Fase predittiva (test): si applica la funzione costruita a un nuovo esempio. Si prende in input x , si computa la risposta dal modello e confrontiamo l'output con il nostro target (che il modello non ha mai visto). Si esegue una valutazione dell'ipotesi predittiva, ad esempio la valutazione della capacità di generalizzazione (valutazione statistica del modello, quanto riesce ad essere accurato sui dati futuri). È bene notare che essere performanti in ML significa predire accuratamente, cioè la performance è stimata dagli errori effettuati calcolando la funzione sul Test Set (che è diverso dal Training set!).

3 Concept Learning

Il Concept Learning anche noto come apprendimento di categoria è definito come "la ricerca di attributi che possono essere utilizzati per distinguere i modelli dai non modelli di varie categorie". Più semplicemente, i concetti sono le categorie che ci aiutano a classificare oggetti, eventi o idee, basandoci sul fatto che ogni oggetto, evento o idea ha un insieme di caratteristiche rilevanti comuni. In un compito di Concept Learning, una macchina apprende come classificare degli oggetti mostrandole prima una serie di esempi associati alle loro etichette di classe. La macchina semplifica quello che ha osservato "condensandolo" sotto forma di un esempio. Questa versione semplificata di ciò che è stato appreso viene quindi applicata a esempi futuri.

Lavoriamo in uno spazio discreto strutturato come lo spazio delle ipotesi, useremo come rappresentazione la congiunzione di letterali e vedremo algoritmi come Find-S ed Candidate Elimination. Praticamente in questa sezione andremo ad vedere in cosa consiste la classificazione del Supervised Learning, cioè come la funzione $f(x)$ restituisce la presunta classe corretta per x .

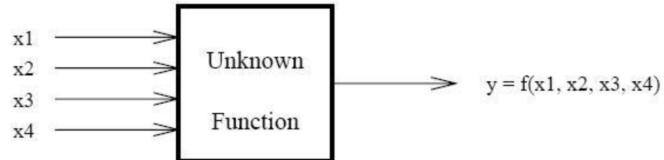
Concept Learning: dedurre una funzione booleana (con dominio X e codominio $\{t,f\}$) da esempi di allenamento positivi e negativi (dove X è lo spazio che contiene le istanze)

Un Training Example è definito come coppia $\langle x, c(x) \rangle$ (le coppie sono contenute nel Training Set). L'ipotesi h : $X \rightarrow \{0, 1\}$ soddisfa x se $h(x)=1$.

Una ipotesi h è consistente con un esempio di allenamento $\langle x, c(x) \rangle$ se $h(x)=c(x)$ con x appartenente a X . È consistente anche con D^1 se $h(x)=c(x)$ per ogni esempio di allenamento $\langle x, c(x) \rangle$ in D .

¹è sempre il Training Set insieme degli esempi

3.0.1 Esempio: Apprendimento Funzione Booleana



Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Questo é un problema mal posto poiché potremmo violare l'esistenza, unicità o stabilità della soluzione. Come possiamo vedere dall'immagine abbiamo in input 4 variabili e in output un singolo valore. Nella tabella troviamo il nostro training set (D) e il nostro compito é quello di trovare la funzione che dati in input quei valori restituisce l'output definito dalla tabella. Ci sono 2^{2^4} possibili funzioni booleane per 4 ingressi, questo perché abbiamo 2^4 possibili ingressi e per ognuno possiamo rispondere 0 o 1. Non possiamo sapere quale sia la funzione corretta fino a che non abbiamo visto tutte le possibili coppie di input/output. Dopo aver analizzato i nostri 7 esempi ci rimangono ancora 2^9 possibilità. Nel caso generale per input/output binari la formula é $|H| = 2^{\text{numero istanze}} = 2^n$ dove $|H|$ é la dimensione dello spazio delle ipotesi ed n é la dimensione dell'input.

Lavoreremo con uno spazio delle ipotesi ristretto: partiamo scegliendo uno spazio delle ipotesi H che é considerevolmente più piccolo dello spazio di tutte le possibili funzioni.

Vedremo:

- proposizioni formate da solo "and" (regole congiuntive) in uno spazio di ipotesi H finito e discreto
- funzioni lineari, in uno spazio di ipotesi H continuo e infinito

3.1 Regole Congiuntive

Quante h diverse possiamo avere? Cioè quante semplici regole congiuntive?

Nel caso generale:

- Letterali positivi: ad esempio $h_1 = l_2$, $h_2 = (l_1 \wedge l_2)$, $h_3 = \text{true}$ ecc... sono semplici regole congiuntive. Abbiamo ridotto lo spazio delle ipotesi così da ottenere $|H| = 2^n$ basti immaginare l_i come una stringa di bit di lunghezza n.
- Letterali: (di cui fa parte anche il $\text{not}(l_i)$) ottenendo uno spazio delle ipotesi di dimensione $3^n + 1$

Vedremo come organizzare e cercare in modo efficiente attraverso uno spazio di ipotesi, sfruttando algoritmi per un insieme di ipotesi molto limitato semplificandoci il lavoro usando dati non rumorosi.

3.2 Problema: Enjoy Sport

Concept: "giorni in cui piace fare sport acquatici"

Task: predire il valore di "Enjoy Sport" per un giorno arbitrario in base ai valori degli attributi.

Sky	Temp	Humid	Wind	Water	Forecast	Enjoy Sport?
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Una riga della tabella è una istanza.

3.3 Rappresentare le Ipotesi

L'ipotesi h è una congiunzione di vincoli sugli attributi. Ogni vincolo può essere:

- Uno specifico valore: ad esempio Water = Warm
- Un valore irrilevante: ad esempio Water = ?
- Nessun valore consentito (ipotesi nulla): ad esempio Water = 0

Esempio di ipotesi h :

Sky	Temp	Humid	Wind	Water	Forecast
Sunny	?	?	Strong	?	Same

corrispondente alla funzione booleana

$$Sky = Sunny \wedge Wind = Strong \wedge Forecast = Same$$

L'ipotesi più specifica (risponde sempre false) e più generale (risponde sempre true) sono rispettivamente

Sky	Temp	Humid	Wind	Water	Forecast
0	0	0	0	0	0 //specifico
?	?	?	?	?	? //generale

3.4 TASK del Concept Learning

Vengono dati:

- Le istanze X : i possibili giorni vengono descritti da attributi Sky, Temp, Humid, Wind, Water e Forecast.
- Funzione Target: $c: EnjoySport : X \rightarrow \{0, 1\}$.
- Spazio delle ipotesi (H): insieme finito di coniunzioni di letterali
- Esempi di allenamento (D): esempi positivi e negativi: $\langle x_1, c(x_1) \rangle, \dots, \langle x_n, c(x_n) \rangle$

Bisogna trovare:

- Una ipotesi h in H tale che $h(x)=c(x)$ per tutti gli x in X

L'apprendimento sta nella ricerca nello spazio delle ipotesi H .

3.5 Apprendimento Induttivo

Ipotesi Apprendimento Induttivo: "Qualsiasi ipotesi trovata per approssimare la funzione target sugli esempi di allenamento, approssimerà anche la funzione target sugli esempi non osservati". Ma purtroppo non è detto che il Training Set sia corretto (dati rumorosi, ...)

Quindi $h(x)=c(x)$ per ogni x in D (funzione consistente con il training set) ma $h(x)=c(x)$ per ogni x in X ? Cioè la nostra ipotesi classificherà in modo corretto tutte le istanze possibili? (Problema fondamentale del ML).

3.6 Qual è il numero di istanze, concetti e ipotesi di un problema?

- Sky: Sunny, Cloudy, Rainy
- Temp: Warm, Cold
- Humid: Normal, High
- Wind: Strong, Weak
- Water: Warm, Cold
- Forecast: Same, Change

La scelta della rappresentazione di H determina lo spazio di ricerca!

Numero di possibili istanze: $3*2*2*2*2*2 = 96$

Numero di concetti distinti: $2^{96} = 2^{\text{numero istanze}}$

Numero di ipotesi sintatticamente distinte: $5*4*4*4*4*4$ (poiché per ogni attributo devo aggiungere 0 o ?)

Numero di ipotesi semanticamente distinte: $1+4*3*3*3*3*3$ (poiché le ipotesi con almeno uno 0 sono equivalenti a false, i valori sono in and!)

Strutturare uno spazio di ricerca può aiutare a cercare in modo più efficiente.

3.7 Da ordine generale a ordine specifico

Consideriamo due ipotesi

1. $h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$
2. $h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ?, ? \rangle$

L'insieme di istanze coperte da h_1 e da h_2 sono differenti, infatti h_2 impone meno vincoli rispetto a h_1 e quindi classifica più istanze x positive ($h_2(x)=1$).

Siano h_j e h_k funzioni booleane definite su X . h_j è più generale o equivalente a h_k ($h_j \geq h_k$)
se solo se $\forall x \in X : [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$
In questo caso $h_2 \geq h_1$

Possiamo sfruttare questo ordine parziale per organizzare più efficientemente la nostra ricerca in H .

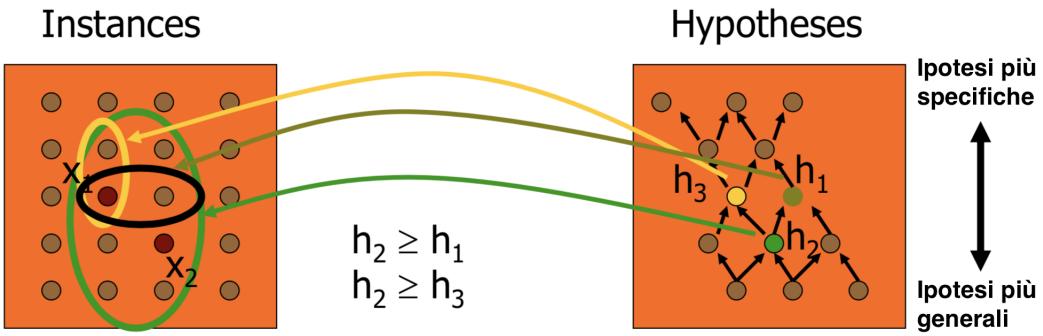


Figure 2: Partial Order

Istanze:

- $x_1 = < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Same} >$
- $x_2 = < \text{Sunny}, \text{Warm}, \text{High}, \text{Light}, \text{Warm}, \text{Same} >$

Ipotesi:

- $h_1 = < \text{Sunny}, ?, ?, \text{Strong}, ?, ? >$
- $h_2 = < \text{Sunny}, ?, ?, ?, ?, ? >$
- $h_3 = < \text{Sunny}, ?, ?, ?, \text{Cool}, ? >$

3.8 Algoritmo Find-S (Specific)

L'algoritmo Find-S sfrutta l'ordine parziale di gestione per cercare in modo efficiente una h consistente (senza elencare esplicitamente ogni h in H).

1. Inizializza h con l'ipotesi più specifica nello spazio delle ipotesi H ($h_0 = < 0, 0, 0, 0, 0, 0 >$)

2. Prendo il prossimo esempio di training:

- Per ogni attributo a_i dell'ipotesi h , se a_i è soddisfatto in h da x allora non fare nulla (quindi se l'esempio è negativo non ci sono cambiamenti sull'ipotesi), altrimenti se a_i NON è soddisfatto e l'esempio è positivo, sostituisci a_i con il prossimo vincolo più generale soddisfatto da x .
Ripeto il passaggio per ogni esempio positivo.

3. Viene dato in output l'ipotesi h

L'algoritmo parte dall'ipotesi più specifica, poi scende in maniera conservativa (generalizza il minimo possibile) in modo da rimanere consistente sui positivi, e di conseguenza contemporaneamente sui negativi. Trovo un'ipotesi che copre tutti i positivi e tutti i negativi. In questo modo evito di esplorare tutto H , scorrendo solo una volta il training set, perché si segue il Partial Order.

3.8.1 Esempio su Enjoy Sport

1. $h_0 = \langle 0, 0, 0, 0, 0, 0 \rangle$
2. $x_1 = \langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$ questo esempio è positivo² e h_0 non è soddisfatta perché False. Quindi devo prendere i vincoli più generali da x_1 e metterli in h , in questo caso sono le singole istanze poiché tutte più generali di 0.
3. $h_1 = \langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle$
4. $x_2 = \langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Warm}, \text{Same} \rangle$ esempio positivo, notiamo che High è in contrasto con Normal. Quindi cambio quell'attributo con uno più generale, in questo caso è "?".
5. $h_2 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$
6. $x_3 = \langle \text{Rainy}, \text{Cold}, \text{High}, \text{Strong}, \text{Warm}, \text{Change} \rangle$ esempio è negativo quindi non faccio nulla
7. $h_3 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} \rangle$
8. $x_4 = \langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Change} \rangle$ esempio positivo
9. $h_3 = \langle \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? \rangle$

3.8.2 Proprietà di Find-S

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 consistente con gli esempi negativi, a condizione che il concetto target sia contenuto in H . Sceglio l'ipotesi più specifica poiché nel caso in cui ci fossero diverse ipotesi consistenti con gli esempi di training, Find-S trova la più specifica.

3.8.3 Aspetti negativi di Find-S

Non so se il sistema di apprendimento converga con il concetto target, nel senso che non è in grado di determinare se ha trovato l'unica ipotesi coerente con gli esempi di allenamento. Inoltre non so quando i dati di allenamento sono incoerenti, in quanto ignora gli esempi di allenamento negativi → nessuna tolleranza al rumore sui dati!

3.9 Version Spaces

Find-S fornisce una singola ipotesi da H che è coerente con gli esempi di allenamento, questa è solo una delle tante ipotesi da H che potrebbero adattarsi ugualmente bene ai dati di allenamento. L'idea è quindi quella di dare in output l'insieme di tutte le possibili h consistenti con D .

Il Version Space $VS_{H,D}$ rispetto allo spazio delle ipotesi H e il training set D , è il sottoinsieme delle ipotesi prese da h coerenti con tutti gli esempi di training:

$$VS_{H,D} = \{h \in H \mid \text{Consistent}(h, D)\}$$

dove $\text{Consistent}(h, D) = \forall \langle x, c(x) \rangle \in D \mid h(x) = c(x)$

Si dice che un esempio x soddisfa l'ipotesi h quando $h(x)=1$, indipendentemente dal fatto che x sia un esempio positivo o negativo del concetto target. Tuttavia, se un tale esempio sia coerente con h dipende dal concetto target e, in particolare, se $h(x)=c(x)$.

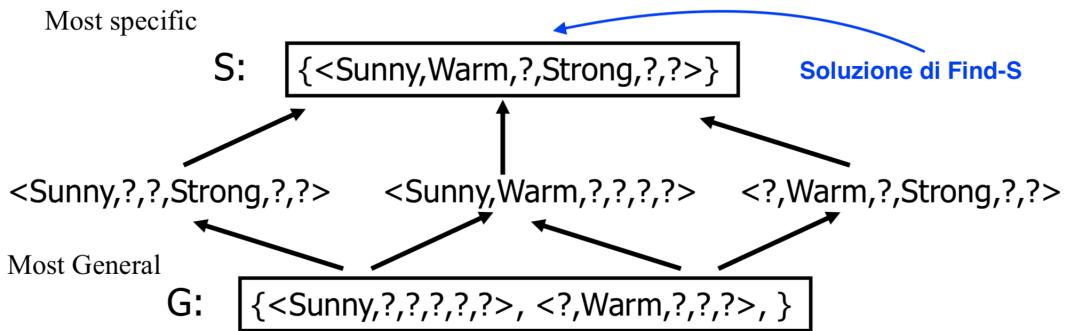
²l'esempio è preso dalla tabella a pagina 10

3.10 Algoritmo List-Then-Eliminate

L'algoritmo List-Then-Eliminate inizializza il Version Space con tutte le ipotesi contenute in H eliminando volta per volta tutte le ipotesi che non sono consistenti con gli esempi di training. Il Version Space si riduce quindi man mano che si osservano più esempi, fino a quando idealmente rimane solo un'ipotesi coerente con tutti gli esempi osservati, presumibilmente, questo è il concetto target desiderato. Se non sono disponibili dati sufficienti per restringere lo spazio l'algoritmo può generare l'intero insieme di ipotesi coerenti con i dati osservati. L'algoritmo List-Then-Eliminate può essere applicato ogni volta che lo spazio di ipotesi H è finito. Ha molti vantaggi, tra cui il fatto che è garantito il risultato di tutte le ipotesi coerenti con i dati di allenamento. Sfortunatamente, richiede un elenco esaustivo di tutte le ipotesi in H , un requisito non realistico per tutti gli spazi di ipotesi tranne quelli più banali.

1. Iniziamo impostando il nostro Version Space con una lista contenente ogni ipotesi in H
2. Per ogni esempio di allenamento $\langle x, c(x) \rangle$ rimuoviamo dal Version Space ogni ipotesi che è inconsistente con gli esempi di allenamento cioè $h(x) \neq c(x)$
3. Viene dato in output la lista delle ipotesi ora contenute nel Version Space

3.10.1 Esempio di Version Space



$$\begin{aligned}
 x_1 &= <\text{Sunny Warm Normal Strong Warm Same}> + \\
 x_2 &= <\text{Sunny Warm High Strong Warm Same}> + \\
 x_3 &= <\text{Rainy Cold High Strong Warm Change}> - \\
 x_4 &= <\text{Sunny Warm High Strong Cool Change}> +
 \end{aligned}$$

Il Version Space dell'algoritmo viene rappresentato solo dal suo membro più generale G e dal più specifico S (che è la soluzione di Find-S!).

Teorema: ogni membro del Version Space si trova tra:

$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq h \geq s)\}$$

dove $x \geq y$ significa x è più generale o uguale a y .

3.11 Algoritmo Candidate Elimination

L'algoritmo di Candidate Elimination calcola il Version Space contenente tutte le ipotesi di H che sono coerenti con una sequenza osservata di esempi di addestramento. L'algoritmo calcola il Version Space senza elencarne esplicitamente tutti i suoi membri, ciò si ottiene utilizzando l'ordinamento più generale che parziale e mantenendo una rappresentazione compatta dell'insieme di ipotesi coerenti.

Inizia inizializzando il Version Space sull'insieme di tutte le ipotesi in H , cioè inizializzando il set di limiti G per contenere l'ipotesi più generale in H $G_0 = <?, ?, ?, ?, ?, ? >$ e inizializzando il set di limiti S per contenere l'ipotesi più specifica $S_0 = <0, 0, 0, 0, 0, 0>$. Questi due insiemi di limiti delimitano l'intero spazio delle ipotesi, poiché ogni altra ipotesi in H è sia più generale di S_0 sia più specifica di G_0 . Dopo che tutti gli esempi sono stati elaborati, lo spazio versione calcolato contiene tutte le ipotesi coerenti con questi esempi.

1. Inizializza G con l'insieme di ipotesi massimamente generali in H $G_0 = <?, ?, ?, ?, ?, ? >$
2. Inizializza S con l'insieme di ipotesi massimamente specifiche in H $S_0 = <0, 0, 0, 0, 0, 0 >$
3. Per ogni esempio di training $d = <x, c(x)>$:
 - se d è positivo:
 - Rimuovere da G ogni ipotesi incompatibile con d
 - per ogni ipotesi s in S che non è coerente con d : (Generalizziamo S)
 - * Rimuovere s da S
 - * Aggiungi a S tutte le minime generalizzazioni h di s tali che h è consistente con d e alcuni membri di G sono più generali di h
 - * Rimuovi da S ogni ipotesi più generale di un'altra ipotesi in S
 - se d è negativo:
 - Rimuovere da S ogni ipotesi incompatibile con d
 - per ogni ipotesi g in G che non è coerente con d : (Specializziamo G)
 - * Rimuovere g da G
 - * Aggiungi a G tutte le minime specializzazioni h di g tali che h è consistente con d e alcuni membri di S sono più specifici di h
 - * Rimuovi da G ogni ipotesi meno generale di un'altra ipotesi in G

3.11.1 Esempio di Candidate Elimination

S: $\{\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$

G: $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle +$

S: $\{\langle \text{Sunny Warm Normal Strong Warm Same} \rangle\}$

G: $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

$x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle +$

S: $\{\langle \text{Sunny Warm ? Strong Warm Same} \rangle\}$

G: $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

$x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle -$

S: $\{\langle \text{Sunny Warm ? Strong Warm Same} \rangle\}$

G: $\{\langle \text{Sunny, ?, ?, ?, ?, ?} \rangle, \langle ?, \text{Warm, ?, ?, ?, ?} \rangle, \langle ?, ?, ?, ?, \text{Same} \rangle\}$

$x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle +$

S: $\{\langle \text{Sunny Warm ? Strong ? ?} \rangle\}$

G: $\{\langle \text{Sunny, ?, ?, ?, ?, ?} \rangle, \langle ?, \text{Warm, ?, ?, ?} \rangle\}$

esempio di allenamento negativo, ora specializzo G lascio il più generale dove non collide

G al passaggio 3 contiene le specializzazioni minime che mi permettono di coprire x_3 come negativa, senza rendere G più specializzato di S. S e G sono le "guardie" di tutti i positivi e di tutti i negativi trovati finora. Il fatto che se specializzo troppo G rendo false alcuni input già accettati, deriva dal partial order: intelligentemente il controllo di consistenza di tutti i vecchi input non è effettuato dall'algoritmo. Infatti, il limite S del Version Space costituisce un riepilogo degli esempi positivi riscontrati in precedenza che possono essere utilizzati per determinare se una ipotesi è coerente con questi esempi. Il limite G riassume le informazioni da esempi negativi riscontrati in precedenza. Ogni ipotesi più specifica di G è garantita per essere coerente con esempi negativi del passato.

4 Inductive Bias

Il nostro spazio delle ipotesi non è in grado di rappresentare target disgiuntivi come ad esempio $(Sky = Sunny) \vee (Sky = Cloudy)$.

Se avessimo due esempi del tipo:

$x_1 = < Sunny, Warm, Normal, Strong, Cool, Change >$ (positivo)

$x_2 = < Cloudy, Warm, Normal, Strong, Cool, Change >$ (positivo)

troveremo che $S = \{ < ?, Warm, Normal, Strong, Cool, Change > \}$

che è inconsistente con l'esempio $x_3 = < Rainy, Warm, Normal, Strong, Cool, Change >$ (negativo)

trovando $S = \{ \}$, cioè S collassa, perché la funzione target è fatta solo di and.

PROBLEMA: abbiamo vincolato il sistema di apprendimento considerando solo ipotesi congiuntive.

Abbiamo bisogno di uno spazio di ipotesi più espressivo.

4.1 Sistema di apprendimento Unbiased

La soluzione ovvia al problema di assicurare che il concetto target sia nello spazio di ipotesi H è di fornire uno spazio di ipotesi in grado di rappresentare ogni concetto insegnabile.

Significa che H è l'insieme di tutti i possibili sottoinsiemi di X (questo insieme viene chiamato anche l'insieme di potenza $P(X)$). In EnjoySport $|H| = 96$, $|P(X)| = 2^{96}$ concetti distinti. Un'ipotesi può essere rappresentata con disgiunzioni, congiunzioni e negazioni delle nostre precedenti ipotesi. H sicuramente contiene il concetto target. Con questa modifica quali sono i nostri G e S ?

NUOVO PROBLEMA: il nostro algoritmo di apprendimento dei concetti non è ora in grado di generalizzare oltre gli esempi osservati. Assumiamo tre esempi positivi (x_1, x_2, x_3) e due esempi negativi (x_4, x_5). $S = \{ (x_1 \vee x_2 \vee x_3) \}$ e $G = \{ \neg(x_4 \vee x_5) \} \rightarrow$ Non abbiamo generalizzazione, S sarà sempre la disgiunzione degli esempi positivi e G la disgiunzione di quelli negativi!

Proprietà: un sistema di apprendimento non vincolato non è in grado di generalizzare

Prova: ogni nuova istanza verrà classificata positivamente precisamente da metà delle ipotesi nel Version Space e negativa dell'altra metà. Poiché H è l'insieme di potenza di X e sia x_i una istanza non vista precedentemente, $\forall h$ consistente con $x_i(test)$, $\exists h'$ identico ad h con eccezione che $h'(x_i) \neq h(x_i)$ cioè $h \in VS \rightarrow h' \in VS$ (sono identici in D)

4.1.1 Il learning Unbiased è utile?

Un sistema di apprendimento che non fa assunzioni preliminari riguardo all'identità del concetto target non ha basi razionali per classificare eventuali istanze non conosciute. Un sistema di apprendimento dovrebbe essere in grado di generalizzare a partire dai dati di allenamento al fine di classificare istanze mai viste precedentemente. Il "bias" non è solo assunto per efficienza, ma è necessario per la generalizzazione, tuttavia non ci dice quale sia la migliore soluzione per la generalizzazione.

4.2 Definizione Formale di Inductive Bias

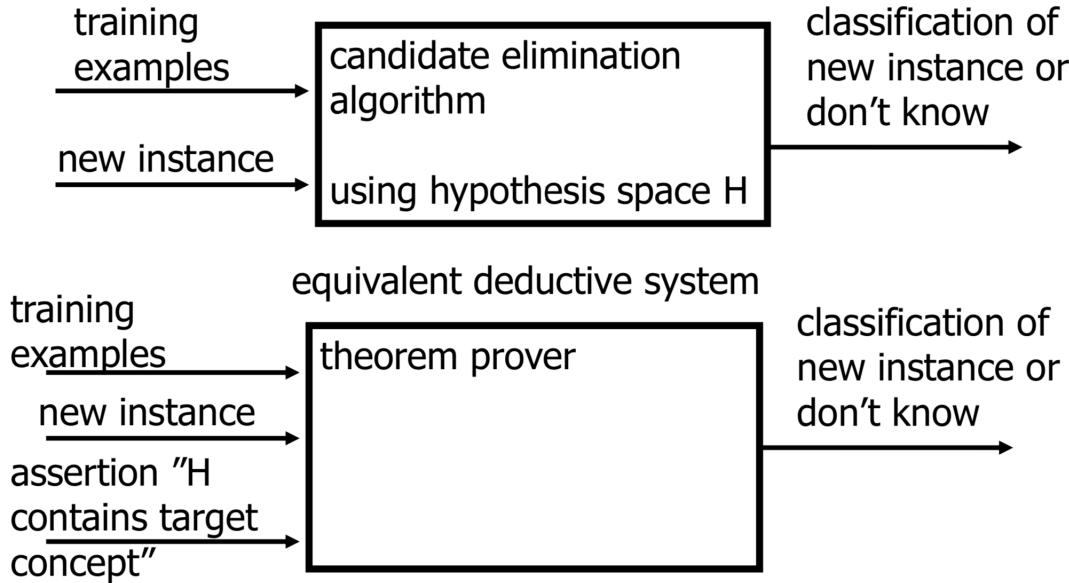
Consideriamo:

- Un algoritmo di apprendimento concettuale L
- Un insieme di istanze X e un concetto target c
- Sia $D_c = \{ < x, c(x) > \}$ un insieme di esempi di training di c
- Sia $L(x_i, D_c)$ la classificazione assegnata all'istanza x_i da L dopo l'allenamento su D_c .

L'Inductive Bias di L è un insieme minimo di asserzioni B tale che per qualsiasi concetto target c e corrispondenti dati di allenamento D_c :

$(\forall x_i \in X)[B \wedge D_c \wedge x_i] \vdash L(x_i, D_c)$ dove $A \vdash B$ significa che B è deducibile da A .

4.3 Sistemi Induttivi e Deduttivi equivalenti



Il comportamento input-output dell'algoritmo Candidate Elimination usando uno spazio delle ipotesi H è lo stesso di un sistema deduttivo in cui diamo come input l'asserzione " H contiene il concetto target". Questa asserzione è chiamata Bias Induttivo dell'algoritmo.

Il Bias è l'insieme delle asserzioni che ci permette di trasformare il problema da induttivo a deduttivo.

4.4 Tre (algoritmi) sistemi di apprendimento con Bias differenti

1. Rote Learner: (lookup table) L'apprendimento corrisponde semplicemente alla memorizzazione di ogni esempio di allenamento osservato nella memoria. Le istanze successive vengono classificate osservandole nella memoria. Se l'istanza viene trovata in memoria, viene restituita la classificazione memorizzata. In caso contrario, il sistema rifiuta di classificare la nuova istanza. In poche parole salva gli esempi, classifica x se e solo se si "accoppiano" con uno degli esempi visti precedentemente.
Non c'è Bias Induttivo → nessuna generalizzazione
2. Candidate Elimination: Le nuove istanze vengono classificate solo nel caso in cui tutti i membri del Version Space corrente concordino la classificazione. In caso contrario, il sistema rifiuta di classificare la nuova istanza.
Bias Induttivo → il concetto target può essere rappresentato nel suo spazio di ipotesi.
3. Find-S: Questo algoritmo, descritto in precedenza, trova l'ipotesi più specifica coerente con gli esempi di addestramento. Quindi utilizza questa ipotesi per classificare tutte le istanze successive.
Bias Induttivo → il concetto target può essere rappresentato nel suo spazio delle ipotesi e tutte le istanze sono istanze negative a meno che l'opposto non sia implicato da altre sue conoscenze.

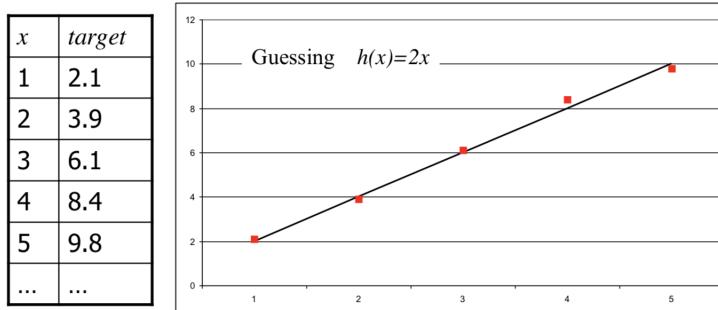
5 Modelli Lineari

Abbiamo visto che lo spazio delle ipotesi H costituisce l'insieme delle funzioni che possono essere realizzate dal sistema di apprendimento. Si assume che la funzione da apprendere f possa essere rappresentata da una ipotesi h in H (selezione di h attraverso i dati di apprendimento) o che almeno una ipotesi h in H sia simile a f (approssimazione). Abbiamo compreso che un Algoritmo di Ricerca nello Spazio delle Ipotesi è rappresentato in ML tramite un algoritmo di apprendimento (ad esempio adattamento dei parametri liberi del modello al task). NOTA: H non può coincidere con l'insieme di tutte le funzioni possibili ma la ricerca deve essere esaustiva → Bias Induttivo.

Sia la regressione che la classificazione (quest'ultima vista nel Concept Learning) appartengono alla categoria delle tecniche di Supervised Learning, ovvero un tipo di apprendimento che fornisce a priori una serie di dati e informazioni al sistema, prima che questo cominci ad apprendere. Andremo a vedere come i Modelli lineare possono essere usati sia per la regressione (Regressione lineare) sia per la classificazione. La regressione lineare si differenzia nettamente dalla classificazione, poiché la classificazione si limita a discriminare gli elementi in un determinato numero di classi, mentre nella regressione l'input è un dato e il sistema ci restituisce un output reale approssimando una funzione. La regressione è un processo statistico che cerca di stabilire una relazione tra due o più variabili. Fornendo a un modello di regressione un valore x , questo restituirà il corrispondente valore y generato dall'elaborazione di x .

5.1 Esempio di regressione

La regressione come abbiamo detto è un processo di stima di una funzione a valori reali sulla base di una serie finita di campioni "rumorosi" (conosciamo le coppie $(x, f(x)+\text{random noise})$).

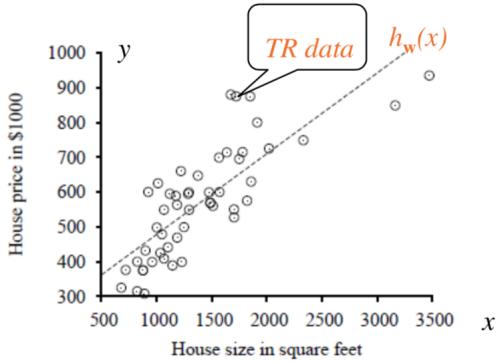


5.2 Modello di Regressione Lineare Semplice (Univariata)

Si parte con una variabile input x e una in output y , assumiamo il modello $h_w(x)$ espresso come $y = w_1x + w_0$ dove w_0 e w_1 sono parametri liberi a valori reali da apprendere. Usiamo la lettera w perché immaginiamo questi coefficienti come weights (pesi) poiché il valore y cambia in base a questi valori. Come si può intuire cerchiamo di adattarci ai dati con una linea retta.

Lavoriamo in uno spazio di ipotesi infinite (i valori w_i sono continui) ma abbiamo una buona soluzione grazie alla matematica classica. Gauss dimostrò che se i valori y hanno del "rumore" distribuito allora i valori w_1 e w_2 possono essere trovati minimizzando la somma dei quadrati degli errori (dopo vedremo meglio cosa significa).

Assumiamo che la variabile y sia (linearmente) correlata a un'altra variabile x o ad altre variabili per cui $y = w_1x + w_0 + \text{noise}$, dove i w_i sono i parametri liberi. Il "noise" è l'errore nella misurazione dei valori target con distribuzione normale. Cerchiamo di costruire un modello (trovare le variabili w_i) per predire/stimare la y per dei valori x non visti precedentemente.



Un esempio di regressione lineare per il problema sovrastante (bisogna associare il prezzo delle case in base alla metratura. y costo della casa, x metratura) è la funzione $h_w(x) = 0.232 * x + 246$ ma come arrivare a questa funzione?

5.3 Costruzione della funzione via LMS

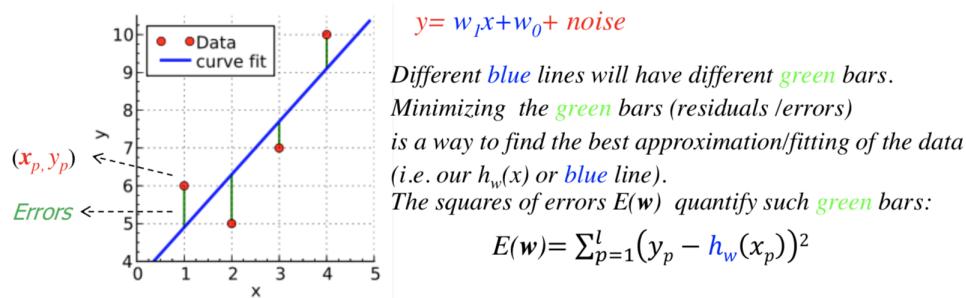
L'algoritmo LeastMeanSquare consiste nel trovare i w_i per cui l'errore è minimizzato (miglior adattamento dei dati sul training set con l esempi).

- Dato un insieme di l esempi di allenamento salvati come coppie (x_p, y_p)
- Trovare $h_w(x) = w_1x + w_0$ che minimizza l'errore medio sul Training Set
- Per calcolare la funzione di errore Loss usiamo la somma dei quadrati delle differenze tra il valore dato dall'esempio y_p e il valore calcolato dalla funzione $h_w(x)$. Il quadrato serve per avere solo valori positivi (si potrebbe anche usare il valore assoluto, ma è meglio di no, e il perché lo vedremo in seguito)

$$Loss(h_w) = E(w) = \sum_{p=1}^l (y_p - h_w(x_p))^2 = \sum_{p=1}^l (y_p - (w_1 * x_p + w_0))^2 \quad (1)$$

dove x_p è il p-esimo input e y_p è il p-esimo output dell'esempio p. Dividendo per l ottengo la media.

Quindi dobbiamo trovare l'argomento minimo w tale che l'errore è minimo in L2 (norma 2, le norme inducono una distanza): $w = argmin_w Loss(h_w) = argmin_w E(w)$ in L2. A livello grafico lo rappresentiamo così:



Il metodo dei minimi quadrati è un approccio standard alla soluzione approssimata di sistemi sovrardeterminati, ovvero insiemi di equazioni in cui vi sono più equazioni che incognite.

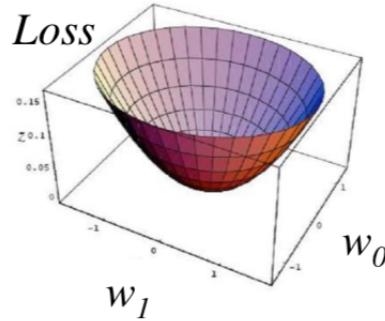
Il minimo locale é un punto stazionario dove il gradiente é nullo.

$$\frac{\partial E(w)}{\partial w_i} = 0 \quad \text{per } i = 0 \dots \text{dimensione_spazio} \quad (2)$$

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

$$\frac{\partial E(x)}{\partial w_0} = 0 \quad \frac{\partial E(x)}{\partial w_1} = 0 \quad (3)$$

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



per w_1 abbiamo che

$$w_1 = \frac{l * (\sum x_p y_p) - (\sum x_p)(\sum y_p)}{l * (\sum x_p^2) - (\sum x_p)^2} \quad (4)$$

mentre per w_0

$$w_0 = \frac{\sum y_p - w_1 * \sum x_p}{l} \quad (5)$$

Un approccio differente serve invece per quelle equazioni per cui il minimo della funzione di Loss spesso non ha una soluzione definita: infatti useremo una sorta di Hill Climbing iterativo che segue il gradiente discendente. Sfruttiamo le seguenti regole delle derivate parziali:

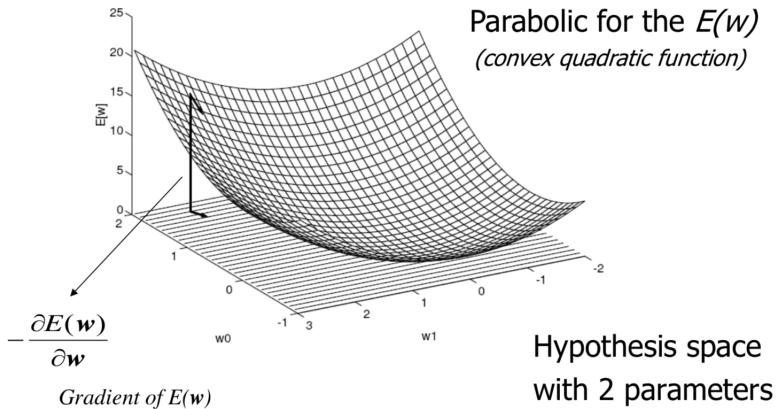
$$\frac{\partial}{\partial w} k = 0 \quad \frac{\partial}{\partial w} w = 1 \quad \frac{\partial}{\partial w^2} w = 2w \quad \frac{\partial(f(w))^2}{\partial w} = 2f(w) \frac{\partial f(w)}{\partial w} \quad (6)$$

ora cerchiamo il gradiente

$$\frac{\partial E(w)}{\partial w_i} = \frac{\partial}{\partial w_i} (y - h_w(x))^2 = 2(y - h_w(x)) \frac{\partial}{\partial w_i} (y - h_w(x)) = 2(y - h_w(x)) \frac{\partial}{\partial w_i} (y - (w_1 x + w_0)) \quad (7)$$

applicando w_0 e w_1 nella derivata ottengo due equazioni

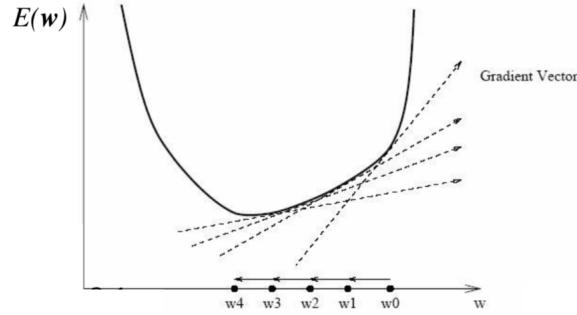
$$\frac{\partial E(w)}{\partial w_0} = -2(y - h_w(x)) \quad \text{e} \quad \frac{\partial E(w)}{\partial w_1} = -2(y - h_w(x)) * x \quad (8)$$



5.3.1 Gradiente discendente

La derivazione precedente suggerisce la costruzione di un algoritmo iterativo basato su $\partial E(w)/\partial w_i$. Il gradiente ci indica la direzione di salita, possiamo spostarci verso il minimo con discesa del gradiente $\Delta w = -\text{gradiente}E(w)$. 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.

$$w_{new} = w_{old} + \eta * \Delta w \text{ dove } \eta \text{ é una costante che indicare il rateo di apprendimento}$$



Le nostre regole di correzione dell'errore (chiamate delta-rule) cambiano i valori w proporzionalmente all'errore:

$$\Delta w_0 = -\frac{\partial E(w)}{\partial w_0} = 2(y - h_w(x)) \quad e \quad \Delta w_1 = -\frac{\partial E(w)}{\partial w_1} = 2(y - h_w(x)) * x \quad (9)$$

- se $(y_{target} - output) = 0$ significa che non abbiamo errore e quindi nessuna correzione da fare
- se $(output > y_{target})$ cioè $(y - h) < 0$ vuol dire che l'output trovato é troppo alto quindi:
 - Δw_0 negativo → riduco w_0
 - se $(x_{input} > 0)$ e Δw_1 negativo → riduco w_1 altrimenti incremento w_1
- se $(output < y_{target})$ cioè $(y - h) > 0$ vuol dire che l'output trovato é troppo basso quindi:
 - Δw_0 negativo → aumento w_0
 - se $(x_{input} < 0)$ e Δw_1 positivo → aumento w_1 altrimenti diminuisco w_1

Questo ci permette la ricerca in uno spazio delle ipotesi infinito, e può essere applicato facilmente per spazi H continui e con funzione di Loss differenziabile (altrimenti non posso calcolare il gradiente!). Per questo motivo il valore assoluto è scomodo: potrebbe rendere non differenziabile la funzione. Arriviamo quindi a definire le funzioni su l esempi:

$$\Delta w_0 = -\frac{\partial E(w)}{\partial w_0} = 2 \sum_{p=0}^l (y_p - h_w(x_p)) \quad e \quad \Delta w_1 = -\frac{\partial E(w)}{\partial w_1} = 2 \sum_{p=0}^l (y_p - h_w(x_p)) * x_p \quad (10)$$

Ora può venirci in mente una domanda, aggiorniamo i w_i dopo aver ispezionato un insieme di esempi di allenamento l , oppure dopo ogni singolo esempio analizzato?

- Batch Algorithm: 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 . Può essere molto lento...
- On-Line Algorithm: calcoliamo il gradiente su un pattern p e poi aggiorniamo subito w_i . Solitamente è più veloce del Batch...

5.4 Modello di Regressione Lineare Multivariato

Il caso standard è con due variabili, ma in realtà possiamo avere in input centinaia di variabili, l'esempio sul prezzo delle case in realtà conteneva anche le variabili che indicavano il numero delle stanze, l'età della casa ecc...

5.4.1 Notazione dei dati

Pattern	x_1	x_2	x_j	x_n
Pat 1	$x_{1,1}$	$x_{1,2}$		$x_{1,n}$
...				
Pat p	$x_{p,1}$	$x_{p,2}$	$x_{p,j}$	$x_{p,n}$
...				

X è una matrice l righe * n colonne (numero di pattern * numero di variabili), ogni riga della matrice X_i è un vettore che rappresenta un esempio. $x_{i,j}$ è uno scalare cioè la componente j della riga i . Stiamo quindi dicendo che ogni esempio x_p è un vettore di n elementi.

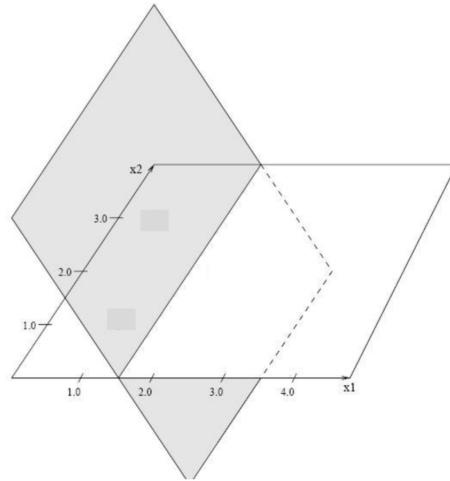
Supponendo vettori colonna x e w , con numero di esempi l e dimensione in input n , possiamo scrivere

$$w^T x + w_0 = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n \quad (11)$$

dove w_0 è il bias (non c'entra con il Concept Learning) è la distanza dalla componente 0. Possiamo anche scrivere che $x_0 = 1$ in modo tale da poter scrivere che $w^T x = x^T w$ quindi: $x^T = [1, x_1, \dots, x_n]$ e $w^T = [w_0, w_1, \dots, w_n]$

5.5 Hyperplane

Anziché adattare una retta, adattiamo un iperpiano.

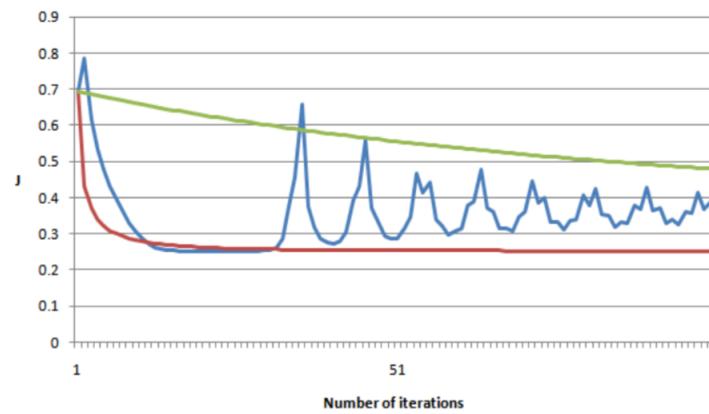


quindi abbiamo che $w^T x = w_1 x_1 + w_2 x_2 + w_0$ e $h(x_p) = x_p^T w = \sum_{i=0}^n x_{p,i} w_i$ dove $x_{p,i}$ è la componente i-esima del p-esimo esempio.

5.6 Gradiente Discendente Algoritmo

1. si inizia con il vettore dei pesi $w_{iniziale}$ e si fissa un $0 < \eta < 1$
2. si calcola il gradiente $\Delta w = -\text{gradiente}E(w) = -(\partial E(w)/\partial w)$
3. si calcola $w_{new} = w_{old} + \eta * \Delta w$
4. ripetere dal passo 2 finché $E(w)$ è sufficientemente piccolo

La versione batch dell'algoritmo si occupa di calcolare Δw dopo un insieme di esempi l. La versione online invece aggiorna Δw dopo ogni esempio p al posto di aspettare la sommatoria su l, ma ha bisogno di uno step size η più piccolo.



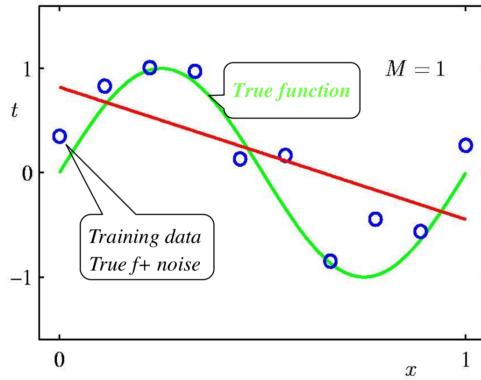
Questi sono tre esempi di curve, dove noi andremo a preferire quella che diminuisce nel minor tempo possibile (quella che impiega meno iterazioni).

5.7 Vantaggi dei modelli lineari

Se funziona bene è un modello "meraviglioso", molto semplice dove tutte le informazioni sui dati sono contenute nel vettore w . Facile da interpretare tant'è che viene usato in medicina, biologia, chimica, economia... Una delle cose fondamentali è che sono ammessi dati rumorosi e viene usato/incluso in modelli più complessi.

5.8 Limitazioni

La regressione lineare presenta delle limitazioni quando si parla di problemi non lineari.



Notiamo che $h_w(X) = w_1x + w_0$ definita "lineare" non significa che è una linea retta, ma piuttosto come il modo in cui i coefficienti w occorrono nell'equazione di regressione (cioè lineare in w , non in x). Quindi, possiamo anche trasformare gli input, come x, x^2, x^3, \dots con relazioni non lineari tra input e output. Sfruttando l'algoritmo LMS utilizzato fino ad ora:

$$y(x, w) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j \quad (12)$$

5.8.1 Esempio

Rispetto all'equazione scritta precedentemente in questo caso abbiamo $M=2$. La funzione non deve essere lineare nell'argomento (variabili in input) ma solo nei parametri che sono determinati per dare il miglior adattamento.

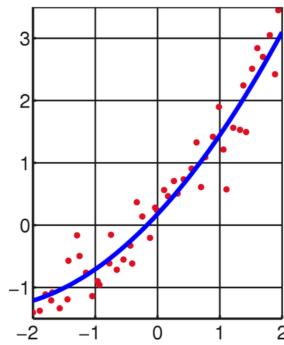


Figure 3: fitting su una funzione quadratica (linea blu) attraverso un set di dati (punti rossi)

5.9 Linear Basis Expansion

Prima calcolavamo il prodotto scalare tra w e x (w trasposto per x) ora prendo delle trasformazioni (anche non lineari) del vettore x e le uso come un'espansione della base, combinandole in maniera lineare in accordo con la funzione ϕ_k .

$$h_w(x) = \sum_{k=0}^K w_k \phi_k(x) \quad (13)$$

La funzione ϕ può essere espressa come rappresentazioni polinomiali del tipo $\phi(x) = x_j^2$ oppure $\phi(x) = x_j * x_i$, o ancora trasformazioni non lineari come $\phi(x) = \log(x_j)$. Il modello è rimasto lineare nei parametri (è sempre x !) quindi possiamo usare lo stesso algoritmo di prima.

5.9.1 Esempi

- dimensione di $x = 1$ e $\phi_j(x) = x^j$

$$h(x) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j \quad (14)$$

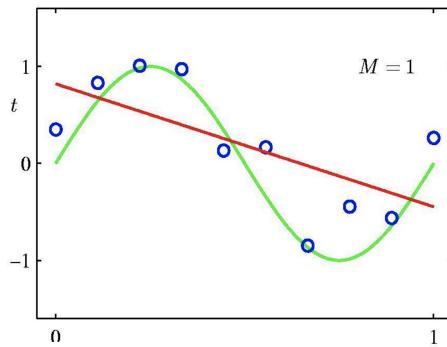
regressione polinomiale ad 1 dimensione ($K=M$)

- $\phi(x) = \phi([x_1, x_2])$

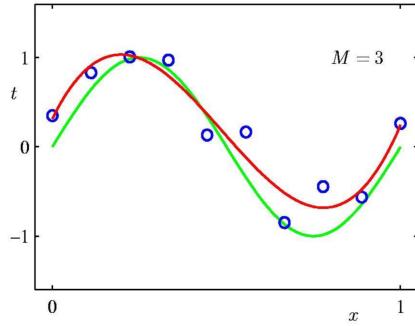
$$h(x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1 x_1 + w_5 x_2 x_2 \quad (15)$$

Quale ϕ scegliere? Adottiamo un modello in stile dizionario? I punti a favore sono la maggiore espressività poiché può modellare relazioni più complesse di quelle lineari. I punti contro sono l'avere una grossa base di funzioni che richiedono metodi per controllare la complessità del modello, vediamo perché...

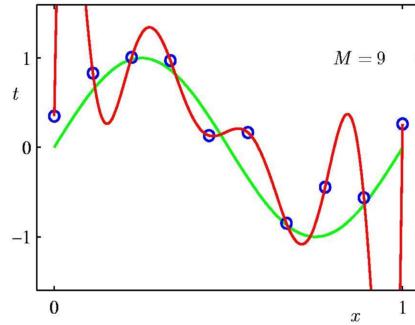
- grado del polinomio = 1, notiamo che è esegue troppo "underfitting".



- ordine del polinomio = 3, va bene ma potrebbe servirci più flessibilità.



- grado del polinomio = 9, notiamo che è flessibilissimo ma eccessivo. Cioè $E(w)=0$ ma dobbiamo tenere conto degli errori sul test set, perché se i dati fossero rumorosi noi stiamo adattando al rumore.



5.9.2 Tradeoff per la complessità

- Se scegliamo un modello troppo semplice non fitta bene i dati e abbiamo una soluzione vincolata (ce ne accorgiamo perché $E(w)$ è alto) → Underfitting
- Se scegliamo un modello troppo complesso siamo troppo sensibili alla piccole perturbazioni dei dati → Overfitting

Vogliamo scegliere la regolarizzazione per bilanciare bias e varianza e lo faremo attraverso il controllo della complessità del modello.

5.10 Regolarizzazione - Ridge Regression

Possiamo evitare l'overfitting penalizzando le funzioni complesse mantenendo comunque la flessibilità dello spazio delle ipotesi (un po' come il rasoio di Occam, si preferisce lo spazio delle ipotesi più semplice che fitta i dati).

La Ridge Regression, chiamata anche regolarizzazione di Tikhonov, è sempre un modello LMS ma regolarizzato. Per esempio è possibile aggiungere vincoli alla sommatoria del valore di $|w_j|$ favorendo modelli "sparsi" (sono modelli con meno termini dovuti a pesi $w_j = 0$).

Un modello non regolarizzato fitta bene ma generalizza male (non ci serve a niente). Vorrei un modello semplice (magari con alcuni coefficienti che valgono zero). Come si fa? Aggiungiamo una penalizzazione alla complessità del modello della loss, sommando la norma del vettore w , moltiplicato per un parametro λ chiamato coefficiente di regolarizzazione. Con il parametro lambda (che è una costante), non ho più bisogno di modificare il grado dei polinomi, o analizzare funzioni più complesse.

$$\text{definito } E_D(w) + \lambda E_W(w) \text{ otteniamo } \text{Loss}(h_w) = \sum_p (y_p - h_w(x_p))^2 + \lambda \|w\|^2 \quad (16)$$

Ottenendo quindi la nuova regola di apprendimento caratterizzata dal weight decay (aggiunta di $2\lambda w$)

$$w_{new} = w_{old} + \eta * \Delta w - 2\lambda w_{old} \quad (17)$$

Per via di $-2\lambda w_{old}$ decremento il parametro se positivo, lo incremento se negativo (in pratica lo avvicino a zero → weight decay). Quindi possiamo controllare la complessità del polinomio sfruttando solamente λ .

Finora abbiamo posto tutta la complessità nel bias di linguaggio, cioè un vincolo posto sul modello (lineare, quadratico, etc). Con la regolarizzazione sposto tutto sul bias di ricerca, cioè sull'algoritmo, come l'algoritmo riesce a trovare la soluzione. Noi non poniamo a priori una complessità al modello, ma sarà il mio metodo di ricerca intelligente a scegliere la funzione più semplice.

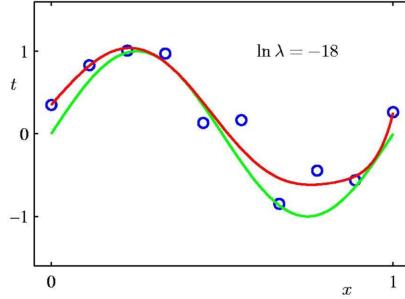


Figure 4: Polinomio di grado 9 con $\log_e(\lambda) = -18$ quindi λ basso

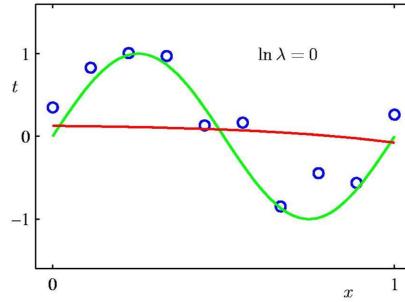


Figure 5: Polinomio di grado 9 con $\log_e(\lambda) = 0$ quindi λ troppo alto

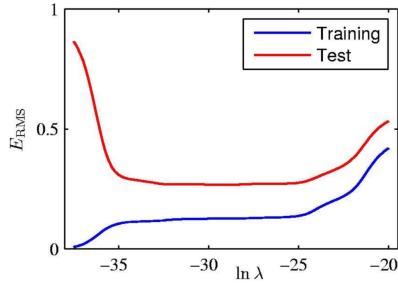


Figure 6: E_{RMS} vs $\log_e(\lambda)$

Possiamo concludere quindi dicendo che se λ è basso \rightarrow overfitting, mentre se λ alto \rightarrow underfitting.

5.11 Limitazione delle funzioni a base fissa

Quando facciamo un'espansione polinomiale, la dimensionalità del problema cresce molto, e i dati che prima erano densi, potrebbero diventare sparsi. Quando la dimensione dell'input aumenta, il problema diventa enormemente difficile (la difficoltà cresce esponenzialmente, proprio come il volume con le dimensioni). Inoltre, se inizio a mettere dati a caso, rischio di aumentare il rumore. Avere la funzione base che opera su ogni dimensione di uno spazio implica che lo spazio di input richieda un numero combinatorio di funzioni. Per esempio un polinomio dell'ordine 3, utilizza tutte le combinazioni di variabili di input dovute ai prodotti $x_1x_2, x_2x_3, \dots, x_1x_2x_3$ ecc... $\rightarrow D^3$ (approssimare all'aumentare di D)

In altri modelli, vedremo come possiamo farcela con meno funzioni di base. Si scelgono le funzioni dal dizionario e si mettono dei parametri liberi, per adattare la ϕ ad H. Ho il vantaggio di essere più generale, ma il modello diventa non lineare rispetto a w (vedremo nelle reti neurali).

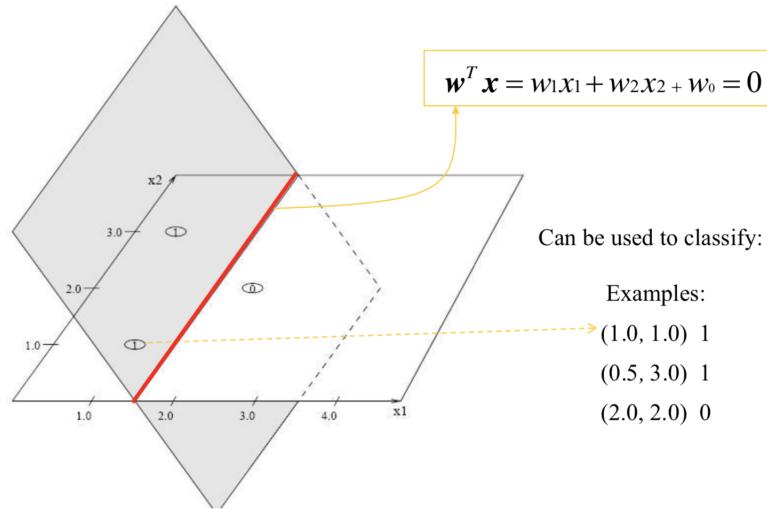
In altri modelli invece di calcolare esplicitamente la ϕ , viene fatta in maniera implicita, col concetto di funzioni kernel, controllando la complessità del modello direttamente con la funzione obiettivo, non più con una penalità aggiunta dopo (SVM).

5.12 Classificazione con il modello lineare

Possiamo utilizzare il modello lineare per la classificazione, in questo caso usiamo un iperpiano ($w^T * x$) assumendo valori positivi o negativi. Sfruttiamo questi modelli per decidere se un punto x appartiene alla zona positiva o negativa dell'iperpiano. Quindi vogliamo impostare w vettore in modo tale da ottenere una buona precisione di classificazione.

5.12.1 Visione geometrica dell'iperpiano

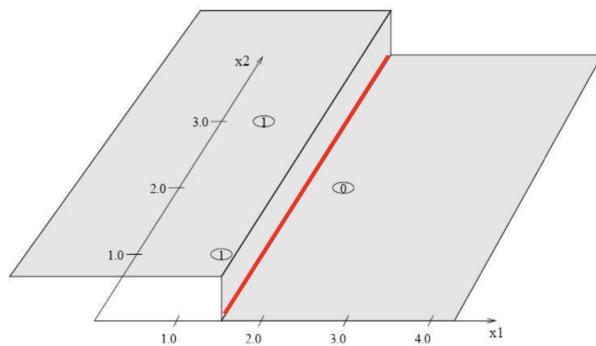
$w^T * x$ (prodotto di vettore riga per vettore colonna) definisce un iperpiano, il Decision Boundary è il luogo geometrico³ dei punti dove l'iperpiano vale zero e può essere utilizzato per classificare.



Quindi possiamo definire una funzione di classificazione del tipo:

$$h(x) = \begin{cases} 1 & \text{if } w^T x + w_0 \geq 0. \\ 0 & \text{altrimenti.} \end{cases} \quad (18)$$

che possiamo scrivere come $h(x) = \text{segno}(w^T x + w_0) \rightarrow h(x_p) = \text{sign}(x_p^T w) = \text{sign}(\sum_{i=0}^n x_{p,i} w_i)$

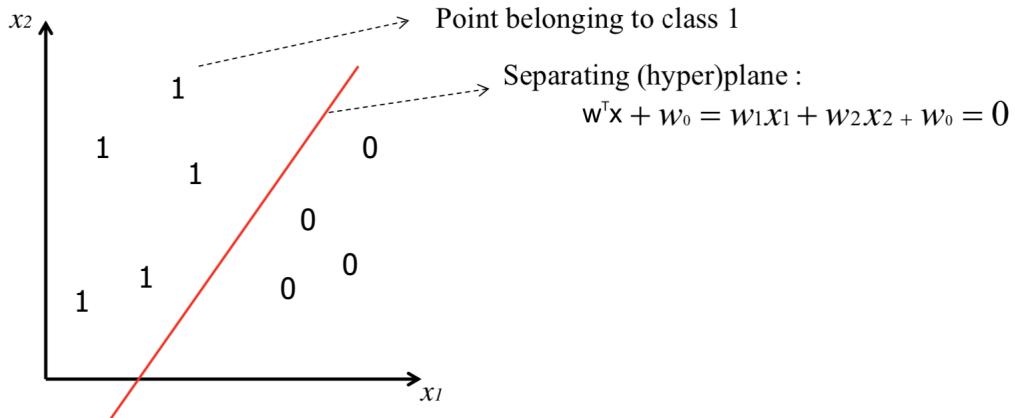


³è una linea, o un piano (quando ci troviamo in dimensioni più gradi)

5.12.2 Classificazione attraverso Decision Boundary lineare

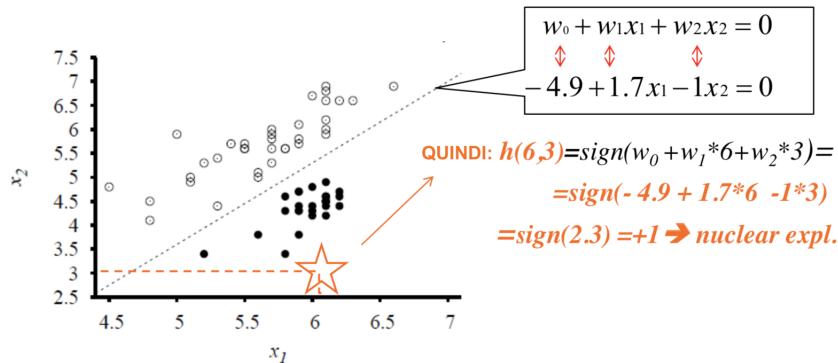
La classificazione può essere vista come l'allocazione dello spazio in input in diverse regioni di decisione. Ad esempio in uno spazio di due dimensioni $x = (x_1, x_2)$ in R^2 e con $f(x) = 0 \vee 1$ il decision boundary sarà una retta.

Linear Threshold Unit (LTU) = indicatore di funzioni, quanti ne abbiamo? un insieme di divisioni indotte da iperpiani, sono tutti i modi che abbiamo per muovere la linea rossa.



5.12.3 Esempio Terremoti/Esplosioni Nucleari

Trovare h in modo tale che il dato (x_1, x_2) ritorni $(0 \vee -1)$ per i terremoti e (1) per esplosioni nucleari. I dati sismici sono rappresentati da x_1 che è la magnitudine dell'onda del corpo e x_2 che è la magnitudine dell'onda della superficie. L'algoritmo trova il decision boundary.



5.12.4 Esempio Email Spam

Trovare $h(\text{mail})$ in modo tale che restituisca $+1$ se è spam e -1 se non è una mail spam.

Ad esempio $\phi(\text{mail}) = \text{txtmail.contains("denaro gratis")}$, w vettore di pesi per le funzioni di input che aiutano alla previsione, cioè peso positivo per "denaro gratis", negativo altrimenti. $w^T x$ è la combinazione dei pesi. $h_w(x)$ provvede a dare il limite per decidere se una mail è spam o non è spam.

$$h_w(x) = \text{segno}(\sum_k w_k \phi_k(x)) \quad (19)$$

5.13 Il problema di apprendimento (per classificatori lineari)

Assumendo che usiamo ancora il metodo dei minimi quadrati, dato un insieme grande l di esempi di allenamento, bisogna trovare w che minimizza la somma residua dei quadrati

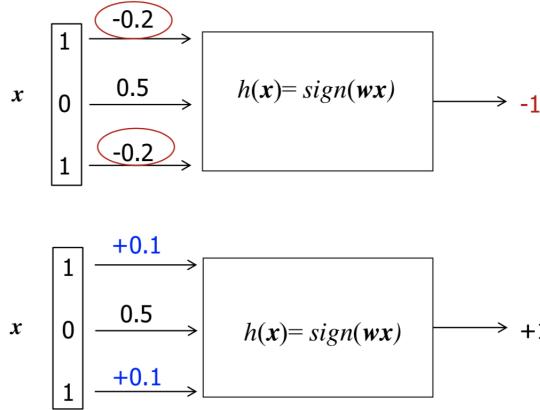
$$E(w) = \sum_{p=0}^l (y_p - x_p^T w)^2 = \|y - x^T w\|^2 \quad (20)$$

se $y_p = 1$ allora $x_p^T w \rightarrow 1$ mentre se $y_p = 0 \vee -1$ allora $x_p^T w \rightarrow 0 \vee -1$. Possiamo utilizzare l'algoritmo di discesa del gradiente:

$$\Delta w_i = -\frac{\partial E(w)}{\partial w_i} = \sum_{p=1}^l (y_p - (x_p^T w)) * x_{p,i} \quad (21)$$

Nella derivata ignoriamo il 2 che viene fuori dal quadrato. È una costante che non incide sull'eta (il coefficiente di apprendimento).

5.13.1 Δw come regola della correzione dell'errore



Se un input viene classificato erroneamente dalla funzione $h(x) = \text{segno}(w^T x)$, quindi ad esempio il Δ negativo per w_1 e w_3 , bisogna diminuire⁴ w_1 e w_3 proporzionalmente (con il coefficiente eta). Eseguiamo una correzione d'errore alzando i valori di Δw dove sono troppo bassi, e viceversa.

5.14 Classificazione dei pattern

Il modello viene *allenato* su un insieme di esempi con il calcolo LMS su $w^T x$ dall'algoritmo di discesa del gradiente, usato per la regressione lineare. Il modello viene *utilizzato* per la classificazione applicando la funzione di soglia $h(x) = \text{segno}(w^T x)$. L'errore può essere calcolato come errore di classificazione o numero di pattern classificati erroneamente.

$$L(h(x_p), d_p) = \begin{cases} 0 & \text{se } h(x_p) = d_p \\ 1 & \text{altrimenti.} \end{cases} \quad \text{mean_err} = \frac{1}{l} \sum_{i=1}^l L(h(x_i), d_i) \quad (22)$$

dove d_p è l'output dato dagli esempi di allenamento, quindi stiamo classificando i valori calcolati dalla nostra h con i valori degli esempi etichettati.

⁴notare il meno nella formula di Δw_i quando si esegue $y_p - (x_p^T w)$

Accuratezza = media dei pattern correttamente classificati = $(l - numerr)/l$
dove $numerr = \sum_{i=1}^l L(h(x_i), d_i)$

L'algoritmo è lo stesso come per la regressione → Algoritmo gradiente descendente (anche l'espansione lineare e Tikhonov possono essere applicati). Notare che l'algoritmo converge asintoticamente al MinLeastSquare, non necessariamente al numero minimo di errori di classificazione.

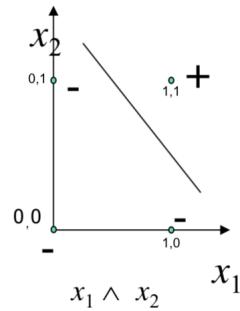
5.15 Congiunzioni nel caso di Modello Lineare

Possiamo rappresentare congiunzioni con i modelli lineari, ad esempio:

$x_1 \wedge x_2 \wedge x_4 \Leftrightarrow y$ cioè $1x_1 + 1x_2 + 0x_3 + 1x_4 > 2$ dove $w_1 = 1, w_2 = 1, w_3 = 0, w_4 = 1, w_0 = 2$ (vettore dei pesi)

Altro esempio con grafico:

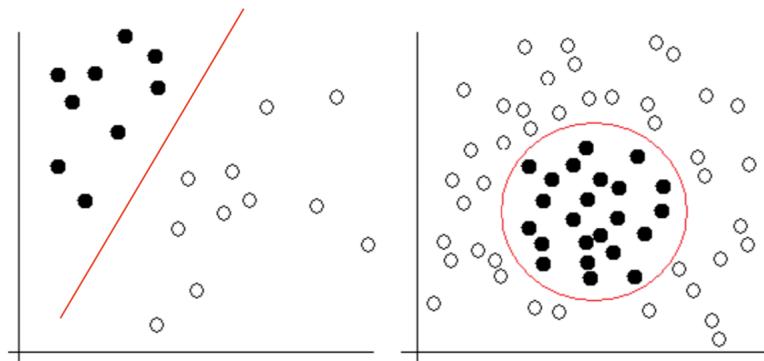
$x_1 \wedge x_2$ cioè $1x_1 + 1x_2 > 1$



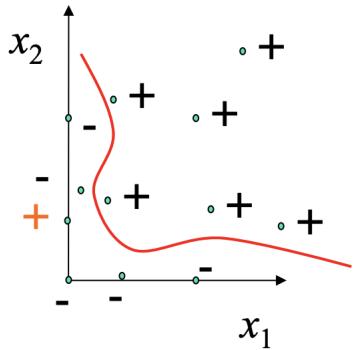
In generale w può essere appreso per trovare la soluzione.

5.15.1 Limitazioni

Nella geometria due insiemi di punti in un grafico a due dimensioni sono linearmente separabili quando possono esser separati da una singola linea.

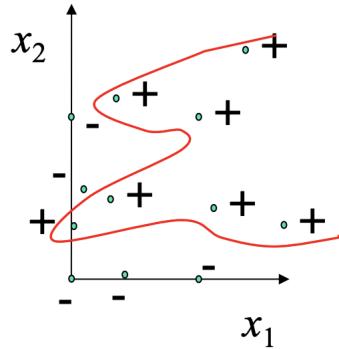


In generale due gruppi di punti sono linearmente separabili in uno spazio di dimensione n se possono essere separati da un iperpiano di dimensione $n - 1$. Il confine lineare dà la soluzione esatta solo per insiemi di punti linearmente separabili.



Non linear decision boundary
(by a basis expansion)

In this example 1 error is admitted
(orange)



A **not-smooth** classifier
that may need
regularization

5.16 Altri sistemi di apprendimento per la classificazione

- Linear Discriminant Analysis
- Logistic Regression: Stima la/le proprietà di x dato y , $P(y/x)$ partendo modellando la densità della classe come densità nota. Abbiamo una soglia soft (continua, differenziabile) con funzione logica (anziché soglia hard 0/1)
- Reti Neurali e SVM: modelli flessibili che includono approssimazione non lineare sia per la regressione che per la classificazione.

5.17 Conclusioni sui Modelli Lineari

I modelli lineari sono un approccio fondato di base per la regressione e la classificazione, è un modo molto compatto per rappresentare la conoscenza (tutti i dati sono nel vettore w) ma con una forte assunzione sulla relazione tra i dati (Assumiamo che ci sia una relazione lineare tra i dati, o comunque in accordo alla ϕ che abbiamo scelto. È una assunzione molto forte sullo spazio delle ipotesi). Abbiamo visto un algoritmo di correzione iterativa dell'errore (LMS) che ricerca spazi di ipotesi continui, una visione dei limiti ad approcci lineari e delle esigenze per modelli di ML più flessibili e anche i loro problemi: un'estensione del modello lineare per attività non lineari e un'introduzione al controllo della complessità (regolarizzazione).

6 Alberi di decisione

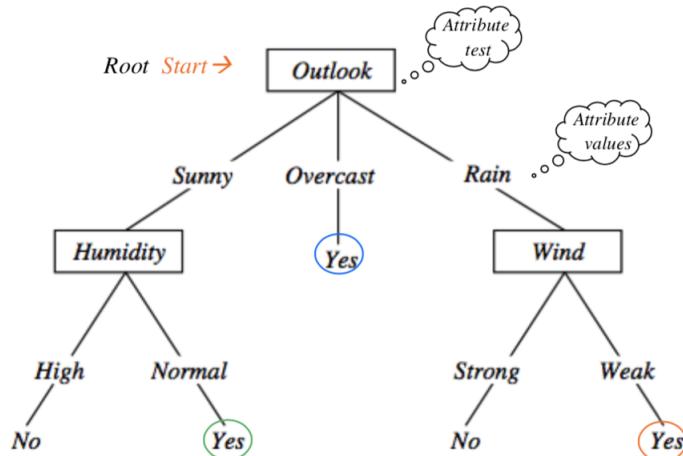
Un albero di decisione rappresenta una funzione che prende in input un vettore di attributi e restituisce una "decisione" (un singolo valore in output). I valori sia in input che in output possono essere valori continui, ma per adesso vedremo solo su valori discreti. Gli alberi di decisione aiutano a superare la restrizione della descrizione delle ipotesi solo tramite regole congiuntive (and). Ogni nodo dell'albero rappresenta un test su un attributo e ogni ramo corrisponde alla scelta del valore per quell'attributo, mentre le foglie corrispondono ad una classificazione.

6.1 Problema del PlayTennis

Vogliamo capire in quali giorni al nostro amico piace andare a giocare a Tennis. Il nostro Training Set è il seguente:

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

Il nostro albero di decisione ha invece questa forma:



L'albero rappresenta una disgiunzione di congiunzioni dei vincoli sui valori degli attributi, dove le foglie rappresentano la classificazione ottenuta. In particolare il nostro albero rappresenta questa formula:

$$\begin{aligned} & (\text{Outlook}=\text{Sunny} \wedge \text{Humidity} = \text{Normal}) \vee \\ & (\text{Outlook}=\text{Overcast}) \vee \\ & (\text{Outlook}=\text{Rain} \wedge \text{Wind}=\text{Weak}) \end{aligned}$$

Possiamo, in altro modo, definire l'albero come un insieme di regole if-then-else.

6.2 (Alg. ID3) Induzione Top-Down sugli alberi di decisione

L'algoritmo ID3 è un algoritmo base per l'apprendimento che sfrutta gli alberi di decisione. Dato un insieme di esempi, l'algoritmo per la costruzione dell'albero di decisione ha un approccio Top-Down eseguendo una ricerca Greedy senza backtracking. La domanda che ci dobbiamo porre quando stiamo creando l'albero è: "qual è il prossimo attributo da testare?" che tradotto in termini di alberi di decisione è scegliere il prossimo nodo che ci restituisce più Information Gain (definito successivamente). Viene quindi creato un nodo discendente per ogni possibile valore dell'attributo e gli esempi vengono partizionati in base a quel valore. Il processo viene ripetuto per ciascun nodo successore fino a quando tutti gli esempi sono stati classificati correttamente o non sono rimasti attributi.

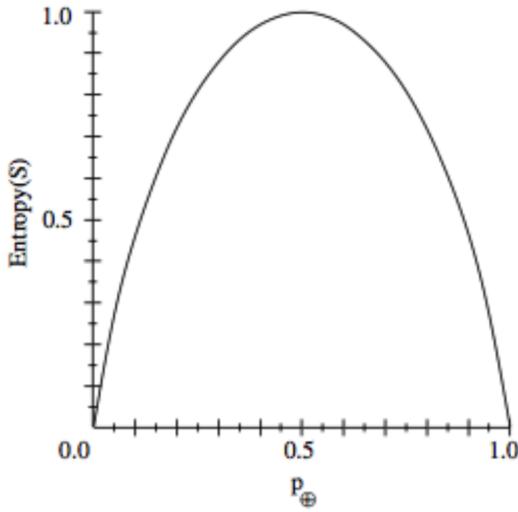
```
ID3(X, T, Attrs)      X: training examples:  
                        T: target attribute  
                        Attrs: other attributes, initially all attributes  
  
Create Root node  
If all X's are +, return Root with class +  
If all X's are -, return Root with class -  
If Attrs is empty 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  $A = v_i$   
        -  $X_i \leftarrow$  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  $X_i$   
        - else add the subtree generated by ID3( $X_i$ , T, Attrs - {A})  
return Root
```

6.3 Entropia: Come scegliere il miglior Attributo

La scelta principale in ID3 riguarda quale attributo testare ad ogni nodo. Per definire l'Information Gain dobbiamo usare la nozione di entropia, la quale misura l'impurità di una collezione di esempi.

- S è una collezione di esempi
- p_+ è una porzione di esempi positivi in S
- p_- è una porzione di esempi negativi in S

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$



Se gli esempi sono tutti positivi o tutti negativi (quindi appartengono tutti alla stessa classe) otteniamo una entropia che vale 0. Otteniamo una entropia massima se gli esempi positivi e negativi sono in ugual dimensione ($\frac{1}{2}+, \frac{1}{2}-$).

Prendiamo come esempio $S=14$ attributi:

$$\text{Entropy}([14+, 0-]) = -\frac{14}{14}\log_2(\frac{14}{14}) - 0\log_2 0 = 0$$

$$\text{Entropy}([7+, 7-]) = -\frac{7}{14}\log_2(\frac{7}{14}) - \frac{7}{14}\log_2(\frac{7}{14}) = 1$$

$$\text{Entropy}([9+, 5-]) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.94$$

Sia l'entropia che la variabile p hanno valori tra lo 0 e 1 compresi. Una interpretazione dell'entropia è che essa specifica il numero minimo di bit necessari per rappresentare la classificazione di un membro random di S , per questo si utilizza il logaritmo in base 2.

6.4 Information Gain

L'Information Gain misura la riduzione aspettata dell'entropia causata dal partizionamento degli esempi rispetto all'attributo. Cioè:

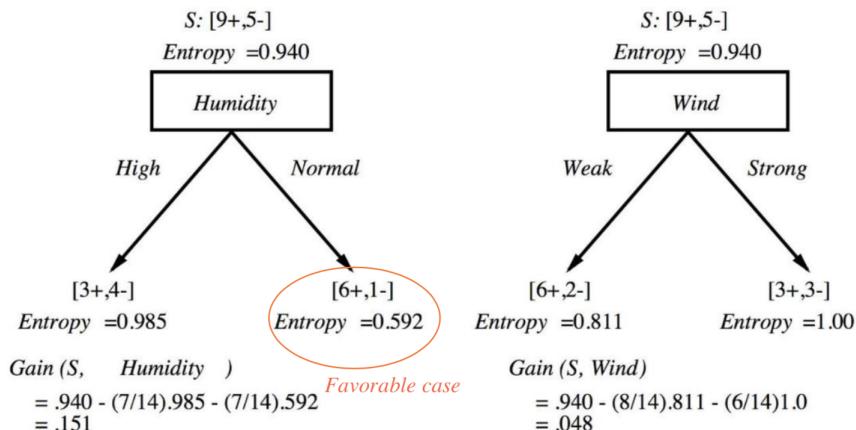
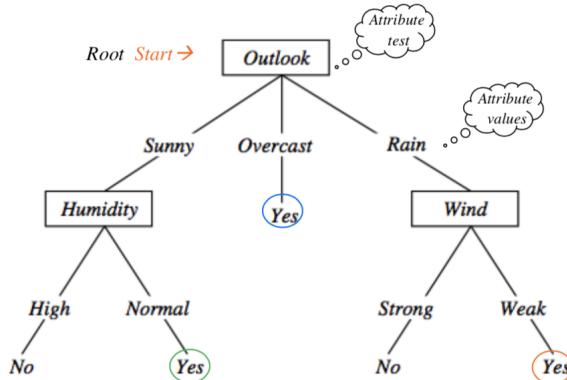
$$\text{Gain}(S, \text{Attr}) = \text{Entropy}(S) - \sum_{v \in \text{Values}(\text{Attr})} \frac{|S_v|}{|S|} \text{Entropy}(S_v) \quad (23)$$

dove $\text{Values}(\text{Attr})$ è l'insieme dei possibili valori associabili all'attributo, mentre S_v è il sottoinsieme degli esempi S per i quali l'attributo ha valore v .

Notare come il primo membro dell'equazione è l'entropia su tutto l'insieme S , mentre il secondo è l'entropia attesa dopo che S è stato partizionato secondo l'attributo Attr. Maggiore è l'Information Gain, più efficace è l'attributo nella classificazione dei dati di training. Usiamo l'entropia per misurare l'omogeneità della classe del sottoinsieme degli esempi, quindi dobbiamo scegliere Attr in modo tale da massimizzare il Gain. Stiamo sottraendo $\sum_{v \in \text{Values}(\text{Attr})} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$ all'entropia generale, ci aspettiamo di ottenere una maggiore omogeneità e quindi un Information Gain maggiore. Lo scopo è quello di separare gli esempi sulla base del target, trovando l'attributo che discrimina gli esempi che appartengono a classi diverse.

6.4.1 Esempio

L'Information Gain viene calcolato per tutti gli attributi, e ID3 lo sfrutta (prende quello con Inf. Gain maggiore) per scegliere il prossimo attributo da testare. Nel nostro esempio $\text{Gain}(S, \text{Outlook})=0.246$ ed è quello con information gain maggiore rispetto a agli altri 3. Quindi per il nodo radice è stato scelto l'attributo Outlook. I dati vengono partizionati rispetto ai valori di Outlook e associati ai nodi successivi. Notiamo che se Outlook=Overcast la risposta è sempre YES, quindi andiamo a valutare $\text{Gain}(S_{\text{Sunny}}, \text{Humidity})$, $\text{Gain}(S_{\text{Sunny}}, \text{Temperature})$ e $\text{Gain}(S_{\text{Sunny}}, \text{Wind})$. Trovando che il massimo gain si ottiene con Humidity. Se l'entropia non è zero l'albero continua a crescere fino ai nodi di classificazione.



Humidity fornisce più informazioni di Wind

6.5 Problemi con Information Gain

L'information Gain favorisce gli attributi con molti valori possibili. Mettiamo caso che nel nostro problema PlayTennis ci fosse stato un attributo che indicava il giorno e il mese, ogni giorno a quel punto sarebbe stato un insieme puro con entropia 0 e quindi avrebbe avuto l'Information Gain più alto di tutti gli altri attributi. Facendo così sarebbe scelto come nodo radice, di conseguenza otterremmo un albero di profondità 1. Ma questo mi è inutile poiché il giorno non è un valore significativo per classificare le istanze future che ancora non abbiamo visto. Quindi come evitiamo la creazione di piccoli sottoinsiemi che mi riducono la generalizzazione?

6.6 Gain Ratio

$$GainRatio(S, Attr) = \frac{Gain(S, Attr)}{SplitInformation(S, Attr)} \quad (24)$$

dove

$$SplitInformation(S, Attr) = - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \quad (25)$$

dove c é il numero di valori possibili per quell'attributo. S_i sono gli insiemi ottenuti partizionando sul valore i di Attr.

SplitInformation misura l'entropia di S rispetto ai valori di A. Più i dati sono dispersi in modo uniforme, più è alto. GainRatio penalizza gli attributi che dividono gli esempi in tante piccole classi come per l'esempio dell'attributo della data. Sia $|S| = n$ numerodate divide gli esempi in n sottogruppi.

$$SplitInformation(S, Date) = -[(\frac{1}{n} \log_2 \frac{1}{n}) + \dots + (\frac{1}{n} \log_2 \frac{1}{n})] = -\log_2 \frac{1}{n} = \log_2 n \quad (26)$$

Il problema che può nascere é che lo SplitInformation può essere 0 o molto piccolo quando $|S_i|$ é circa $|S|$ per alcuni valori i . Ad esempio in un caso estremo in cui $|S_1| = 0$ e $|S_2| = 1$ otteniamo uno $SplitInformation = -[0/n * log0/n + 1/n * log1/n] = -0 - 0 = 0$. Per evitare questo effetto viene utilizzata la seguente euristica:

1. Calcolare il Gain per ogni attributo
2. Applicare il GainRatio solo a quegli attributi con il Gain superiore alla media

6.7 Considerazioni sulla ricerca nello Spazio delle Ipotesi (in DT Learning)

Nell'algoritmo ID3 la ricerca nello spazio consiste nella ricerca dell'albero dal più semplice al più complesso. Mettendolo a confronto con l'algoritmo Candidate Elimination troviamo che:

- Lo spazio delle ipotesi é completo perché rappresenta tutte le funzioni a valori discreti relative agli attributi disponibili (mentre prima avevamo solo funzioni rappresentate da and)
- La ricerca mantiene solo una singola ipotesi corrente perché è un albero di decisione (mentre prima si manteneva il set di tutte le ipotesi consistenti)
- ID3 nella sua forma pura non utilizza backtracking e non c'è garanzia di trovare l'ottimo (perché è un algoritmo greedy e potrebbe fermarsi sui minimi locali)
- Usa tutti gli esempi a disposizione (mentre prima si guardava un esempio alla volta, in ID3 si usa tutto l'insieme degli esempi, siamo meno suscettibili all'errore sul singolo).
- Può terminare prima accettando classi rumorose (mentre prima non accettavamo classi rumorose)

6.8 Inductive Bias in DT Learning

Qual é il nostro Inductive Bias negli alberi di decisione? Cioè qual é il modo che ha ID3 per generalizzare dagli esempi di training?

L'algoritmo ID3 seleziona preferibilmente alberi corti rispetto a quelli con molti livelli e quelli che hanno un Information Gain alto vicino alla radice. Gli alberi di decisione non sono limitati nel rappresentare tutte le possibili funzioni, la restrizione non riguarda lo spazio delle ipotesi ma riguarda la strategia di ricerca.

Abbiamo due tipi di Bias:

1. Bias di ricerca: è legato alla strategia di ricerca. ID3 ricerca uno spazio completo di ipotesi; la strategia di ricerca è incompleta poiché essendo un algoritmo Greedy una volta che scelgo il primo attributo sto eliminando tutte le ipotesi successive che non corrispondono.
2. Bias di linguaggio: è legato all'insieme delle ipotesi esprimibili o considerabili. Candidate Elimination cerca uno spazio di ipotesi incompleto ma la strategia di ricerca è completa.

Perché preferiamo un Bias di ricerca? Se iniziamo con uno spazio di ricerca troppo stringente potremmo escludere la funzione target. Conviene avere uno spazio più ampio che poi andiamo a restringere con la strategia di ricerca⁵. In ML generalmente si usano approcci flessibili (capacità universale dei modelli), senza escludere a priori la funzione target sconosciuta ma ovviamente, flessibile → Overfitting!

6.9 Problemi con gli alberi di decisione

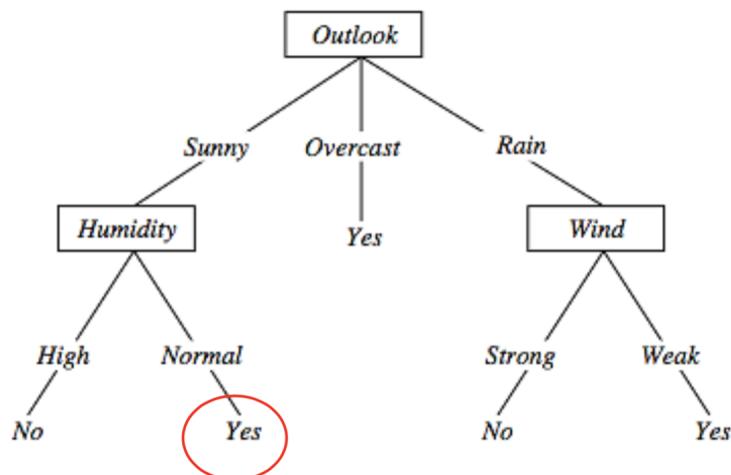
Il problema principale che può presentarsi quando si usano gli alberi di decisione è l'Overfitting (errore di taglio ridotto). Costruire alberi che si adattano troppo agli esempi di training portano all'Overfitting. Definiamo l'errore di una ipotesi h su:

- Dati di training: $error_D(h)$ // errore empirico
- Tutti i dati contenuti in X : $error_X(h)$ // errore atteso

L'ipotesi h "overfitta" i dati di allenamento se c'è un'altra ipotesi h' tale che:
 $error_D(h) < error_D(h')$ e $error_X(h') < error_X(h)$

detto a parole l'ipotesi h si comporta meglio sul singolo dato ma peggio sull'insieme di dati. Questo comporta che h' funziona male sul training set, ma molto meglio sui dati futuri non visti precedentemente.

Un esempio con un dato rumoroso è il seguente: $\{Outlook = Sunny, Temp = Hot, Humidity = Normal, Wind = Strong, PlayTennis = No\}$ da come valore target NO, invece rispetto al nostro albero di decisione la risposta dovrebbe essere YES.



Questo nuovo esempio rumoroso provoca la divisione del secondo nodo foglia creando un albero più complesso. Adattandosi anche al rumore, il nuovo DT è perfetto per i dati di training ma non per i nuovi dati.

⁵Rasoio di Occam: "prefer the simplest hypothesis that fits the data"

6.9.1 Evitare l'Overfitting

Ci sono diversi approcci per evitare l'overfitting che possono essere racchiusi in due classi:

1. fermare la crescita dell'albero prima che abbia una classificazione perfetta
2. permettere all'albero di overfittare i dati e successivamente eseguire un post-potatura dei rami

Come possiamo valutare l'effetto di queste decisioni?

- Possiamo dividere il training set in due parti (training e validazione) e utilizzare il set di validazione per valutare l'utilità della post potatura
- Possiamo utilizzare tutti i dati di training e in seguito eseguire un test statistico per stimare l'effetto di espansione o potatura
- Possiamo usare un meccanismo di misurazione della complessità per la codifica degli esempi di allenamento e dell'albero. Cioè usare il principio della lunghezza minima della descrizione e fermare la crescita dell'albero quando la dimensione dell'encoding è minima.

6.9.2 Potatura con errore ridotto

Come possiamo usare il validation set per evitare l'overfitting? Nel reduced error pruning ogni nodo è un candidato per la potatura: la potatura consiste nella rimozione di un sottoalbero radicata 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. Uno svantaggio è sacrificare dei dati del training set per formare il validation set.

6.9.3 Regola della post-potatura

Nella pratica un metodo che funziona spesso è la post-pruning rule. Le seguenti regole sono solo euristiche: non garantiscono l'ottimo a priori.

1. Si crea l'albero di decisione a partire dal training set fino a che i dati di training fittano al loro meglio. (stiamo facendo overfitting appositamente)
2. Si converte l'albero in un insieme equivalente di regole
 - Ogni path corrisponde a una regola
 - Ogni nodo lungo un path corrisponde a una pre-condizione
 - Ogni nodo foglia classifica

per esempio $(Outlook = Sunny) \wedge (Humidity = High) \rightarrow (PlayTennis = No)$

3. Potare (generalizzare) ogni regola rimuovendo quelle pre-condizioni la cui rimozione migliora l'accuracy sia sul validation set, sia sul training set.
4. Ordinare le regole in ordine di precisione stimato e considerarle in sequenza quando si classificano nuove istanze

Perché trasformare ogni path in regola? Ogni percorso distinto produce una regola diversa, la rimozione di una condizione può essere basata su un criterio locale. La potatura delle pre-condizioni è specifica delle regole mentre la potatura dei nodi è globale e influisce su tutte le regole!

6.10 Problema: Valori continui degli attributi

Fino ad ora abbiamo usato valori discreti per gli attributi, come possiamo affrontare quelli a valori continui?

Dato un attributo a valore continuo A , si crea dinamicamente un nuovo attributo A_c tale che: $A_c = \text{True}$ se $A < c$, False altrimenti.

Ma come determinare il valore di soglia c ? Un esempio sul tennis potrebbe essere:

<i>Temperature</i>	40	48		60	72	80		90
<i>PlayTennis</i>	No	No	<i>54</i>	Yes	Yes	Yes	<i>85</i>	No

Cioè possiamo determinare il valore utilizzando la media tra due esempi consecutivi dove però abbiamo un cambio di classificazione: nell'esempio $(48+60)/2=54$ e $(80+90)/2=85$.

6.11 Problema: Dati di training incompleti

Come possiamo invece risolvere il problema dei dati mancanti? La strategia è utilizzare altri esempi per "indovinare" l'attributo in due modi:

- Si assegna il valore più comune tra tutti gli esempi di training sul nodo o quelli della stessa classe.
- Si assegna una probabilità a ciascun valore possibile da assegnare, in base alle frequenze, si associa il valore all'attributo mancante secondo la distribuzione di probabilità.

I valori mancanti in nuove istanze da classificare vengono trattati di conseguenza e viene scelta la classificazione più probabile.

6.12 Problema: attributi con costi differenti

In alcuni problemi le istanze degli attributi possono avere un costo associato, cioè possiamo dare più importanza ad alcuni attributi rispetto ad altri. Preferiamo gli alberi che usano costi bassi per gli attributi. L'algoritmo ID3 può essere modificato per lavorare anche con i costi:

- Tan e Schlimmer

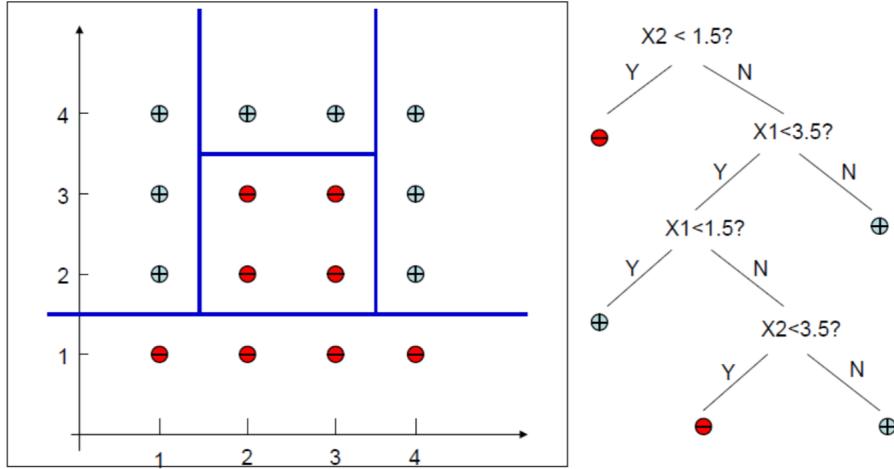
$$\frac{\text{Gain}^2(S, \text{Attr})}{\text{Cost}(S, \text{Attr})} \quad (27)$$

- Nunez

$$\frac{2^{\text{Gain}(S, \text{Attr})} - 1}{(\text{Cost}(\text{Attr}) + 1)^w} \text{ con } w=0/1 \quad (28)$$

6.13 Visione Geometrica (Decision Boundaries dei DT)

I decision boundaries, che possono essere prodotti da un albero di decisione, dividono lo spazio di input in rettangoli paralleli agli assi ed etichettano ogni rettangolo con una delle classi K (foglia dell'albero).



6.14 Conclusioni sugli alberi di decisione

Gli alberi di decisione sono un approccio molto utilizzato per la classificazione con un numero discreto di classi. Lo spazio delle ipotesi comprende l'area degli approcci proposizionali quindi basati su regole. Si comporta bene con dati rumorosi e gestisce anche i valori degli attributi mancanti. È facile da capire (regole if-then-else, a meno che non lo siano troppe). ID3 ricerca in uno spazio completo di ipotesi, con una strategia greedy INcompleta (perché manca il backtracking, una volta che fissa il primo nodo, non torna indietro). Presenta un approccio flessibile: l'overfitting è un problema importante che affrontata con il post-potatura e la generalizzazione delle regole indotte dall'albero.

7 Validation

Un problema fondamentale del ML è valutare la capacità di generalizzazione del nostro modello, una domanda che dobbiamo porci è: quando un modello è un buon modello? L'apprendimento, come abbiamo visto, consiste nel trovare una buona funzione in uno spazio di funzioni costruito a partire dai dati conosciuti. "Buona" in relazione al fatto di effettuare un basso errore di generalizzazione, per sapere quanto accuratamente il modello si comporta su nuovi dati non visti precedentemente.

In particolare, qualsiasi h (ipotesi) che approssima bene f (la nostra funzione da cercare) sui dati di training, approssimerà bene anche su istanze non conosciute? La generalizzazione è quindi un punto cruciale del ML, dobbiamo trovare un modo in cui scegliamo il miglior modello tra svariati modelli calcolati. Abbiamo visto che si procede con una fase di apprendimento (Learning phase) dove si costruisce (training, fitting) il modello sfruttando i dati conosciuti. Segue poi una fase predittiva (Predictive phase) dove si applica il modello a nuovi dati sconosciuti e si valuta l'ipotesi predittiva (ad esempio si valuta la capacità di generalizzazione), quest'ultima è una fase di test.

Per valutare generalmente usiamo due misurazioni:

- Per la classificazione: Mean Square Error sulla funzione di Loss e l'accuratezza o Mean Error Rate per il risultato.
- Per la regressione: Mean Square Error o Mean Absolute Error...

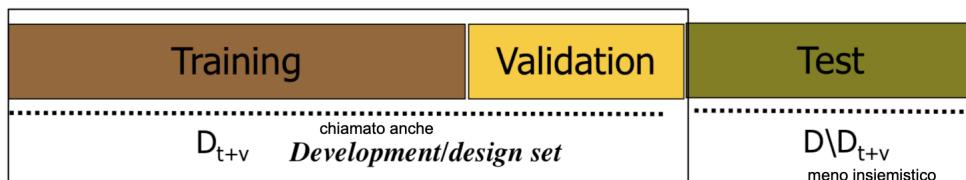
7.1 Obiettivi Validazione

La validazione ha due obiettivi principali:

- Model Selection: stima la performance (errore di generalizzazione) di differenti modelli di apprendimento in modo tale da scegliere il migliore. Il compito di selezione del modello ovviamente restituisce un modello.
- Model Assessment: dopo aver scelto un modello finale, stimiamo/valutiamo il suo errore/rischio di previsione (errore di generalizzazione) su nuovi dati di test mai visti prima (magari dati dal cliente). Il compito di valutazione di un modello restituisce una stima.

7.2 Holdout Cross-Validation

Per ottenere una valutazione accurata di un modello dobbiamo valutarlo su esempi nuovi mai visti. Ma noi abbiamo solo un insieme dati limitati e non dobbiamo sprecarli. Dobbiamo quindi suddividere il nostro data set D in Training Set, Validation Set e Test Set.



Il Training Set verrà usato dall'algoritmo per la creazione dell'ipotesi h . Il Validation Set viene usato per scegliere il miglior modello tra differenti modelli costruiti. Infine usiamo il Test Set solo per la valutazione del modello.

Cosa succede se il test set viene utilizzato in un ciclo ripetuto di design? Stiamo eseguendo una selezione del modello e non una valutazione dell'errore di generalizzazione affidabile. Purtroppo non saremo in grado di valutare il modello su esempi non visti, poiché ormai "ce li siamo bruciati". In questo caso, dato che abbiamo usato il test set abbiamo una valutazione sovraottimistica. La regola d'oro è tenere una separazione tra i compiti e usare insiemi separati per Training, Validation e Test.

7.3 Meta-Algoritmo per l'utilizzo del Data Set

- separare TR, VL e TS
- cercare la migliore ipotesi $h_{w,\lambda}$ cambiando gli iper-parametri λ del modello
- per ogni valore differente dell'iper-parametro λ : cercare la migliore ipotesi che minimizza l'errore / perdita empirica (fittando i dati sul TR set) trovando i migliori parametri w .
- selezionare la migliore ipotesi $h_{w,\lambda}$ dove migliore significa con il minor errore sul Validation Set!
- opzionale: è possibile far fittare h su TR+VL con λ ormai fissato
- valutare la h finale sul Test Set

La ricerca dei migliori iper-parametri può consistere in una ricerca in una griglia di valori candidati, per ogni modello calcolare il risultato su Validation Set e prendere quello che con il minimo errore a massima accuratezza.

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

Nell'immagine Res1 è calcolato sul Validation Set dal modello con un polinomio di grado 1 e un $\lambda = 0.1$ (modello allenato sul training set). Il migliore è Res3, perché ha un polinomio di grado 4 e $\lambda = 0.1$. Questi calcoli sono facili da parallelizzare.

7.3.1 Esempio

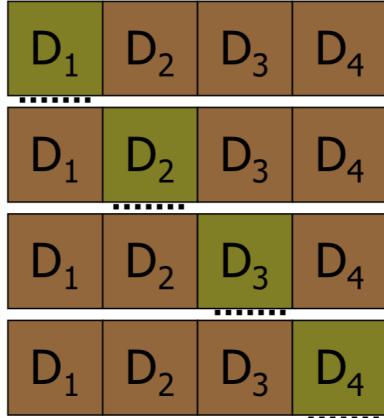
λ	TR	VL	TS
0.5	75	70	70
0.1	80	75	70
0.01	90	70	72

- In che ordine si usano le porzioni di dati per calcolare i valori in tabella?
Prima il training, poi validation e infine il test
- Quale modello (ossia lambda) si sceglie?
Si sceglie il migliore sul Validation Set, NON si usa il Test Set per il model selection!

7.3.2 Esempio perché separare VL e TS

Ci troviamo in una situazione con 20-30 esempi, 1000 variabili di input, come output abbiamo target 0/1 calcolato randomicamente. Trovo un modello con una sola variabile che indovina "per caso" al 99% su training e validation set. Otteniamo un risultato perfetto (cioè un modello con accuratezza al 99%)? 99% non è una buona stima dell'errore di test (quella corretta è 50% visto che gli 0/1 in output sono randomici). L'errore stimato su training o validation per il model selection NON è utile! Usare tutto il data set per feature/model selection lede la correttezza della stima. Se effettuo la selezione del modello su tutto il data set e poi eseguo training e validation su sottoinsiemi, il test fornirà risultati biased (FS-bias). Un test set esterno fornisce invece la stima corretta del 50%.

7.4 K-fold cross validation



Hold out cross validation, che abbiamo visto prima, può effettuare un uso insufficiente di dati. Il seguente metodo ogni volta riparte da zero e non tiene conto di quello che è successo prima.
K-fold cross validation:

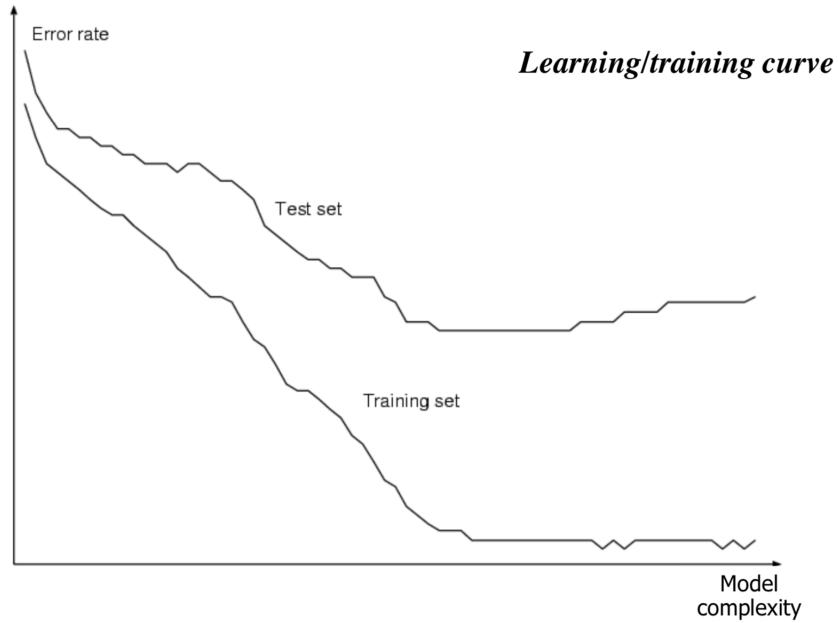
Scegliere un λ . Dividere il data set in D porzioni mutualmente esclusive di dati. Allenare l'algoritmo di apprendimento sull'insieme $D - D_i$ e testarlo su D_i . Calcolo la media sui risultati trovati su D_i . Cambio λ e rieseguo il tutto. Alla fine scelgo il modello con il minimo errore di validation, sulla base del valore calcolato con la media sui vari D_i .

Utilizza tutti i dati per training, validazione o test. NOTA: questa tecnica può essere utilizzata sia per il validation set, sia per il test set. In quante parti dobbiamo dividere? 3,5,10... ma spesso è computazionalmente costoso.

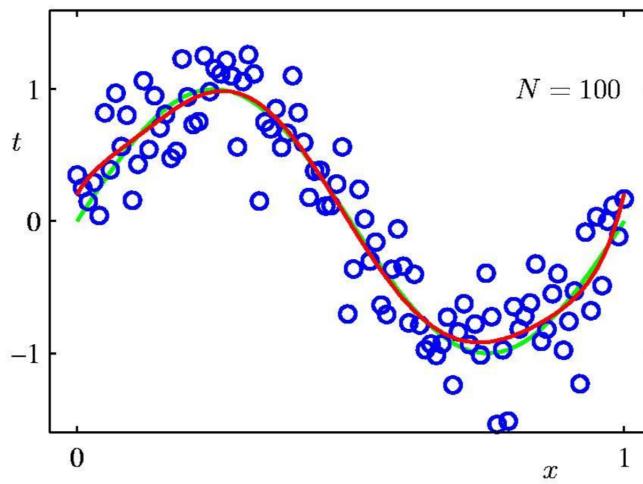
7.5 Esempio di Model Selection e Assessment

Dividere i dati in Training Set e Test Set (sfruttando Hold out o K-fold), usare K-fold internamente sul Training Set per trovare i migliori iper-parametri λ del modello. Eseguire una ricerca su griglia con tutti i possibili valori dell'iper-parametro e scegliere il miglior λ . A questo punto allenare su tutto il TR set il modello finale e valutarlo su un test set esterno.

7.6 Comportamento tipico di un algoritmo di apprendimento



All'inizio sia l'errore di training che quello di test sono alti, perché? Succede un po' quello che succedeva con il grado del polinomio troppo basso, che non fittava bene i dati → Underfitting.
 Alla fine invece l'errore di training è basso mentre l'errore sul test è alto, perché? È il caso in cui si sta overfittando sui dati di training e quindi il grado del polinomio è molto alto, ma non si sta comportando bene riguardo la generalizzazione sui dati di test sconosciuti.



Se prendiamo un polinomio con grado alto, ma abbiamo molti dati in più, predico meglio la funzione target. Ma quindi il problema era l'alto grado del polinomio o i pochi dati a disposizione? Per dare una risposta a questa domanda esiste della teoria di supporto.

7.7 Statistical Learning Theory

Mettendo insieme:

- La capacità di generalizzazione (misurata come errore di test) di un modello rispetto all'errore di training, delineando le zone di overfitting o underfitting
- Il ruolo della complessità del modello
- Il ruolo del numero di dati

troviamo la Statistical Learning Theory (SLT):

- si approssima una funzione sconosciuta $f(x)$
- si cerca di minimizzare la funzione di rischio $R = \int Loss(t, h(x)) dP(x, t)$ (Integrale della Loss su tutti i dati).
- vengono dati un valore target t , una probabilità di distribuzione $P(x, t)$ e una funzione Loss del tipo $L(h(x), t) = (t - h(x))^2$
- il compito finale è quello di cercare una ipotesi h nello spazio delle ipotesi H , dove il valore di rischio generale R è minimo, ma noi abbiamo solo un data set finito $TR(x_i, t_i)$ con $i=1\dots l$
- per cercare h dobbiamo quindi minimizzare l'errore empirico (training error) trovando il miglior valore per il modello con parametri liberi w .

$$R_{emp} = \frac{1}{l} \sum_{i=1}^l (t_i - h(x_i))^2$$

Questo si chiama principio induttivo della minimizzazione del rischio empirico (Empirical Risk Minimization). La domanda che dobbiamo porci è: possiamo usare R_{emp} per approssimare R ?

7.8 Teoria di Vapnik-Chervonenkis + SLT

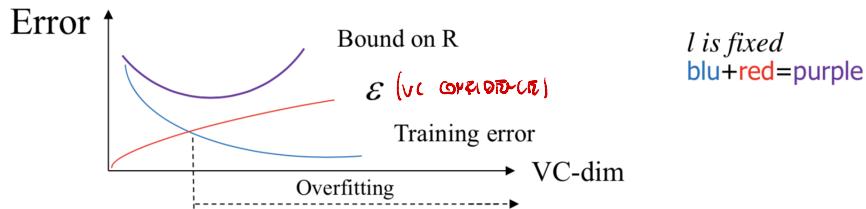
- VC-dimension è una misura per la complessità dello spazio delle ipotesi H (flessibilità per fittare i dati, come ad esempio il numero di parametri per modelli lineari/grado del polinomio)
- VC-bounds afferma che con una probabilità di $1 - \delta$ che

$$R \leq R_{emp} + \epsilon \left(\frac{1}{l}, VC, \frac{1}{\delta} \right)$$

cioè che il rischio generale R ha un limite superiore definito dal rischio empirico (che decresce con la complessità dei modelli) più il VC-confidence. Il VC-confidence (ϵ) aumenta all'aumentare della complessità del modello (indicata da VC) e diminuisce quando il numero dei dati aumenta. ($1/\delta$ indica la probabilità che valga questo bound)

Possiamo usare il rischio empirico per approssimare R ? Sì, il Machine Learning è ben fondato, l'intuizione sta nel fatto che un numero alto di dati portano ad un R basso, mentre un alto VC -dim abbassa R_{emp} ma potrebbe alzare R (overfitting).

7.9 Structural Risk Minimization



Sfruttiamo il concetto di controllo della complessità del modello, eseguiamo un trade-off tra la complessità del modello e l'accuratezza sul TR set.

7.10 Conclusioni

La STL permette di inquadrare formalmente il problema della generalizzazione e overfitting, fornendo limitazioni superiori analitiche e quantitative al rischio R di predizione su tutti i dati, indipendentemente dal tipo di learning algorithm o dettagli del modello. Il ML è ben fondato, il rischio del learning (e dell'errore di generalizzazione) può essere analiticamente limitato! Si può trovare un buona approssimazione della f target da esempi, a patto di avere un buon numero di dati e una adeguata complessità del modello (misurabile formalmente con la VC-dim). Questo ci porta a nuovi modelli (SVM) (e altri metodi che direttamente considerino il controllo della complessità nella costruzione del modello). La STL fonda uno dei principi induttivi sul controllo della complessità.

Alcuni esempi di controllo della complessità sui modelli lineari sono il numero di parametri liberi w o la dimensione in input. Mentre per i Decision Trees il numero di nodi. Vedremo un approccio diretto alla complessità attraverso il modello SVM.

8 Support Vector Machine

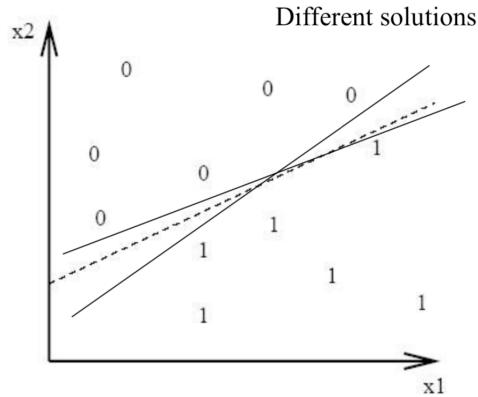
Il Support Vector Machine (o SVM) è l'approccio più famoso per il supervised learning. È un classificatore derivato dalla Statistical Learning Theory, dopo anni di sviluppi teorici, SVM è diventato famoso quando, usando immagini come input, dava una accuratezza comparabile a reti neurali.

8.0.1 Obiettivi utilizzo SVM

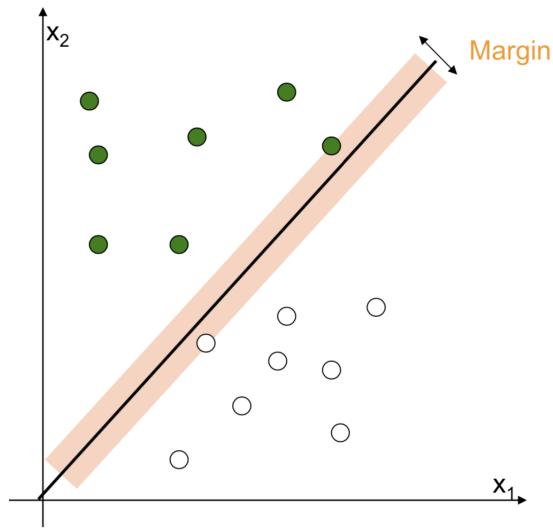
1. Controllare la complessità del modello attraverso un approccio di ottimizzazione, approssimando la minimizzazione del rischio strutturale (Max Margin Classifier).
2. Usare efficientemente l'espansione lineare via kernel in modo tale da ottenere un approccio flessibile per il Supervised Learning non lineare (Kernel).
3. Evitare di utilizzare SVM nel modo sbagliato.

8.1 Maximum Margin Classifier (controllo complessità)

Viene utilizzato nella classificazione binaria di problemi, mettiamoci nel caso in cui abbiamo un sistema linearmente separabile, non tutti gli iper-parametri che risolvono il problema sono uguali

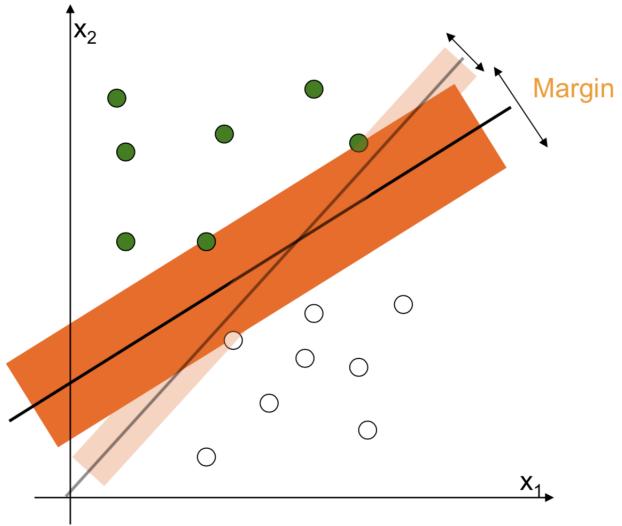


Grazie a SVM abbiamo un criterio per scegliere le soluzioni: minimizzare il rischio strutturale.



Il margine è (il doppio della) la distanza tra l'iperpiano e il dato più vicino.

Notiamo che però non tutti gli iperpiani che risolvono il problema sono uguali, variando l'iperpiano anche il margine cambia.



Ed è per questo motivo che cerchiamo il classificatore con il margine massimo!

8.2 Rappresentazione canonica dell'iperpiano e Support Vector

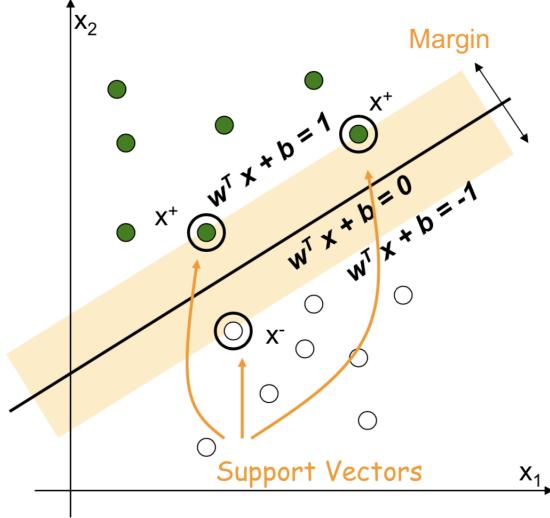


Figure 7: nota: $w_0 = b$

Support Vectors: $x_i : |w^T x_i + b| = 1$

Tutti i punti sono classificati correttamente se $(w^T x_i + b)y_i \geq 1 \quad \forall i$

8.3 Verso l'ottimizzazione del margine

Consideriamo il problema di apprendere un modello lineare per classificazione binaria $h(x) = \text{sign}(wx + b)$. Il problema consiste nel trovare il vettore w e il valore b in modo tale che tutti i punti sono classificati correttamente e il margine è massimizzato.

La rappresentazione canonica dell'iperpiano è definita come:

Il punto (x_i, y_i) è classificato correttamente per tutti gli i (tutti gli esempi del training set) se:

- $w^T x_i + b \geq 0$ se $y_i = 1$
- $w^T x_i + b < 0$ se $y_i = -1$

è possibile scalare w e b in modo tale che i punti più vicini all'iperpiano soddisfino $|w^T x_i + b| = 1$ e quindi

- $w^T x_i + b \geq 1$ se $y_i = 1$
- $w^T x_i + b \leq -1$ se $y_i = -1$

Due fatti a noi utili sono:

1. Margine = $2/|w|$ e sappiamo anche che $|w|^2 = (w^T w)$.
Quindi per massimizzare il margine → minimizzare $|w| \rightarrow$ minimizzare $|w|^2/2$

2. Il VC-dim del SVM è inversamente proporzionale al margine, quindi controlliamo la complessità del modello utilizzando il margine

Come abbiamo accennato precedentemente, l'iperpiano ottimo è l'iperpiano che massimizza il margine e ovviamente risolve il problema di training.

*Questa nostra
nella
di controllo
minimizzando
il margine*

8.4 Problema di ottimizzazione quadratica

Il nostro problema è quindi diventato

minimizzare $|w|^2/2$ in modo tale che $(w^T x_i + b)y_i \geq 1$ per tutti gli i .

Questa è la nostra forma primale del problema, ne esiste anche una duale. Nota: cerchiamo la minimizzazione diretta della complessità del modello (funzione di ottimizzazione). Essendo il problema linearmente separabile, dobbiamo mantenere la soluzione tramite i vincoli. In relazione alla SLT minimizzando la VC-dim, diminuiamo anche il bound di errore.

8.5 Nuovo classificatore $h(x)$ (Problema Duale)

La soluzione ottima può essere trovata anche massimizzando

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \text{ con } \alpha_i \geq 0$$

Il nostro compito è cercare un vettore α ottimale, dopo averlo trovato (calcolato dalla forma duale) possiamo calcolare i vettori $w = \sum \alpha_i y_i x_i$ e $b = y_k - w^T x_k$ per qualsiasi $\alpha_k > 0$. Possiamo adesso definirci un nuovo classificatore

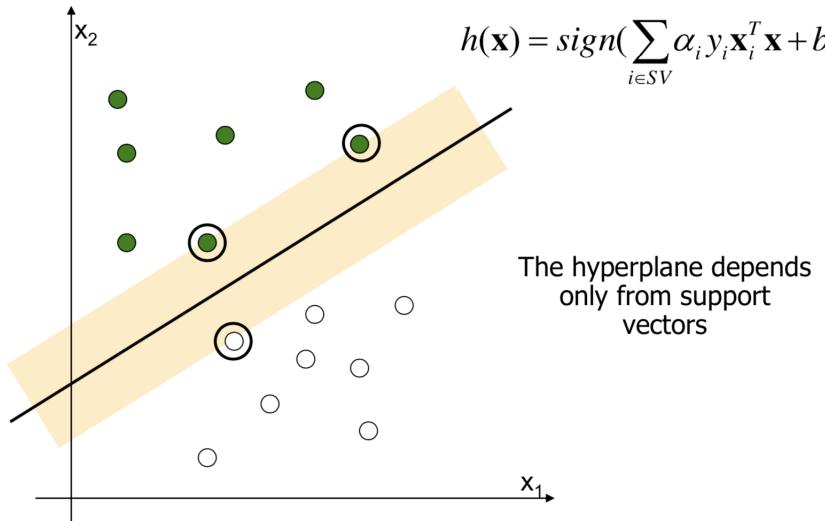
$$h(x) = \text{sign}(w^T x + b) = \text{sign}\left(\sum_{i=1}^l \alpha_i y_i x_i^T x + b\right) = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i x_i^T x + b\right) \quad (29)$$

$x_i^T x$ è il prodotto scalare tra le x dei vettori di supporto (presi ovviamente dal training set) e la x da classificare.

Una proprietà importante è che

i pesi α_i associati ad ogni dato sono uguali a zero ad eccezione dei Support Vector. Cioè se $\alpha_i \neq 0$ allora x_i è un Support Vector.

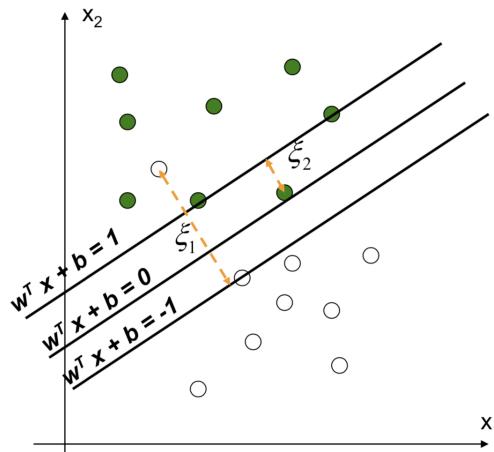
La soluzione è sparsa e formulata solo nei termini dei Support Vectors, in poche parole l'iperpiano dipende solo dai Support Vectors! Inoltre è una forma speciale della soluzione in cui non dobbiamo calcolare esplicitamente (w, b) per classificare i punti.



8.6 Margine Soft

Purtroppo però, nella realtà, non abbiamo quasi mai dei problemi che sono linearmente separabili e con zero errore di classificazione. Anche se il problema fosse linearmente separabile, rischieremmo di avere un margine piccolissimo. Ammettiamo quindi errore introducendo le slack-variables (variabili di rilassamento).

Le variabili slack ξ_i possono essere aggiunte per permettere classificazioni errate o punti "molto rumorosi"



Modifichiamo il nostro problema in forma primale:

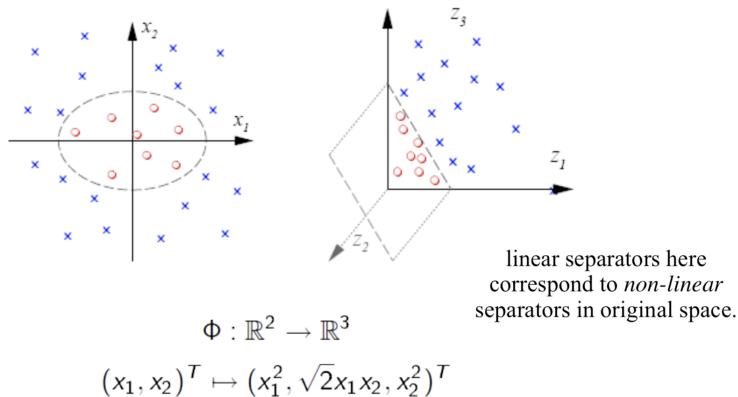
$$\text{minimizzare } |w|^2/2 + C \sum_i \xi_i \text{ tale che } (wx_i + b)y_i \geq 1 - \xi_i \text{ e } \xi_i \geq 0 \text{ per ogni } i$$

Possiamo vedere ξ come la distanza tra il vettore di supporto e il punto che stiamo analizzando, mentre $C > 0$ gestisce il numero di errori permessi (C è un valore definito dall'utente). Con un C basso stiamo accettando tanti errori sul TR Set (perché gli ξ possono crescere molto) e quindi potremmo sfociare in un possibile underfitting. Con un C alto non permettiamo errori sul TR Set e quindi potremmo cascpare in overfitting.

8.7 Mapping per Spazi Dimensionali Ampi

Il compito di SVM è quello di creare un iperpiano che separa linearmente i dati, ma spesso non è possibile dividere linearmente un problema in una certo spazio dimensionale... ed è in questi casi che viene utilizzato il "kernel trick". Cioè spesso i dati non sono linearmente separabili nello spazio dimensionale in input, ma potrebbero essere divisibili in uno spazio dimensionale più grande.

Abbiamo visto cosa succede in casi non lineari: dobbiamo utilizzare efficientemente l'espansione della base via kernel in modo tale da ottenere un approccio flessibile anche per il Supervised Learning non lineare.



Ma questo lo sapevamo già, quando usavamo la funzione $\phi(x)$ al posto di x .

$$h_w(x) = \text{sign}(\sum_k w_k \phi_k(x))$$

Sappiamo anche che l'utilizzo di spazi ad alta dimensione (funzioni di espansione della base) può essere non conveniente nel calcolo dal punto di vista computazionale e può facilmente portare a un overfitting, nel caso in cui non controlliamo la dimensione dello spazio e la complessità del classificatore (la complessità in questo caso è correlata alla dimensionalità dell'input).

Useremo l'approccio del Kernel per gestire (implicitamente) lo spazio nel contesto della modellizzazione regolarizzata (la complessità dipende dal margine). Pertanto, grazie alla regolarizzazione automatizzata di SVM, la complessità del classificatore può essere mantenuta ridotta indipendentemente dalla dimensionalità nel nuovo spazio.

In SVM non è necessario calcolare w e nemmeno calcolare direttamente la ϕ . Poiché sfruttiamo il Kernel. Il Kernel è il risultato del prodotto scalare. Non ci importa di come è fatta la phi: non ho bisogno di espanderla per poi fare il prodotto scalare, ma mi basta il risultato di $K(x_j, x_k)$.

$$h(x) = \text{sign}(\sum_{i \in SV} \alpha_i y_i \phi(x_i)^T \phi(x)) = \text{sign}(\sum_{i \in SV} \alpha_i y_i K(x_i, x))$$

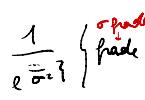
8.7.1 Esempi di Kernel

- Lineare: $K(x_i, x_j) = x_i^T x_j$
Mapping $\phi: x \rightarrow \psi(x)$ dove $\psi(x)$ è x stesso
- Polinomiale: $K(x_i, x_j) = (1 + x_i^T x_j)^p$ con p iper-parametro che indica il grado del polinomio
Mapping $\phi: x \rightarrow \psi(x)$ dove $\psi(x)$ ha dimensione esponenziale rispetto a p
- RBF (radial-basis-function) Gaussiana: $K(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma^2}$ dove σ è un iper-parametro
Mapping $\phi: x \rightarrow \psi(x)$ dove $\psi(x)$ è di dimensione infinita
(molto potente sul TR ma può portare ad overfitting)

8.8 Riassunto procedimento SVM con Kernel Fun.

- Scegliamo un parametro C (trade-off)
- Scegliamo una funzione di Kernel K (e i suoi parametri)
- Troviamo il miglior α
- Usiamo il modello finale

$$h(x) = \text{sign}(\sum_{i \in SV} \alpha_i y_i K(x_i, x))$$



 è molto flessibile
 definisce boundary (regola)
 x punto (del TR)
 ↓
 σ lungo, il set è stretto, n. classific.
 se questo il punto è allontanato niente,
 classifica solo in base a x interno, quindi
 o errore di training
 ma n. re in OVERFITTING

8.9 Utilizzare bene SVM

Evitare errori di interpretazione tipici nell'uso di SVM. È possibile che si verifichi un overfitting senza un'attenta selezione dei parametri: C, funzione Kernel, parametri del Kernel, ecc... Il trattamento implicito dello spazio dimensionale ampio deve avvenire nello spazio delle caratteristiche e non in quello di input. Anche la tecnica di validation vista fino ad ora per la selezione del modello e la valutazione del modello devono essere utilizzate rigorosamente. Quindi:

- Trasformare i dati in un formato leggibile da un software SVM (ad esempio {red, green, blue} → (0,0,1), (0,1,0), (1,0,0))
- Eseguire una semplice scalatura dei dati (ad esempio in un intervallo da [-1,1] o [0,1])
- Considerare il kernel $K(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma^2}$
- Usare la cross-validation per trovare i migliori parametri C e σ
- Riutilizzare il TR set però con i migliori C e σ trovati
- Eseguire il testing su un Test Set esterno

9 K-Nearest Neighbors

I modelli precedenti che abbiamo visto sono modelli parametrici, nel senso che sono modelli che si comportano in modi diversi in base ad alcuni parametri modificabili. Esistono anche i modelli non parametrici, cioè quei modelli che non sono caratterizzabili/vincolabili attraverso dei parametri. K-NN fa parte del Supervised Learning, ed è un modello basato su istanze (è una sorta di algoritmo table lookup efficiente).

9.1 1-Nearest Neighbor

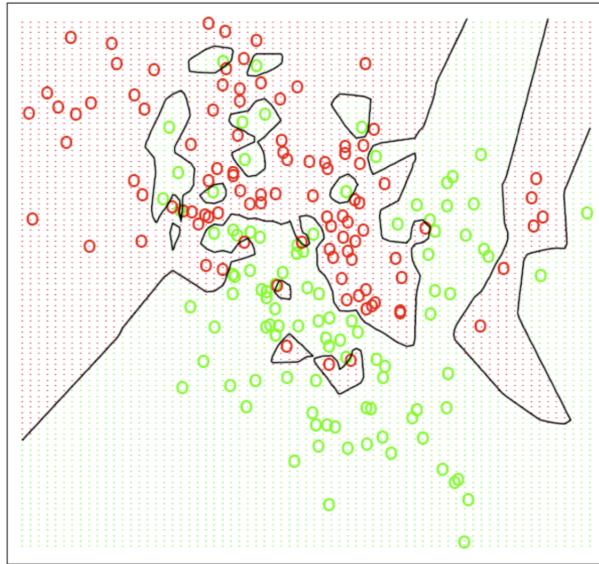
Questo algoritmo non impara, ma sfrutta tutti i valori del training set.

- Salviamo i dati di training nella forma $\langle x_j, y_j \rangle$ con $j=1\dots l$
- Dato un input x di dimensione n dobbiamo trovare l'esempio di training più vicino x_i tale che $d(x, x_i)$ è minimo
dove

$$d(x, x_j) = \sqrt{\sum_{t=q}^n (x_t - x_{jt})^2} = \|x - x_j\|$$

x_t è la componente t-esima di x , x_{jt} è la componente t-esima di x_j

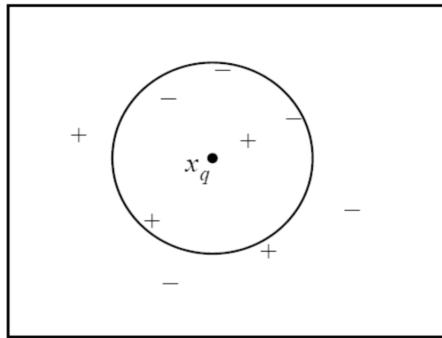
- Diamo come output y_i (in pratica stiamo vedendo l'esempio che più assomiglia ai nostri dati e rispondiamo come ha risposto lui)



È molto flessibile, non c'è errore di classificazione sui dati del TR Set... ma il Decision Boundary non è più lineare, potrebbe portare all'overfitting?

9.2 K-NN

È la versione in cui si "guarda" il comportamento dei K vicini.



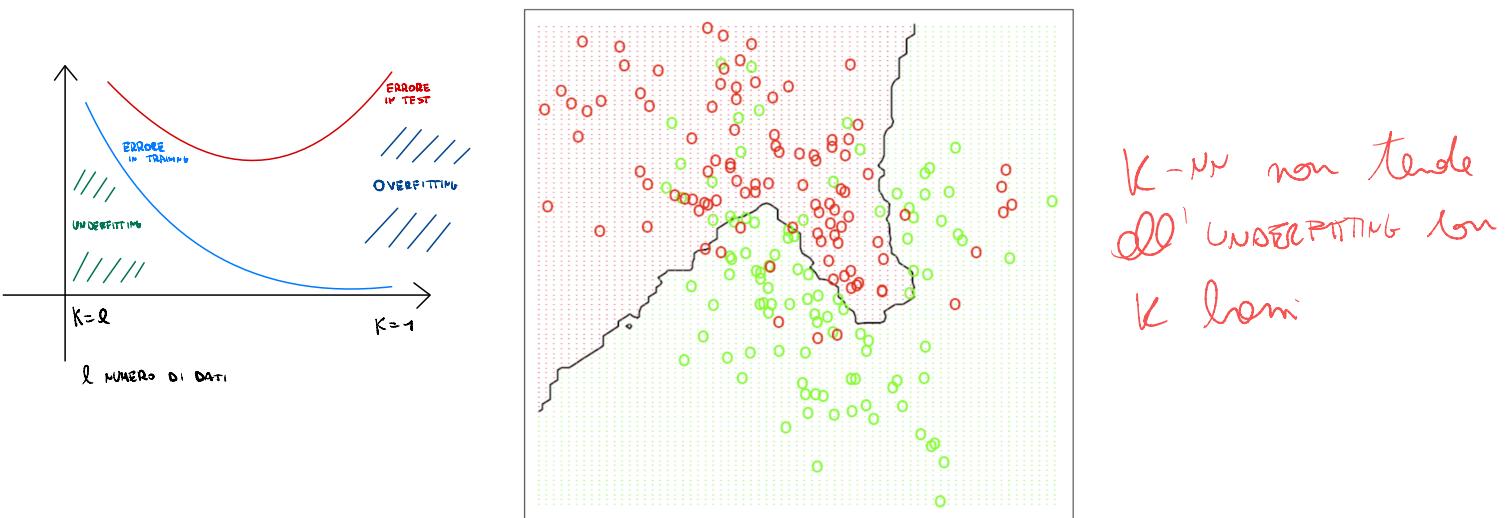
In questo caso 1-NN risponderebbe + per x_q , valutando invece con 5-NN viene restituito - per x_q . Come succedeva con il polinomio di alto grado che andava in overfitting, anche qui dobbiamo renderlo più "smooth" valutando su un insieme di vicini.

Per questo possiamo "dare un occhiata" a tutti i K vicini e restituire una media:

$$avg_k(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

dove $N_k(x)$ è il vicinato di x che contiene esattamente K vicini. Se c'è una chiara dominanza di una delle classi nel vicinato di x , allora è probabile che anche x appartenga a quella classe. Quindi la regola di classificazione è la maggioranza che vota tra i membri di $N_k(x)$.

$$h(x) = 1 \text{ se } avg_k(x) > 0.5 , 0 \text{ altrimenti}$$



Dobbiamo trovare il giusto trade-off tra underfitting e overfitting trovando il giusto valore K. (Nota: è possibile usare K-NN anche se ci sono più classi).

9.3 Considerazioni su K-NN

Non c'è una ipotesi globale per tutte le istanze: non c'è quindi nessun modello da allenare. Dobbiamo semplicemente memorizzare gli esempi di input, infatti è un metodo basato sulla memoria, sull'istanza e sulla distanza. Il Bias induttivo è scegliere il risultato in base alla distanza.

Notare che K-NN fa una approssimazione locale della funzione target per ogni nuovo esempio da classificare, il costo computazionale è tutto contenuto nella fase di predizione. Inoltre è molto costoso a livello computazionale perché per ogni nuovo input bisogna calcolare le distanze dall'esempio a tutti i vettori memorizzati. Il tempo è proporzionale al numero di modelli memorizzati, questo ci fa notare che anche il costo per lo spazio è alto, dato che tutti i dati sono memorizzati.

Quando abbiamo molte variabili in input K-NN spesso fallisce a causa del "curse of dimensionality": quando la dimensione aumenta, il volume dello spazio aumenta in modo tale che i dati disponibili diventano sparsi. Ad esempio la quantità di dati che servono per supportare un risultato spesso cresce esponenzialmente con la dimensione. Troviamo anche il problema denominato "curse of noisy": se il target dipende da poche altre variabili, potremmo trovarci un "modello simile" con la somiglianza dominata dal gran numero di funzioni irrilevanti.

9.4 Distance Based Methods

Quando utilizzo degli approcci basati su distanza (come K-NN o alcune funzioni Kernel) devo assicurarmi che la distanza calcolata sia davvero un fattore discriminante. Stiamo misurando quando una coppia di pattern si assomiglia, inoltre dare una metrica pone un Bias rilevante sulla soluzione. Possiamo imparare in qualche modo la metrica? Sì (Reti Neurali, Learning K...).

10 Unsupervised Learning (ripasso)

Nel Unsupervised Learning non abbiamo esempi etichettati, quindi senza un output associato, siamo noi come sistema di apprendimento che dobbiamo trovare correlazioni tra i dati e raggrupparli (clustering). Dobbiamo quindi ripartire i dati, in cluster (sottoinsiemi di dati simili). Il goal che ci poniamo è ripartire in modo ottimo una distribuzione sconosciuta di dati, in uno spazio di dimensione x , in regioni (approssimando poi usando un cluster).

Se dovessimo lavorare sullo spazio delle ipotesi H , lo rappresenteremmo come un insieme di quantizzatori di vettori $x \rightarrow c(x)$ cioè dato un vettore in input voglio sapere a quale cluster appartiene. Passiamo da uno spazio continuo ad uno spazio discreto.

Una funzione di Loss comune è

$$L(h(x_i)) = \|x_i - c(x_i)\|^2$$

cioè la distanza tra x e il centroide. Il valore medio sulla distribuzione degli input si chiama errore di quantizzazione.

Unsupervised Learning viene spesso usato in Data Analysis per scoprire caratteristiche comuni tra i dati, viene anche usato per il Preprocessing dei dati da utilizzare poi in seguito per altri approcci di ML. Inoltre dobbiamo notare il fatto che raccogliere dati non etichettati è più "economico", rispetto a trovarne labeled.

10.1 K-means

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, tuttavia, presenta diversi inconvenienti e limitazioni che vedremo dopo...

1. Scegliere k centri di cluster in modo che coincidano con k modelli scelti casualmente (o k punti definiti casualmente all'interno dell'ipervolume contenente il set di pattern) $c_1 \dots c_k$ con k fissato che sceglio noi.
2. Assegnare ad ogni modello il centro del cluster più vicino (il vincitore) per ogni x calcoliamo

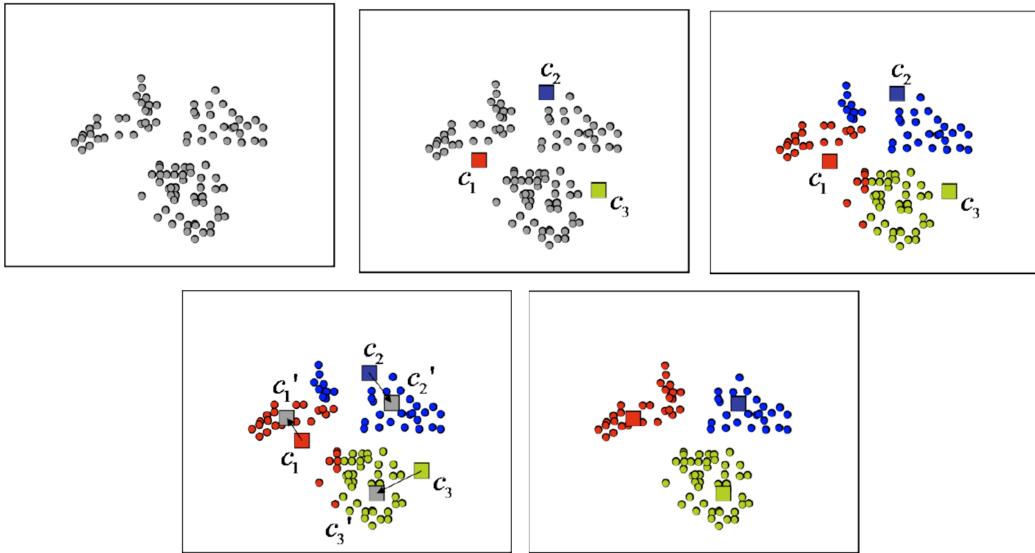
$$i^*(x) = \operatorname{argmin}_i \|x - c_i\|_2^2 = \sum_{j \rightarrow n} (x_j - c_{ij})^2$$

cioè provo tutti i c_i e quello che ha distanza minima è il vincitore (mi interessa l'indice del vincitore i^*). Adesso x appartiene al cluster i^*

3. Ricalcolare i centri del cluster (centroide) utilizzando le nuove appartenenze al cluster corrente

$$c_i = \frac{1}{|\text{num di membri cluster}_i|} \sum_{x_j \in \text{cluster}_i} x_j$$

4. 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 oppure una minima diminuzione dell'errore quadratico)



10.2 Limitazioni K-means

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.

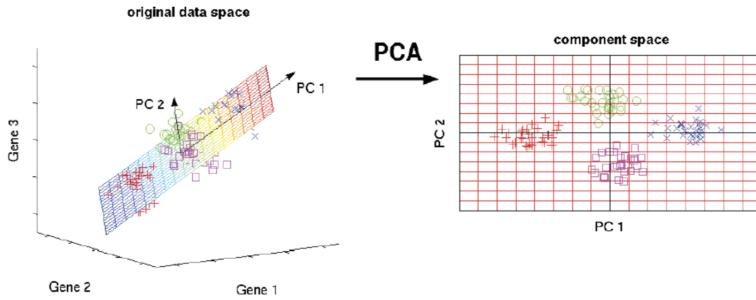
Come viene valutato l'output di un algoritmo di clustering? Che cosa caratterizza un risultato di raggruppamento "buono" e uno "scarso"? Nel clustering esiste ben poco in termini di "standard di riferimento", tranne nei sottodomini specifici in cui conosco la metrica (conosco bene il problema e so riconoscere una classificazione sensata). Misure oggettive (non trattate qui) come ad esempio l'errore di quantizzazione. Noi sappiamo a quali classi appartengono i dati, ma usiamo un algoritmo di clustering, e controlliamo che le classi corrispondano. È un modo sbagliato per valutare la bontà di un algoritmo di clustering!

10.3 Preprocessing dei dati

Abbiamo visto che Unsupervised Learning viene utilizzato per il preprocessing dei dati, ma in che modo? Menzioniamo la riduzione della dimensionalità cioè trasformare la dimensione dei dati in una più piccola. (non è importante la quantità, bensì la dimensione dei dati!)

$$\langle x_1, x_2, \dots, x_n \rangle \rightarrow \langle x'_1, x'_2, \dots, x'_m \rangle \text{ con } n > m.$$

Un esempio è il PCA (principal component analysis) dove i nuovi assi sono calcolati nella direzione di massima varianza dei dati



Un altro approccio per il preprocessing dei dati è la feature selection: è una particolare tecnica di riduzione della dimensionalità nella quale scelgo le variabili più significative per il task, anziché trasformarle. In pratica scegliamo un sottoinsieme di tutte le caratteristiche. Un ultimo approccio dal preprocessing che vediamo è Outlier detection: si cercano valori inusuali dei dati che non sono consistenti con gli altri (valori scorretti a causa di misurazioni errate).

11 Altri Task del Machine Learning

Altri task che non fanno parte del supervised o unsupervised learning.

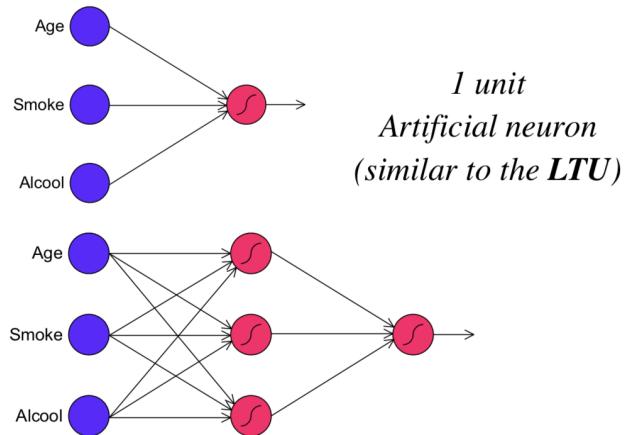
- Reinforcement Learning: si utilizza come metodo di adattamento per sistemi autonomi (in particolare in robotica). L'algoritmo apprende un criterio su come agire data un'osservazione del mondo. Ogni azione ha un certo impatto sull'ambiente e l'ambiente fornisce un feedback che guida l'algoritmo di apprendimento. Invece di avere una supervisione per ogni passaggio, abbiamo informazioni su vittorie / perdite per lo stato finale. Le azioni devono massimizzare la quantità di vittorie ricevute. L'apprendimento decide quali azioni sono state maggiormente responsabili di vittorie / perdite.
- Semi-Supervised Learning: combina esempi etichettati e non etichettati (in genere in numero maggiore) per generare una funzione o un classificatore appropriati.
- Learn to Rank: (utilizzato per i motori di ricerca) quando l'input è un insieme di oggetti e l'output desiderato è una classifica (un ranking) di tali oggetti.
- On-Line Learning: nuovi esempi sono imparati al momento
- Structured domain learning and relational learning: il dominio di input e output può essere strutturato sotto forma di sequenze (segnali, serie temporali, ...) o in modo più complesso: alberi, grafici, reti sociali.

12 Altri Modelli del Machine Learning

12.1 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. Questa nuova rappresentazione semplifica il compito di classificazione nell'ultimo livello della rete.

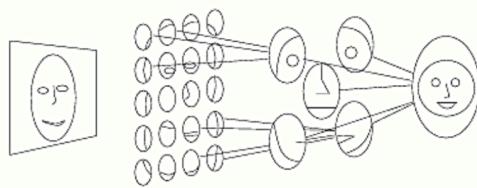
Abbiamo una espansione della base non lineare in w , e le ϕ sono imparate automaticamente, ma purtroppo questo scaturisce in un problema di ottimizzazione non lineare.



12.2 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.

Più tecnicamente, quando si parla di Deep Learning, si fa riferimento a reti neurali multistrato profonde, nel senso che abbiamo molteplici layer di unità di processing non lineari. Indipendentemente se si usa il Supervised o Unsupervised Learning per la rappresentazione di caratteristiche in ogni livello, i vari livelli vanno a formare una gerarchia di caratteristiche / rappresentazioni da basso livello ad alto livello (diversi livelli di astrazione). Imparare automaticamente come rappresentare i dati, e capire come classificarli aumenta il livello di astrazione tra i vari layer. Ad esempio, un'immagine può essere rappresentata in molti modi: come un vettore di pixel o in un modo più astratto come un insieme di bordi, regioni di forma particolare, ecc... Per questo il Deep Learning funziona meglio quando i dati in input hanno una sorta di struttura: spaziale, temporale, linguistica ecc...



I vantaggi del Deep Learning sono svariati, come ad esempio la possibilità di sfruttare la composizionalità della rappresentazione interna che porta ad un guadagno esponenziale nel potere di rappresentazione. I concetti più semplici vengono rappresentati in uno strato della rete, i quali poi possono essere sfruttati come dati primitivi dal livello successivo per rappresentare concetti più complessi. Sono necessari meno esempi per raggiungere una buona capacità di generalizzazione. Le Deep Networks erano difficili da addestrare in passato, ma combinando tecniche per l'addestramento di modelli grandi, HPC (ad esempio GPU) e grandi raccolte di dati da applicazioni reali (ad esempio milioni di immagini), al giorno d'oggi lavorano molto bene conseguendo una rivoluzione nell'approccio AI alle soluzioni del mondo reale.