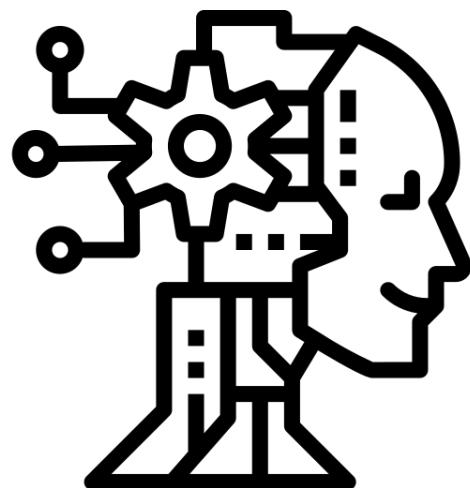


Intelligenza Artificiale Appunti

Andrea Mansi UniUD

II° Semestre 2020/2021
Big Data Analytics



Contents

1 Introduzione	5
1.1 Livello simbolico e subsimbolico	5
1.2 Concetto di Strong vs Weak AI	6
1.3 Approccio tradizionale vs AI	6
1.4 AI basata su conoscenza	7
1.5 Conoscenza vs Informazioni vs Dati	8
1.6 Quale conoscenza vogliamo rappresentare?	8
1.7 Architettura di un sistema intelligente knowledge-based	8
1.8 Knowledge Representation - Ipotesi di Smith	9
1.9 Concetto di entailment - IA vista come processo di inferenza	10
1.10 Alcuni criteri di valutazione di un sistema di KR	10
2 Metodi di KR	12
2.1 Logica	12
2.2 Reti semantiche	12
2.3 Tre concetti generali della KR	14
2.4 Grafi concettuali	16
2.5 Altri metodi di KR basati sulle Reti Semantiche	24
2.6 Alcuni problemi sul NLP	25
2.7 Frames	27
2.8 SCRIPTs	32
2.9 MOPs	34
2.10 Production Rules	35
2.11 Case-Based Reasoning	48
2.12 Prospettive evolutive di RBS, CBR e ML	51
3 Problem Solving	53
3.1 Definizioni e concetti base	53
3.2 Strategie basilari per la ricerca di una soluzione	56
3.3 Esempi:	60
4 DAI - Distributed Artificial Intelligence	66
4.1 Architetture multi-agente distribuite	66
4.2 Architetture per agenti intelligenti	68
5 Knowledge Modeling and Diagnostic Reasoning	73
5.1 Knowledge Modeling	74
5.2 Metodi AI per la diagnosi	76
5.3 Conceptual Modeling di un Dominio di Conoscenza	81
6 Ontologie	84

7 Machine Learning	85
7.1 Introduzione al ML	86
7.2 Association Rule learning e Frequent Item Set discovery (Unsupervised)	90
7.3 Inductive Learning Techniques	93
7.4 ML come approssimazione funzionale	96
7.5 Linear Regression e Gradient Descent - Introduzione	97
7.6 Logistic Regression [G.S.]	98
7.7 Gradient Descent [G.S.]	102
7.8 SVM - Support Vector Machine	105
7.9 Inductive Learning of Decision Trees/Rules	106
7.10 Clustering - Introduzione	107
8 Neural Networks	109
8.1 Neuroni e reti artificiali - basic concepts	109
8.2 Reti neurali artificiali - rappresentazione	114
8.3 Loss Functions	114
8.4 Multiclass Classification	115
9 Deep Learning	116
9.1 Tipologie di ANN per il DL	117
9.2 Convolutional Neural Networks	117
9.3 Esempio per la classificazione di immagini	118
9.4 Recurrent ANN:	121
9.5 Metodologia di sviluppo di un sistema ML	122
9.6 Valutazione della qualità di un sistema ML	123
9.7 Considerazioni finali sulle ANN	126
10 NLP & Text Mining - Applicazioni DL	127
10.1 Tecniche di base	127
10.2 Tecniche di base per la rappresentazione dei significati testuali	128

Informazioni sugli appunti

Questi appunti (non ufficiali) del corso di Intelligenza Artificiale (Prof. Carlo Tasso e Prof. Giuseppe Serra - Università degli studi di Udine) mirano a riassumere i principali concetti teorici trattati nel corso. Gli appunti sono basati sulle lezioni e sulle slide dei relativi professori.

TODO: Reinforcement Learning Section

1 Introduzione

Iniziamo analizzando due definizioni di **intelligenza**.

“ L'intelligenza è la capacità di acquisire e applicare conoscenza. La facoltà di pensiero e di ragionamento.

American Heritage Dictionary - 1970

“ L'intelligenza è la capacità cognitiva di un individuo di imparare dall'esperienza, di ragionare bene, di ricordarsi informazioni importanti e di affrontare gli obiettivi della vita giornaliera.

Sternberg - 1994

Una definizione di intelligenza artificiale di connotazione informatica può essere la seguente:

“ L'intelligenza artificiale è una disciplina della computer science che studia i metodi e le tecniche per la progettazione e la costruzione di sistemi capaci di fornire performance tipicamente considerate esclusive dell'intelligenza umana (comportamenti intelligenti).

1.1 Livello simbolico e subsimbolico

In ambito del ragionamento per l'acquisizione di conoscenza possiamo distinguere due livelli:

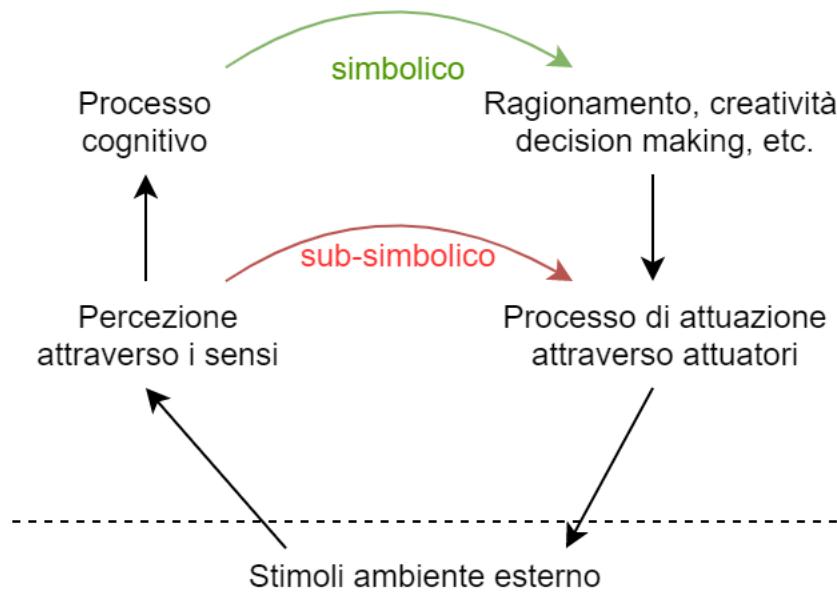
- **Simbolico:** esplicito, simbolicamente rappresentato, ragionamento reso esplicito, per intero, più facilmente spiegabile nel dettaglio, conoscenza rappresentata esplicitamente, adeguata ai livelli cognitivi superiori, ...

[ESEMPIO: un sistema esperto basato su KR esplicito]

- **Sub-Simbolico:** implicito, 'automatico', rappresentato da parametri numerici, difficile da spiegare, segue scorciatoie, collegamenti impliciti tra input e output (performance), più efficiente, adatto ai livelli 'più bassi', verso percettivo e motorio, come il modello 'stimolo-risposta', anche se applicato ad un 'alto' livello cognitivo, ...

[ESEMPIO: rete neurale]

Possiamo paragonare il livello simbolico al ragionamento che porta un essere umano all'acquisizione di conoscenza a partire da stimoli esterni, mentre il livello sub-simbolico è paragonabile agli automatismi della mente umana che portano al compimento di una azione in risposta a degli stimoli esterni, senza passare per la fase di processo cognitivo e/o ragionamento.



1.2 Concetto di Strong vs Weak AI

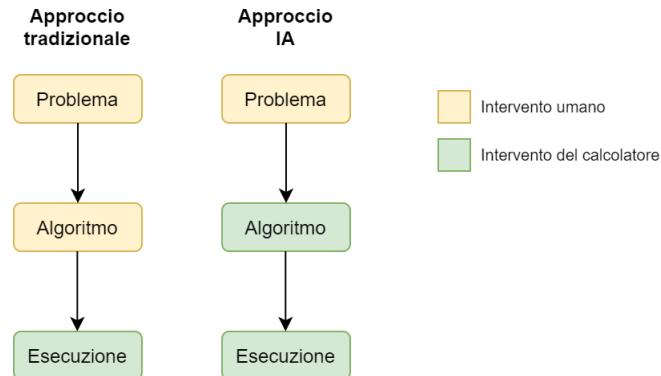
È importante distinguere il concetto di **strong** e **weak** AI. Si parla di:

- **Strong AI** quando si cerca di emulare il funzionamento e la struttura della mente umana all'interno del calcolatore;
- **Weak AI** quando si è interessati più al risultato e viene meno l'emulazione dei processi cognitivi umani.

Possiamo dire quindi che la strong AI segue un approccio simulativo, mentre la weak AI un approccio emulativo.

1.3 Approccio tradizionale vs AI

È importante fare chiarezza sulla principale differenza che distingue l'approccio tradizionale da quello basato sull'IA relativamente alla risoluzione dei problemi.



Come vediamo dallo schema, nell'approccio IA, è il calcolatore ad occuparsi della formulazione dell'algoritmo in grado di risolvere il problema. In questo approccio, l'IA si occupa

di simulare il processo cognitivo della mente umana nel trovare una soluzione (un algoritmo) per la risoluzione del problema di input (è capace di svolgere problem solving).

1.4 AI basata su conoscenza

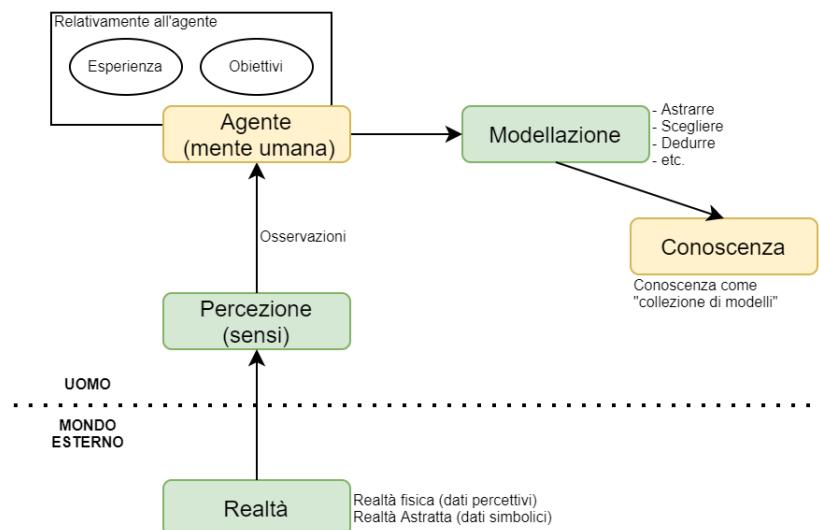
La conoscenza ha un ruolo **fondamentale** nell'ambito dell'intelligenza artificiale. Al fine di costruire un'intelligenza artificiale sono necessarie tre componenti/fasi principali.

- Acquisire conoscenza (KA - Knowledge Acquisition);
- Rappresentare conoscenza (KR - Knowledge Representation);
- Processare conoscenza (Knowledge Reasoning).

Forniamo una prima definizione di conoscenza:

“ La conoscenza ha origine nella mente di un individuo (lo stato mentale di avere idee, fatti, concetti, dati e tecniche, registrati nella memoria di un individuo) e si basa su informazioni che vengono trasformate e arricchite da esperienze, credenze e valori personali con significato decisionale e rilevante per l'azione. La conoscenza formata da un individuo potrebbe differire dalla conoscenza posseduta da un'altra persona che riceve le stesse informazioni. ”

Importante è interpretare il concetto di conoscenza anche come capacità di agire e ragionare (know-how): saper fare, agire efficacemente, saper risolvere problemi, prendere decisioni e scegliere fra alternative. Segue uno schema del processo di acquisizione di conoscenza.



1.5 Conoscenza vs Informazioni vs Dati

È importante distinguere questi tre concetti separatamente:

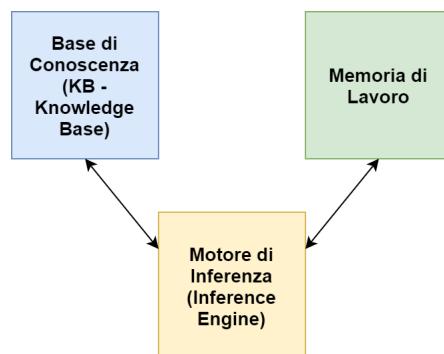
- **Dati:** descrizione oggettiva di un aspetto, di un fenomeno, di un evento, ecc., È un concetto oggettivo in quanto rappresenta un aspetto della realtà (fisica o astratta). [avere un dato significa conoscere un aspetto di un fenomeno. Metafora utile: vedere i dati come materia prima. I dati sono statici/passivi];
- **Informazioni:** è il risultato di un'attività di trattamento svolta su uno o più dati, significativa per il suo destinatario e utile ai suoi processi decisionali; da questa prospettiva, l'informazione ha un carattere più soggettivo (ciò che è informativo per te potrebbe non esserlo per me perché lo so già, perché non ne ho bisogno, ecc. - vedi ad esempio come due persone leggono un giornale in un modo diverso. [avere informazioni significa conoscere un aspetto di un fenomeno che non è necessariamente osservabile direttamente, e che richiede elaborazione. Nella metafora del "dato come materia prima", l'informazione è un "prodotto finito". Anche l'informazione, se vista isolatamente, può essere considerata come qualcosa di statico/passivo];
- **Conoscenza:** sono informazioni/dati contestualizzati all'interno di un processo di ragionamento (processo decisionale e, più in generale, problem solving). Avere conoscenza significa non solo essere informati su un determinato dominio, ma significa essere in grado di utilizzare le informazioni come parte di un processo di problem solving, significa 'sapere come fare', 'sapere cosa-fare', sapere come risolvere problemi (di ogni tipo), saper ragionare. [La conoscenza è un concetto più attivo/dinamico].

1.6 Quale conoscenza vogliamo rappresentare?

Concetti e relative relazioni, definizioni, fatti, descrizioni di eventi e procedure, processi, fatti incerti, euristiche, vincoli, ipotesi, regole di decisione, strategie, metodi di ragionamento (deduzione, induzione, generalizzazione, generazione di ipotesi, propagazione di vincoli, ragionamento per analogia, etc.) ...

1.7 Architettura di un sistema intelligente knowledge-based

Con il seguente schema si introduce l'architettura di un sistema di IA basato su conoscenza.



1.8 Knowledge Representation - Ipotesi di Smith

Il ruolo di come l'informazione/conoscenza vada rappresentata all'interno di un sistema intelligente ha un ruolo fondamentale nella ricerca in ambito dell'IA. Il campo che va sotto il nome di **Knowledge Representation (KR)** ha l'obiettivo di identificare e studiare linguaggi formali per questo obiettivo. Gli obiettivi della KR possono essere espressi facendo riferimento alla cosiddetta ipotesi di Brian Smith.

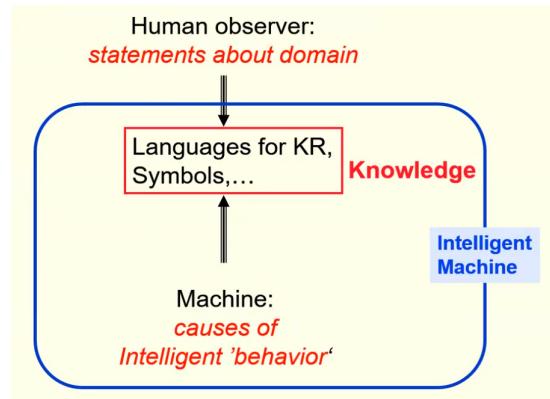
L'ipotesi della rappresentazione di conoscenza - Brian Smith

“ Any mechanically embodied intelligent process will be comprised of structural ingredients that:

- we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and
- independent of such an external semantical attribution, play a formal but causal and essential role in engendering the behavior that manifests that knowledge.

”

La cui rappresentazione grafica è la seguente:



Consideriamo una macchina intelligente che sia in grado di svolgere un processo intelligente X . Allora Smith afferma che: Tale macchina dovrà contenere delle strutture simboliche, di tipo 'linguistico', che in qualche modo rappresentano le conoscenze e le credenze (i.e., ciò che si crede) che si può affermare (o che noi uomini di solito possiamo pensare) caratterizzino/siano incluse nel processo X . Strutture che per noi costituiscono una rappresentazione mentale e che hanno un significato. Esiste poi un processo (tecnico) interno alla macchina che 'calcola' in base a queste rappresentazioni, ossia un processo che reagisce a queste strutture, al loro contenuto ed alla loro organizzazione, senza alcun riguardo a ciò che esse significano o rappresentano (per noi uomini), fornendo all'esterno il comportamento intelligente X .”

Possiamo riassumere il concetto di KR come segue:

“ La KR è la branca dell'IA che mira a definire linguaggi che permettano di formalizzare la conoscenza al fine di essere in grado di trarne informazioni utili (fare inferenza). Un sistema di KR è costituito da:

- Un **linguaggio di rappresentazione** con una sintassi per i simboli utilizzati, in grado di codificare le informazioni che devono essere rappresentate nel sistema software;
- Un **insieme di regole/operazioni/procedure** che permettono di manipolare le strutture sintattiche in accordo con il significato ad esse attribuito per fare inferenza.

”

1.9 Concetto di entailment - IA vista come processo di inferenza

Lo scopo dell'IA può essere visto come il processo di fare inferenza e ricavare informazioni/conoscenza nuova (entailment - deduzioni) a partire da conoscenza già acquisita. Siamo quindi non solo interessati a quello che è esplicitamente scritto nella base di conoscenza, ma anche a quello che è implicito, eseguendo dell'inferenza (conseguenza logica, deduzione, calcolo degli entailments).

Ricavare informazioni esplicitamente archiviate nella base di conoscenza può essere ricondotto all'approccio database, mentre ricavare informazioni implicite dalla base di conoscenza tramite il calcolo di deduzioni può essere ricondotto all'approccio IA.

Uno strumento per eseguire inferenza è la logica formale, utilizzabile per il calcolo delle deduzioni/entailments (ad esempio con il primo ordine logico - FOL). Esempio:

- **Base di conoscenza:** tutti gli uomini sono mortali, Socrate è un uomo (esplicito);
- **Entailment:** Socrate è mortale (implicito).

Abbiamo calcolato un entailment (nuova conoscenza) a partire dalla nostra base di conoscenza tramite il calcolo di un predicato (FOL). Il FOL tuttavia è indecidibile (non esiste una procedura generale di derivazione) ed è NP-Completo qualora termini.

1.10 Alcuni criteri di valutazione di un sistema di KR

Il primo criterio di valutazione è chiamato **adeguatezza espressiva**: il sistema KR esprime adeguatamente tutti gli aspetti che sono importanti? con la giusta granularità? distingue gli elementi che sono distinti nella realtà?

Un altro criterio è l'**adeguatezza inferenziale**: permette di eseguire le inferenze necessarie? Si ripresenta il problema della contrapposizione, o meglio, del continuum fra l'atteggiamento dell'IA debole e dell'IA forte! Emulazione vs. Simulazione ...

Un ulteriore criterio è l'**adeguatezza/plausibilità cognitiva**: quanto è "vicino" al ragionamento umano, rispecchia le modalità simboliche umane? c'è corrispondenza con il modo della mente umana di concepire la conoscenza?

In ambito informatico, un criterio importante è quello dell'**efficienza** del motore inferenziale in termini di spazio e tempo. In termini cognitivi è efficiente? (è in grado di riprodurre le scorciatorie sub-simboliche della mente umana?).

Una delle decisioni più importanti e difficili quando si progetta un sistema di intelligenza artificiale con conoscenza esplicita, è determinare quale debba essere la forza espressiva del metodo KR utilizzato: più espressiva sarà, più facile e più precisa sarà la rappresentazione di una data entità, tuttavia una maggiore espressività risulterà anche in un maggior numero di entità rappresentate e, di conseguenza, sarà più difficile derivare automaticamente inferenze. Si parla in questo caso di "trade-off" tra espressività e efficienza.

2 Metodi di KR

2.1 Logica

Perchè viene utilizzata la logica come metodo di KR? Il linguaggio naturale sarebbe ideale ma è troppo complesso. La logica ignora la struttura fine di una frase (bassa granularità) ed è un buon trade-off.

Il **calcolo proposizionale** è semplice ma non abbastanza potente: il ragionamento è limitato, in quanto consente solo di costruire tabelle di verità che elencano tutti i possibili valori di una espressione.

Il **calcolo dei predicati (FOL)** aggiunge i predicati, variabili individuali e quantificatori: è più potente. Il FOL però è difficile da leggere, in particolare per chi non lo conosce.

In generale usare la logica come metodo di KR è un metodo elegante e semplice (ma può richiedere strutture poco leggibili per frasi complesse), è eseguibile, computazionalmente poco efficiente, è uno standard per il KR.

2.2 Reti semantiche

È un meccanismo molto generale per la rappresentazione della conoscenza che sta alla base di molti approcci specifici, come i grafici concettuali, ontologie, mappe concettuali, mappe tematiche, reti di dipendenza concettuale, etc., e linguaggi, come RDF, OWL, etc. È stato introdotto per la prima volta nel 1968 da Ross Quillian, per modellare la natura associativa della memoria umana.

Con **associativo** ci si riferisce al fatto che la nostra memoria è in grado di rappresentare singole entità (singole, unitarie, non strutturate) che si associano tramite specifiche relazioni tra loro.

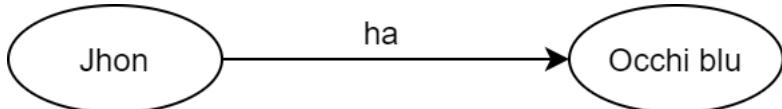
Sono state introdotte diverse tipologie di reti semantiche per rappresentare il significato delle frasi in linguaggio naturale, piuttosto che i processi di risoluzione dei problemi. Alcuni tipi di conoscenza utilizzati per l'elaborazione del linguaggio naturale:

- **Lessico:** riguardante la costruzione delle singole parole (singolare/plurale, coniugazione, etc.);
- **Sintassi:** riguardante la costruzione e la strutturazione di frasi mediante l'unione e/o combinazione di più parole;
- **Semantica:** riguardante la rappresentazione dei significati (oggetti, concetti, azioni, eventi, proprietà, relazioni, etc.) riferiti dalle parole in una frase;
- **Pragmatica:** su come le persone comunicano i significati e come e perché (con quale scopo) producono frasi adeguate al contesto. Comprende aspetti sociolinguistici/culturali.

Definizione: Le reti semantiche sono meccanismi KR in grado di rappresentare singoli concetti e relazioni/associazioni tra loro tramite un insieme di nodi collegati tra loro da archi.

- Nodi: oggetti, concetti, azioni, eventi, proprietà, etc.
- Archi: relazioni tra i nodi.

Esempio:



Importante è la relazione **is-a**: una relazione che permette la riduzione della ridondanza della rappresentazione.

Il processo di passare da testo in linguaggio naturale a una rete semantica è detto "Interpretazione/Understanding". Il passaggio di passare dalla rete semantica al testo in linguaggio naturale è detto "Generazione".

Come si ragiona sulle reti semantiche? i principali algoritmi di ragionamento sono:

- **Ereditarietà:** un nodo eredita tutte le proprietà dei nodi "più generali" della rete;
- **Ricerca di intersezioni:** identificazione delle relazioni tra due concetti. Ottenuto mediante una ricerca di percorsi nella rete (mediante attivazione per diffusione) che collegano i due nodi.

Ipotesi del mondo chiuso

L'ipotesi cita: "conosco solo ciò che è rappresentato, esplicitamente o implicitamente, nella rappresentazione contenuta nella KB".

In altri termini, in un sistema intelligente, non posso aspettarmi di ottenere inferenze (entailments) su entità o proprietà che non sono (esplicitamente o implicitamente) rappresentate.

Le reti semantiche sono molto generali e di natura associativa/dichiarativa. Hanno una elevata adeguatezza cognitiva, sono flessibili ed estensibili. Permettono definizioni intensionali e algoritmi di ragionamento ad-hoc. Tuttavia gli algoritmi possono essere inefficienti, l'eredità può causare lo spread delle definizioni (bassa località).

2.2.1 Tipologia di conoscenza adatta alle reti semantiche

Conoscenza descrittiva, conoscenza associativa o relazionale (concettuale), tassonomie ben stabilite (is-a, has-part, ...). Si concentra più sulla conoscenza descrittiva piuttosto che sul reasoning.

2.2.2 Pro e Contro

Pro: generale, dichiarativa, di natura associativa, buona adeguatezza cognitiva, flessibile, estendibile, permette definizioni estensionali ed intensionali, permette algoritmi di reasoning ad-hoc.

Contro: fase di modellazione complessa, eccezioni, ereditarietà multipla, distinzione tra classi e istanze, non troppo efficiente, rappresentazione "sparsa".

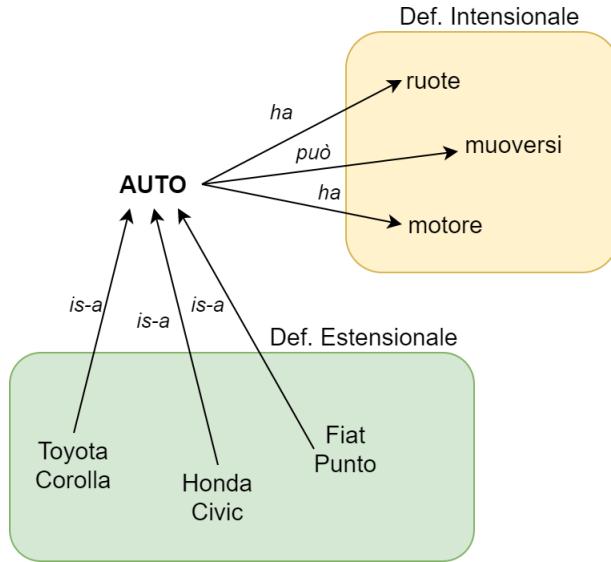
2.3 Tre concetti generali della KR

Illustriamo tre concetti generali applicabili ai metodi di KR.

2.3.1 Definizioni intensionali e estensionali dei concetti

Intensionale: insieme di proprietà che comportano l'appartenenza (condizioni necessarie e sufficienti) per dire che l'oggetto appartiene a quella classe. **Estensionale:** elenco di esempi di istanze della classe.

Segue un esempio:



Intensionale:

- più generale (nuovi oggetti possono essere riconosciuti);
- computazionalmente più costosa;
- richiede più conoscenza (proprietà, etc.).

Estensionale:

- meno generale (solo gli oggetti listati possono essere riconosciuti);
- più efficiente;
- cognitivamente più semplice (richiede solo pattern matching).

2.3.2 Distinzione type-token

In molte reti semantiche si fa una distinzione tra **concetti generali** (chiamati **tipi**) e **concetti specifici** (chiamati **token**, rappresentati con qualche tipo di marchio).

Di solito in questo caso, ci sono due tipologie di nodi relazione is-a:

- is-a: che rappresenta la relazione di inclusione di una classe in un'altra;
- instance-of (o a-k-o - a kind of): che rappresenta l'appartenenza di un token a una classe.

2.3.3 Concettualizzazione del dominio

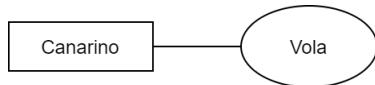
È il processo dedicato alla scelta di "cosa includere del mondo nella rappresentazione che inseriremo nel sistema", quali concetti e istanze (generali), quali proprietà, quali relazioni, etc. La concettualizzazione si fa analizzando il dominio di conoscenza e selezionando/definendo oggetti, proprietà, funzioni e relazioni. La concettualizzazione è un'attività di modellazione.

2.4 Grafi concettuali

I grafi concettuali (CG) (Sowa, 1984; Chein & Mugnier, 2009; Syandard ISO 2007) sono un metodo KR che appartiene alla classe delle reti semantiche. Sowa aveva l'obiettivo di rappresentare il significato dei testi in linguaggio naturale. Più specificamente, la struttura di un grafo concettuale è costituita da un grafo finito, connesso e bipartito. In un CG ci sono due tipi di nodi che rappresentano rispettivamente:

1. concetti;
2. relazioni concettuali.

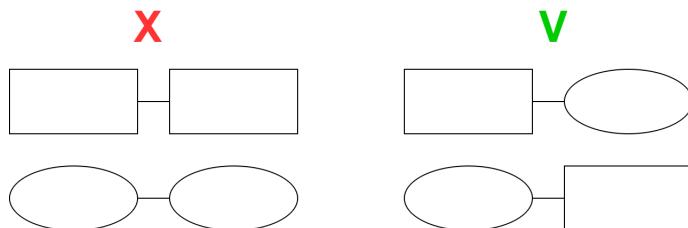
Gli archi (senza frecce) non sono etichettati e possono collegare solo concetti con relazioni concettuali o viceversa. Ogni grafo concettuale è un'asserzione riguardante gli individui che esistono nel dominio considerato. Una base di conoscenza contiene diversi CG, gerarchie di tipi e relazioni, e un catalogo di individui. Di solito, nella rappresentazione grafica, i concetti sono rappresentati da rettangoli e le relazioni da ellissi (o ovali).



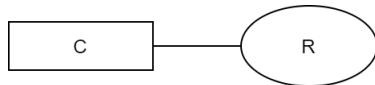
Una rappresentazione analoga, in "forma lineare" è la seguente: [canarino]–(vola)

Archi

Gli archi non possono collegare relazioni con relazioni o concetti con concetti.

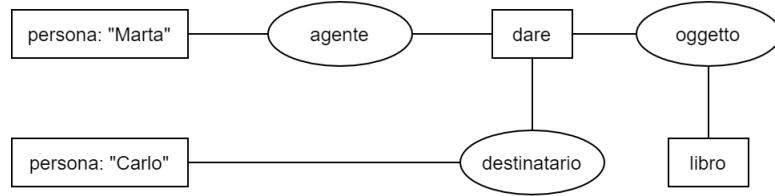


Diciamo che l'arco "appartiene alla relazione" e che "collega la relazione R a C".



In generale, alle relazioni dovrebbe essere assegnato un ruolo astratto/"meta". Il contenuto specifico dell'asserzione è rappresentato principalmente attraverso i nodi concettuali.

Esempio: un esempio completo



2.4.1 Gerarchie dei tipi

I CG supportano il ragionamento classificatorio. I tipi di concetti rappresentano classi di entità, attributi, stati, eventi, etc.

Sull'insieme dei tipi possiamo definire una relazione di ordine parziale " $<$ " ; questa relazione è chiamata gerarchia dei tipi. Se t e s sono due tipi, e $t < s$, allora diciamo che t è sottotipo di s (o, viceversa, che s è un supertipo di t), e ciò significa che ogni elemento di t è anche un elemento di s .

Per esempio: Gatto $<$ Mammifero $<$ Animali $<$ Oggetto fisico

In un sistema specifico che utilizza i CG, viene anche definito un insieme di tipi.

La gerarchia dei tipi è rappresentata da un reticolato, la struttura tipica dell'ereditarietà multipla. Un tipo può avere più di un supertipo e più di un sottotipo. Tuttavia, ogni coppia di tipi ha un minimo comune supertipo e un massimo comune sottotipo.

Consideriamo per esempio il caso:

Lepre $<$ Animale selvatico

Pantera $<$ Animale selvatico

Lepre $<$ Mammifero

Pantera $<$ Mammifero

Per mantenere la struttura reticolare è richiesta la considerazione di un nuovo tipo, per esempio MammiferoSelvatico. Ma poiché questa operazione non è sempre naturale, vengono introdotti due ulteriori tipi nella gerarchia dei tipi, il tipo "universale" e il tipo "absurde", che sono rispettivamente supertipo di tutti i tipi e sottotipo di tutti i tipi.

2.4.2 Tipi e referenti

In un CG, ogni concetto rappresenta un individuo specifico di un dato tipo: Un concetto è l'istanza di un dato tipo.

Ogni nodo rappresentato da un rettangolo corrisponde a un concetto individuale ed è etichettato con il tipo del concetto (al quale il concetto appartiene).

In ogni CG, ogni concetto ha un tipo e un referente. Il referente identifica l'individuo specifico a cui si fa riferimento.

La specificazione del "tipo" è rappresentata sul lato sinistro e il "referente" sul lato destro.

Il campo referente può essere vuoto.

Un individuo specifico X può essere rappresentato nel campo referente aggiungendo la specifica dell'individuo X a destra della specifica del tipo.

Per esempio:

Gatto: "Silvestro"

indica l'individuo Silvestro, appartenente all'insieme di tutti i gatti.

Se l'individuo non è specificato, allora il concetto rappresenta un individuo non specificato di quel tipo (\exists , cioè quantificazione esistenziale).

Un referente può anche riferirsi a valori numerici (per esempio 3,1415...) o a insiemi esplicitamente definiti (per esempio, {Tokyo, Toledo, Roma}).

Referente: Il referente indica a quale concetto specifico (singolo) si fa riferimento, o quale insieme di concetti, o quanti, o il fatto che il concetto referenziato è un concetto complesso, che è rappresentato per mezzo di un intero CG.

Il referente di un concetto può essere specificato da:

- **Quantificatore:** esistenziale o universale o definito per riferire quantità o quantità (numero)
- **Designatore:** che specifica il referente, per esempio con un letterale (un numero o una stringa), o un nome (che identifica un individuo), un marcitore individuale (concetto unico nel catalogo degli individui), o indefinito
- **Descrittore:** un grafico concettuale (possibilmente vuoto), che descrive il referente

Descriptori

Un CG "nested" nel campo referente, ad esempio:

[Proposition:[Man]-(Agent)-[eat]]

Designatori

Servono a specificare chi, cosa etc. Esempi:

Name: "Boston", "Mario Rossi"

È possibile riferirsi a specifici individui, senza considerare il loro nome, ma utilizzando gli individual markers. I marker sono rappresentati dal simbolo # seguito da un numero. Identificano univocamente un individuo.

Individual Marker: [Canary:#56]

[Canary:Twitty#56]

Significa che un canarino di nome "Twitty" è identificato da 56.

Tramite markers possiamo distinguere tra gli individui e i suoi nomi. Il nome può essere rappresentato da una esplicita relazione, come ad esempio:

[Tweety]-(name)-[Canary:#56]-(color)-[yellow]

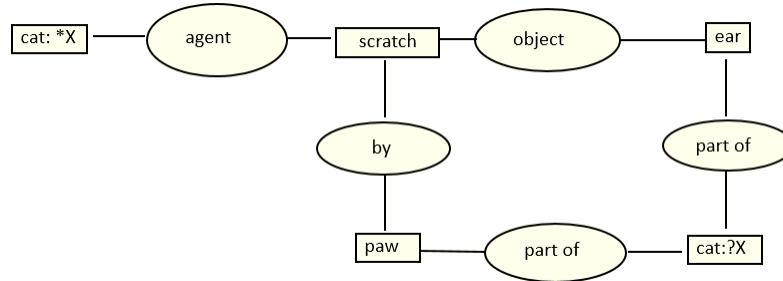
La distinzione tra individuo e nome è una feature che aumenta l'espressività della rappresentazione delle CGs, permettendo di rappresentare conoscenza del tipo:

"Marta è un nome da femmina"

È possibile specificare un generico individuo con il marker generico * che per convenzione si omette: canary*

Quando in più di un nodo è necessario riferirsi allo stesso individuo senza specificare il nome, si possono usare variabili, rappresentate da un asterisco seguito dalla variabile di nome *X. È possibile usare variabili per riferirsi a insiemi.

Variable: [Canary: *X]



Il gatto gratta il suo orecchio con la sua zampa o la sua zampa con l'orecchio?

L'etichetta [cat:*X] definisce la variabile "Un gatto" e ?X fa riferimento a *X, già definita.

Collezioni: insiemi di entità specificate come {a,b,c} o non specificati {*}.

Entrambi sono detti insiemi plurali.

Ad esempio: [Bird:{*}] significa "Ci sono degli uccelli", o semplicemente "Uccelli".

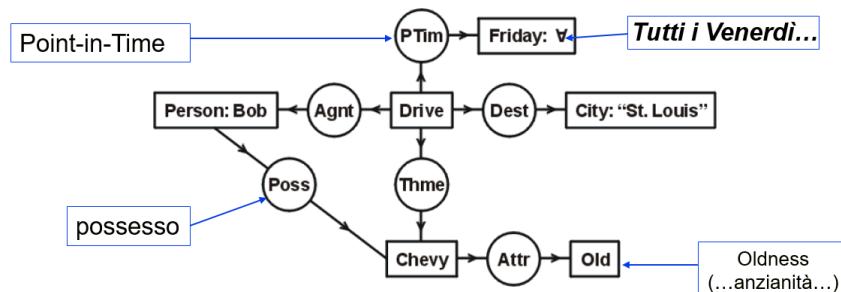
[Bird:{*}@n *X] significa "C'è un insieme (che posso referenziare con ?X) di n uccelli".
Potremmo limitare n con [integer:n]-(>=)-[integer:5]

Quantificatori

Quantificatore universale, etc. come ad esempio:

[Cat: ∀]-[On]-[Balcony]

Esempio completo: (vecchia sintassi con le frecce! non si usano, ma si usavano nelle prime versioni).



@ seguito da n indica una quantità di entità di un tipo.

[Person]-(has part)-[Leg:@2]

Il cui significato è: esiste una persona con due gambe.

Oppure:

[Guest; {*} @40]-(Agent)-[Give]-(Recpt)-[Person:Alfred]-(Thme)-[Gift:{*}@37*X]

Il cui significato è: i 40 ospiti hanno dato 37 regali ad alfred (l'insieme dei 37 regali può essere referenziato dalla variabile *X).

[Number:18] c'è un numero, il 18

[Number:@18] ci sono 18 numeri

[Number:@18 18] ci sono 18 numeri 18

[Type:{*}@n] insieme di n (non specificato) elementi del tipo Type

Possono venir applicati vincoli su n, ad esempio n maggiore di 30.

[Integer:n]-(>)-[Integer:30]

2.4.3 Relazioni Concettuali

Le relazioni concettuali codificano il ruolo astratto che ogni concetto ha in relazione ad un altro concetto. Esempi tipici sono: attributo, agente, oggetto, modalità, medium, possessione, target, causa, tempo, verbo, locazione, etc.

Una relazione può essere:

- **unaria:** ad esempio: (past)-[proposizione:...]
- **binaria:** ad esempio [go]-(tense)-[time-interval: T1]
- **ternaria, quaternaria, etc.**

Valenza: indica il numero di archi che appartengono/caratterizzano una relazione - analogamente al concetto di arità di un predicato.

Firma/Signature: Ogni relazione è vincolata in rispetto al tipo dei concetti con cui si può collegare. Ad esempio, la relazione agente, può collegare due concetti, uno del tipo BeAnimated e uno del tipo Act. La sequenza BeAnimated e Act è la signature della relazione e rappresenta un vincolo sul tipo dei concetti che possono essere collegati della rel. Agent.

Esempio: Between < entity, entity, entity >

[Entity]-(Between)-1-[entity]-2-[entity]

2.4.4 Basi canoniche

I grafi concettuali rappresentano asserzioni su un specifico dominio (a livello asserzionale o dei token). C'è però anche il livello terminologico (o livello dei tipi, che è un altro livello di conoscenza più astratto), riguardante:

- la descrizione di tipi e concetti tipici, la specificazione di informazione di default, di concetti, firme, restrizioni riguardanti l'uso dei concetti, definizioni di nuovi tipi e concetti, etc.

Questa informazione è rappresentata mediante:

- grafi di definizione, per definire nuovi tipi e relazioni
- grafi canonici, che forniscono una descrizione di base del tipo di un dominio specifico

Lo standard proposto è quello delle lambda-expressions. L'insieme di questi grafi di definizione è chiamato insieme canonica del dominio. Operazioni sui grafi sono usate per derivare nuovi grafi dalla base canonica.

Estendere la gerarchia dei tipi: iniziando da Absurd e Universal più un insieme di tipi primitivi, nuovi tipi possono venir definiti. Con le espressioni lambda possiamo definire nuovi tipi e possiamo specificare dove i nuovi tipi sono localizzati nella gerarchia.

Esempio: definiamo il concetto di MaineFarmer con una lambda expr.

MaineFarmer = [Farmer: λ]-[location]-[State:Maine]

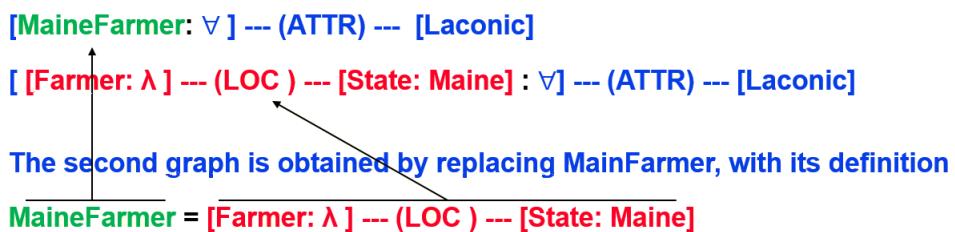
Entità del tipo appena definito sono ristrette al tipo farmer e λ è il parametro della definizione. In altre parole, il tipo di λ è un supertipo del nuovo tipo definito.

Ovvero: Farmer > MaineFarmer

Alternativamente, la notazione originale (grafo di definizione):

Type MaineFarmer (*X) is [Farmer:?X]-[location]-[State:Maine]

La definizione e il nuovo tipo definito sono intercambiabili. I due seguenti grafi sono equivalenti:

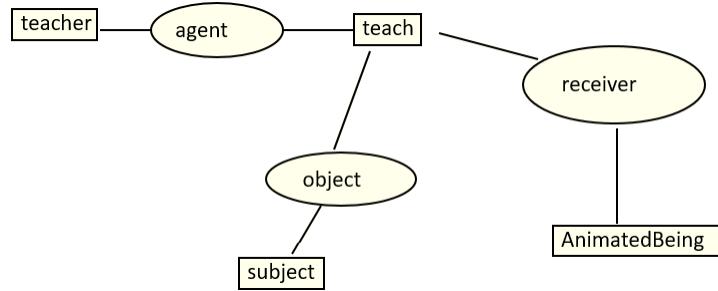


2.4.5 Grafi canonici

Un grafo canonico rappresenta il pattern base della rappresentazione di un concetto (o una relazione concettuale), e fornisce vincoli sui possibili concetti (o relazioni) ad esso collegabili (idea simile al concetto di firma/signature delle relazioni).

I grafi canonici sono una tipica rappresentazione terminologica e definizione intensionale di concetti.

Esempio:



2.4.6 Gerarchia delle relazioni

Si può definire una gerarchia anche tra relazioni. Due relazioni sono correlate come tipo/-sottotipo se hanno la stessa valenza (arietà) e la signature è consistente.

Ad esempio:

[AnimatedBeing]-(go)-[Location]
 [AnimatedBeing]-(go)-[FriulianLocation]

2.4.7 Definire relazioni tramite lambda espressioni

Partiamo con un esempio: definiamo la relazione Going-To che verrà definita tra "una persona che va" $\lambda 1$ e una "destinazione" $\lambda 2$

[Relation:GoingTo]-(Def)-[LambdaExpr.: [Person: $\lambda 1$]-(Agnt)-[Go]-(Dest)-[City: $\lambda 2$]]

Spiegazione: la relazione GoingTo è definita da una lambda espressione che associa a una persona $\lambda 1$, che è l'agente del concetto [Go], a una città $\lambda 2$ che è la destinazione di [Go].

Il parametro λ (equivalente a $*X$) rappresenta un parametro formale, e quindi è un placeholder per ciò che rimpiazzerà quando verrà usata la definizione.

Un altro esempio: [boy:?X]-(son)-[person:*]-(son)-[person:?Y]
 ovvero la relazione brother($*x, *y$) di fratellanza

2.4.8 Operazioni sui grafi concettuali

Le operazioni sui grafi rappresentano le regole canoniche per generare nuovi grafi derivabili dalla base canonica. In questo senso, sono anche chiamati regole di formazione canonica.

Sono "funzionali", nel senso che prendono in input uno o più CG e producono in output uno o più nuovi CG.

Le operazioni sui grafi non garantiscono il mantenimento della verità, ma solo che i grafi ottenuti rappresentino della conoscenza (non per forza plausibile o corretta nel dominio considerato).

6 operazioni:

- **copy** e **simplification**: trasformano un CG in un CG equivalente. Copy esegue solo la copia mentre simplification semplifica togliendo le parti ridondanti.
- **restrict** e **unrestrict**: la prima rende un CG più specializzato, mentre unrestrict produce un CG più generale.
- **join** e **detatch**: la prima trasforma due CG in un solo CG specializzato, mentre detatch esegue l'opposto: separa un CG in due o più CG più generali.

Join (specializzazioni) e **restrict** (generalizzazioni) sono operazioni che riducono la generalità. Un grafo derivato canonicamente da altri grafi è detto una specializzazione del grafo originale.

La relazione di specializzazione definisce un ordine parziale tra i grafi derivati dalla base canonica. Se il grafo g è una specializzazione del grafo f , allora diciamo che il grafo f è una generalizzazione del grafo g .

La generalizzazione è ottenuta con le operazioni unrestrict e detach.

La gerarchia delle generalizzazioni gioca un ruolo chiave in numerose tecniche di inferenza e di reasoning classificatorio: sono la base per l'ereditarietà e il "common sense reasoning".

Si noti come la relazione di generalizzazione è "truth-preserving".

Di solito le operazioni sono utilizzate in combinazione.

Combinando restrizioni e join possiamo ottenere l'ereditarietà: per mezzo della restrizione possiamo trasformare un grafico e quindi applicare il join. La sostituzione di un marker generico con individui porta l'ereditarietà di un individuo dal suo tipo.

Allo stesso modo, possiamo avere la sostituzione di un tipo con un sottotipo, di un tipo con la sua definizione, di una relazione con la sua definizione, ecc.

Se una situazione stereotipata nella base canonica è descritta da "molti" dettagli, possiamo apportare assunzioni plausibili su situazioni specifiche. Ciò si ottiene rappresentando la situazione specifica (tipicamente nota da una serie di dettagli inferiori a quella che caratterizza più generalmente nella base canonica) da un grafico e cercando di utilizzare il join (possibilmente usando anche restrizioni) tra questo grafico e grafici nella base canonica.

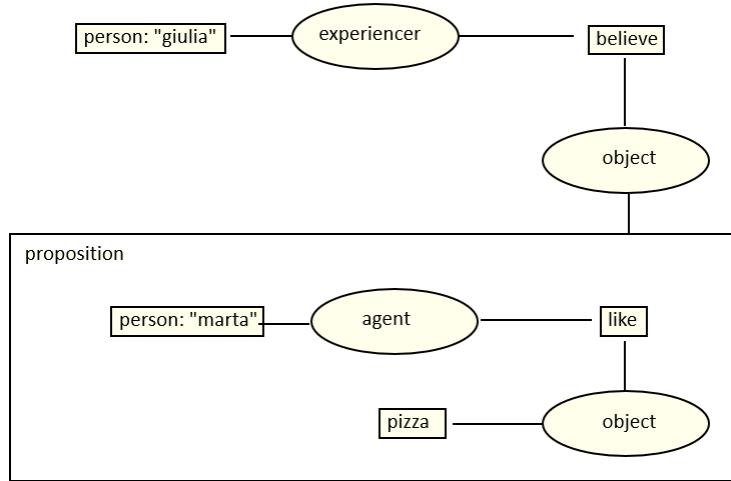
Nodi proposizionali

Fino ad ora abbiamo visto grafi concettuali per rappresentare affermazioni/proposizioni su oggetti del dominio. I CG possono essere utilizzati anche per rappresentare relazioni tra proposizioni. Per fare ciò, utilizziamo il nuovo tipo "proposizione" che ha un grafo come referenti.

Una proposizione asserisce la congiunzione del grafo a cui si riferisce. Graficamente una proposizione è indicata da un rettangolo che contiene il grafo a cui si riferisce.

Esempio:

Negazione La negazione è ottenuta da una relazione unaria "not" che prende come argomenti un nodo proposizionale.



2.4.9 Grafi concettuali e calcolo dei predicati

La potenza espressiva dei grafi concettuali è equivalente a quella del FOL. Ogni cosa rappresentabile dal calcolo dei predicati può essere rappresentato dai grafi concettuali, e viceversa.

2.4.10 Considerazioni finali

- Applicazioni: i CG sono adeguati per rappresentare queries su database come schemi di database. Sono adeguati a rappresentare il significato delle frasi in linguaggio naturale. Sono utilizzati come metalinguaggio/linguaggio intermedio per la rappresentazione della conoscenza (ad esempio per rappresentare modelli concettuali, etc.).
- Valutazione: i CG sono considerati flessibili. Sono equivalenti al FOL (calcolo dei predicati del primo ordine). I CG sono molto più espressivi e potenti del formalismo entità-relazione. Non sono adeguati per la rappresentazione di operazioni e relazioni matematiche complesse.

2.5 Altri metodi di KR basati sulle Reti Semantiche

Molti autori, oltre a Sowa, proposero vari metodi per rappresentare reti semantiche: tutti sono basati sulla rappresentazione di concetti singoli (nodi) il cui significato è espresso per mezzo dei nodi a cui sono collegati e il tipo di archi sfruttati nella connessione.

2.5.1 Dipendenza Concettuale - Roger C. Schank

Principio base: date due sentenze il cui significato è equivalente, ci dovrebbe essere solo una rappresentazione di tale conoscenza, anche se le due sentenze hanno parole diverse. Il concetto si basa su un insieme finito di primitive semantiche che dovrebbero essere capaci di rappresentare ogni tipologia di significato.

Ci sono 11(+1) azioni primitive, 8 stati base, 7(+1) relazioni concettuali.

2.6 Alcuni problemi sul NLP

È molto difficile identificare la "rappresentazione interna" del significato di frasi in linguaggio naturale. Questo è principalmente causato dalla proprietà del LN di essere ambiguo. Prima di affrontare il tema rivediamo alcune definizioni/concetti:

- **Lessico:** concerne la costruzione delle singole parole (singolare, plurale, coniugazione etc.);
- **Sintassi:** concerne la costruzione e la strutturazione di frasi di significato dall'unione e combinazione di più parole;
- **Semantica:** concerne la rappresentazione del significato (oggetti, concetti, azioni, eventi, proprietà, etc.) riferito da parole nella frase;
- **Pragmatica:** concerne come le persone comunicano il significato e come o perché producono frasi appropriate nel contesto. Include gli aspetti sociolinguistici.

Il NLP è stato considerato come consistente di più livelli. I livelli sono:

- **Analisi lessicale:** analisi della forma delle parole;
- **Analisi sintattica:** processamento della struttura;
- **Analisi semantica:** rappresentazione del significato;
- **Analisi del discorso:** processamento di frasi combinate;
- **Analisi pragmatica:** l'intento dell'uso delle frasi nel contesto.

L'ambiguità può avvenire in tutti questi livelli. Esempi:

Ambiguità lessicale: ambiguità di una singola parola. Una parola può essere ambigua rispetto alla sua categoria lessicale (nome, verbo, aggettivo, etc.).

Ad esempio: la parola **silver** può essere usata come nome, aggettivo o verbo:

- She bagged two silver medals (nome)
- She made a silver speech (aggettivo)
- His worries had silvered his air (verbo)

L'ambiguità lessicale può essere risolta dalla disambiguazione della categoria lessicale, ad esempio: parts-of-speech tagging (POS-tagging).

Ambiguità semantico-lessicale (polisemia): avviene quando una singola parola è associata a più significati (polisemia).

Ad esempio: John went to the bank. (riva o banca?)

L'ambiguità semantico-lessicale può essere risolta con tecniche di disambiguazione del significato di una parola (WSD), dove WSD mira ad assegnare automaticamente il significato della parola in base al contesto.

Ambiguità sintattica di scope: l'ambiguità di scope coinvolge operatori e quantificatori.

Ad esempio: Old men and women were taken to safe locations.

Old è applicato solo a men o anche a women?

Lo scope dei quantificatori non sempre è chiaro e crea ambiguità.

Esempio: Every man loves a woman.

Tutti gli uomini amano una donna diversa o tutti la stessa?

Ambiguità sintattica di attaccamento: una frase ha ambiguità sintattica di attaccamento se un costituente si adatta a più di una posizione nell'albero di parsing. Questo tipo di ambiguità nasce dall'incertezza di attaccare una frase o clausola a una parte della frase.

Ad esempio: The man saw the girl with the telescope on the hill.

La donna aveva il telescopio o l'uomo? "with" a chi è "attaccato"?

Ambiguità semantica: avviene quando il significato delle parole stesse può essere ambiguo. Anche dopo che la sintassi e il significato della parola è stato risolto ci sono molti modi di interpretare una frase.

Ad esempio:

The car hit the pole while it was moving

Chi si muoveva? L'ambiguità semantica tipicamente emerge dal fatto che generalmente un calcolatore non è in grado di distinguere ciò che è logico da quello che non lo è. Questo è possibile solo tramite tecniche di disambiguazione semantica.

La disambiguità semantica avviene anche quando una sentenza contiene una frase o parola ambigua. Può essere quindi la conseguenza di altre tipologie di disambiguità, come quella semantica, lessicale, sintattica, etc.

Ad esempio: We saw his duck

duck può riferirsi all'animale di una persona o il suo movimento (nuotare sott'acqua).

Ambiguità anaforica: le anafore sono le entità che sono state precedentemente introdotte nel discorso. Può esserci ambiguità nell'identificazione delle entità riferite nell'anafora.

Ad esempio: The horse ran up the hill. It was very steep. It soon got tired.

La referenza anaforica di "it" nei due casi causa ambiguità. Ovviamente un essere umano capisce facilmente a chi si riferisce "it" ma per un calcolatore senza conoscenze di background è un task difficile. Sono necessarie informazioni di contesto e conoscenza/senso di base (common sense).

Ambiguità pragmatica: si riferisce alla situazione dove il contesto di una frase fornisce interpretazioni multiple. Una delle più difficili task del NLP. Il problema coinvolge il processamento dell'intenzione dello user, sentimento, credenze, etc.

Ad esempio: Tourist (checking out of the hotel): "Waiter, go upstairs to my room and see if my sandals are there, do not be late; I have to catch the train in 15 minutes"

Waiter (running upstairs and coming back): "Yes sir, they are there"

Chiaramente la cameriera non ha capito la pragmatica della situazione, ignorando quale fosse il goal della domanda.

Dobbiamo quindi riconoscere obiettivi e piani dello speaker in un dialogo. L'ambiguità pragmatica nasce quando la frase non è specifica e il contesto non fornisce le informazioni necessarie a chiarificare la frase. Mancano informazioni che vanno ricavate.

2.7 Frames

I frames sono un meccanismo di KR proposto nel 1975 da Minsky. Possono essere usati per rappresentare oggetti, eventi, qualsiasi astrazione o entità strutturata. Nello stesso periodo sono stati proposti da Roger Schank gli scripts e poi i MOPs, per rappresentare eventi strutturati stereotipati. Da queste proposte si sono poi sviluppate altre idee, come i CBR e l'idea di oggetti nell'OOP.

I frames rappresentano situazioni generali riguardo entità, luoghi, oggetti, etc. strutturati e possibilmente stereotipati.

Con stereotipati si intende che sono tipici, occorrono spesso con quella struttura e elementi e quindi, un frame fornisce informazioni su cosa aspettarsi: quali dati, oggetti, eventi etc.

I frame combinano conoscenza dichiarativa e procedurale.

Esempio di un frame per rappresentare l'albero genealogico di Adam:

sex: Male

spouse: Beth

child: (Charles, Donna, Elle)

sex, spouse e child sono detti slot (possono essere riempiti da uno o più valori).

Esempio di conoscenza stereotipata: se andiamo al ristorante attiveremo il suo frame che ci aiuta a controllare la situazione fornendo informazioni a riguardo delle operazioni da svolgere. I frame aiutano a farsi delle aspettative. Le tipologie di conoscenza adatte per i frame sono tipicamente concetti e eventi stereotipati, tassonomie, conoscenza di dominio dove sono chiare le aspettative relativamente alla forma e contenuto dei dati.

Una definizione più tecnica di frame può essere: Un frame è un metodo KR molto generale, utilizzato per rappresentare differenti tipologie di conoscenza, inclusi concetti, astrazioni etc. Combina un insieme strutturato di conoscenza procedurale e dichiarativa che descrive un oggetto o una classe di oggetti.

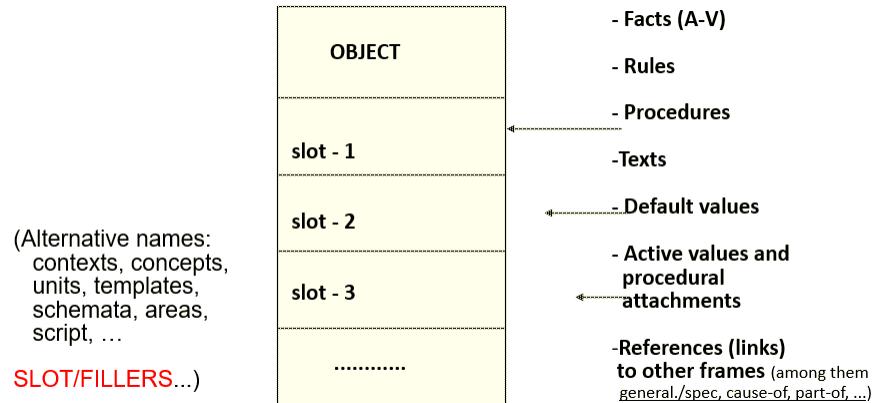
La differenza tra le due tipologie di conoscenza è:

- **Conoscenza procedurale (come):** orientata alle istruzioni, sapere come. Si concentra su come ottenere i risultati: si pensi alla programmazione procedurale, ovvero un insieme di azioni definite per ottenere qualcosa.
- **Conoscenza dichiarativa (cosa):** orientata alle asserzioni, sapere cosa. Descrive un oggetto e eventi specificandone le proprietà che lo caratterizzano. Non fa attenzione sulle azioni necessarie per ottenere il risultato ma solo sulle proprietà. Tipico della programmazione logica.

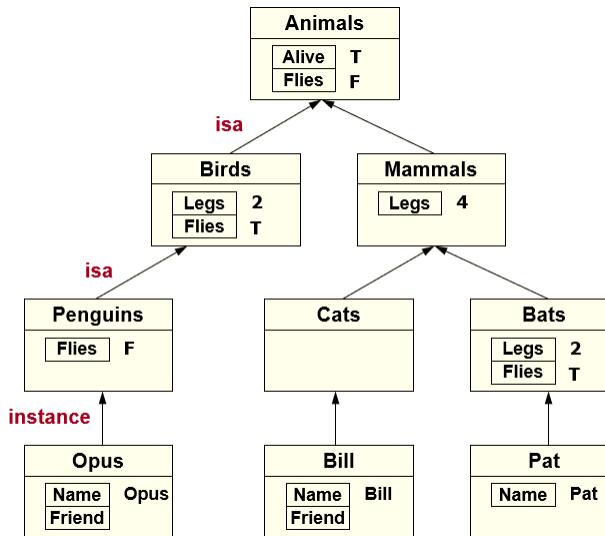
Abbiamo che:

- **Procedurale:** efficiente, bassa modificabilità, bassa adeguatezza cognitiva.
- **Dichiarativa:** maggiore astrazione, buona modificabilità, affidabilità e matching cognitivo. Bassa efficienza computazionale.

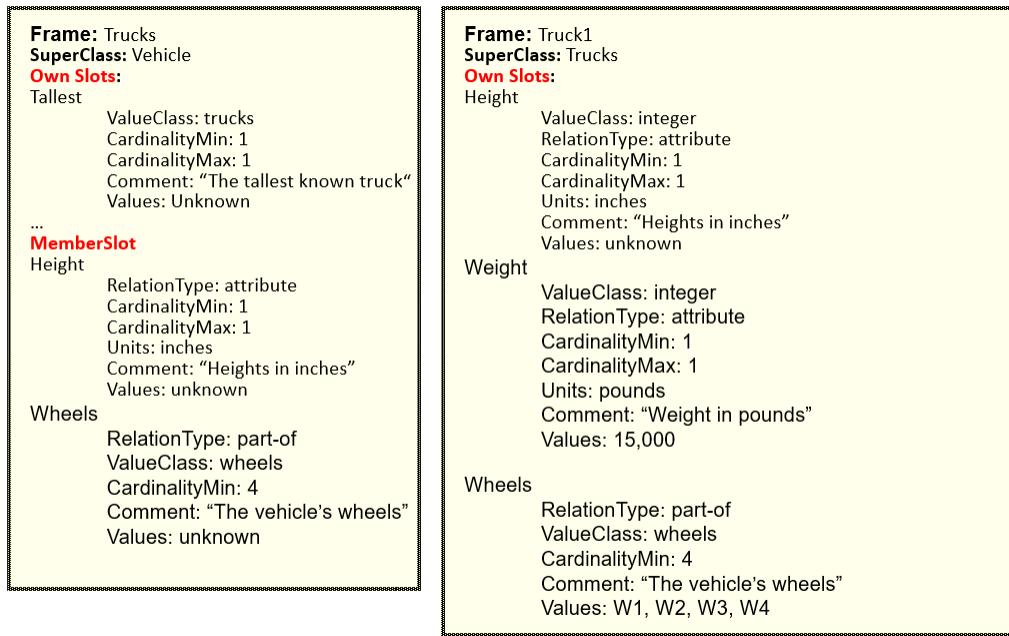
Struttura di un frame:



Esempi:



Esempi: Own e member slot



2.7.1 Caratteristiche:

Range dei valori degli slot: il valore di ciascuno slot può essere associato a un tipo o a qualche vincolo specifico (min, max, etc.).

Valori di default: valori usati in modo temporaneo, possibilmente approssimanti, quando una specifica informazione relativa allo specifico valore non è presente. Possono essere sovrascritti successivamente quando saranno disponibili nuove informazioni. Esempio: un elefante è tipicamente grigio, ha 4 zoccoli e 2 zanne.

Frame: elefante
colore: (default: grigio)
n° zoccoli: (default: 4)
n° zanne: (default: 2)

In assenza di informazioni possiamo ragionevolmente usare i valori di default.

2.7.2 Algoritmi di ragionamenti sui frame

- matching
- istanziazione
- derivazione di fatti non osservati

- attaccamento procedurale (daemons)
- e sfruttando i link is-a: ereditarietà e ricerca di intersezione

Matching: fornita un'entità sconosciuta di cui conosciamo alcune caratteristiche (osservate) e un insieme di frame relativi a classi di entità, l'operazione di matching verifica che l'oggetto sconosciuto appartenga o meno a una delle classi rappresentate dai frame.

Si può avere matching completo e parziale. In generale il matching può essere usato per diversi scopi: classificazione, misurazione di similarità. È anche una operazione di base del case-based reasoning.

- classificazione: A matcha B allora A è di tipo B
- giudizio di similarità: A matcha parzialmente B allora A è simile a B
- case-based reasoning: A è il miglior match parziale di B che si può trovare in memoria, allora usiamo l'informazione attaccata ad A per processare B, tenendo in considerazione le possibili discrepanze.

Istanziazione: Dato un oggetto e sapendo che appartiene ad una data classe rappresentata da un frame (ad esempio come risultato di un'operazione di matching), l'oggetto viene inserito nella knowledge base (o nella memoria di lavoro), viene allocato uno spazio di memoria adeguato e viene stabilito un collegamento verso la sua classe, eventualmente ereditando alcune proprietà note a livello di classe (proprietà che aumenteranno le proprietà che già conoscevamo dell'oggetto, perché, ad esempio, le abbiamo osservate prima).

Attaccamento procedurale: È il meccanismo per integrare conoscenza procedurale e dichiarativa. Consiste nell'associare conoscenza procedurale ad uno slot; tipicamente una procedura. Tali procedure sono attivate (a run-time) quando "qualche condizione è soddisfatta", per esempio quando una specifica operazione viene eseguita nello slot, come read o write.

Le computazioni legate al problem-solving accadono largamente per via di side-effect sul flusso dei dati in entrata e uscita dai frame. Tipiche categorie di attaccamenti procedurali sono:

- IF-ADDED
- IF-REMOVED
- IF-NEEDED
- IF-REPLACED
- IF-NEW
- IF-IN-RANGE

La semantica di un attaccamento procedurale è data dai suoi eventi (read, write, etc.) che permettono l'esecuzione della procedura (event-driven). Una procedura di questo tipo, che viene automaticamente invocata quando uno specifico evento avviene è anche detta "daemon". Esempio:

```

Frame: condenser
...
Slot-2: temperature
  value: integer between Tmin and Tmax
  IF-NEEDED: procedure for acquisition from sensors
  IF-ADDED: procedure for graphical display

Slot-3: pressure
  value: integer between ...
  IF-NEEDED: ...
  IF-ADDED: IF value > threshold THEN call ALARM
  IF-REMOVED: procedure for:
    - disabling sensors
    - updating screen
    - ...
  
```

2.7.3 Differenze con l'OOP

La programmazione orientata agli oggetti è derivata dai frames: knowledge vs computation information.

La differenza sostanziale è il meccanismo di attivazione per il codice procedurale: metodi vs attaccamenti procedurali; in altre parole: invocazione esplicita vs side-effect.

Il livello di astrazione nei frames è maggiore che nell'OOP.

2.7.4 Valutazione dei frame: pro e contro Pro:

- struttura ricca
- buona integrazione (naturale ed efficiente) di conoscenza dichiarativa e procedurale (buona adeguatezza cognitiva)
- buon livello di encapsulazione per fare astrazioni: la conoscenza è localizzata piuttosto che distribuita
- scomponibilità
- buona modularizzazione della conoscenza
- buona efficienza computazionale se gli attaccamenti procedurali sono sfruttati

Cons:

- semantica potenzialmente complessa da capire
- problemi di ereditarietà multipla
- scelta delle proprietà essenziali e accidentali

2.8 SCRIPTs

I ricercatori nella rappresentazione della conoscenza - in particolare Roger Schank e i suoi collaboratori (Riesbeck) - hanno ideato alcune interessanti variazioni sul tema degli oggetti strutturati. In particolare, hanno proposto l'idea degli scripts (trad. Italiana della parola 'script': 'copione').

Simili ai frame, aggiungono sequenze di eventi, obiettivi e piani degli attori coinvolti.

Uno script è una descrizione di una classe di eventi in termini di contesti, partecipanti e sotto-eventi. Gli slot sono ordinati (viene definita una sequenza).

Un copione può essere considerato uno schema per episodi generalizzati, consentendo allo studente di fare inferenze sulle situazioni compilando le informazioni mancanti.

Uso degli script nella deduzione e nel decision making

Schank crede che la comprensione umana sia sviluppata da una miscela di osservazioni esperienziali e inferenze fatte da esperienze precedenti (script) archiviate nella memoria (una libreria di script).

Gli script, attraverso la conoscenza stereotipata che codificano, sono in grado di supportare:

- comprensione con aspettative e
- processo decisionale con conoscenza del passato

Componenti di uno script:

- **Condizioni di ingresso** che devono essere vere per l'attivazione dello script;
- **Condizioni di terminazione** che sono vere quando lo script viene terminato;
- **Elementi e oggetti** che supportano lo script;
- **I ruoli** sono le **azioni che i singoli partecipanti** devono eseguire;
- Le scene spezzano il copione in **sottosequenze** che:
 - occorrono sequenzialmente e
 - forniscono alternative (se la condizione A then scena 1 else scena 2)

Esempio: a person dines in a restaurant: finding a seat, reading the menu, ordering drinks etc.

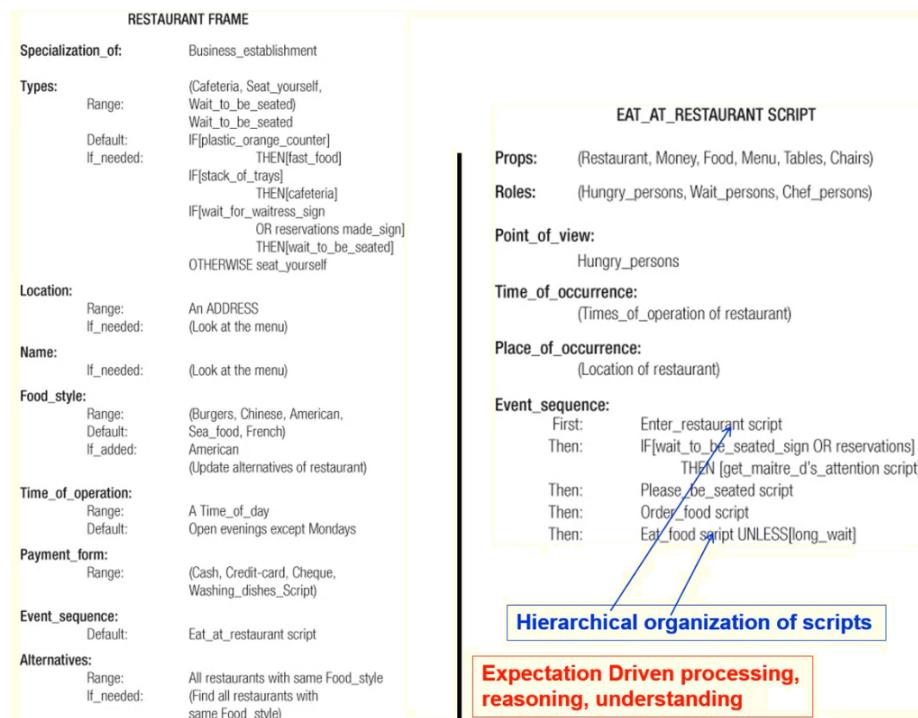
Entry conditions:
 Customer is hungry
 Customer has money

Results:
 Customer has less money
 Owner has more money
 Customer is not hungry
 Customer is pleased
 (optional)

Participants:
 Customer,
 Owner,

...

Esempio: ristorante



2.9 MOPs

Memory Organization Packets (MOPs) proposti da Schank (1989) sono frames utilizzati per rappresentare memoria episodica; ovvero memoria di scene, specifici eventi o la loro generalizzazione.

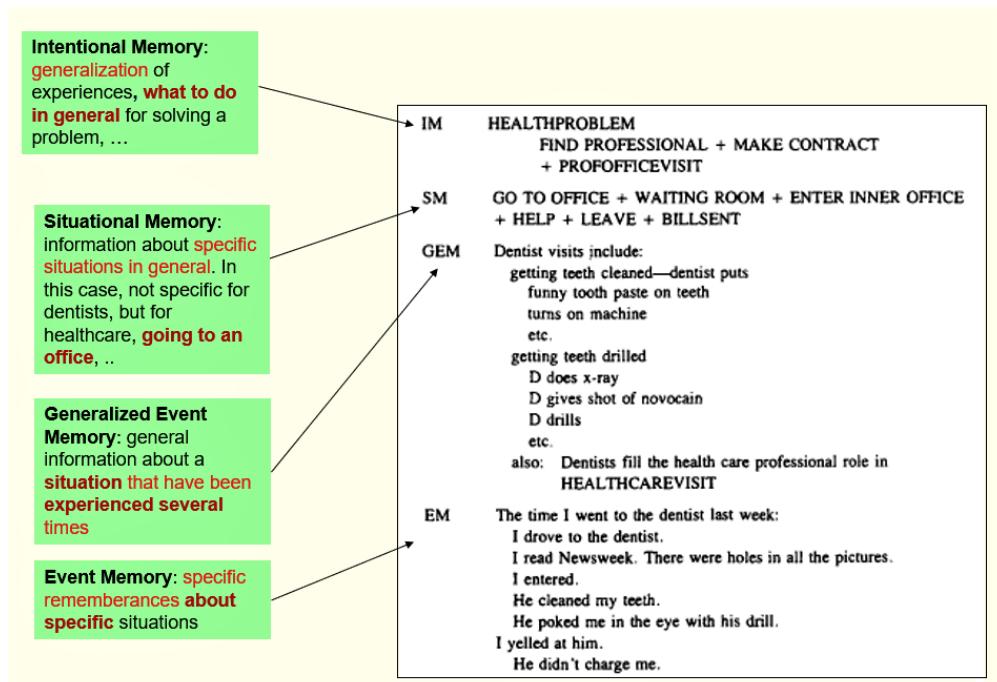
"All we know is embodied in stories. We understand everything in terms of stories we already know." Roger C. Schank.

Definizione di MOPs: Un MOP consiste in una serie di scene dirette al raggiungimento di un obiettivo. Un MOP ha sempre una scena principale il cui obiettivo è l'essenza o lo scopo degli eventi organizzati dal MOP.

Poiché i ricordi si trovano nelle scene, una parte molto importante dell'organizzazione della memoria è la nostra capacità di viaggiare da una scena all'altra. Un MOP è un organizzatore di scene. Trovare il MOP appropriato, nella ricerca della memoria, consente di rispondere alla domanda "Cosa succederebbe dopo?" e la risposta è un'altra scena. I MOP forniscono informazioni su come le varie scene sono collegate tra loro.

Trovare un MOP non è un processo cosciente. Non ci sediamo a dire a noi stessi: "Mi chiedo quale MOP funzionerebbe bene qui?" Ma cerchiamo di sapere dove siamo, cosa sta succedendo e cosa.... Ciò significa sapere in quale scena ci troviamo e quale scena verrà dopo.

Esempio di MOP: memorie di storie ed eventi dal dentista.



2.10 Production Rules

Le regole di produzione sono un metodo KR per rappresentare conoscenza "esperta" (euristiche) e ragionare con essa.

Il sistema basato sulle regole di produzione fu proposto nel 1972 da Newell & Simon come un modello della cognizione umana (come il cervello umano processa le informazioni: dato un specifico insieme di circostanze, attiviamo azioni, conclusioni, decisioni, conoscenza).

Le regole di produzione corrispondono a un pattern di ragionamento umano tipico, come:

- ragionamento per eccezione;
- ragionamento organizzato in casi, che sono riconosciuti da un insieme di condizioni;
- ragionamento deduttivo o abduttivo.

Le regole di produzione derivano da un sistema proposto da E. Post nel 1942 che consiste in un meccanismo di manipolazione/produzione di simboli. N. Chomsky usò questo formalismo come sistema di riscrittura sfruttato dalle grammatiche formali.

Definizione: Le regole di produzione (Newell & Simon, 1972) costituiscono un formalismo che consente la rappresentazione della conoscenza frammentaria, organizzata in due parti:

- la prima specifica le condizioni di applicabilità delle conoscenze contenute nella regola;
- la seconda parte specifica le azioni da eseguire (procedurali) o le condizioni da impostare (più dichiarative) quando viene applicata la regola.

Struttura:

IF <condizione> **THEN** <action>

Esempio:

IF near-destination **THEN** go-walking

IF there is a device ?D with temperature ?T and ?T>450° **THEN** turn off ?D

?D e ?T sono dette variabili.

Esempio con conoscenza procedurale:

IF engine_does_not_start **AND** starting engine OK **AND** lights OK
THEN check(spark plugs)

Esempio con conoscenza dichiarativa:

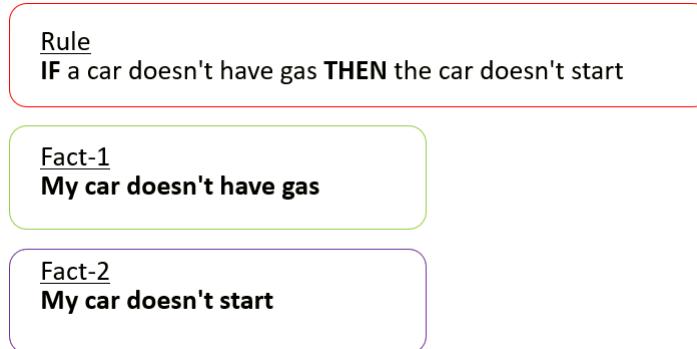
IF nurse (?X, child) **THEN** is-a (?X, mammal)

2.10.1 Sfruttamento base delle regole di produzione nelle procedure di ragionamento

I passi elementari di ragionamento sono:

- **Deduzione**
- **Abduzione**
- **Giustificazione**

Considerando i seguenti tre frammenti di conoscenza:



Deduzione: assumendo che il fatto-1 sia conosciuto (My car doesn't have gas) per via della regola-1 (IF a car doesn't have gas THEN the car doesn't start) eseguendo il matching fatto-1 con la parte sinistra (LHS) della regola-1 possiamo asserire sicuro il fatto-2: My car doesn't start.

Dedurre = asserire nuova conoscenza certa, precedentemente non nota.

Abduzione: assumendo che il fatto-2 sia conosciuto, per via della regola-1 possiamo eseguire il match del fatto-2 con la parte destra (RHS) della regola-1, ipotizzando così il fatto-1.

Ipotizzare = individuare un'ipotesi plausibile (non certa).

Giustificazione: assumendo conosciuto il fatto-1 e il fatto-2. Per via della regola-1 possiamo fornire una spiegazione plausibile allo stato dei due fatti.

Giustificare = fornire una spiegazione dello stato dei fatti, indicare una relazione che esiste fra due fatti.

2.10.2 Ragionare con più regole di produzione

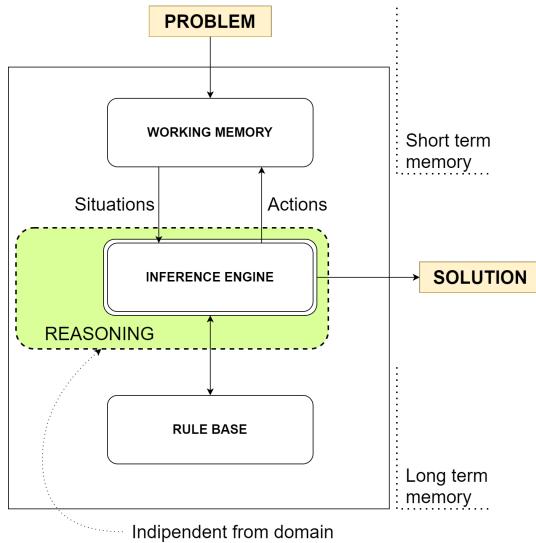
Concatenare i risultati dell'applicazione di una regola con l'applicazione di un'altra regola.

1. Ragionamento deduttivo, tramite **forward chaining** di una o più regole (data driven);

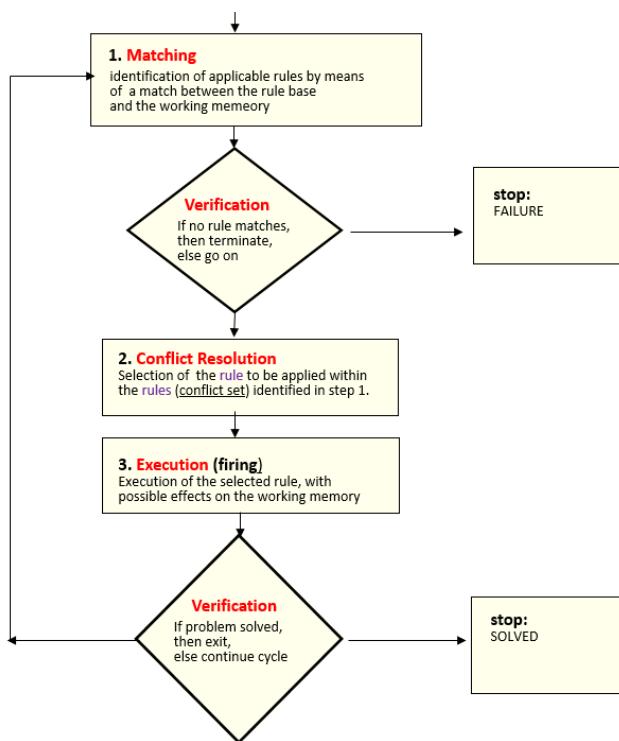
2. Ragionamento abduttivo, tramite **backward chaining** di una o più regole (goal driven).

2.10.3 Struttura base di un interprete di regole di produzione

Segue una rappresentazione grafica:



Come funziona? il classico **recognize-act cycle** di un **interprete** per production rules (inference engine - motore inferenziale).



Istanziazione di una regola

Una istanza di una regola è una coppia costituita dalla regola e i valori delle variabili (della regola) che soddisfa le condizioni di applicabilità della regola.

L'associazione variabile-valore è detta binding. Il conflict set è costituito da istanze di regole. Esempio:

RULE BASE	WORKING MEMORY
... ... RULE 19 IF there is a device ?D with temperature ?T and ?T > 450° F THEN turn off ?D 12. (device power-supply-1 460° F) . 14. (device modem-13 760° F) . 16. (device power-supply-34 290° F) .

Il matching produrrebbe 2 istanze della regola 19. Il conflict set sarebbe:

- (RULE 19, ?D = power-supply-1 corresponding to work. memory element no. 12)
- (RULE 19, ?D = modem-13 corresponding to work. memory element no. 14)

Regole di produzione e statement condizionali: sembrano simili al costrutto ma sono due cose diverse e usate in modo diverso.

- Lo statement if-then-else è un costrutto di controllo tipicamente utilizzato per linguaggi procedurali, è eseguito durante l'esecuzione di una istruzione di un programma, in un punto preciso del codice.
- La production rule invece è un meccanismo di KR, è eseguito se fa match con il contenuto della working memory e se è selezionato dal meccanismo di conflict resolution tra tutte le regole (istanze) che matchano il contenuto della wm. Quando: dipende.

Forward chaining

Processo deduttivo, data driven. Nella working memory sono contenute, inizialmente, i dati a disposizione (fatti certi) e cresce con altri fatti, ottenuti da passi deduttivi. Il matching viene eseguito sulla parte sinistra della regola. I passi deduttivi consistono nell'inserire in WM i fatti certi dedotti presenti nella parte destra della regola.

Segue un esempio: transportation problem.

Rule Base

FACTUAL KNOWLEDGE:

R1: IF railway-station THEN +near-destination
 R2: IF stadium THEN +far-destination
 R3: IF hospital THEN +far-destination, +in-the-city-destination

EXPERT KNOWLEDGE:

R10: IF near-destination THEN +(go walking)
 R11: IF far-destination THEN +(go by-car)
 R12: IF far-destination AND in-the-city-destination THEN +use-public-transportation
 R13: IF use-public-transportation THEN +(go by-bus)
 R14: IF use-public-transportation AND very-short-time THEN +(go by-taxi)

CONTROL KNOWLEDGE:

R20: IF <START> THEN → "Where do you like to go?", +←
 R21: IF (go X) THEN → "Go X", → "Session terminated", +HALT

Esecuzione

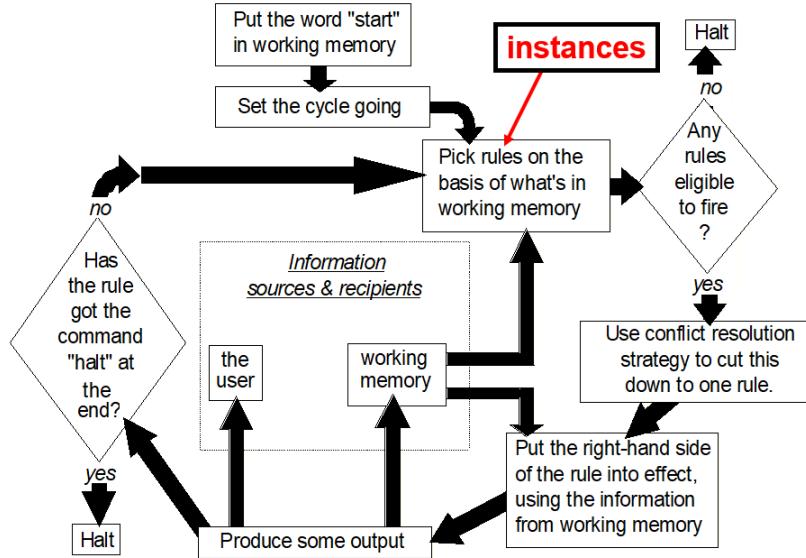
1.	WM: <start> CS: R20 CR: R20 Exec: R20: → "Where do you like to go?" → (... ← Hospital)	4.	WM: <start>, Hospital, Far-d., in-city-d., use-pub.transport. CS: R20, R3, R11, R12, R13 CR: R13 (Recency) Exec: R13: +(go by-bus)
2.	WM: <start>, Hospital CS: R20, R3 CR: R3 (Refraction) Exec: R3: +far-destination, +in-the-city-destination	5.	WM: <start>, Hospital, Far-d., in-city-d., use-pub.transport., (go by-bus) CS: R20, R3, R11, R12, R13, R21 con X= by-bus CR: R21 (Recency) Exec: R21: → "Go by-bus", → "Session terminated", +HALT
3.	WM: <start>, Hospital, Far-d., in-city-d. CS: R20, R3, R11, R12 CR: R12 (Specificity) Exec: R12: +use-public-transportation		WM: , HALT

Il ciclo Recognize-act nel Forward chaining:

1. Considerando i dati attualmente nella WM.
2. Si controlla se ci sono istanze di regole la cui parte sinistra sia soddisfatta da dati nella WM. Si registrano i bindings di tale istanza che matcha.
3. Se ci sono tali istanze, si proceda con la fase di conflict resolution, identificando l'unica regola-bindings da usare per il passo deduttivo.
4. Si proceda con il passo deduttivo, che consiste nell'esecuzione della parte destra della regola inferenziata nell'istanza selezionata nello step precedente.
5. Return to step 1.

Output e condizioni di terminazione:

- Se nello step 3 il conflict set è vuoto, il ciclo termina con un messaggio di failure.
- Se nello step 5 la condizione di terminazione è soddisfatta il ciclo termina con messaggio di successo.



Backward chaining

Processo goal driven. La WM è strutturata in due parti:

- Parte A: fatti che sono certi;
- Parte B: obiettivi (goals);

Si esegue il matching sulla parte destra (sui goals correnti). Step abduttivo: si rimpiazza il goal corrente con il nuovo subgoal SG (che consiste nelle condizioni presenti nella parte sinistra della regola attivata).

IF SG_A & SG_B & SG_C THEN GOAL

LHS: condizioni/SubGoals che vanno soddisfatti per soddisfare il GOAL a destra (RHS). Se i subgoals sono soddisfatti allora il starting goal è anche soddisfatto.

Il ragionamento guidato dagli obiettivi (o concatenamento all'indietro - backwards chaining), è un modo efficiente per risolvere problemi che possono essere modellati come problemi di "selezione strutturata": scegliere la soluzione ("obiettivo") tra diverse possibili soluzioni potenziali. Per ogni soluzione sappiamo quali sono le sue precondizioni, ovvero quali sono le condizioni che, se soddisfatte, permettono di concludere la soluzione.

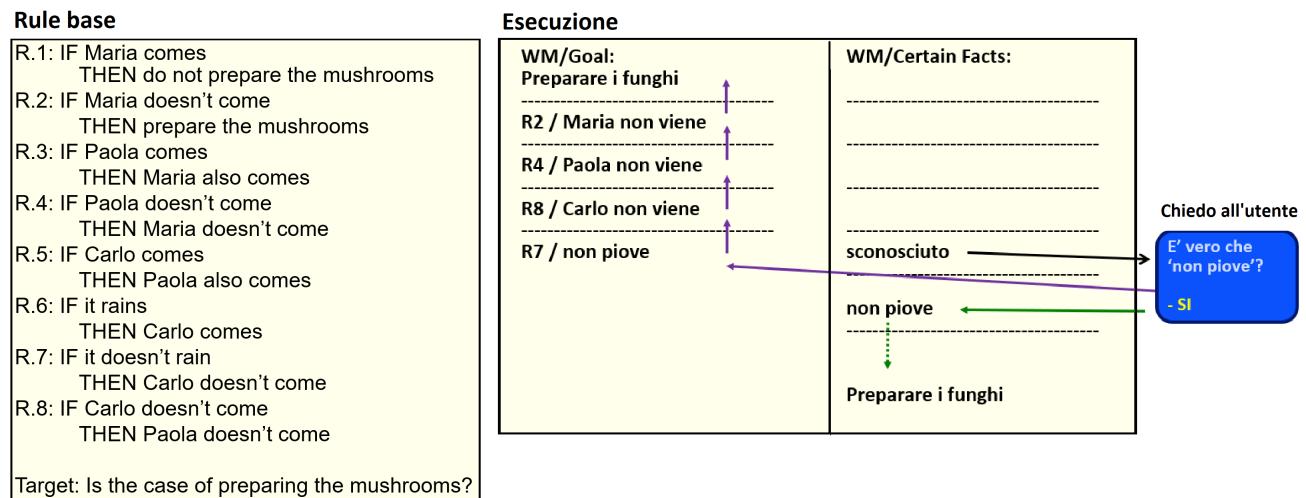
Lo scopo del sistema è scegliere la scelta migliore tra molte possibilità enumerate. Ad esempio, i problemi di classificazione dell'identificazione rientrano in questa categoria. Anche i sistemi diagnostici si adattano a questo modello, poiché lo scopo del sistema è trovare la diagnosi corretta (da un insieme noto a priori di potenziali diagnosi), dati alcuni modelli di osservazioni e sintomi.

La conoscenza è strutturata in regole che descrivono come ciascuna delle possibilità potrebbe essere selezionata: cioè quali condizioni dovrebbero essere soddisfatte per selezionare una specifica possibilità.

La regola suddivide il problema in sottoproblemi, in altre parole, l'obiettivo in sotto-obiettivi (riduce il problema in sottoproblemi). L'obiettivo è quindi qualcosa da dimostrare, non un fatto certo.

Esempi di ragionamento sul concatenamento all'indietro: Diagnosi, Classificazione, Analisi Incidenti / Catastrofi / Crimine, Recupero di un oggetto smarrito, Decision Making partendo dagli obiettivi che voglio raggiungere (ragionamento what-if partendo dalle conclusioni), Anomaly Detection, etc.

Esempio backwards chaining:



Il ciclo Recognize-act nel Backward chaining:

1. Si consideri l'attuale (sub)goal
2. Si controlli se si trova nella WM (fatti certi) un fatto che lo soddisfa. Se è il caso termino: il goal corrente è soddisfatto e quindi anche tutti i subgoals precedentemente considerati, fino al goal di partenza.
3. Se non trovo un fatto certo che matcha il subgoal corrente allora provo se ci sono istanze di regole la cui parte destra soddisfa il subgoal (matching).
4. Se tale/i istanza/i è trovata registro il binding e procedo con la fase di conflict resolution, identificando la regola-binding da usare nello step abduttivo.
5. Procedo con lo step abduttivo, che consiste nel rimpiazzare il goal corrente con la parte sinistra della regola (nuovi subgoal da considerare), istanziati con i bindings.
6. Return to step 1.

Output e condizioni di terminazione:

- Se il conflict set nello step 3 è vuoto, provo a chiedere all'utente (come nell'esempio) se il goal è soddisfatto. Se l'utente conferma allora è soddisfatto, termina: inserisco la risposta nella WM/fatti certi e rieseguo la catena deduttiva in WM dall'ultimo al primo goal iniziale.

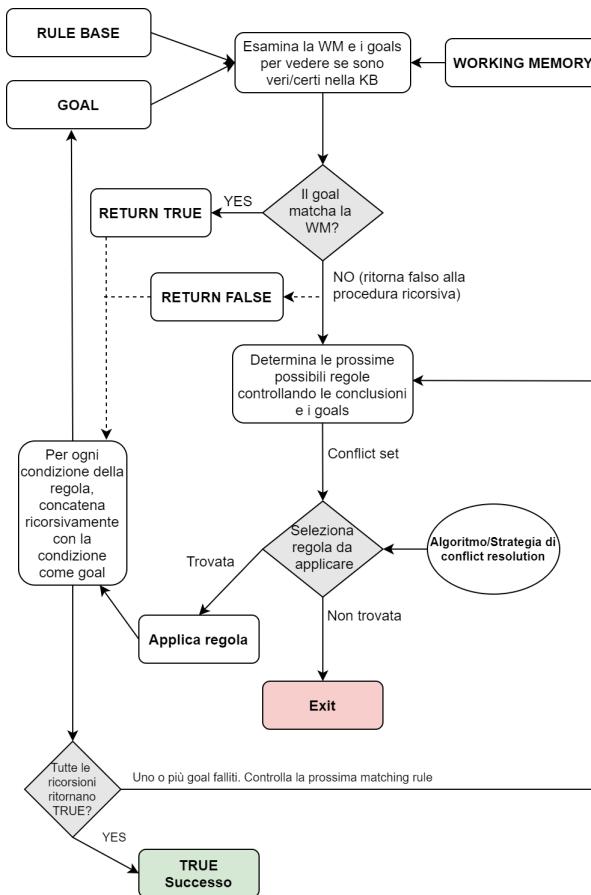
- Altrimenti se l'utente non conferma: fail. Cancello i goal correnti e ritorno allo step 1.

Capacità di spiegazione del concatenamento all'indietro: Alla fine, o quando il sistema chiede all'utente qualcosa in relazione a un obiettivo, se l'utente chiede "Come hai trovato questo risultato? Perché mi è stata posta questa domanda?", si può costruire una spiegazione come risposta, considerando la catena abduttiva.

Spiegazione: la capacità di illustrare in dettaglio e motivare il processo di ragionamento (tematica della XAI - eXplainable AI).

Lo step 4 prende la forma di una riduzione da Goal in Subgoals, come la riduzione di un problema nei suoi sottoproblemi:

IF S-problem1 solved & S-problem2 solved THEN Problem solved



2.10.4 Backward vs Forward chaining

Forward:

- Facile da implementare
- Sensibile ai cambiamenti ai dati
- Inefficiente

- Non trasparente, valuta molte alternative

Adeguato per problemi le cui soluzioni sono numerose e non conosciute a priori.

Backward:

- Efficiente con pochi goals
- Si concentra sulle regole rilevanti per il goal, viene valutato quello che è necessario per provare il goal
- Permette più facile spiegazione, trasparenza
- Non adeguato per risposte rapide ai cambi dei dati
- Difficile da implementare

Adeguato per problemi le cui soluzioni non sono numeri e sono note a priori.

2.10.5 Approccio generale al conflict resolution

Durante la fase di conflict resolution vengono applicati specifici **criteri di esclusione** al conflict set che comportano l'eliminazione di alcuni elementi fino a che non rimane una sola istanza che verrà poi selezionata per l'esecuzione.

Criteri:

- **Evitare cicli (Rifrazione):** mai rieseguire istanze già eseguite
- **Reagire al cambiamento (Attualità) :** preferire il matching di dati recenti; seguire la corrente linea di ragionamento finché non si capisce se è corretta o meno
- **Sfruttare regole più specifiche alla situazione attuale (Specificità) :** preferire il matching con elevato numero di condizioni
- **Considerare criteri ad-hoc per specifici domini:** assegnare priorità alle regole e poi preferire dati/regole con priorità maggiore. Criteri programmabili (statici o dinamici).

2.10.6 Concetto di Meta-Conoscenza: La meta-conoscenza (meta knowlendge) può essere definita come: "conoscenza riguardo conoscenza". Un esempio: conoscere cosa scegliere nel prossimo step, come fare conflict resolution, come organizzare i passi di reasoning, etc. L'effetto della meta-conoscenza non è sulla descrizione del problema e sui risultati intermedi del processo di ragionamento, ma ha influenza sul motore di inferenza e su come procede.

- **Conoscenza di dominio:** Da dati di dominio D_D in WM e regole di dominio R_D in RB si ricavano, in generale, effetti I_D in WM.
- **Meta-Conoscenza:** Da dati di dominio D_D in WM e da meta-regola R_{MK} in RB, tipicamente si ricavano effetti I_{MK} sul funzionamento del motore inferenziale MI (ad esempio sulla fase della conflict resolution).

Esempio:

DOMAIN RULE	META-RULE
IF patient in consultation known diabetic under hypoglycemic treatment slightly perturbed glycoregulation THEN slightly unbalanced diabetes	IF low hemoglobin level normal amount of blood cells increased average blood cell volume THEN Look for rules ABOUT: Blermer anaemia Cirrosis Acute alcholic hepatitis

Una Meta-regola può essere quindi vista come "una regola che governa l'applicazione di altre regole".

2.10.7 Incertezza e Ragionamento approssimato nei sistemi Rule-Based

L'idea alla base è che in molti domini la conoscenza non è certa. L'incertezza può essere causata da molteplici cause:

- Non è chiara/nota l'evoluzione di un fenomeno, quindi siamo incerti su quello che succederà
- Abbiamo informazioni incomplete
- Non siamo capaci di eseguire test che ci forniscano certezza
- Il concetto è definito in modo vago/ambiguo
- Le euristiche (regole) derivano da esperienza, ma non sono sempre vere, ci sono eccezioni, a volte capita di aver osservato un comportamento diverso.

Esempio:

IF the identity of the germ is not known with certainty
AND the germ is gram-positive
AND the morphology of the organism is "rod"
AND the germ is aerobic
THEN there is a strong probability (0.8) that the germ is of type enterobacteriaceae

Approcci al ragionamento approssimato:

- Teorema di Bayes, probabilità condizionale;
- Logica Fuzzy;
- Fattori di certezza.

Fattori di certezza

Un fattore di certezza è associato a ciascun fatto e ciascuna regola. È calcolato come la differenza di due valori: MB (Misura di credenza - belief) e MD (Misura di discredenza - disbelief). MB e MD variano da 0 a 1, la loro somma non è necessariamente 1. La differenza (CF) può essere un valore tra -1 e 1.

Se ad ogni fatto è associato un CF, quanto è il valore della CF della congiunzione di più fatti? e che dire della disgiunzione? ovvero del risultato dell'applicazione di una regola (che è anche caratterizzata da una CF), applicata ai fatti (con una CF associata a ciascuna uno)? Esempio:

IF F_1 has C.F. = 0.9 and F_2 has C.F. = 0.7, then
conjunction
(F_1 and F_2) have CF = 0,7 (the minimum of the two)
disjunction
(F_1 or F_2) have CF = 0,9 (the maximum of the two)
Negation
The negation of F_2 is – 0,7

**IF F_1 and F_2 and F_3
THEN F_4 , with C.F. = 0.8**

If we know F_1 with C.F. = 0.9
 F_2 with C.F. = 0.7
 F_3 with C.F. = 0.75
then we can conclude F_4 with certainty factor
 $0.7 \times 0.8 = 0.56$

Fuzzy Logic

La funzione member-of che usualmente restituisce valore binario (appartiene o non appartiene), è rimpiazzata da una funzione il cui valore è nel range 0-1 (inclusi gli estremi). Indica il grado di appartenenza.

2.10.8 Sistemi di produzione vs programmi tradizionali

	Programmi tradizionali	Sistemi di produzioni
<i>Programmi</i>	Costituiti da una sequenza finita di istruzioni	Insieme di regole indipendenti
<i>Conoscenza</i>	Non più presente o distribuita nel codice (mixata con il controllo) e dati	Separata dal controllo contenuto nella base di regole e nella WM
<i>Controllo</i>	Pre-determinato ed esplicito	Non esplicito e determinato a run-time dalla catena inferenziale
<i>Esecuzione</i>	Segue la direzione delle strutture di controllo	Calcolato dal motore inferenziale
<i>Efficienza</i>	Alta	Bassa
<i>Modificabilità</i>	Bassa	Alta
<i>Adatto per:</i>	Situazioni in cui il controllo è gestibile	Situazioni in cui il controllo è complesso e la conoscenza è frammentata
<i>Trasparenza</i>	Molto bassa	Alta
<i>Sviluppo</i>	Ciclo tradizionale del software	Sviluppo incrementale

Tipologie di conoscenza adatte per le regole di produzione:

- Regole di decisione (euristiche - certe o meno)
- Ragionamento deduttivo o ipotetico
- Domini di conoscenza: frammentari, incompleti, complessi, data driven
- Domini con sviluppo nel tempo

Le Regole si adattano molto bene a ragionamenti in cui, in un flusso definito, si possono inserire grandi quantità di eccezioni, ciascuna corrispondente all'applicazione di una regola deduttiva. Si tratta di un modello molto tipico di certi settori di business. Ed è utilizzato il termine business rule.

In applicazioni del genere, sorge il problema di gestire tutte le regole corrispondenti.

Diverse conseguenze:

- Una modalità di modellare il ragionamento degli esperti
- La necessità di disporre di adeguati tool sw per la gestione di queste regole, in modo 'separato' dal flusso principale
- Nascono i BRE: Business Rule Engine, tool sw integrati/abili ai sistemi ERP per la costruzione di applicazioni in cui le regole sono gestite in modo indipendente

I Business Rule Engines, sono componenti sw integrati/abili ai sistemi ERP per la costruzione di applicazioni in cui le regole sono gestite in modo indipendente

Utilizzabili da NON programmati per aggiungere specifiche conoscenze/politiche decisionali a flussi più generali.

La specificazione della business rule risulta quindi separata dagli aspetti implementativi (codice) che la implementa e dal flusso generale cui è associata (in ben precisi punti)

Si parla di BRMS: Business Rule Management Systems.

2.10.9 Valutazione delle production rules

Pro:

- molto naturali e facili da usare
- semplici, intuitive, cognitivamente adeguate
- molto modulari
- ottimo grado di modificabilità ed estensibilità
- dichiarativo (il controllo globale è determinato dall'interprete), ma inferiore alla logica
- calcolabile in tempo lineare
- molti strumenti, shell, lingue: OPS5, YAPS, CLIPS, JESS;
- regole per il web semantico: RuleML e SWRL, un linguaggio di regole web semantico che combina OWL e RuleML

Contro:

- inefficiente (90% del tempo circa è impiegato per eseguire matching)
- non molto adeguato per gerarchie IS-A
- computazione non sempre totalmente trasparente
- non molto expressive: troppo uniformi, poca struttura
- domain e meta-knowledge mischiate
- programmare con astrazioni può rendere la base di regole molto illeggibile
- non supporta tipicamente il backtracking, guadagno di efficienza ma con effetti negativi sulle modifiche distruttive della WM

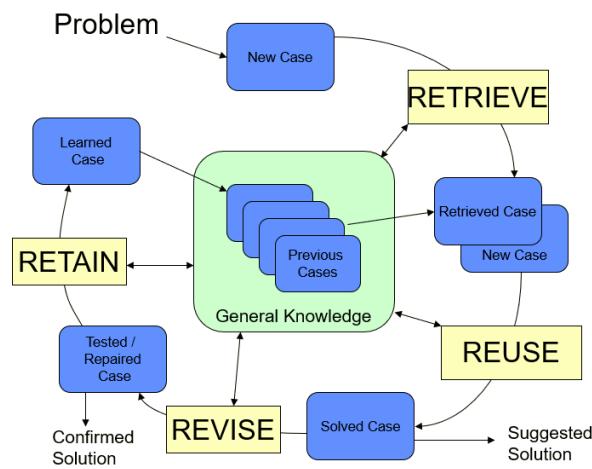
2.11 Case-Based Reasoning

CBR: imparare e ragionare per analogia. CBR è un processo di risoluzione di nuovi problemi basati sulle soluzioni di problemi simili risolti in passato.

Esempio: un meccanico che ripara un motore richiamando un caso di un'auto che ha esibito un sintomo simile.

Una delle assunzioni base del metodo CBR è che problemi simili hanno soluzioni simili.

Ciclo CBR generico:



Retrieve: uno dei tanti casi dalla case-base è selezionato sulla base di qualche misura di similarità (matching).

L'attività di recupero è definita come la ricerca di un piccolo numero di casi dalla base dei casi con la massima somiglianza con la query.

Possibili soluzioni algoritmiche: k-nearest-neighbor, alberi di decisioni, supervised ML algorithms, funzioni di similarità etc.

Quando la base di casi cresce, l'efficienza del recupero decresce. Esistono però metodi per migliorare l'efficienza di questa fase, come l'indexing o l'utilizzo di particolari strutture dati.

Reuse: riutilizzare una soluzione recuperata può essere abbastanza semplice se la soluzione viene restituita invariata come soluzione proposta per il nuovo problema. Se richiesto viene svolto dell'adattamento: selezionare gli elementi da cambiare e capire come cambiarli. Esistono molte tecniche di adattamento nel CBR (sostituzione, aggiustamento parametri, model-based, etc.).

Revise: in questa fase viene ottenuto il feedback relativo alla soluzione costruita fino a quel punto. Il feedback può venir fornito in forma di correttezza del risultato o in forma di un caso revisionato manualmente. Il caso revisionato (o altre forme di feedback) entrano nel sistema CBR (case-base) per il suo uso nella fase di retain successiva.

Retain: la fase di retain è quella di apprendimento per un sistema CBR (aggiungere nuovi casi revisionati alla case-base). Nuova esperienza entra a far parte della case-base.

Esempio:

Case-Base

Case 1	Case 2
<u>Problem & Features</u> <ul style="list-style-type: none"> • Problem: Front light not working • Car: VW Golf, 2.0L • Year: 1999 • Battery voltage: 13.6V • State of lights: OK • State of light switch: OK 	<u>Problem & Features</u> <ul style="list-style-type: none"> • Problem: Front light not working • Car: Passat • Year: 2000 • Battery voltage: 12.6V • State of lights: surface damaged • State of light switch: OK
<u>Solution</u> <ul style="list-style-type: none"> • Diagnosis: Front light fuse defect • Repair: Replace front light fuse 	<u>Solution</u> <ul style="list-style-type: none"> • Diagnosis: Bulb defect • Repair: Replace front light

Nuovo problema

- Observations define a new problem
- Not all feature values may be known
- New problem = case without solution

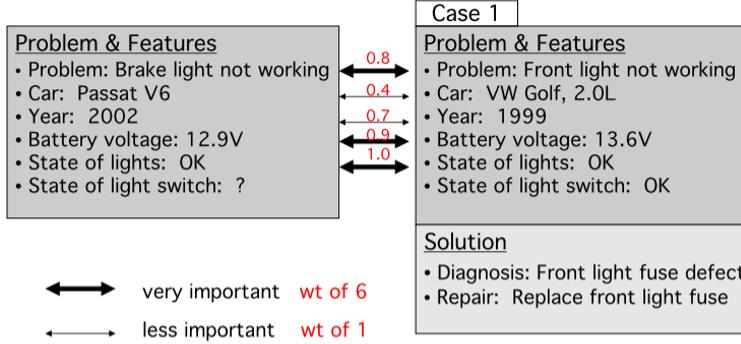
Problem & Features
• Problem: Brake light not working
• Car: Passat V6
• Year: 2002
• Battery voltage: 12.9V
• State of lights: OK
• State of light switch: ?

Trovare casi simili



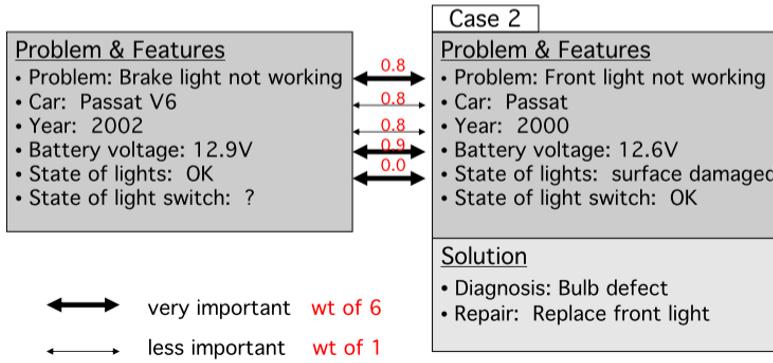
- Compare similarity of each feature
- But some features may be more important

Confronto con caso1



Similarity by wted avg = $1/20 (6*0.8 + 1*0.4 + 1*0.7 + 6*0.9 + 6*1.0) = 0.87$

Confronto con caso2

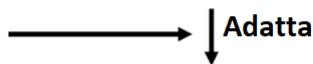


Similarity by wted avg = $1/20 (6*0.8 + 1*0.8 + 1*0.8 + 6*0.9 + 6*0.0) = 0.59$

Riuso caso 1

Problem & Features
• Problem: **Brake light** not working
• ...

Case 1
Problem & Features
• Problem: **Front light** not working
• ...
Solution
• Diagnosis: **Front light** fuse defect
• Repair: Replace **front light** fuse



New Solution
• Diagnosis: **Brake light** fuse defect
• Repair: Replace **Brake light** fuse

Salva nuovo caso3

Case 3
Problem & Features
• Problem: Brake light not working
• Car: Passat V6
• Year: 2002
• Battery voltage: 12.9V
• State of lights: OK
• State of light switch: OK
Solution
• Diagnosis: Brake light fuse defect
• Repair: Replace brake light

2.12 Prospettive evolutive di RBS, CBR e ML

I sistemi basati su regole (RBS), e più in generale i sistemi basati sulla conoscenza tradizionali (KBS), richiedono un'ampia elicitazione, acquisizione, analisi e modellazione della conoscenza del dominio considerato e dei processi di risoluzione dei problemi dei compiti considerati.

Questa attività necessaria per costruire un modello concettuale esplicito del dominio può essere molto impegnativa e richiede uno sforzo significativo da parte degli ingegneri della conoscenza e degli esperti del dominio.

L'idea di CBR cerca di superare questo problema:

1. CBR non richiede un modello di dominio esplicito e quindi l'elicitazione diventa un compito di raccolta di case history.
2. L'implementazione si riduce all'identificazione di caratteristiche significative che descrivono un caso, un compito più semplice rispetto alla creazione di un modello esplicito.
3. I sistemi CBR possono apprendere acquisendo nuove conoscenze come casi. Questo e l'applicazione delle tecniche di database facilitano la manutenzione di grandi volumi di informazioni.

I 3 punti sopra, e nello specifico il punto 2, indicano un percorso convergente verso l'approccio del Machine Learning. A proposito, il Machine Learning (ML) era a quel tempo ('90) nella sua prima fase di evoluzione, in un'epoca che potremmo chiamare era "Pre-Neural/pre-DeepLearning".

Da qui la nascita di tre differenti strategie per costruire problem solvers:

- **KBS:** analizzare e modellare manualmente il dominio di conoscenza, ragionamento e strategie di problem solving;
- **ML:** imparare automaticamente le relazioni tra dati in input e soluzioni del problema;
- **CBR:** salvare casi dell'esperienza passata e adattare vecchie soluzioni ai nuovi casi simili.

Il grande sviluppo di ML/DL ha portato a inserire nel modello generale del CBR degli step realizzati mediante ML/DL. In effetti il modello CBR è molto generale ed include fasi che sono dedicate a compiti molto generali e genericci che quindi possono essere affrontati con diverse tecniche specifiche. Di conseguenza, molte tecniche di ML/DL possono venir applicate nei vari passaggi del ciclo CBR.

Ad esempio possono essere implementate tecniche di ML per: la costruzione della libreria dei casi, per il matching con la libreria dei casi, per la scelta delle feature più utili, dei pesi, etc.; per il miglior adattamento al caso corrente.

3 Problem Solving

Iniziamo esplicitando il concetto di **problema**:

- Abbiamo un problema quando abbiamo un goal da raggiungere e non sappiamo cosa fare al fine di raggiungerlo.
- È necessario ragionare, per prendere decisioni, considerare differenti alternative, valutare i suoi effetti, valutare i suoi vantaggi e svantaggi, criticare il risultato ottenuto, considerare possibili alternative.
- La situazione (stato) che stiamo affrontando è differente dalla situazione (stato) in cui vorremmo essere; non è immediatamente chiaro come colmare il gap.
- Il processo di problem solving può essere considerato come un processo di ricerca mirato a raggiungere una sequenza di azioni capaci di raggiungere il goal.
- Un algoritmo prende in input e produce in output una soluzione, espressa come una sequenza di azioni che portano al goal desiderato.

Gli approcci e strategie generali sono gli algoritmi (metodi deterministici per trovare una soluzione) o le euristiche, ovvero regole pratiche e non generali che ci porteranno probabilmente a produrre una soluzione (probabilmente). Ad esempio ridurre un problema in sottoproblemi, la cui soluzione è già conosciuta o partire all'indietro dalla soluzione.

I principali fattori che hanno un impatto sulle strategie di ricerca della soluzione sono l'organizzazione dei dati (che costituiscono la rappresentazione del problema) e l'approccio di ricerca.

3.1 Definizioni e concetti base

- **Stato:** una specifica configurazione delle variabili che caratterizzano il problema;
- **Stato di partenza:** configurazione all'inizio del processo di problem solving;
- **Operatore:** azione che consente il passaggio da uno stato del problema a un nuovo stato del problema. Non tutti gli operatori possono essere applicabili in uno stato specifico. Dato uno stato s , possiamo identificare gli operatori che possono essere applicati su s (applicabile in s). Un operatore O viene descritto mediante la transizione che viene eseguita su uno stato s , quando O viene applicato a s , determinando in tal modo l'evoluzione dalla forma s ad un altro stato.
- **Spazio stati:** insieme di tutte le configurazioni (stati) raggiungibili dallo stato iniziale, tramite una qualsiasi sequenza di azioni. Solitamente è rappresentato mediante un grafo, i cui nodi corrispondono a stati e un arco che collega S_1 e S_2 corrisponde all'applicazione di un operatore (applicabile) allo stato S_1 che provoca la transizione allo stato S_2 .

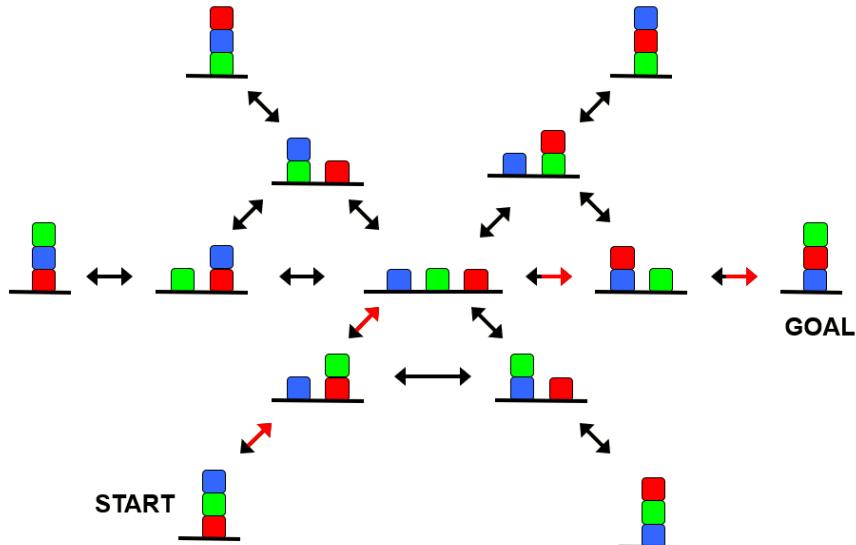
Definizioni alternative:

- Spazio stati: insieme di tutte le possibili configurazioni (stati) che sono teoricamente raggiungibili;
- Spazio del problema: insieme di tutte le possibili configurazioni (stati) che sono raggiungibili dallo stato iniziale.

Lo spazio degli stati rappresenta l'universo dove si cerca di operare. Altre definizioni:

- **Cammino:** una sequenza di stati attraverso lo spazio stati;
- **Stato goal/finale:** stato corrispondente all'aver raggiunto la soluzione;
- **Goal test:** test che valuta se lo stato attuale è lo stato finale;
- **Soluzione del problema:** un sequenza di operazioni (azioni) che permettono di trasformare lo start state in final state.
- **Soluzione ottimale:** le soluzioni non sono tutte uguali, possono differire in base a criteri di ottimizzazione.
- **Ricerca:** processo sistematico per analizzare gli stati, con lo scopo di trovare una soluzione;
- **Spazio di ricerca:** insieme di tutti gli stati attualmente raggiunti e considerati con gli operatori sfruttati;
- **Algoritmo di ricerca:** algoritmo devoto alla ricerca.

Esempio: the block world game



Quando si esegue problem solving si è tipicamente interessati alla miglior soluzione. Esistono dei criteri per valutare quale sia la migliore soluzione, ad esempio:

- **Costo operatore:** una funzione che assegna un valore di costo corrispondente all'applicazione di un operatore;
- **Costo cammino:** una funzione che assegna un costo a un cammino (la somma di ciascuna azione individuale appartenente al cammino);
- **Costo della soluzione:** costo del cammino che porta alla soluzione.

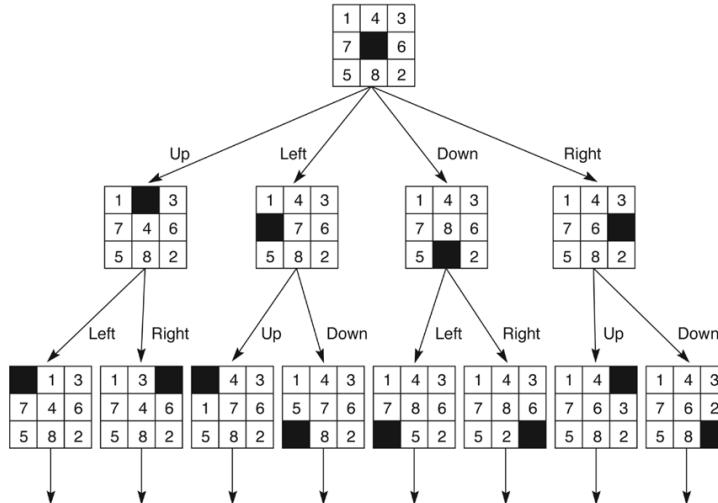
I criteri di ottimizzazione possono essere quindi basati sulla minimizzazione del costo; sul cammino più breve, etc.

- **Costo di ricerca:** si riferisce allo spazio di ricerca; è più alto se lo spazio di ricerca è maggiore.

Cercare una soluzione ottimale può essere molto costoso. Ci si accontenta tipicamente di un compromesso/trade-off tra costo minimo e soluzione ottimale (a volte cercare la soluzione ottimale può richiedere l'uso di molte risorse per un beneficio insignificante rispetto ad altre soluzioni "quasi-ottimali").

Ritornando al concetto di problema, possiamo riassumere i suoi componenti in:

- Stato di partenza
- Possibili operatori e transizioni corrispondenti
- Test del goal
- Funzione di costo per i cammini



3.2 Strategie basilari per la ricerca di una soluzione

Tipiche strategie per la ricerca di una soluzione sono:

- **Forward search:** data-driven search
- **Backward search:** goal-driven search
- **Bi-directional search**

Quale approccio scegliere dipende da vari fattori, uno di questi è il branching factor; si consideri l'esempio:

Napoleone is your ancestor?

10 generations, hypothesis that each human has 3 descendants

Bkw Search:

Each human has 2 ancestors $\rightarrow 2^{10}$

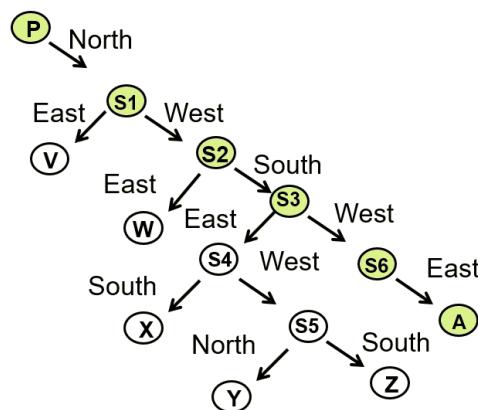
Fwd Search:

Each human has three descendants (more likely at the time of Napoleon) $\rightarrow 3^{10}$

Un'altra strategia è quella del **backtracking**: si procede verso la soluzione fino a raggiungere l'obiettivo (e si termina con successo!) o fino a quando non si può procedere oltre. In tal caso, torni al punto precedente in cui le scelte aperte erano ancora disponibili e prendi una di quelle opzioni.

Backtracking cronologico = tornare all'ultima scelta dove ci sono ancora opzioni disponibili.

Un esempio di problema a cui si adatta la strategia del backtracking è trovare la soluzione ad un labirinto. Esempio:



In generale, meno step inutili vengono fatti, più "intelligente" è la nostra strategia. Se spremiamo passi, dovremmo fare del backtracking. In generale una strategia procedere nel seguente modo:

1. seleziona uno stato su cui operare
2. sceglie l'operatore da applicare

3.2.1 Uniformed vs Informed Search

Un'altra categorizzazione delle strategie di ricerca è la seguente:

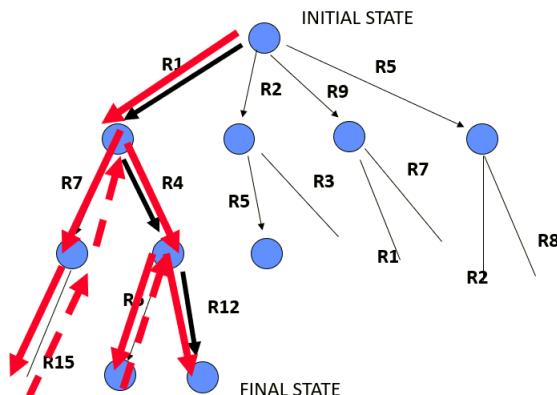
- **Uniformed Search (Blind Search):** la procedura è indipendente dalla qualità degli stati (tutti gli stati sono considerati equivalenti). Non molto intelligente per problemi realisticamente complessi.
- **Informed Search (Heuristic Search):** la procedura considera anche informazioni specifiche del dominio, al fine di considerare prima gli stati più promettenti.

Uniformed Search

Esempio di due uniformed search: depth e breadth first search.

- Depth first search: ad ogni step espando l'ultimo nodo tra i nodi generati

Esempio:



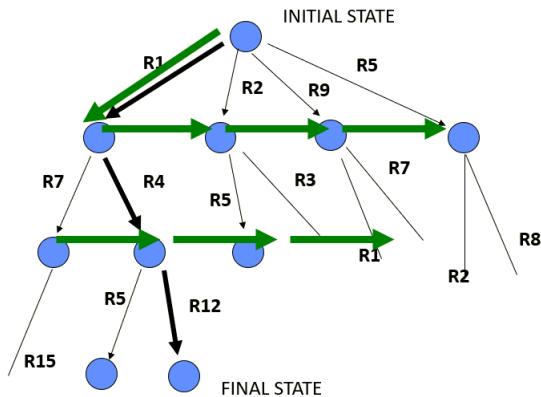
Il problema con depth first search è la presenza di loop e di cammini infiniti. Per quanto riguarda l'uso della memoria, poiché in ogni fase stiamo procedendo lungo un percorso risolutivo, il numero di stati che dobbiamo memorizzare per procedere (e applicare il backtracking, se necessario), cresce linearmente con la profondità del percorso.

La complessità è $O(r * p)$ dove r è il fattore di branching e p è la profondità del cammino della soluzione.

Il numero di nodi visitati cresce esponenzialmente man mano che la ricerca procede. La presenza di cammini infiniti può rendere impossibile il ritrovamento della soluzione ottima.

Una alternativa è il breadth first search:

- Breadth first search: un nodo di un dato livello è visitato solo dopo che tutti i nodi del precedente livello sono stati visitati.



Esempio:

I nodi sono visitati in maniera ordinata, livello dopo livello. Se una soluzione esiste, verrà trovata. Poiché si procede prima con i nodi dei livelli più vicini allo stato iniziale (nodo), la soluzione trovata sarà la soluzione più breve.

I loop e i percorsi infiniti non compromettono la ricerca di una soluzione.

Tuttavia, l'utilizzo della memoria è molto elevato: è necessario memorizzare tutti i nodi visitati in precedenza.

Complessità $O(r^p)$, dove r è branching factor e p la profondità del percorso della soluzione.

Anche in questo caso il numero dei nodi visitati cresce in modo esponenziale man mano che la ricerca va avanti.

Informed Search

Le tecniche fin qui considerate sono esaustive (ad eccezione dei loop infiniti): garantiscono di trovare una soluzione, poiché visitano tutto lo spazio degli stati.

Il loro costo è però troppo alto in termini di spazio di ricerca, in genere mostrano una sorta di esplosione combinatoria, indipendentemente dalla velocità della CPU.

Un obiettivo plausibile è quindi rendere più efficiente il processo di ricerca.

L'idea di base è quella di sfruttare informazioni specifiche sulla situazione per la selezione delle alternative!. Gli algoritmi di questo tipo sono chiamati euristiche.

Il problema di ricerca può essere considerato come il problema di visitare (esplorare) un grafo (un albero è un caso particolare di grafo), a partire da un nodo (stato iniziale). Ad ogni passaggio generico, considero il nodo n (stato n) cercando di:

- Valutare se il nodo n rappresenta uno stato finale.
- Se n non è uno stato finale, si identifica l'insieme di nodi che sono successori (cioè raggiungibili tramite 1 operatore applicato a n) del nodo n appena valutato (si dice che il nodo n viene espanso). Tali nodi successori vengono aggiunti ad un insieme S di successori non ancora espanso (visitato), chiamato elenco/lista aperto/a.
- Quindi, secondo un criterio specifico, si seleziona un nodo appartenente a S come nodo successivo da espandere e si torna al punto 1.

Esempio:

IF	selection criteria: «take the first node of the Open List S» S is an ordered set the new successors are always inserted <u>at the beginning of S</u> ,
THEN	The algorithm is DEPTH-FIRST

IF	selection criteria: «take the first node of the Open List S» S is an ordered set the new successors are always inserted <u>at the end of S</u> ,
THEN	The algorithm is BREADTH-FIRST

Commenti:

- Ovviamente, se cambiamo i criteri per l'espansione, ovvero come vengono identificati e inseriti i successori di N in S e come viene selezionato il nuovo nodo da espandere, otteremo diversi algoritmi di ricerca.
- La selezione da S (lista aperta) del nuovo nodo N da considerare corrisponde alla selezione dello stato su cui lavorare (ovvero la parte del problema su cui ragioniamo).
- Il criterio di espansione di N corrisponde alla selezione dell'operatore da applicare.
- La dimensione dello spazio di ricerca e il costo della soluzione (il percorso dal nodo iniziale al nodo finale) dipende dalle scelte sopra citate.
- La generazione di meno successori ("potatura dell'albero") porterà a uno spazio di ricerca più limitato (ma bisogna fare attenzione a non sfoltire il percorso verso la soluzione).
- Se i nodi candidati (nodi non ancora espansi) sono adeguatamente ordinati in modo ottimale, possiamo arrivare prima alla soluzione o raggiungere una soluzione migliore.

Le euristiche vengono utilizzate in problemi che non possono avere una soluzione esatta (sia per ambiguità nella formulazione del problema sia per ambiguità o incompletezza dei dati disponibili), o per problemi che possono avere una soluzione esatta (algoritmica), ma il costo per trovarlo è troppo alto (ad esempio uno spazio degli stati con un numero di stati che aumenta in modo esponenziale - per l'esplosione combinatoria - e quindi gli approcci alla cieca non sono accettabili entro i limiti di tempo dati).

Le euristiche rappresentano quali criteri utilizzare per scegliere quale opzione (tra quelle disponibili) è più (o la più) conveniente per raggiungere la soluzione.

Di solito, le euristiche non sono esatte e si basano sull'esperienza e sull'intuizione, dipendono dal problema specifico e non sono necessariamente sempre corrette/complete (non sono adeguate per tutti i casi).

Un'euristica è una ipotesi informata/intelligente sulla prossima mossa da eseguire e sfrutta una quantità limitata di informazioni, che di solito è costituita da una certa conoscenza dello stato corrente o degli stati inclusi nella Open List o, in generale, alcune informazioni "locali", cioè non sullo spazio degli stati completo (1000^{40} stati per gli scacchi).

Una buona euristica dovrebbe essere semplice e allo stesso tempo efficace (compromesso). Possiamo essere soddisfatti se l'euristica ci porta a una soluzione subottimale, ma, in alcuni casi, potrebbe non portare a nessuna soluzione.

L'euristica può essere sfruttata nella ricerca mediante una funzione di valutazione euristica, in grado di associare ad ogni stato un valore che rappresenta quanto è buono il corrispondente percorso verso la soluzione. Questa funzione viene sfruttata ad ogni passo, per selezionare lo stato più promettente da seguire (ovvero il nodo successivo della lista aperta da espandere)

3.2.2 Approcci alla valutazione:

Esistono diversi approcci alla valutazione:

- Scegliere il più grande, corto, costoso, pesante, etc.
- Generare una possibile soluzione e poi valutarla (generate & test)
- Varie modalità di calcolare la distanza dallo stato valutato e lo stato del goal (look forward)
- Best first: espandere lo stato più promettente, analizzando lo stato e le variabili (look forward)
- Hill-climbing: espandere uno stato "chiuso" (vicino a qualche metrica, ad esempio di successore) che è il più promettente
- Branch & Bound: espandere un nodo che ha la più alta probabilità di essere nel cammino soluzione

Ritornando al concetto di trade-off del costo: ci sono tipicamente due obiettivi in conflitto tra di loro:

- minimizzare il costo di ricerca
- trovare una soluzione ottimale

3.3 Esempi:

GRID search: algoritmo in machine learning che cerca esaustivamente attraverso un insieme specificato manualmente di hyperparametri dell'algoritmo target.

Vengono provate tutte le possibili combinazioni dei valori degli hyperparametri, ognuno dei quali è fatto variare in un range finito di possibili valori che sono ritenuti plausibili. Rischia l'inefficienza e non garantisce l'ottimo, ma quantomeno restringe l'attenzione su un range di possibilità più limitato.

NAS - Neural Architecture Search: tecnica di DL basata su un algoritmo di search.

Problema: trovare un'architettura DL che ottimizza la soluzione al problema di classificazione di uno specifico dataset.

Search space: I building block della soluzione sono: vari tipi di layer DL (convolutivi, max pooling, etc.), vari tipi di connessioni tra i layer (skip, etc.), le varie funzioni di attivazione, gli iperparametri, etc.

Search strategy: Vengono generate varie combinazioni (provarle tutte potrebbe richiedere magari 10^{30} diverse possibilità – esempio realistico).

Si restringe lo spazio di ricerca limitando le possibilità, utilizzando alcuni blocchi di soluzione predefiniti, limitando il range dei valori degli iperparametri (grid search).

Per ogni architettura candidata con il metodo descritto si esegue il training e si valutano le prestazioni ottenute.

Interviene il solito trade-off tra spazio "largo" che garantisce soluzione migliore ma allunga i tempi e spazio "tagliato" più ristretto, che velocizza ma può dare soluzioni meno ottimali.

Best-first - concetto generale: mantiene la Lista Aperta (nodi da visitare) e la Lista Chiusa (nodi già visitati, non verranno più visitati, alcuni costituiranno la soluzione).

Ad ogni passaggio, seleziona il nodo più promettente in base a una funzione di valutazione euristica.

- Caso "Look forward": valuta (euristicamente) la distanza dello stato corrente dallo stato dell'obiettivo ed espande il nodo con una distanza minima.
- Caso "Guarda indietro": prende in considerazione (euristicamente) il percorso seguito dal nodo iniziale al nodo corrente.

Strategie look forward: valutano quanto uno stato candidato da espandere è lontano dalla soluzione. In alcune euristiche, si cerca di misurare la distanza dal candidato allo stato goal.

Esempio: gioco del tris (tic-tac-toe): strategia: piazza la X nella posizione con il maggior numero di opportunità di vittoria (euristica).

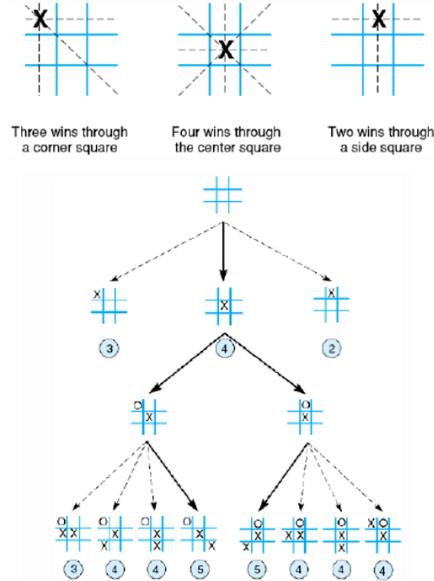
La simmetria permette di ridurre a 3 le possibilità iniziali (le altre erano equivalenti). L'euristica ha ridotto ulteriormente lo spazio di stato a una mossa delle 3 possibilità per la prima mossa. E così via...

Hill-Climbing: Iniziamo con un esempio:

When climbing towards the top of a mountain, a terrible fog appears. I do not see anything more, and I do not have a map.

Ho una bussola e voglio arrivare in cima. Decido di muovermi in una delle 4 possibili direzioni Nord, Est, Sud e Ovest. (in questo modo riduco le dimensioni dello spazio di ricerca).

Possibile strategia euristica: ad ogni passo seleziono, tra le 4 mosse possibili, quella che mi permette di arrivare al livello più alto possibile (guarda avanti: considera solo la distanza



ancora da percorrere, e non la distanza già percorsa: non ricorda il percorso già percorso e non è in grado di correggere errori precedenti).

Se raggiungo un punto che ha intorno solo posizioni inferiori, mi fermo. Può essere efficace, ma esistono alcune situazioni critiche come i minimi locali ("sotto-colline"); plateau, etc. Simile al Gradient Descent.

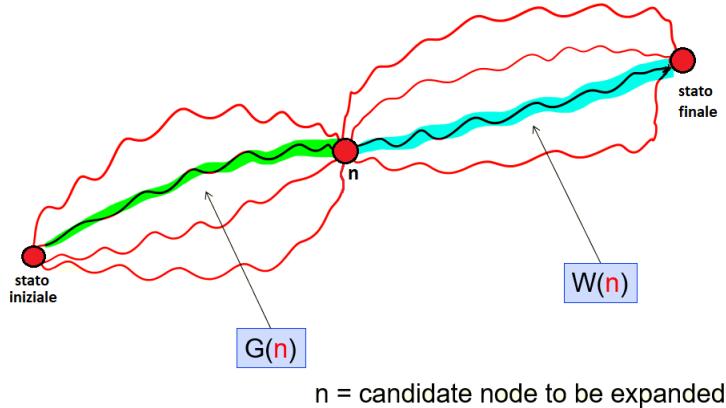
Strategie look backward (Branch & Bound): Considerano anche la distanza già percorsa (dal nodo di partenza al nodo corrente), oltre alla distanza dal nodo corrente alla soluzione. Un algoritmo branch-and-bound consiste in un'enumerazione sistematica di tutte le soluzioni candidate, in cui grandi sottoinsiemi di candidati infruttuosi vengono scartati in massa, utilizzando i limiti stimati superiore e inferiore della quantità da ottimizzare.

Ordered Search: Espandere il nodo con minimo valore di f (corrisponde più o meno a cercare una soluzione in meno mosse). Un particolare algoritmo di Ordered Search è detto **A***.

La funzione di valutazione di A* è: $f(n) = G(n) + W(n)$ dove:

- $G(n)$ è la stima del costo del cammino minimo dal nodo di partenza al nodo n .
- $W(n) =$ stima del costo del cammino minimo tra il nodo n e il goal state.

In altre parole $f(n)$ è la stima del cammino di costo minimo che deve includere n . Il nodo espanso per primo sarà quello con valore minimo n .



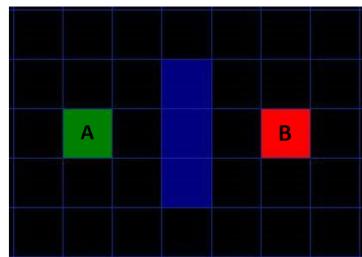
Si può dimostrare che l'algoritmo A* ha la massima efficienza in ricerca e che trova il cammino di costo minimo. Identifica, in poche parole, un cammino ottimale.

Esempio:

Problema: andare da A a B. I due punti sono separati da un muro che io posso vedere solo se sono adiacente ad esso.

Euristica (assunzione di semplificazione): lo spazio è suddiviso in una griglia. L'assunzione su come lo spazio sia rappresentato rappresenta un'astrazione.

In tale modo lo spazio di ricerca è significativamente ridotto e possiamo operare con un numero gestibile di punti nel suolo.

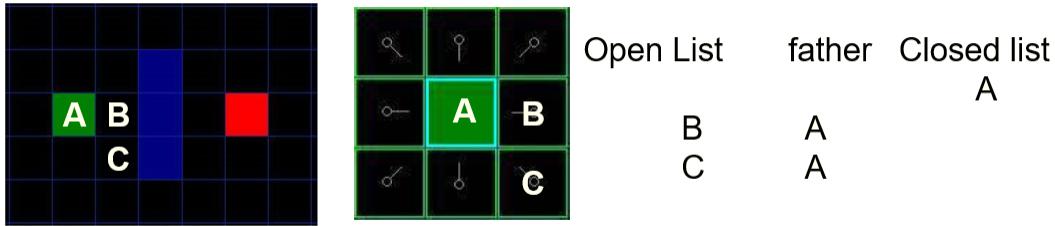


1. Inizia dalla casella A. Aggiungi A all'elenco aperto. Questa lista è come una lista di cose da fare: all'inizio contiene solo A, altre caselle arriveranno dopo. L'elenco include i quadrati che possono essere sul percorso cercato, ma non necessariamente, quindi deve essere controllato.
2. Individua ora tutte le caselle sulle quali puoi muoverti con un solo passo (la zona del muro o altre zone vietate sono quindi escluse). Aggiungi tali quadrati all'elenco aperto. Per ciascuno di essi, memorizza il quadrato "padre" (il quadrato da cui puoi spostarti). Queste informazioni saranno utili in seguito.
3. Togli il quadrato A dalla Open List e inseriscilo nella Closed List, che contiene i nodi che non dobbiamo più visitare.

4. Seleziona uno dei quadrati dell'elenco aperto ed elaboralo nello stesso modo in cui è stato fatto per A. Qui viene applicata la funzione di valutazione per la migliore prima selezione.

```
//A* Search
A* Search (Root_Node, Goal)
{
Create Queue Q
Insert Root_Node into Q
While (Q_Is_Not_Empty)
{
    G = Remove from Q
    Mark G visited
    If (G= goal) Return the path from Root_Node to G;
    Else
        Add each child node's estimated distance to current distance.
        Insert the children of G which have not been previously visited into the Q
        Sort Q by path length
    } // end while
Return failure
}// end of A* function.
```

Esempio esplicativo (su 3 nodi, per capire):



Il piccolo trattino al centro di ogni quadrato indica il quadrato padre. Quando aggiungo un quadrato alla Open List, controllo se il quadrato non è già presente, e se è presente, controllo se il nuovo percorso per quel nodo è migliore del precedente (cioè ha un inferiore $G(n)$). In tal caso, vengono memorizzate le informazioni sul costo inferiore. (cioè è stato trovato un percorso a basso costo per il nodo).

Come vengono calcolati i nuovi stati?

Per ogni nodo nella Open List calcoliamo: $F = G + W$

Dove G = stima del costo per spostarsi da A alla casella considerata. Stima, poiché abbiamo astratto, ci spostiamo sui 'quadrati', non sul terreno reale.

W = stima costo del percorso per andare dal quadrato considerato alla destinazione finale B. W è un'euristica, poiché rappresenta una stima incerta: non conosciamo la distanza finché non conosciamo esattamente il percorso e, inoltre non sappiamo se ci sono ostacoli, muri, etc.

Come stimiamo le distanze?

Assegnamo ad ogni tipo di mossa un costo stimato: 10 per verticali e orizzontali e 1.41... per i movimenti diagonali.

In generale, G è calcolato come il costo del padre +10 o +14 a seconda del tipo di mossa. Per quanto riguarda H, usiamo la distanza Manhattan (euristica): totale n. di mosse (verticali e / o orizzontali) necessarie per raggiungere la destinazione finale, ignorando le mosse diagonali e ignorando eventuali ostacoli (che non conosciamo). Quindi lo moltiplichiamo per 10 (costo di un movimento orizzontale o verticale).

È chiaro che questa distanza è un'approssimazione, poiché, ad esempio nel nostro caso, non tiene conto del muro (che a-priori non so né dove!).

FINIRE, RIVEDERE

3.3.1 Funzione di valutazione qualitativa: La funzione di valutazione può contenere anche conoscenze espresse in modo diverso. Gli esempi mostrati includevano formule numeriche e quantitative, ma potremmo anche usare regole di produzione o altri schemi di KR e di Reasoning così come metodi ML/DL.

3.3.2 Limiti dell'approccio di ricerca nel problem solving: La risoluzione dei problemi e la ricerca sono tecniche classiche nell'intelligenza artificiale. Hanno iniziato prima dei metodi KR. Per diversi motivi, non sono stati considerati di grande successo (con poche eccezioni). Questo fatto ha aperto la strada ai metodi KR. I principali problemi dell'approccio di ricerca sono i seguenti:

- Enorme dimensione dello spazio di ricerca
- Funzioni di valutazioni: risolvo la ricerca blind ma:
 - Non sono semplici da ideare (si cerca di ridurre il problema legato alla conoscenza alla semplice esecuzione di una formula...)
 - Problemi ad-Hoc specifici
 - Problemi di minimo locale
 - Tipicamente non cognitivamente plausibili

Queste sono le principali ragioni per cui l'approccio non fu molto di successo e fu superato dai metodi KR, metodi basati su euristiche (CBR,...) etc.

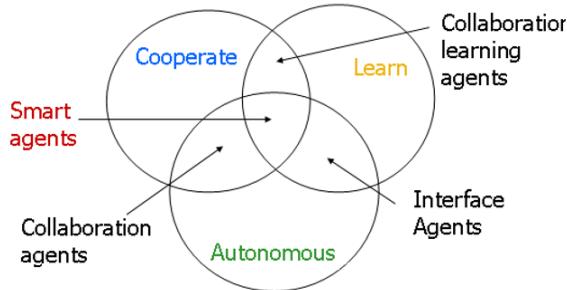
Con l'attuale ritorno di metodi sub-simbolici (Machine Learning) le cose stanno cambiando. ML&DL sfruttano alcuni approcci generali derivati dal problem solving e ricerca. Ad ogni modo, insieme alla logica, questo approccio rimane una referenza molto importante e elegante.

Human problem solving: vediamo gli step:

1. **Scomposizione:** scomporre un problema in sottoproblemi più semplici, di cui si conosce la soluzione e che si risolvono indipendentemente.
2. **Ricerca di pattern specifici:** schemi specifici con i quali abbiamo familiarità e sui quali sappiamo come procedere. Se raggiungiamo uno schema noto, è facile procedere da esso.
3. **Applicazione di regole "pratiche":** regole che suggeriscono azioni, - di solito, ma non sempre - possono essere utilizzate in alcune circostanze specifiche. A differenza del pt. 2, tali regole sono applicate in modo opportunistico, se e quando si verificano alcune condizioni, quindi in modo meno pianificato rispetto al caso 2: situazioni semplici ma risultati non garantiti.

4 DAI - Distributed Artificial Intelligence

Si parla di DAI quando si ha distribuzione del processo di risoluzione dei problemi, ovvero molti risolutori/attori/più **agenti** che cooperano insieme per risolvere un problema. Questo comporta numerosi problemi: cooperazione, interazione, comunicazione, autonomia/indipendenza, gerarchia, ridondanza, interfacce, protocolli, coordinazione, etc.



A partire dal 1975 circa, l'Intelligenza Artificiale Distribuita (DAI) si è sviluppata rapidamente e ha avuto interazioni (bidirezionali) con diverse altre discipline (dalla sociologia all'economia, dall'ingegneria alla filosofia).

DAI è lo studio, la costruzione e l'applicazione di sistemi multi-agente, cioè sistemi in cui diversi agenti interagenti e intelligenti perseguono una serie di obiettivi o eseguono una serie di compiti [Gerhard Weiss, 2005]

Le principali aree dell'informatica coinvolte sono:

- AI
- Ingegneria del software
- Sistemi distribuiti

Esempio: Architettura distribuita ”BlackBoard”, ovvero: un’ organizzazione Architettonica con un repository centralizzato (BlackBoard) condiviso da più moduli (Knowledge Source - KS) che non comunicano direttamente fra di loro e sono in grado di risolvere parte del problema, attivandosi in modo opportunistico, in base al contenuto della BlackBoard.

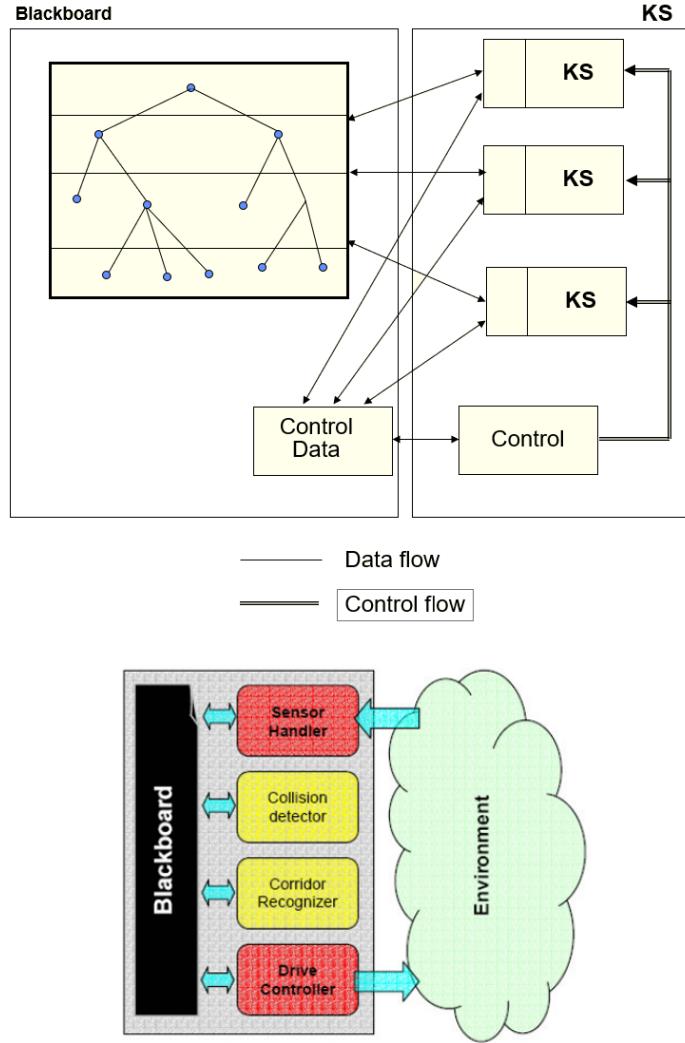
4.1 Architetture multi-agente distribuite

Un agente è un sistema situato in qualche ambiente. Un sistema multi-agente è un insieme di agenti che comunicano e interagiscono in un certo ambiente. Ciascun agente opera su una specifica parte dell’ambiente. Gli agenti possono sovrapporsi e possono diventare indipendenti. Esempio con l’architettura blackboard:

L’agente è capace di percepire/controllare e modificare/influenzare (tipicamente in modo parziale) l’ambiente per via di azioni autonome al fine di completare il suo scopo.

Un ambiente può essere di diverso tipo:

- **Accessibile:** completo, accurato e con informazioni aggiornate; o parzialmente accessibile/osservabile.



- **Deterministico:** l'effetto di una azione è conosciuta a priori; o stocastico (non deterministico).
- **Episodico:** o sequenziale (lo stato dell'ambiente dipende dagli stati precedenti).
- **Statico:** cambia solo dopo che l'agente ha eseguito un'azione; oppure è dinamico.

Esempio di un singolo semplice agente: ciascun sistema di controllo può essere visto come un agente. Un termostato digitale ad esempio:

- Percepisce la temperatura T dell'ambiente e verifica se T è sopra o sotto dei dati thresholds.
- Modifica lo stato dell'ambiente accendendo o spegnendo il dispositivo di riscaldamento.

Altri esempi: SW daemons o triggers dei database.

Un agente è intelligente se il processo di selezione da eseguire è svolto massimizzando qualche indicatore di performance (se ad esempio sfrutta tecniche KR).

Reattività: gli agenti intelligenti sono in grado di percepire il loro ambiente e rispondere in modo tempestivo ai cambiamenti che si verificano in esso al fine di soddisfare i loro obiettivi di progettazione.

Se l'ambiente è accessibile, deterministico e statico, possiamo progettare a-priori l'agente includendo in esso tutte le conoscenze necessarie (un sistema funzionale I/O standard), tuttavia:

- Spesso l'ambiente è dinamico (modificato anche dagli altri agenti)
- L'ambiente si evolve in modi "casuali" (modi che non possono essere previsti in anticipo)
- Le precondizioni/assunzioni vere all'inizio diventano false o cambiano nel tempo

Esempi: robot ambulante, agente di guida autonoma, etc.

Proattività: La reattività non è sufficiente. Vogliamo che gli agenti agiscano per nostro conto. Essi devono essere capaci di prendere iniziative, in modo mirato (non solo comportamento guidato dagli eventi (reattivo)). Gli agenti intelligenti sono in grado di esibire un comportamento diretto all'obiettivo prendendo l'iniziativa per soddisfare i loro obiettivi di progettazione.

Esempio reattivo: termostato

Esempi proattivi:

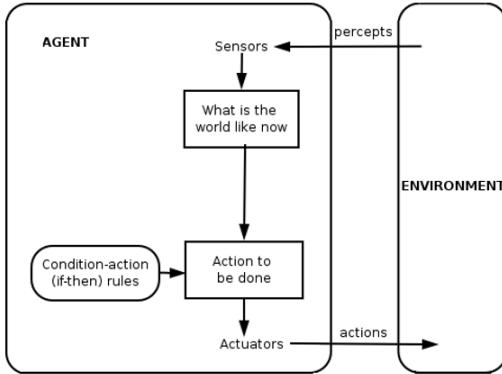
- Assistente personale: conosce preferenze, dove siamo, e ci suggerisce, ricorda, avvisa, etc.
- Home Automation System: ad esempio dedicato agli anziani o ai malati: monitora, vede dove siamo, verifica se ci stiamo comportando in modi diversi dal solito, conosce i nostri obiettivi e progetti, avvisa i parenti, invia allarmi, monitora sensori, segue strategie, imposta dispositivi e sensori, etc.

Gli agenti intelligenti sono in grado di interagire con altri agenti (e possibilmente umani) per raggiungere i loro obiettivi di progettazione, tramite un linguaggio di comunicazione.

4.2 Architetture per agenti intelligenti

Le architetture degli agenti includono le entità (ambiente, agenti, etc.) e le funzioni (percezione e azione) che costituiscono un sistema multi-agente. Le architetture astratte devono quindi essere implementate per mezzo di vari strumenti software.

Esempio di un agente reattivo:



Alcuni concetti: Sensore va inteso in senso generale; l'ambiente infatti non è necessariamente un ambiente fisico: può essere un ambiente "IT" "virtuale", con archivi, repository, moduli di elaborazione, sistemi sw e qualsiasi dispositivo in grado di fornire e ricevere dati e informazioni in forma digitale. L'esplosione dell'era dell'IoT: moltissimi nuovi sensori compaiono ogni giorno.

Ambiente, azioni e percezioni

L'ambiente può essere descritto, in un dato istante, da uno stato. Può trovarsi in uno qualsiasi di un insieme finito S di stati discreti/istantanei:

$$S = s_1, s_2, \dots$$

Ogni ambiente è associato a un insieme di azioni che sono in grado di trasformare l'ambiente dallo stato corrente a un nuovo stato:

$$A_c = a_1, a_2, \dots$$

Una funzione di transizione di stato (solitamente non deterministica) specifica come un'azione cambia lo stato.

L'informazione che un agente può percepire sull'ambiente è rappresentata per mezzo di un insieme di percetti (input percettivo; input/dati percepiti).

La capacità di un agente di percepire l'ambiente consiste nella sua capacità di associare le percezioni allo stato dell'ambiente. L'insieme delle percezioni di un termostato potrebbe essere:

$$Per = OK, NOT OK$$

oppure:

$$Per = OK, NOT OK/sopra, NOT OK/sotto$$

La funzione di percezione di un agente è un mapping tra gli stati S dell'ambiente e i percetti Per

$$See : S \rightarrow Per$$

In un termostato, ad esempio:

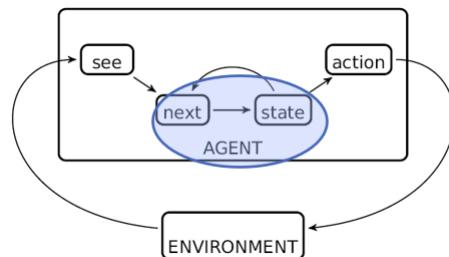
$See = OK$ se la temperatura dell'ambiente è OK , altrimenti $NOT\ OK$

Le decisioni rispetto l'azione da intraprendere dipendono dalla sequenza dei percetti precedenti. La funzione di azione di un agente mappa ciascuna possibile sequenza di percetti in un'azione.

$$Action : Per^* \rightarrow Ac$$

Agente con stati

Un agente con stato è un agente caratterizzato da uno stato interno, che rappresenta la storia dell'agente e dell'ambiente fino all'istante corrente. L'agente tiene traccia dello stato percepito dall'ambiente e delle azioni compiute attraverso lo stato interno. I percetti aggiornano lo stato interno e l'azione selezionata dipende dai percetti e dallo stato interno.



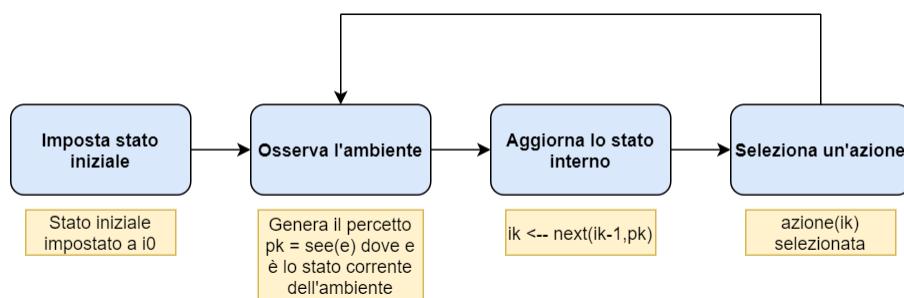
La selezione dello stato dipende dallo stato attuale: l'agente è in uno dei suoi stati interni (insieme I). La funzione di percezione non cambia, ma la funzione d'azione ora dipende anche dallo stato interno.

$$Action : I \rightarrow Ac$$

L'aggiornamento dello stato interno avviene per mezzo dei percetti, in base a una funzione di update. Questa funzione associa un nuovo stato ad uno stato interno e a un percepito:

$$Next : I \times Per \rightarrow I$$

Ciclo di controllo di un agente:



Le migliori/peggiori capacità e competenze di un Agente, l'intelligenza di un agente sono 'racchiuse' nella funzione che decide quale prossima azione svolgere, in base all'ambiente, allo stato, e al fine di raggiungere lo specifico scopo dell'agente stesso; nella funzione azione.

L'azione può essere scelta principalmente seguendo due approcci:

- **Knowledge-based agents:** in cui il processo decisionale è implementato in una qualche forma di ragionamento basato sulla conoscenza per lo specifico compito di risoluzione dei problemi coinvolto dall'obiettivo dell'agente.
- **ML based self-learning agents:** in cui l'agente impara in autonomia a rispondere all'ambiente e ad altri agenti (concorrenti), mediante tecniche di ML (tra queste, RL - Reinforcement Learning); Esempi: giochi per computer, robot autonomi, ecc.

4.2.1 Self-Sufficiency: L'autosufficienza è la capacità di un attore di prendersi cura di se stessa. I principali elementi coinvolti in un agente autosufficiente sono:

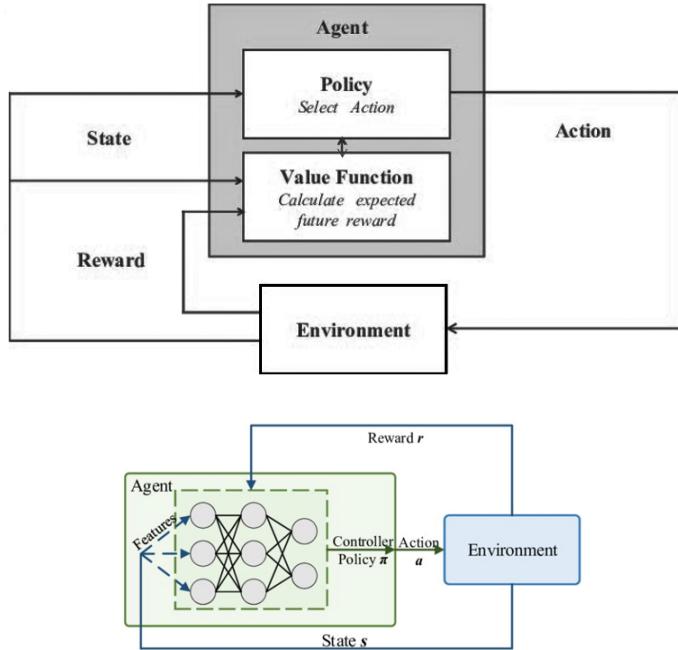
- **Stato:** la situazione attuale dell'agente.
- **Azione:** la decisione o la mossa intrapresa dall'agente di cambiare il suo stato attuale in quello futuro, al fine di raggiungere il suo obiettivo (come l'obiettivo di un veicolo autonomo, di un robot che cammina, di un giocatore di un robot RoboCup, etc.).
- **Ambiente:** il luogo in cui l'agente interagisce ed esegue l'azione, l'ambiente può essere fisico o virtuale.
- **Agente:** Quello che addestriamo (nel nostro problema di Reinforcement Learning) per affrontare l'ambiente ("affrontare", cioè guidare, camminare, giocare a calcio, etc.).
- **Ricompensa:** il feedback fornito all'agente dopo il passaggio dallo stato attuale a quello futuro. Il segnale di feedback è scalare. La ricompensa è maggiore più la mossa aiuta a raggiungere meglio l'obiettivo, in altre parole, l'obiettivo è massimizzare i premi futuri totali.
- **Politica:** la funzione di azione. I criteri/strategia che un agente adotta per spostarsi da uno stato all'altro e raggiungere il proprio obiettivo.
- **Funzione valore:** una funzione che stima la ricompensa che un agente otterrebbe intraprendendo un'azione nello stato corrente per passare allo stato futuro successivo. Un modo per selezionare l'azione migliore in un dato momento e stato (sia lo stato dell'ambiente che lo stato interno).

Esempio di architettura autosufficiente:

Si parla di **Deep Learning Reinforcement Learning** quando un agente computazionale impara a fare decisioni per trial and error.

L'approccio di Reinforcement Learning considera il problema di un agente computazionale che da solo è in grado di imparare a prendere decisioni per tentativi ed errori.

Deep Reinforcement Learning (Deep RL) incorpora il deep learning nella soluzione, consentendo agli agenti di prendere decisioni da dati di input non strutturati (la variabile che descrive l'ambiente) senza l'ingegneria manuale dello spazio degli stati.



Gli algoritmi Deep RL sono in grado di accettare input molto grandi e decidere quali azioni l'agente deve eseguire per ottimizzare un obiettivo.

L'apprendimento per deep learning è stato utilizzato per una serie diversificata di applicazioni tra cui, a titolo esemplificativo ma non esaustivo, robotica, videogiochi, elaborazione del linguaggio naturale, visione artificiale, istruzione, trasporti, finanza, sanità etc.

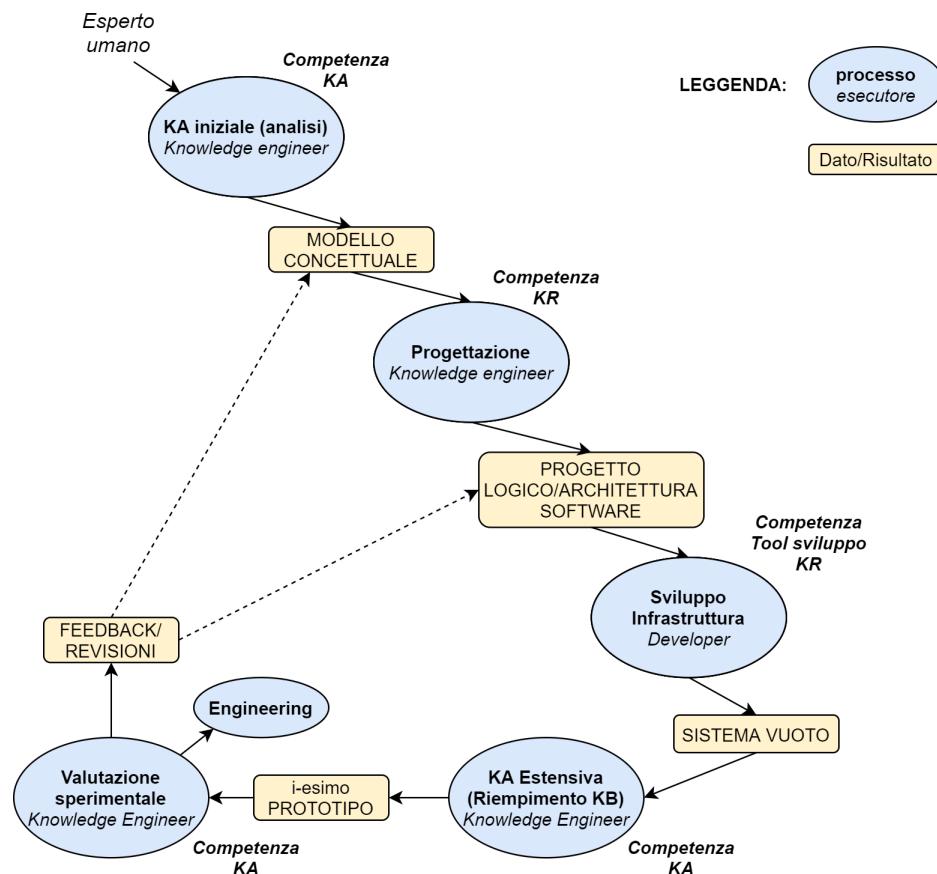
5 Knowledge Modeling and Diagnostic Reasoning

La modellazione della conoscenza è un processo di creazione di un modello di conoscenza interpretabile dal calcolatore.

Gli step nella progettazione di un KBS sono:

1. Analisi del processo di risoluzione del problema: acquisizione della conoscenza (Initial-KA);
2. Costruzione di un modello concettuale del dominio e del processo di problem solving;
3. Design architettonale e costruzione dell'infrastruttura;
4. Riempimento della KB: Acquisizione della conoscenza (Extensive-KA)
5. Completamento, ingegnerizzazione, deployment, etc.

Recap su grafico:



5.0.1 Livello di conoscenza

Secondo Newell esistono due diversi livelli nella rappresentazione della conoscenza:

1. Livello della conoscenza
2. Livello simbolico

Questo permette di separare chiaramente i due casi:

1. Livello della conoscenza (world oriented view): i metodi generici sfruttati dall'uomo per il problem solving, ovvero le strategie (indipendenti dal dominio) che vengono sfruttate per derivare la soluzione ad un dato task applicativo. (ndr: modellazione della conoscenza, modellazione cognitiva, analisi del compito cognitivo, etc.). Un'astrazione separata dai dettagli di implementazione.
2. Livello simbolico (system oriented view): i meccanismi di rappresentazione specifici (linguaggi, simboli, regole di sintassi, etc.), che sono associati ai dati specifici e agli algoritmi di ragionamento sfruttati.

5.1 Knowledge Modeling

Iniziamo elencando le principali classi di task svolte da un KBS:

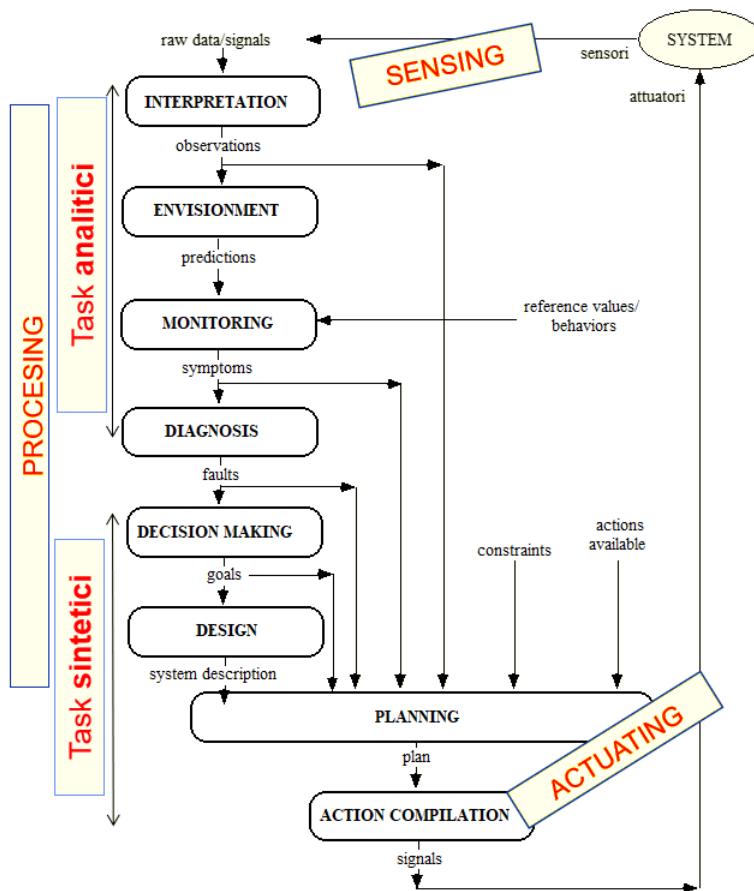
- **Task analitica:** finalizzata all'identificazione di proprietà sconosciute di un sistema descritto attraverso un dato insieme di conoscenze riguardanti la sua struttura, comportamento, funzione e scopo;
- **Task sintetica:** finalizzata a definire una descrizione strutturata di un sistema al livello di astrazione e granularità desiderato, a partire da un dato insieme di elementi costitutivi di base e da un insieme di proprietà desiderate.

Più nello specifico abbiamo:

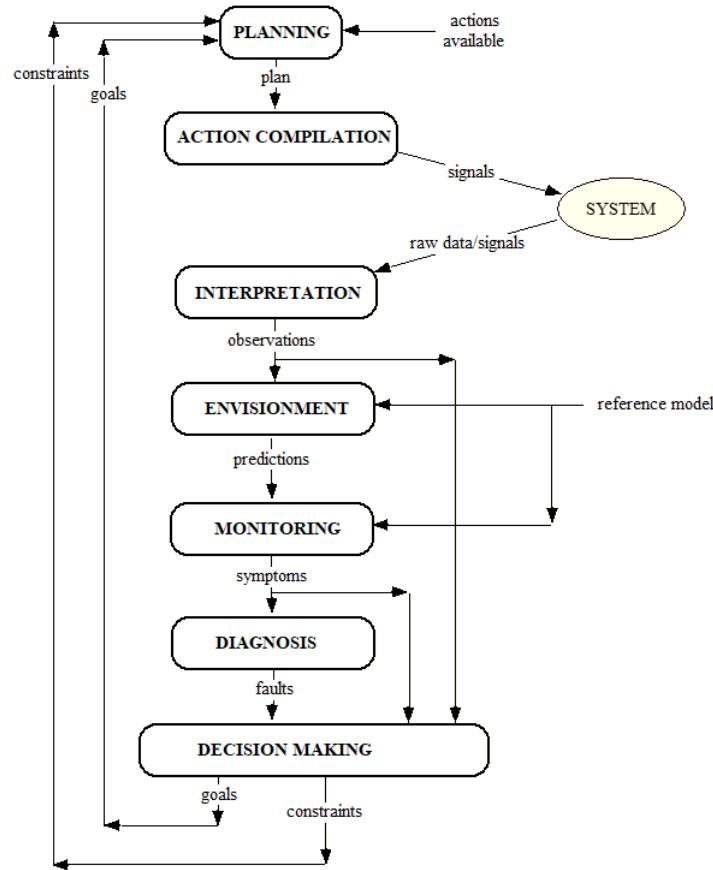
- **Interpretazione/classificazione dei dati**, finalizzata a trasformare i dati grezzi e sensoriali (segnali) in una rappresentazione simbolica (chiamati osservazioni) che descrive in modo significativo un sistema.
- **Simulazione**, finalizzata alla generazione di previsioni (valori o comportamenti attesi) su un sistema, dato un modello di riferimento e uno stato iniziale del sistema.
- **Monitoraggio**, finalizzato a rilevare discrepanze (chiamate sintomi) tra previsioni e osservazioni effettive, o tra osservazioni e valori o comportamenti di riferimento.
- **Diagnosi**, finalizzata all'individuazione delle possibili cause (chiamate colpe) dei sintomi osservati.
- **Processo decisionale - Decision making**, finalizzato all'individuazione degli obiettivi per un nuovo stato del sistema (possibilmente non caratterizzato da sintomi o guasti).

- **Progettazione**, mirata a sviluppare la descrizione di un sistema (design - progetto) in termini di componenti e struttura, data una specifica delle possibili componenti da usare e l'obiettivo da raggiungere o qualche vincolo da soddisfare.
- **Pianificazione**, mirata a costruire un descrizione strutturata temporanea (piano) delle azioni necessarie per raggiungere un goal prestabilito, dato un insieme di risorse e vincoli.
- **Action compilation**, mirato a trasformare il piano in sequenze di segnali (comandi) che influenzano il sistema direttamente.

Esempio di KB predictive maintenance



Planning:



5.2 Metodi AI per la diagnosi

Sottocampo dell'intelligenza artificiale che si occupa dello sviluppo di algoritmi e tecniche in grado di determinare se il comportamento di un sistema è corretto. Se il sistema non funziona correttamente, l'algoritmo dovrebbe essere in grado di determinare, nel modo più accurato possibile, quale parte del sistema si sta guastando e quale tipo di guasto si trova ad affrontare. Il calcolo si basa su osservazioni, che forniscono informazioni sul comportamento corrente.

Definizioni di base:

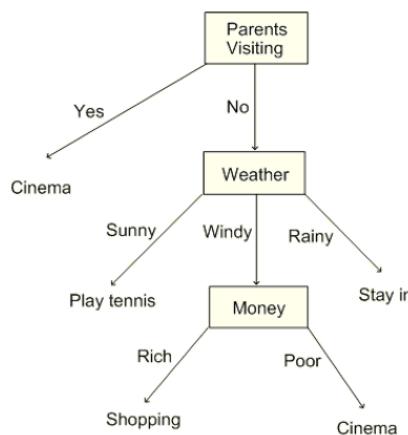
- **Osservazione:** informazioni o dati che descrivono un sistema (staticamente e/o dinamicamente), ottenute direttamente o attraverso misurazioni o attraverso interpretazioni di misurazioni.
- **Comportamento atteso:** stato o evoluzione del sistema che corrisponde al comportamento normale.
- **Sintomo:** discrepanza tra osservazioni e comportamento atteso.

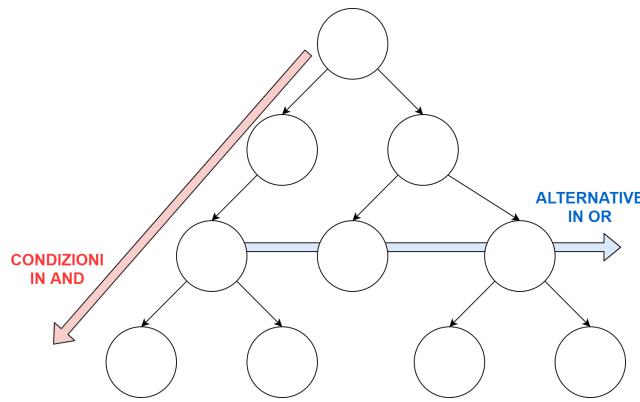
- **Guasto:** causa del sintomo.
- **Diagnosi attiva:** dato un insieme di sintomi, trova un insieme di guasti che giustificano i sintomi.

5.2.1 Modelli/Approcci possibili alla diagnosi

- Alberi di decisione
- Tabelle di decisione
- Fault tree (alberi di guasti)
- Approcchi KB:
 - Fault models e ragionamento abduttivo
 - Classificazione semplice
 - Classificazione basata su euristiche
 - CBR
- Tecniche Machine Learning
 - ML supervised tradizionale
 - Deep Learning

Alberi di decisione: Algoritmo deterministico per decidere quale variabile testare successivamente, in base alle variabili testate in precedenza e ai risultati della loro valutazione, fino a quando è possibile determinare il valore della funzione.



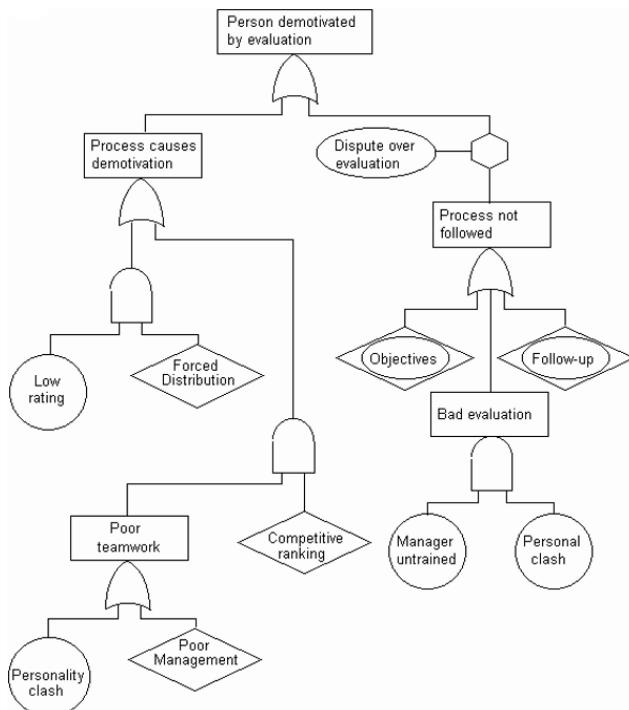


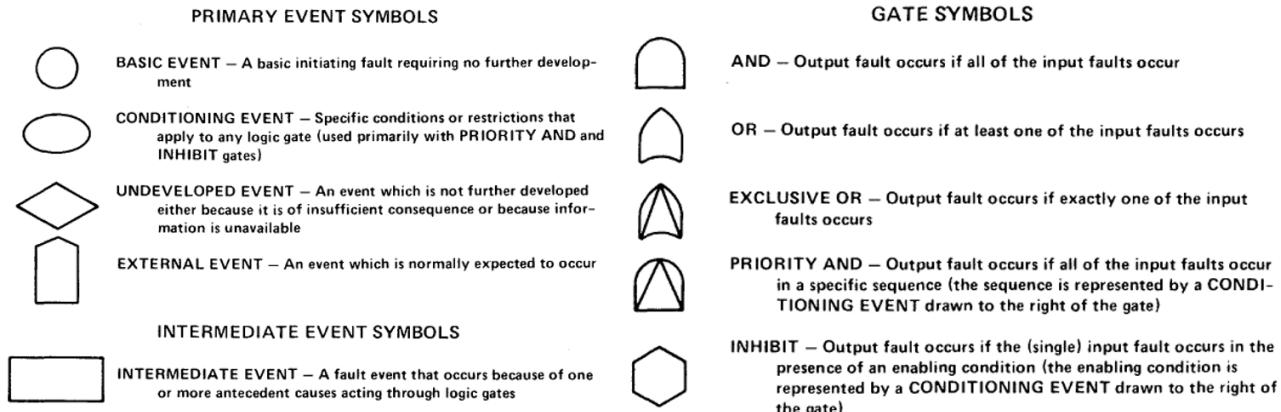
Machine Learning e alberi di decisione

Gli alberi decisionali possono essere appresi automaticamente tramite diverse tecniche di apprendimento automatico (ad esempio, apprendimento induttivo a partire da esempi).

Fault Trees - FTA

Un Fault Tree è una struttura ad albero in cui uno stato di sistema indesiderabile viene analizzato (ricorsivamente) in termini (di combinazioni) di eventi (stati, stati patologici intermedi, osservazioni, etc.) di livello inferiore (cioè che sono effettivamente le cause dello stato indesiderabile). Codificano tipicamente legami causa-effetto.





Da metodi deterministici ad approcci KB

I modelli visti fino ad ora sono deterministici (il controllo è deciso a priori, inserito nel modello), poco flessibili, di non facile modifica e manutenzione, inadeguati in situazioni complesse (cioè livello di ramificazione molto alto). Dobbiamo separare la conoscenza dal controllo. Per fare questo abbiamo bisogno di identificare/isolare/acquisire/modellare la conoscenza diagnostica.

Esistono diverse tipologie di associazioni:

- Causa-effetto
- Osservazioni – sintomo-guasto
- Osservazioni - insieme di sintomi co-occorrenti
- Sintomi - stati intermedi guasti
- Difetto - sintomi
- Comportamento atteso - comportamento guasto
- ...

Molti modelli di ragionamento diagnostico sfruttano tali associazioni.

Modelli di guasto

Permettono di rappresentare la conoscenza su come un componente diventa guasto, come si comporta quando lo diventa, quali sono le conseguenze, etc.

In altre parole, un modello specifica il processo tale che:

IF guasto X **THEN** succede A, B, C

Diagnosi attraverso ragionamento abduttivo: La diagnosi abduttiva cattura il ragionamento comune nella diagnosi (medica), dove spiegare un sintomo (effetto) di solito significa trovare un insieme di cause che implicano il sintomo stesso. Un'ipotesi spiega quindi un sintomo non essendo coerente con esso, ma sottintendendolo logicamente. Perché ciò sia possibile, tuttavia, il modello di sistema deve descrivere cosa accade in presenza di un guasto.

Modelli KB generici per il ragionamento diagnostico

Semplice classificazione: identificare uno o più fenomeni sconosciuti come membri di una classe nota di oggetti, eventi, processi, etc. Di solito, le classi sono stereotipi, organizzate gerarchicamente, e il processo di identificazione consiste semplicemente nel far corrispondere le caratteristiche osservate dell'oggetto sconosciuto con le caratteristiche delle classi.

I dati sono parte della soluzione, l'oggetto classificato è la soluzione. In altri termini, la soluzione è caratterizzata da caratteristiche osservabili.

È un processo di selezione da un insieme di soluzioni, note a priori.

È direttamente fattibile se le soluzioni sono caratterizzate da aspetti osservabili. In altre parole, i dati osservabili caratterizzano e identificano le soluzioni.

È possibile utilizzare una corrispondenza perfetta o parziale.

È anche possibile sfruttare regole di inferenza al fine di fare inferenza dalle osservazioni e ottenere nuovi dati utili da sfruttare per identificare la soluzione.

Questo è utile se le caratteristiche osservabili non sono direttamente parte della soluzione (i.e. le caratteristiche della soluzione non sono direttamente osservabili), e sono collegate alla soluzione per via di passi inferenziali.

I principali meccanismi di inferenza sono:

- **Astrazione:** inferenza di concetti più generali (muoversi verso l'alto nella gerarchia).
- **Raffinamento:** inferenza di concetti più specifici (muoversi verso il basso nella gerarchia).

Tipicamente i meccanismi di astrazione sono preferiti, dato che la conoscenza umana tende a categorizzare piuttosto che considerare un singolo oggetto.

Tre meccanismi di astrazione sui dati:

- **Astrazione di definizione:** considera le caratteristiche essenziali necessarie e sufficienti: i dettagli della definizione portano all'oggetto definito.
- **Astrazione qualitativa:** le informazioni quantitative vengono mappate in informazioni qualitative.
- **Astrazione di generalizzazione:** spostarsi verso l'alto in una gerarchia di sottotipi.

Classificazione con euristiche: Oltre a una corrispondenza diretta tra dati e soluzioni (o indiretta tramite inferenza e astrazione), che non è sempre possibile, si propone di utilizzare anche associazioni dirette (derivate dall'esperienza chiamate relazioni euristiche) che collegano le due strutture (gerarchiche) di dati e di soluzioni.

Relazioni euristiche: incerte, poco capite, valide in alcuni casi tipici (di solito con assunzioni sottostanti). Riducono la ricerca saltando step intermedi. Rappresentate tipicamente tramite regole.

CBR per la diagnosi: un altro approccio basato sulla conoscenza sfruttato per la diagnosi.

Indicato anche come strategie diagnostiche di riconoscimento di pattern e ragionamento induttivo di schema.

Tecniche ML per la diagnosi: già dagli anni 90 si sono applicate tecniche di Machine Learning Supervisionato alla diagnosi:

- Decison tree
- Reti Neurali
- Regressione Logistica

Dall'analisi di molti casi costituiti da dati taggati, quali ad esempio valori di analisi di laboratorio o sintomi o caratteristiche di immagini diagnostiche (radiografie, ecc.) si costruiscono dei modelli predittivi in grado di classificare nuovi casi.

5.3 Conceptual Modeling di un Dominio di Conoscenza

Ritornando al generale, non più considerando il caso specifico di esempio della diagnostica; parliamo di modellazione concettuale.

L'approccio classico dell'AI consiste nel costruire un modello concettuale del dominio, che descrive tutti i passi del processo (cognitivo) di ragionamento che l'esperto umano segue per risolvere il problema considerato.

Il modello concettuale è una rappresentazione strutturata astratta e completa della conoscenza sfruttata in un dato dominio applicativo che include sia una visione statica ("quali tipi specifici di conoscenza vengono utilizzati?") che dinamica ("come vengono utilizzati?").

Describe:

- Tipologie di entità (classi di conoscenza) coinvolte e loro relazioni.
- Le varie funzioni di ragionamento applicate su di esse.
- La strategia globale di risoluzione del problema (strategia di problem solving).

È indipendente dall'implementazione. È il risultato del KA iniziale e dell'attività di "modellazione concettuale". È il punto di partenza per la progettazione dell'architettura logica di un KBS.

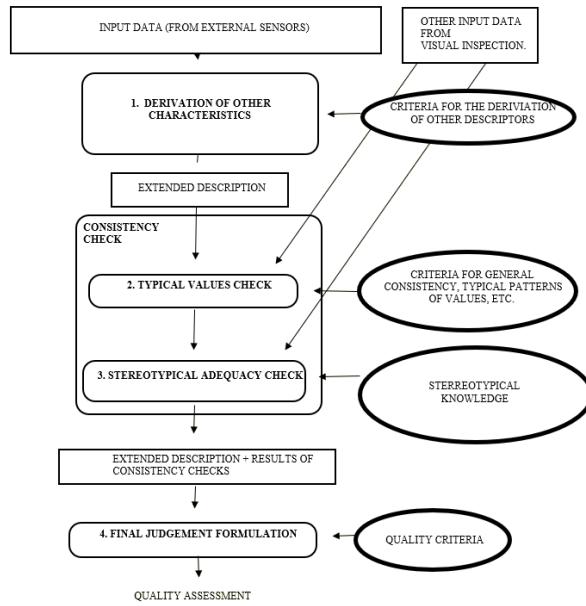
Per la parte **statica**: per ogni tipo di conoscenza: cosa rappresenta, quali entità, quali proprietà, quali relazioni tra entità.

Per la parte **dinamica**: per ogni funzione di ragionamento: Input (quali sono gli input con cui opera, arco entrante), output (cosa viene prodotto, arco uscente), quale trasformazione viene svolta (in termini di processo di ragionamento: deduzione, analogia, generalizzazione etc.). Input e Output intesi come classi di conoscenza.

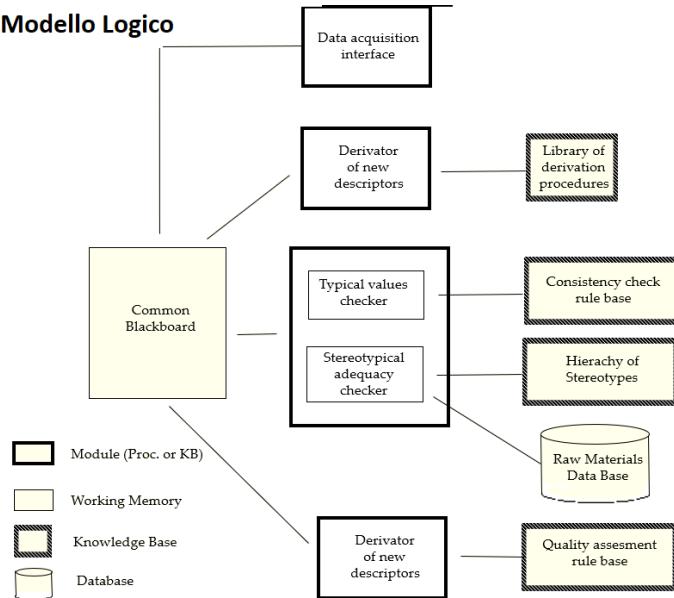
La rappresentazione della strategia di problem solving viene rappresentata tramite strutture di controllo, uso di meta-conoscenza e con relazioni di dipendenza I/O (archi tra le funzioni di ragionamento).

Esempio:

Modello Concettuale



Modello Logico



Le 3 principali nozioni per la costruzione di un modello concettuale sono:

- Tecniche di Knowledge Acquisition and Modeling (KA)
- Tecniche di Knowledge Representation and Reasoning (KR)
- Metodologia di analisi dei processi Knowledge Intensive

Processi Knowledge Intensive: processi la cui esecuzione dell'esperto umano richiede conoscenza non facilmente algoritmizzabile, derivata dall'esperienza, non modellata formalmente.

Grazie al modello concettuale è possibile progettare l'architettura e sviluppare un KBS, sistema SW in grado di risolvere un problema seguendo gli stessi ragionamenti e modalità di problem solving che utilizza un esperto umano. Inoltre, grazie al modello concettuale, si riesce a spiegare quali ragionamenti sono stati fatti e a giustificare le soluzioni ottenute.

6 Ontologie

Un'ontologia è una descrizione dei concetti di un dominio ed è usata per ragionare sugli oggetti in quel dominio e sulle relazioni presenti tra di loro. Le ontologie possono essere sfruttate per la modellazione concettuale.

Per come abbiamo impostato lo studio dei Metodi di KR nel nostro corso, le Ontologie sono una specifica tipologia di KR appartenenti alla Classe delle Reti Semantiche

Definizione di ontologia da parte di Sowa: lo studio di categorie di cose che esistono o possono esistere in qualche dominio. Il prodotto di tale studio, chiamato ontologia, è un catalogo di tipi di cose che si presume esistano in un dominio di interesse D dal punto di vista di una persona L allo scopo di parlare di D. I componenti nell'ontologia sono predicati, sensi di parole o tipi di concetti e relazioni del linguaggio L.

Gli ingredienti di una ontologia sono:

- Individui: gli oggetti di base o "a livello del suolo"
- Classi: insiemi, collezioni o tipi di oggetti
- Attributi: proprietà, funzioni, caratteristiche o parametri che gli oggetti possono avere e condividere
- Relazioni: modi in cui gli oggetti possono essere messi in relazione tra loro (is-a, part-di, etc.)
- Vincoli su individui specifici e sui tipi di relazioni consentiti tra loro

I punti di forza delle ontologie sono l'interpretazione della conoscenza, il riuso, la standardizzazione e interoperabilità.

7 Machine Learning

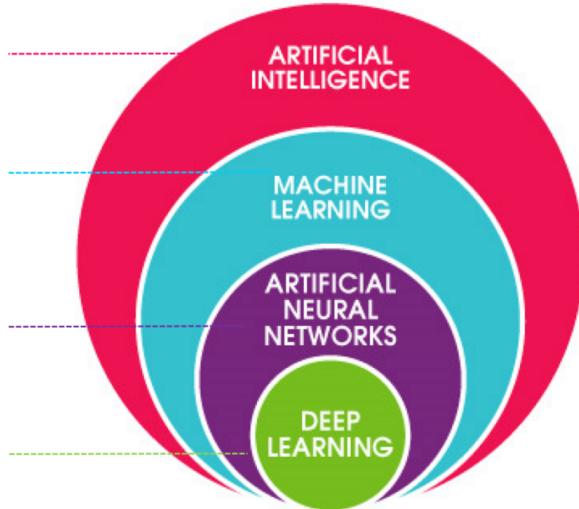
Iniziamo con un recap ricordando la "gerarchia" dell'AI:

AI: studio dei sistemi software intelligenti, capaci di risolvere problemi, capire il significato dei testi e delle immagini, gestire robot, cercare l'informazione in modo semantico, diagnosticare, supportare nelle decisioni, raccomandare, processi knowledge-intensive, sistemi basati sulla conoscenza,

ML: tecniche che, partendo dalle descrizioni di esperienza passate, sintetizzano automaticamente nuova conoscenza utile per costruire modelli predittivi

ANN: tecnica di ML, basata su reti di neuroni artificiali, organizzati in layer, che ricordano l'organizzazione dei neuroni biologici

DL: tecniche di ML basate ANN costituite da molti layer di neuroni artificiali, che rappresentano una gerarchia di elementi del problema, dove i livelli di più alto livello sono definiti sulla base di quelli di basso livello, a partire dai dati elementari in input



Partiamo con una premessa; l'informatica tradizionale si basa sull'elaborazione di dati basandosi su modelli precisi, ben definiti, matematici, con procedure ben definite, talvolta 'per legge'. Si automatizzano le operazioni di attività rutinarie, ripetitive e ben proceduralizzate.

Si sviluppano le applicazioni progettando un algoritmo (programma) che permette l'esecuzione automatica, precisa e deterministica di tutti i passaggi e di tutte le elaborazioni necessarie per portare a termine le procedure di interesse.

Con l'avvento dell'AI e del ML lo scenario cambia profondamente.

I sistemi intelligenti visti fino a questo capitolo si basano sull'approccio classico dell'AI: sistemi basati sulla conoscenza (SBC o KBS). La metodologia classica è la seguente: si costruisce un modello concettuale del dominio, che descrive tutti i passi del processo (cognitivo) di ragionamento che l'esperto umano segue per risolvere il problema considerato.

Per la costruzione di un modello concettuale sono necessari:

- **Tecniche di Knowledge Acquisition and Modelling (KA):** riguardano molteplici tecniche (spesso mutuate dalla psicologia sperimentale) da utilizzare per capire come l'esperto umano ragiona, che elementi considera, che ragionamenti fa, come struttura le varie fasi del processo, ... costruendo il modello concettuale che descrive il processo problem solving dell'esperto.
- **Tecniche di Knowledge Representation and Reasoning (KR):** definiscono come la conoscenza debba essere rappresentata nel calcolatore e come si devono eseguire le varie tipologie di ragionamento.

Grazie al modello concettuale è possibile sviluppare un KBS; un sistema SW in grado di risolvere un problema seguendo gli stessi ragionamenti e modalità di problem solving che utilizza un esperto umano. Inoltre, grazie al modello concettuale, si riesce a spiegare quali ragionamenti sono stati fatti e a giustificare le soluzioni ottenute (trasparenza del sistema).

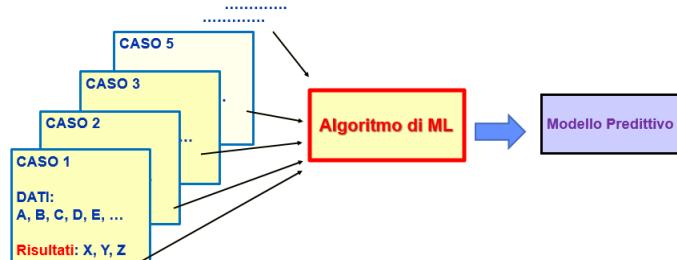
7.1 Introduzione al ML

Il machine learning è un approccio alternativo alla costruzione di moduli SW intelligenti. Le tecniche di Machine Learning lavorano principalmente al livello sub-simbolico; infatti per costruire un sistema ML non è necessario costruire un modello concettuale del dominio e del problema di problem solving da risolvere (passi necessari per costruire un KBS). D'altro canto, il processo di training in ML considera l'input del problema e la sua soluzione al fine di trovare correlazioni tra dati e risultati così da essere in grado di considerare nuovi casi e fornirne la soluzione.

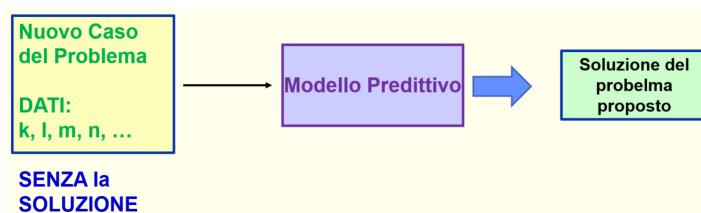
7.1.1 Supervised Learning

Si fanno analizzare al sistema software moltissimi esempi di casi (descritti dai dati che li caratterizzano con le relative soluzioni/decisioni/risultanze) e l'algoritmo di ML costruisce un nuovo Modulo Software, in grado di affrontare casi simili, non già visti in precedenza. Il Modulo Software viene indicato con molti possibili termini. Un termine molto comune è 'Modello Predittivo' (o Classificatore, o semplicemente Modello).

Fase di training:



Fase di utilizzo del modello:



Il calcolatore ha "imparato" autonomamente a risolvere il problema. Dall'analisi dei dati e dalla soluzione di ciascun esempio fornito all'algoritmo di ML, ha "compreso" come trovare la soluzione direttamente dai dati del problema. Qualsiasi sia il dominio è importante avere a disposizione tanti dati relativi ai problemi che si vogliono automatizzare. Più dati ci saranno a disposizione e potenzialmente migliori saranno i risultati.

7.1.2 Definizioni

Una possibile definizione di ML è la seguente:

L'apprendimento automatico (Machine Learning) è un'area dell'Intelligenza Artificiale che si occupa della realizzazione di sistemi che si basano su dati che descrivono osservazioni o esempi o misure (training set) al fine di sintetizzare nuova conoscenza (schemi di classificazione, generalizzazioni, riformulazioni, alberi di decisione, ...) che permette successivamente di risolvere specifici problemi.

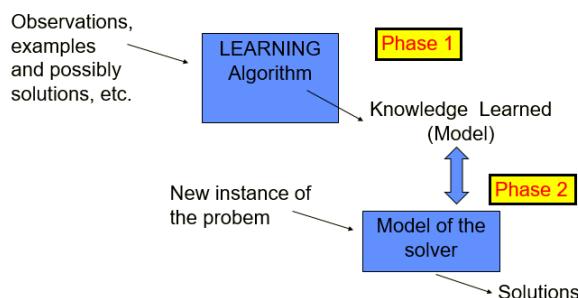
Alcune sotto-definizioni sono:

- L'apprendimento è definito come il processo che consente a un sistema di funzionare meglio nella risoluzione dei problemi di compiti uguali o simili in futuro (Simon, 1983).
- L'apprendimento automatico è un'area fondamentale dell'IA e riguarda lo sviluppo di sistemi e algoritmi che sfruttano le osservazioni come dati, che vengono utilizzati per sintetizzare nuove conoscenze.
- L'apprendimento si basa sulla "cattura di caratteristiche interessanti" estratte da esempi, strutture dati, sensori, al fine di analizzarli e scoprire relazioni con le variabili osservate. (Wikipedia).
- Migliorare le prestazioni/il comportamento attraverso l'esperienza (Mitchell).

Le tecniche di ML hanno l'obiettivo di acquisire/produrre automaticamente conoscenze per l'esecuzione di un determinato compito/problema, partendo da esperienze passate (nel caso del supervised learning), casi specifici, esempi di soluzioni, output attesi, condizioni ambientali, ecc.

Il termine "apprendimento" si riferisce al fatto che tutti gli approcci sono strutturati in 2 fasi:

- Una fase di **apprendimento**, chiamata training, che produce la conoscenza per eseguire un determinato compito/risolvere un determinato problema. In questa fase vengono analizzate le osservazioni (esperienze passate, esempi, casi, ...).
- La fase successiva, in cui la conoscenza prodotta viene sfruttata per risolvere specifiche (nuove) istanze del problema.



7.1.3 Formulazione più dettagliata

La conoscenza sfruttata nel ML riguarda la soluzione di un problema dove un'istanza è solitamente rappresentata per mezzo di alcuni **parametri** (numerici o simbolici) (detti anche attributi/variabili/caratteristiche/features ...) che caratterizzano il problema e - nel caso di ML supervisionato - la soluzione corrispondente (in altre parole l'istanza è pre-classificata, etichettata, labelled): $\langle p_1, p_2, p_3, \dots, p_n, \text{soluzione} \rangle$

Quindi, il ML si occupa di (è adeguato per) problemi che possono essere rappresentati in modo unitario attraverso n-tuple, contenenti sia un insieme di parametri per descrivere il problema e, eventualmente, la corrispondente soluzione.

L'esperienza (osservazioni ed esempi) è rappresentata da molte n-tuple o $(n+1)$ -tuple.

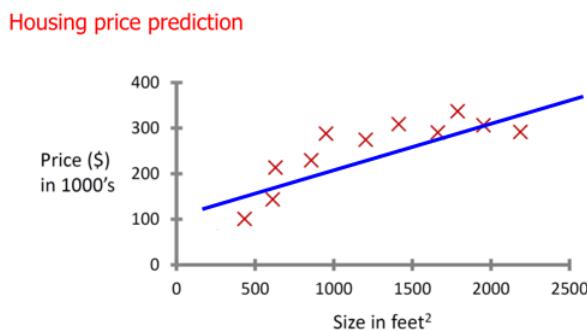
La $(n+1)$ -esima variabile (quella da prevedere) è chiamata variabile target.

7.1.4 Classificazione generale delle tecniche di ML

Abbiamo già visto il caso del supervised learning; più in generale le alternative sono le seguenti:

- **Supervised Learning:** il training è costituito da esempi pre-classificati. Lo scopo è predire i valori per la classificazione di nuovi esempi/casi.
- **Unsupervised Learning:** il training non è costituito da esempi pre-classificati. Lo scopo è scoprire patterns/strutture "nascoste" nei dati non facilmente/direttamente osservabili, cluster di dati, correlazioni, trends, regolarità, anomalie, etc.
- **Reinforcement Learning:** i risultati vengono valutati e viene applicato un rinforzo (feedback), che è "positivo" (in caso di risultati corretti) o "negativo" (in caso di risultati errati). Il rinforzo viene preso in input dall'algoritmo, che lo sfrutta per migliorare le prestazioni, massimizzando alcune misure di "corrispondenza/adattamento" con il rinforzo. Ovviamente non può essere applicato al caso unsupervised.

La statistica ha già proposto delle soluzioni. Per eseguire delle predizioni potremmo approssimare le distribuzioni dei dati eseguendo il fit di modelli statistici come nell'esempio.



ML "as a function": Output = $f(\text{Input})$

- **Supervised Learning:** la funzione viene appresa dai casi di I/O (gli oggetti dell'apprendimento sono: decisioni, regole, classificazioni, ...).

- **Unsupervised Learning:** non si sa nulla dell'output. (gli oggetti dell'apprendimento sono: pattern, schemi, cluster, ...; l'apprendimento non riguarda come calcolare f , ma, d'altra parte, è finalizzato a conoscere meglio x , i dati di input).
- **Reinforcement Learning:** l'apprendimento dipende dal rinforzo ricevuto (ricompensa, feedback sulla pertinenza) che indica i comportamenti e gli stati desiderabili.

I tipici problemi adeguati per i casi supervised learning sono problemi dove si conosce sia i parametri in input (n-tuple) che le corrispondenti soluzioni. Devono essere conosciute molte n-tuple e devono essere etichettate.

Nel caso dell'unsupervised learning lo scopo è di scoprire strutture nascoste, correlazioni, etc. in una collezione di dati strutturati, non classificati a priori (non labellati/etichettati). È importante avere una quantità di dati enorme (big data).

Tipologie di conoscenza apprendibile attraverso il ML

- Associazioni/correlazioni tra entità (soluzioni, frequent itemset, classi, cluster, etc.)
- Regole di decisione
- Alberi di decisione
- Criteri di classificazione (binari o multivariati)
- Valori di una funzione
- Modelli predittivi
- etc.

La definizione appena vista generalizza l'interpretazione del ML come approccio alla costruzione di sistemi intelligenti, ossia in grado di replicare delle capacità di problem solving umano.

L'algoritmo identifica relazioni fra diverse tipologie di dati, valuta somiglianze e diversità fra gruppi di dati, stima valori, associa a delle classi, ecc.

Ma nulla cambia se i dati che analizziamo ed usiamo per il training non sono strettamente relativi ad un processo di problem solving umano, cognitivo. Possono essere dati estratti da una qualsiasi base di dati o da un insieme di basi di dati, anche di natura eterogenea.

Ad esempio il caso del BigData Analytics, nell'ambito del quale grandi masse di dati (la cui analisi manuale è praticamente impossibile) vengono analizzate mediante algoritmi di ML, per costruire modelli predittivi, clustering, ecc.

7.2 Association Rule learning e Frequent Item Set discovery (Un-supervised)

L'algoritmo "A-Priori" è una tecnica unsupervised per identificare regole di associazione. L'algoritmo è basato sulla statistica, e consente di analizzare Data Set strutturati (molto grandi), spesso su transazioni (come Web log di un sito di e-commerce, dati di marketing, dati sui clienti, dati provenienti da sensori e fenomeni simultanei, ecc.).

L'obiettivo è scoprire relazioni interessanti (forse inaspettate).

Una regola di associazione rappresenta un'associazione tra due elementi di dati che è spesso presente nel set di dati.

Ad esempio: Analisi del Web Usage (Mining), per identificare i trend comportamentali dei clienti al fine di fornire un servizio sempre più personalizzato. Cattura le relazioni tra gli articoli, a partire dalla loro presenza nelle transazioni. Web marketing, sistemi di raccomandazione.

Date le sequenze di pagine (o prodotti) visitate dai singoli utenti (pageview), è possibile produrre in output gruppi di pageview che si verificano frequentemente in molte transazioni.

Analisi preliminare di Data Set, per identificare frequenti associazioni di valori.

Esempio di una regola identificata (dominio della navigazione web).

$$\{A.html, B.html\} \rightarrow \{C.html\}$$

Rappresenta la navigazione degli utenti che hanno visitato in sequenza A e B e poi hanno visitato C.

L'insieme contenente C.html è detto **itemset**.

Un **frequent itemset** è un insieme di item che appare frequentemente all'interno di un data set.

Metriche per le regole di associazione

Vediamo ora le metriche per la determinazione delle regole di associazione.

$$\{A.html, B.html\} \rightarrow \{C.html\} [\text{supporto}=0.01, \text{confidenza}=0.75]$$

A sinistra della freccia abbiamo l'antecedente, mentre $\{C.html\}$ è detto conseguente.

- **Supporto** = $P(A,B,C)$: 1% indica che l'itemset $\{A,B,C\}$ è presente solo l'1% del numero totale di sessioni analizzate. Il valore del supporto aiuta a identificare regole che vale la pena considerare in analisi future. Misura di quanto è frequente l'itemset {antecedente + conseguente}.
- **Confidenza** = $P(A,B,C)/P(A,B)$: 75% indica che il 75% delle volte un utente ha visitato la sequenza A,B poi ha visitato anche C.

Quando la frequenza del conseguente è molto alta, anche se l'antecedente ha una frequenza bassa, la confidenza della regola sarà alta. Tuttavia la regola non sarà molto utile.

Per ovviare a questo, viene introdotta la metrica **LIFT**, che "tiene conto" della frequenza del conseguente, calcolando la confidenza. Il LIFT di una regola è il rapporto tra il supporto osservato e quello atteso se X e Y fossero indipendenti.

$$LIFT = P(A, B) / P(A) * P(B)$$

Il LIFT indica quanto la presenza di un antecedente aumenta la probabilità che il conseguente è anche presente.

- $LIFT < 1 \rightarrow$ bassa correlazione di X con Y = alta confidenza insignificante
- $LIFT >>> 1 \rightarrow$ alta correlazione di X con Y = alta confidenza da sfruttare
- $LIFT = 1 \rightarrow$ X e Y sono indipendenti

Esempio:

Il supporto indica quanto frequente l'itemset è presente nel dataset.

Transaction 1	
Transaction 2	
Transaction 3	
Transaction 4	
Transaction 5	
Transaction 6	
Transaction 7	
Transaction 8	

$$\text{Support } \{\text{apple}\} = \frac{4}{8}$$

La confidenza indica quanto spesso la regola è stata trovata "vera": data la mela appare anche la birra

Transaction 1	
Transaction 2	
Transaction 3	
Transaction 4	
Transaction 5	
Transaction 6	
Transaction 7	
Transaction 8	

$$\text{Confidence } \{\text{apple} \rightarrow \text{beer}\} = \frac{\text{Support } \{\text{apple}, \text{beer}\}}{\text{Support } \{\text{apple}\}}$$

(Uno svantaggio della misura di confidenza è che potrebbe travisare l'importanza di un'associazione. Questo perché spiega solo quanto sono popolari le mele, ma non le birre. Se le birre sono anche molto popolari in generale, ci sarà una maggiore possibilità che una transazione contenente mele contenga anche birre, gonfiando così la misura di fiducia. Per tenere conto della popolarità di base di entrambi gli elementi costitutivi, utilizziamo la misura chiamata lift, che misura quanto sono indipendenti le mele e la birra.)

Transaction 1	
Transaction 2	
Transaction 3	
Transaction 4	
Transaction 5	
Transaction 6	
Transaction 7	
Transaction 8	

Lift $\{ \text{apple} \rightarrow \text{beer} \} = \frac{\text{Support } \{ \text{apple}, \text{beer} \}}{\text{Support } \{ \text{apple} \} \times \text{Support } \{ \text{beer} \}}$

Il Lift permette di capire quanto l'antecedente ed il conseguente sono indipendenti tra loro. In altri termini, quanto è probabile che il conseguente (beer) sia acquistato quando l'antecedente è acquistato, tenendo conto di quanto il conseguente è più o meno frequante.

Scenario generale

Dato un Data Set composto da transazioni, ove ciascuna transazione è costituita da un insieme di item (itemset), si cercano situazioni di questo tipo:

Ogni volta che un certo (sotto) insieme X di feature assume una certa configurazione di valori accade con buona probabilità che un altro (sotto) insieme Y di feature assume una certa configurazione di valori.

E ciò con una confidence ed un supporto significativi. (significativi nel contesto applicativo specifico).

Supporto e confidenza minimi

Le regole di associazione sono generalmente richieste per soddisfare un supporto minimo specificato dall'utente, una confidenza minima specificata dall'utente e un incremento minimo specificato dall'utente (> 1) allo stesso tempo.

In altri termini:

- (supporto) alta frequenza della sequenza completa XY
- (confidenza) alta percentuale del caso in cui dato un antecedente X il conseguente Y è accaduto
- (lift) buona probabilità di X dato Y

I valori specifici delle percentuali dipendono dalla specifica applicazioni. Con milioni di transazioni valori come 1% possono essere considerati alti.

A-priori

Dato un itemset, tenta di trovare sottoinsiemi comuni ad almeno un numero minimo C itemset. Apriori utilizza un approccio "bottom up", in cui i sottoinsiemi frequenti vengono estesi di un elemento alla volta (generazione di candidati) e i gruppi di candidati vengono testati in base ai dati. L'algoritmo termina quando non vengono trovate ulteriori estensioni riuscite. Apriori utilizza BFS (breadth first search) e una struttura ad albero per contare in modo efficiente i set di elementi candidati. Genera set di elementi candidati di lunghezza k da set di elementi di lunghezza k-1. Quindi elimina i candidati che hanno un sotto-pattern poco frequente. Successivamente, esegue la scansione del database delle transazioni per determinare i set di elementi frequenti tra i candidati.

Apriori permette di trovare association rules e frequent itemset, utilizza il Data Set ed imponendo dei valori minimi di support, trust, etc.

L'algoritmo è composto da due fasi:

1. Generazione di tutti i frequent itemsets di ogni candidato ma con un supporto sopra una soglia prestabilita.
2. Generazione di tutte le regole di associazione con una confidenza sopra un threshold prestabilito, partendo dai frequent itemsets prodotti in fase 1.

7.3 Inductive Learning Techniques

- Ragionare dal particolare al generale
- Derivare conoscenza generale dall'analisi di casi specifici
- Ragionare a una conclusione su tutti i membri di una classe dall'esaminazione di una sua porzione di istanze
- Identificare regolarità che spiegano un fenomeno

Corrisponde al modello generale di ML introdotto all'inizio.

Ad esempio:

Il cigno 1 è bianco. Il cigno 2 è bianco. → i cigni sono bianchi.

Tuttavia, le conclusioni induttive non possono essere accettate per certe. Future osservazioni potrebbero entrare in conflitto con la generalizzazione ottenuta, e una revisione sarebbe necessaria.

Il ragionamento induttivo produce solo conclusioni 'temporanee' (cioè che sono valide solo se non emergono osservazioni contrastanti in seguito). Tuttavia, il ragionamento induttivo è molto utile per produrre ipotesi di lavoro, permettendo di proseguire con l'attività di ragionamento.

Il ragionamento induttivo è uno dei paradigmi fondamentali di ragionamento della scienza moderna (Fisica, Astronomia, Medicina, ...). Il ragionamento induttivo è la base dell'apprendimento induttivo.

7.3.1 Metodologia generale

- **Training set:** insieme di esempi, forniti (i) dall'esperto o (ii) acquisiti (mediante estrazione e normalizzazione) da un DB o da un sistema informativo, di casi tipici (problematici) nel dominio.
- **Esempio (o istanza):** insieme di attributi e valori corrispondenti (x_1, x_2, \dots). Sono (i) le informazioni (input) considerate dall'esperto per risolvere il problema, oppure i (ii) dati che caratterizzano la situazione che vogliamo analizzare. Gli esempi sono etichettati in base alle (i) diverse soluzioni del problema o (ii) ai diversi risultati delle situazioni ($f(x)$). (supervised learning)
- **L'algoritmo di induzione consente:** la spiegazione delle regolarità nei valori degli attributi. Prevedere la soluzione del problema per i casi non inclusi nel training set (modello predittivo).
- L'output della regressione è una funzione da cui viene calcolato un valore.
- L'output della classificazione può essere costituito dalla conoscenza rappresentata da un valore intero preso da un insieme finito di valori, mediante alberi di decisione o regole, etc.

7.3.2 Training e Test SET

Il set di esempi disponibili (solitamente chiamato dataset) è solitamente suddiviso in:

- **Training set:** costituito da una parte degli esempi disponibili e sfruttato come input per l'algoritmo dedicato all'identificazione della "funzione h che approssima f ". Il set di training può includere, ad esempio, dal 60% all'80% della quantità totale di dati nel set di dati.
- **Test set:** costituito dalla restante parte degli esempi disponibili e sfruttato per valutare l'accuratezza del modello (ovvero il classificatore o la funzione di approssimazione h) nella risoluzione di nuovi casi non visti in precedenza durante il Training. In altri termini, per ogni esempio nel Test Set, i corrispondenti valori x_i (attributi) vengono forniti in input al modello e la soluzione prodotta viene confrontata con l'etichetta precedentemente assegnata a quello specifico esempio. Se corrispondono, la soluzione è corretta, altrimenti non corretta. Il set di test può includere, ad esempio, dal 20% al 40% della quantità totale di dati nel set di dati.

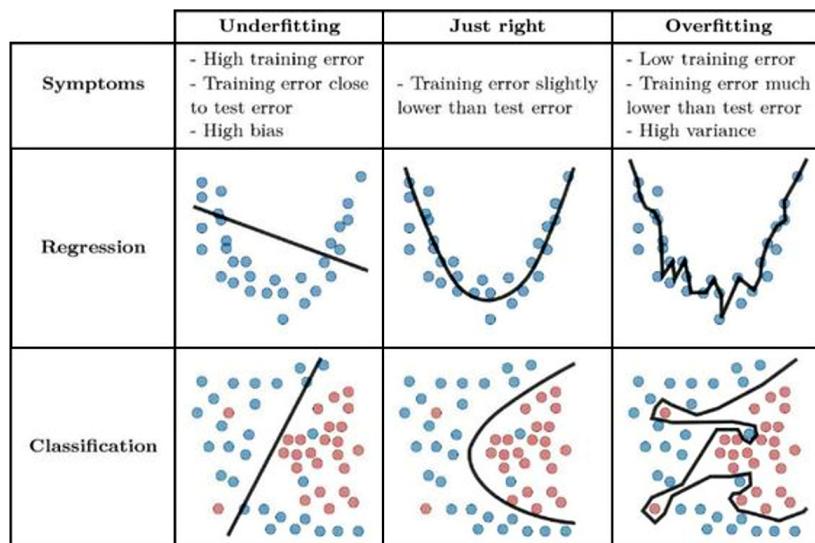
È chiaro che il successo dipende direttamente dai criteri di partizionamento (in training e test set).

Il training set deve essere ben bilanciato su tutti i casi possibili del fenomeno, altrimenti c'è il rischio di un apprendimento parziale, cioè vedo solo una parte del fenomeno, imparo bene una parte del fenomeno e scarto altri casi. In altri termini, so classificare solo in alcuni modi specifici (parziali), ma non nella generalità delle situazioni possibili.

Tale situazione/problema è chiamato **overfitting**. La conseguenza è che potrebbero esserci casi nel Test Set che non sono riconosciuti, non sono adeguatamente elaborati, sono classificati nella classe sbagliata.

Con l'overfitting, il modello è molto preciso su casi simili ai casi del data set, ma può andare molto male su casi mai visti prima. Da un altro punto di vista, l'overfitting corrisponde anche a una scarsa capacità di generalizzazione e ad un adeguamento eccessivo ai dati di training.

Over e Under-fitting: OverFitting (OF) e UnderFitting (UF) sono due fenomeni che si presentano anche se: l'errore di training è molto alto (UF). L'errore di training è molto basso o se è molto più piccolo dell'errore di test: tutti i dati di training vengono 'fittati', senza scoprire tendenze meno puntuali/più generali (OF).

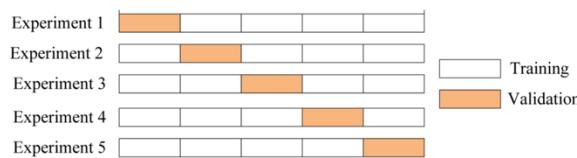


Cross Validation

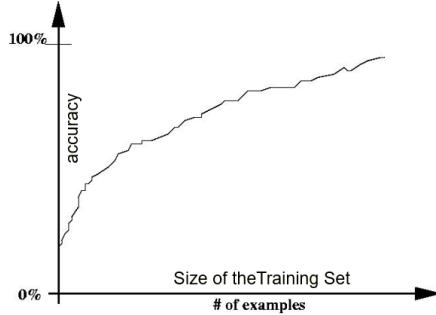
Il metodo noto come Cross Validation ha l'obiettivo di superare le criticità dell'overfitting. In generale, si basa sull'idea di ripetere più volte utilizzando diversi set di test (e corrispondentemente diversi set di training).

Nella cross validation **k-fold** la fase di Training è suddivisa in k "esperimenti", considerando ogni volta un diverso Test Set con una frazione di $1/k$ esempi. Questo viene ripetuto k volte: apprendendo su $(k-1)/k$ del totale n. di esempi e test su $1/k$ del totale n. di esempi, ogni volta diversi. La precisione del modello viene quindi calcolata come media delle k precisioni ottenute in ciascun esperimento.

Valori tipici: $k = 5$ o $= 10$ (cross validation 10-fold).



La qualità della classificazione dipende dal numero di casi/esempi considerati in input. In linea di principio, maggiore è il numero e maggiore è la qualità (se l'overfitting è evitato). Ovviamente, più dati vengono processati durante la fase di training e maggiore potenza computazionale è richiesta.



7.4 ML come approssimazione funzionale

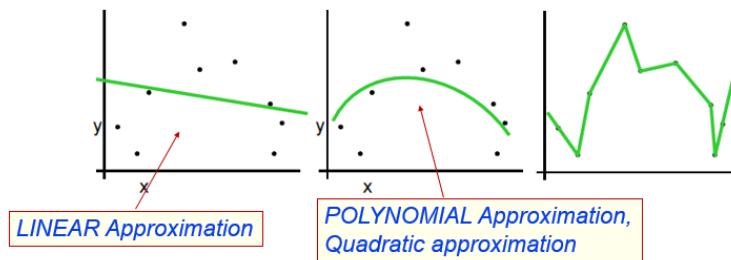
Dato un certo problema (ad esempio bi-dimensionale) potrebbe essere utile conoscere la funzione che mappa i valori in input con quelli in output.

Approssimazione: dato un insieme di valori $\{(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_n, f(x_n))\}$ di una funzione $f(x)$ impariamo una funzione $h(x)$ che approssima i valori di $f(x)$ per ogni input.

La funzione h è detta **hypothesis**. Se h corrisponde esattamente a tutte le osservazioni è detta consistente. Altrimenti, in generale ci potrebbero essere degli errori e lo scopo è di minimizzarli.

Una buona funzione di ipotesi h predice bene/correttamente gli esempi che non sono stati visti/incontrati ancora (si tratta del caso supervised).

Lo spazio di ipotesi (hypothesis space) è un insieme di tutte le funzioni possibilmente considerate per h (funzioni lineari, polinomiali, sinusoidali, etc.).



7.4.1 Regressione e classificazione

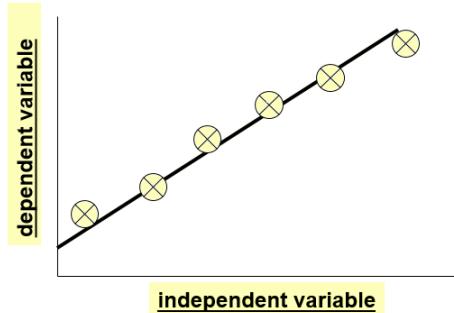
- Se la funzione f da approssimare ha valori continui, la task di approssimazione è qualificata come **regressione**;
- Se la funzione f da approssimare ha valori discreti, la task di approssimazione è qualificata come **classificazione**.

In statistica, l'analisi di regressione è un nome collettivo per tecniche usate nell'analisi di dati numerici. Aiuta a capire le relazioni tra:

- una variabile dipendente $f(x)$ - anche chiamata var. risposta o target var.
- una o più variabili indipendenti x - anche chiamate variabili esplicative o predittori

Il modello di regressione è usato per predirre il risultato della variabile dipendente, dati i valori delle variabili indipendenti.

Esempio: regressione lineare



Se ho una serie di misurazioni di entrambe le variabili, posso stimare una linea retta (un modello lineare). Successivamente ho solo bisogno di misurare la variabile indipendente per poter stimare la variabile dipendente; cioè posso prevedere il valore della variabile dipendente.

7.5 Linear Regression e Gradient Descent - Introduzione

7.5.1 Linear Regression: Nel problema della regressione lineare abbiamo:

- INPUT: Dataset con variabile target y di tipo numerico (x_n, y_n).
- Obiettivo: Trovare una funzione h lineare (retta) che per qualsiasi valore x fornisce il valore della variabile target y riducendo al massimo l'errore nei punti x_1, x_2, \dots, x_n .
- OUTPUT: funzione di una retta.

”Visualmente” possiamo immaginare di ruotare e sposatare la retta tra i punti del dataset al fine di trovare la posizione che minimizza l'errore.

Come si sceglie quindi la retta? bisogna impostare un metodo per definire l'errore. Procedendo iterativamente in più passaggi, si cerca di ricavare la retta che minimizza una Funzione d'Errore (**Funzione di Costo**).

Ad esempio una funzione delle distanze fra i punti ottenuti $(x_i, h(x_i))$ ed i punti che si volevano ottenere (x_i, y_i) .

Spesso si usa lo scarto quadratico medio.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

o la sua radice quadrata:

$$\text{Root Mean Squared Error (RMSE)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

con

y_i = predicted value, \hat{y}_i = actual value

Per "spostare e ruotare" la retta al fine di minimizzare l'errore si utilizza un metodo chiamato **Gradient Descent**.

7.5.2 Gradient Descent: Si provano successivamente più rette, cercando, ad ogni passo di modificare la retta 'nella direzione' di 'minimizzazione dell'errore'.

Ci si muove in base al gradiente della funzione d'errore, infatti il gradiente indica la direzione di maggior crescita di una funzione (di più variabili) e muovendoci in direzione opposta riduciamo (al massimo) l'errore.

Ad ogni passo, quindi, si modificano i parametri della retta, ad esempio inclinazione, intersezione con l'asse x, ecc. L'entità della modifica viene calcolata in base ad un parametro α , denominato learning rate: maggiore è α , e maggiore sarà l'entità della modifica.

Il valore maggiore o minore di α permette di arrivare più o meno velocemente al valore di minimo desiderato.

7.5.3 Ottimizzazione degli Iperparametri

Il metodo della Linear Regression ed in particolare la tecnica del Gradient Descent, utilizza il parametro learning rate α , il cui valore ha un ruolo fondamentale per l'efficacia del metodo.

Praticamente quasi tutti gli algoritmi di ML sono caratterizzati da una serie di parametri (denominati iperparametri), che ne influenzano in vari modi il funzionamento.

Per utilizzare l'algoritmo è necessario settare gli iperparametri ad un valore preciso. Quale? Ci sono molte modalità per farlo, la maggioranza delle quali è fondamentalmente basata su regole empiriche o su un metodo empirico di ottimizzazione del tipo seguente:

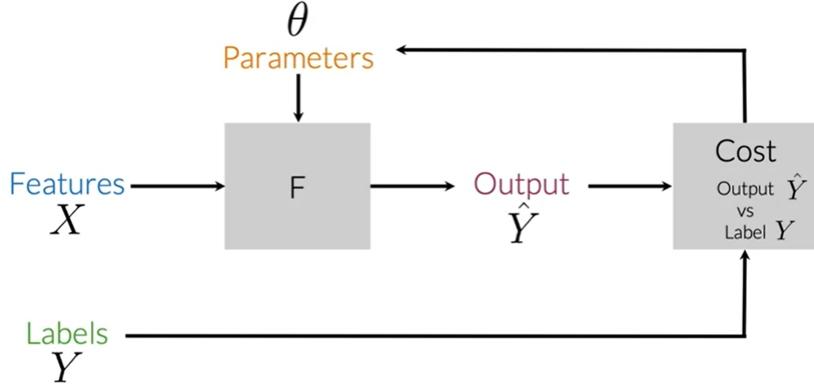
- Provare successivamente diversi valori, valutare i risultati e scegliere il valore del parametro che fornisce i risultati migliori (ad esempio, si possono provare valori di diversi ordini di grandezza, quali 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 1000, etc., per poi concentrarsi sui range di valori che presentano i risultati migliori).
- In certi casi esistono tecniche (statistiche o ML) che permettono di fare scelte ottimizzate.

7.6 Logistic Regression [G.S.]

La regressione logistica è un classificatore utilizzato per modellare la probabilità di una determinata classe binaria (ad esempio: successo/fallimento).

Ingredienti della logistic regression:

- Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$;



- La funzione probabilistica che vogliamo imparare: $\hat{y} = P(y=1 | x)$;
- In particolare, i parametri w e b della seguente funzione (ipotesi):

$$h_{w,b}(x) = g(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}, \text{ dove } g(z) \text{ è la Sigmoid function}$$

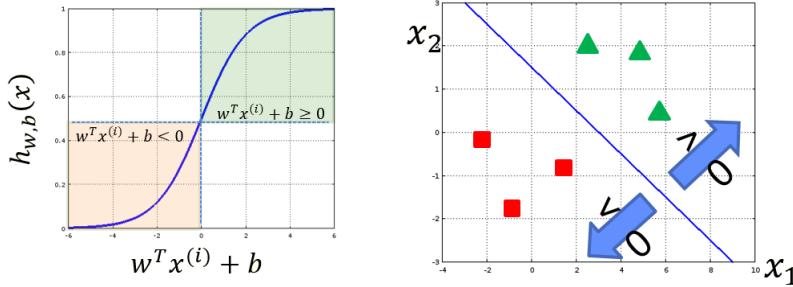
t.c.

$$\begin{aligned} h_{w,b}(x) &\geq 0.5 \text{ if } y = 1 \\ h_{w,b}(x) &< 0.5 \text{ if } y = 0 \end{aligned}$$

- Per ottenere i nostri valori discreti 0 e 1 (di classificazione) mappiamo l'output della funzione ipotesi come segue: $h_{w,b}(x) \geq 0.5 \rightarrow "1"$ $h_{w,b}(x) < 0.5 \rightarrow "0"$
- Funzione di costo: consideriamo il caso 2D: $h_{w,b}(x) = g(b + w_1 x_1 + w_2 x_2)$ con: $b; w = [w_1, w_2]^T, x = [x_1, x_2]^T$.

Usando l'algoritmo di regressione logistica possiamo ottenere: $b = 3; w = [1, 2]$. Si noti che $h_{w,b}(x) = g(w^T x + b) > 0.5$ quando $w^T x + b > 0$. Allora il decision boundary è:

$$h_{w,b}(x) = 0.5 \Rightarrow w^T x + b = 0 \Rightarrow -3 + x_1 + 2x_2 = 0$$



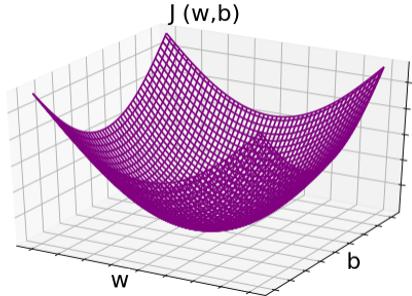
La funzione di costo per il logistic classifier è la seguente funzione:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{w,b}(x^{(i)}), y^{(i)})$$

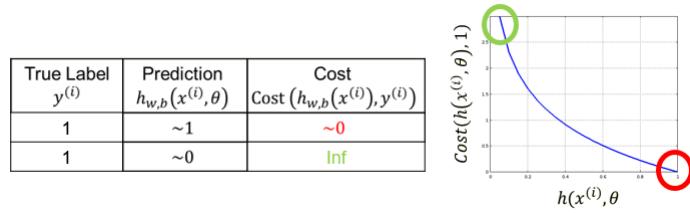
dove

$$\text{cost} (h_{w,b}(x^{(i)}), y^{(i)}) = -y^{(i)} \ln (h_{w,b}(x^{(i)})) - (1 - y^{(i)}) \ln (1 - h_{w,b}(x^{(i)}))$$

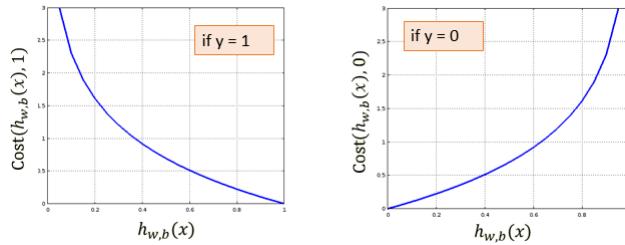
Si noti che tale funzione di costo (anche detta Loss) è derivabile rispetto w e b ed è convessa.



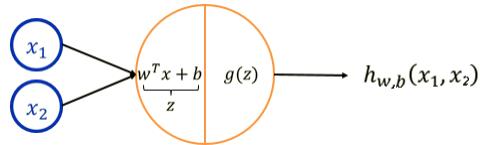
Si noti che se $y^{(i)} = 1 \Rightarrow \text{Cost} (h_{w,b}(x^{(i)}), y^{(i)}) = -\ln (h_{w,b}(x^{(i)}))$ (il secondo termine si semplifica). S



Nel caso $y = 1$ la funzione di costo sarà 0 se la nostra funzione di ipotesi ritorna 1. Se l'ipotesi si approssima a 0 allora al funzione di costo tende a infinito. Nel caso $y = 0$ è il contrario.



Possiamo rappresentare la logistic regression come segue:



Questa rete significa che per computare il valore della funzione di decisione, dobbiamo prima moltiplicare l'input x_1 con θ_1 , il secondo input x_2 con θ_2 , sommare i valori insieme al bias b e poi applicare la sigmoid function g al risultato z .

7.6.1 Logistic regression con confini di decisione non lineari

La funzione di ipotesi h non deve per forza essere lineare. In questo caso il confine di decisione non sarà una retta ma una linea qualunque.

Si supponga che:

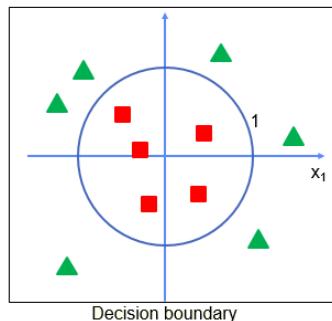
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

e che usando la regressione logistica otteniamo:

$$\theta = [-1 \ 0 \ 0 \ 1 \ 1]$$

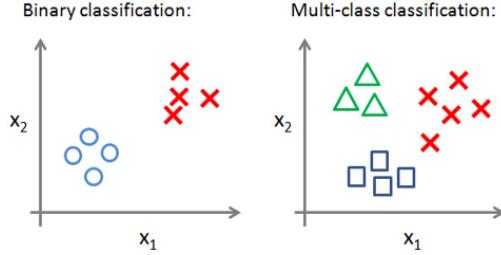
Allora il decision boundary sarebbe:

$$h_\theta(x) = 0.5 \Rightarrow g(\dots) = 0 \Rightarrow -1 + x_1^2 + x_2^2 = 0$$



7.6.2 Classificazione Multiclasse

Si parla di classificazione a classi multiple quando le categorie sono più di due (se sono due 2 = Binary classification).



7.7 Gradient Descent [G.S.]

Il Gradient Descent è un algoritmo utilizzato per minimizzare una funzione derivabile.

Il margine di un classificatore lineare è definito come la larghezza di cui è possibile aumentare il confine prima di raggiungere un punto dati. Vediamo come usarlo per la regressione lineare. Abbiamo una funzione $J(\theta_0, \dots, \theta_n)$ e vogliamo minimizzarla.

Idea:

- Iniziare con valori di θ_n random;
- Continuamo a cambiare questi valori di θ riducendo J ;
- Ci fermiamo quando troviamo un minimo.

L'algoritmo è il seguente:

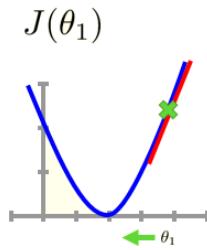
```
repeat until convergence {
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$  (for  $j = 0$  and  $j = 1$ )
}
```

Correct: Simultaneous update $\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ $\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ $\theta_0 := \text{temp0}$ $\theta_1 := \text{temp1}$	Incorrect: $\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ $\theta_0 := \text{temp0}$ $\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ $\theta_1 := \text{temp1}$
---	---

Intuizione del funzionamento del gradient descent nel caso bidimensionale:

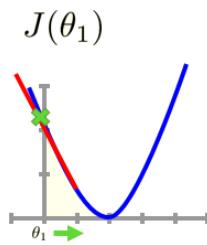
Il primo caso: derivata maggiore di zero e alpha sempre maggiore di zero, quindi avendo il meno davanti lo step sarà sempre negativo (verso sinistra). Nel secondo caso è l'opposto, lo step sarà sempre maggiore di zero (verso destra) e quindi ci si sposta sempre verso il minimo.

È importante notare che Gradient Descent lavora bene solo per valori di α sufficientemente piccoli, altrimenti vengono eseguiti "passi" troppo grandi e non si riesce a raggiungere il minimo. Se α è troppo piccolo, invece, la convergenza sarà molto lenta.



$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_1) > 0$$

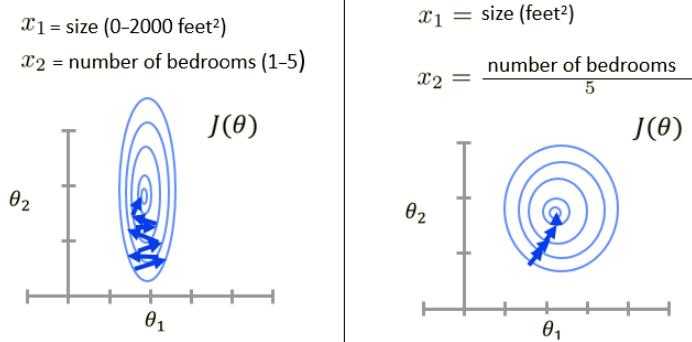


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_1) < 0$$

7.7.1 Scaling delle features

È importante, in ambito ML, prima di addestrare un modello, eseguire lo scaling delle features in uno spazio "proporzionato" per semplificare le operazioni di convergenza.



Vediamo ora i principali metodi di normalizzazione.

Mean Normalization (Standardization)

Un metodo di normalizzazione è il Mean Normalization: si rimpiazza x_i con $x_i - \mu_i$ per avere le features con media circa 0.

$$x_1 = \frac{\text{size}-1000}{2000}$$

$$x_2 = \frac{\text{bedrooms}-2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

Formalmente $x'_i = \frac{x_i - \mu_i}{\sigma_i}$

dove μ_i, σ_i sono la media e la standard deviation sul training set.

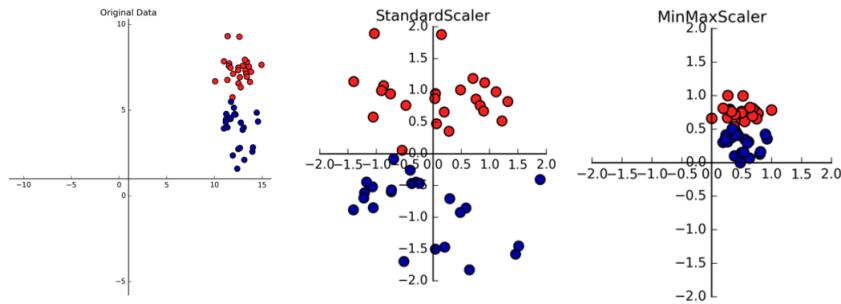
$\mu_i = \frac{1}{m} \sum_{i=1}^m x_i$ e $\sigma_X = \sqrt{\frac{1}{m-1} \sum_{i=1}^m (x_i - \mu_i)^2}$, dove m è il numero di campioni.

Min-Max Normalization

La Min-Max Normalization "sposta" i dati in modo tale che tutte le features ricadano tra 0 e 1.

$$\text{Formalmente: } x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

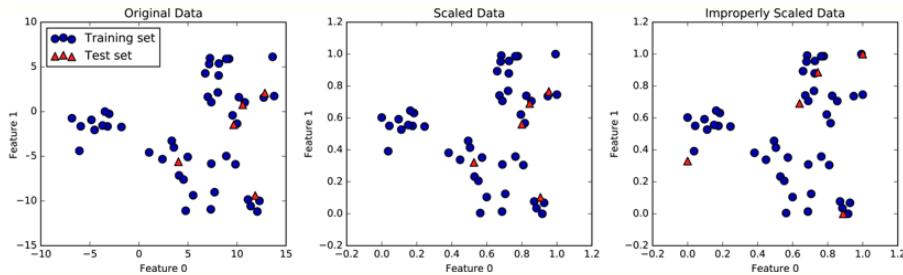
dove x_i sono i valori originali e x'_i è il valore normalizzato.



La figura mostra un dataset a due classi normalizzato con i due metodi visionati.

Scalare Training e Test dataset allo stesso modo

È molto importante applicare la stessa trasformazione ai due dataset di test e training altrimenti stiamo invalidando i dati. L'immagine è esplicativa:



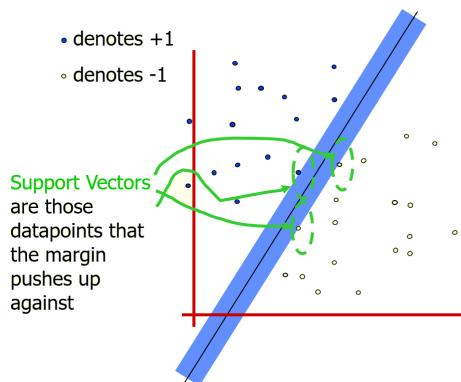
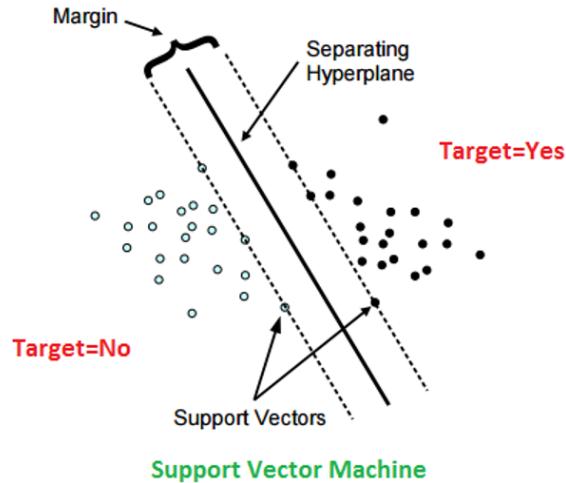
Il primo pannello è un set di dati bidimensionale non in scala, con il set di training illustrato come cerchi e il set di test illustrato come triangoli. Il secondo pannello contiene gli stessi dati, ma ridimensionato utilizzando MinMaxScaler. Qui, abbiamo chiamato fit sul set di training, quindi abbiamo chiamato trasformazione sui set di training e test. Si può vedere che il set di dati nel secondo pannello sembra identico al primo; sono cambiati solo i tick sugli assi. Ora tutte le caratteristiche sono comprese tra 0 e 1. Si può anche vedere che i valori minimo e massimo delle caratteristiche per i dati di test (i triangoli) non sono 0 e 1.

Il terzo pannello mostra cosa accadrebbe se ridimensionassimo separatamente il set di training e il set di test. In questo caso, i valori minimi e massimi delle funzionalità sia per l'training che per il set di test sono 0 e 1. Ma ora il set di dati ha un aspetto diverso. I punti di test si sono spostati in modo incongruo rispetto al set di training, poiché sono stati ridimensionati in modo diverso. Abbiamo modificato la disposizione dei dati in modo arbitrario. Chiaramente questo non è quello che vogliamo fare.

7.8 SVM - Support Vector Machine

L'SVM è un metodo supervisionato che nella versione base permette di realizzare classificatori linerari in modo efficiente.

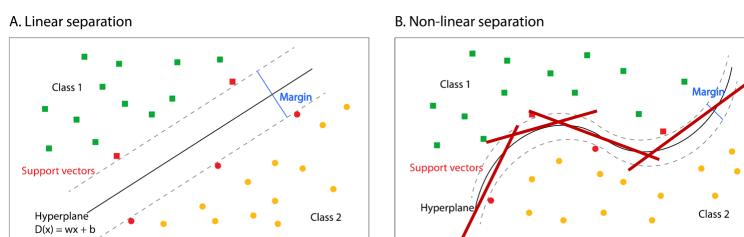
Per insiemi di dati linearmente separabili, individua la retta che separa le due classi e che massimizza il margine (la distanza) tra la retta e i punti più vicini delle due classi.



Questo è il tipo più semplice di SVM (chiamato LSVM)

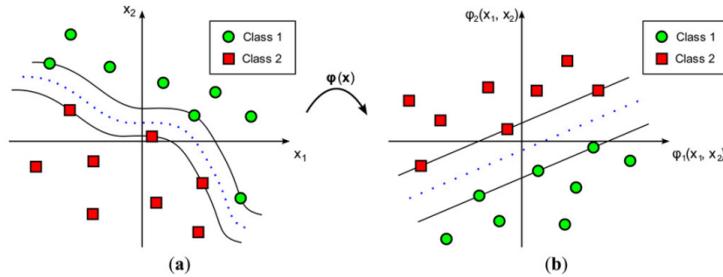
7.8.1 Varianti SVM non lineari

L'approccio LSVM funziona bene per insiemi linearmente separabili. Se i dati non sono linearmente separabili si ricorre alle varianti SVM non lineari.



Il problema viene ridotto all'individuazione di classificatori lineari multipli.

In pratica si utilizza una funzione kernel in (x, y) che opera una trasformazione non lineare dei dati in input, passando dalla configurazione (a) iniziale ad uno spazio delle feature (b) linearmente separabile e ciò riduce il problema al caso lineare.



SVM è un classificatore lineare binario, dove l'obiettivo principale è disegnare un iperipiano per dividere le 2 classi.

In casi anomali, SVM cerca la migliore classificazione (riducendo al minimo una funzione di errore) o, se necessario, ignora i valori anomali.

SVM tende a funzionare molto bene nei problemi in cui abbiamo una chiara separazione dei dati.

Oltre a ciò, gli SVM possono funzionare male nei dataset in cui abbiamo molto rumore.

7.9 Inductive Learning of Decision Trees/Rules

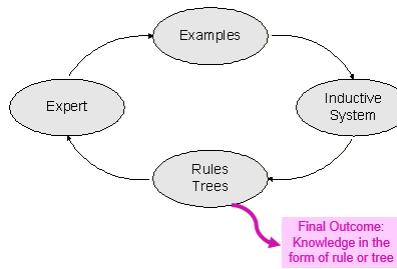
Il "Rule & Tree Induction" è un metodo supervisionato che permette di costruire un classificatore rappresentato da regole, o più in generale, da un albero di decisione.

Un albero di decisione è un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia indica il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, rappresentati nel cammino (path) dal nodo radice (root) al nodo foglia.

Analisi manuale: L'esperto descrive quali parametri influenzano le sue decisioni e in quale misura. L'analista traccia una corrispondenza tra combinazioni di eventi e decisioni o conseguenze.

Bisogna però considerare che: tentare di rappresentare situazioni di XOR o mutua esclusione può far esplodere la complessità dell'albero. Sono presenti frequenti inconsistenze o inesattezze dovute ad errori di elicitazione. Difficilmente si "scopre" nuova conoscenza.

Automatizzare il processo: Viene costruito un training set di dati interessanti (esempi), creati ad hoc o, più interessante, estratti da un database reale. Per ogni entry vengono specificati attributi e decisione/risultato finale. Un algoritmo induttivo costruisce un albero decisionale. Analizzando l'albero è possibile dedurre nuove regole e conoscenze che, altrimenti, sarebbero rimaste tacite.



7.9.1 L'algoritmo ID3

Quinlan (1979) introduce ID3 (Iterative Dichotomiser 3); l'idea di sfruttare il contenuto dell'informazione (concetto derivato dalla Teoria dell'Informazione) come base della funzione di apprendimento euristico. L'euristica consiste nel massimizzare il guadagno di informazioni.

- L'algoritmo ID3 è iterativo.
- Sceglie un sottoinsieme del training set (chiamato finestra) a caso, quindi costruisce un albero decisionale che è in grado di classificare correttamente ogni istanza nella finestra.
- Tutti gli altri oggetti nel set di addestramento vengono quindi classificati utilizzando questo albero. Se ogni oggetto noto è classificato correttamente da questo albero, l'algoritmo termina con successo.
- Tuttavia, su set di addestramento non banali, è più probabile che l'albero iniziale classifichi erroneamente alcune delle istanze. In questo caso, una selezione degli oggetti classificati in modo errato viene aggiunta alla finestra e l'albero viene ricostruito.
- Esistono versioni più recenti dell'algoritmo ID3: AQ11, J48 in WEKA, CN2 in Orange, ...
- ID3 è anche computazionalmente efficiente, in quanto il tempo impiegato per costruire alberi aumenta solo linearmente con la dimensione del problema.

7.10 Clustering - Introduzione

Gli algoritmi di clustering sono un insieme di tecniche non-supervisionate di ML che, a partire da un insieme di elementi, suddividono tale insieme in gruppi omogenei di elementi simili secondo una data metrica (ossia, in termini più semplici, secondo un criterio definito e computabile); i gruppi così formati si chiamano cluster.

Tecniche comuni: K-means, Clustering Gerarchico, Clustering basato sulla densità ...

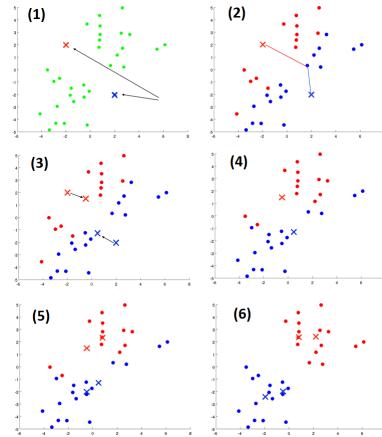
7.10.1 K-means

K-means è il più popolare algoritmo di clustering. k-means riceve in input un insieme di elementi ed un intero k e quindi suddivide l'insieme in k gruppi. Tali gruppi inizialmente

sono generati in modo casuale, ma in seguito vengono progressivamente affinati stabilendo quali elementi appartengono a ciascun gruppo in base alla vicinanza al centroide del gruppo, ossia l'elemento al centro del cluster.

Esempio con $k=2$:

1. Selezioniamo a caso due punti attorno ai quali costruire i 2 cluster, uno blu e uno rosso (I due punti rappresentano i potenziali centroidi dei due cluster in costruzione);
2. Procediamo ad una prima costruzione dei due Cluster. Ogni punto viene assegnato al cluster che ha il centroide più vicino;
3. Per i due cluster appena individuati, calcoliamo l'esatta posizione dei relativi centroidi; (Centroide di un insieme di punti: punto con le coordinate che corrispondono alla media delle coordinate di tutti i punti dell'insieme);
4. Relativamente ai due nuovi centroidi, ricostriamo i cluster come al punto 2.
5. Per i due nuovi cluster, ri-calcoliamo l'esatta posizione dei relativi centroidi;
6. loop - fino a convergenza dei risultati delle iterazioni;



7.10.2 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) connette regioni di punti con densità sufficientemente alta. A partire da un elemento casuale p , DBSCAN inserisce nel cluster di p tutti gli elementi che stanno entro una determinata distanza d da p , a condizione che siano in numero superiore ad un certo minimo threshold minPts . DBSCAN ripete lo stesso procedimento per tutti i punti così trovati. Una volta che non vengono trovati nuovi punti per il cluster, DBSCAN ripete lo stesso procedimento per un nuovo elemento casuale (non ancora incluso), genera un nuovo cluster, e così via fino ad esaurire i punti da analizzare. I punti che non hanno un sufficiente no. di vicini ($> \text{minPts}$), rimangono isolati. DBSCAN non richiede di settare k (no. di cluster), ma lo trova autonomamente.

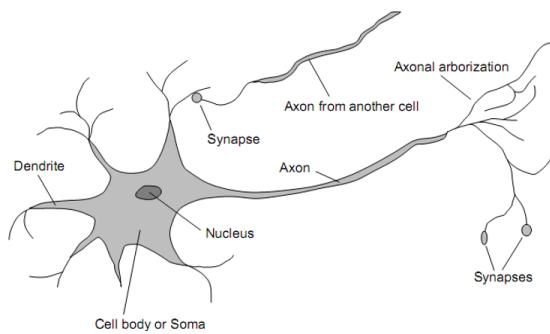
8 Neural Networks

Artificial Neural Networks (ANN): tecnica di ML, basata su reti di neuroni artificiali, organizzati in layer, che ricordano l'organizzazione dei neuroni biologici.

Le reti neurali artificiali (ANN) si ispirano a un modello di cervello umano, costituito da più di 10 miliardi di neuroni, variamente connessi.

Il cervello umano reagisce agli stimoli esterni e adatta il suo comportamento attraverso l'apprendimento. L'apprendimento avviene mediante modifiche nella "forza" delle connessioni tra i neuroni.

In modo analogo, in una ANN l'adattamento avviene modificando i pesi numerici, sulla base del confronto dei risultati prodotti e delle prestazioni desiderate.



Le ANN sono costituite da unità di calcolo molto semplici che lavorano in parallelo, interconnesse in modo tale da ottenere i risultati desiderati.

Neurone biologico: Se è presente un'eccitazione sufficiente, il corpo cellulare (soma) "si accende" e invia un picco di un piccolo segnale elettrico attraverso l'assone fino al fondo.

Sinapsi: Le sinapsi collegano un neurone con un altro: tra un neurone e il successivo c'è un piccolo spazio, che viene riempito con un fluido conduttivo, che permette la trasmissione elettrica.

Gli ormoni cerebrali (o sostanze come la caffeina) influenzano il grado di condutività e, di conseguenza, la forza del segnale elettrico al dendrite collegato.

Le connessioni sinapsi rappresentano le porte di collegamento per il passaggio dell'informazione tra neuroni.

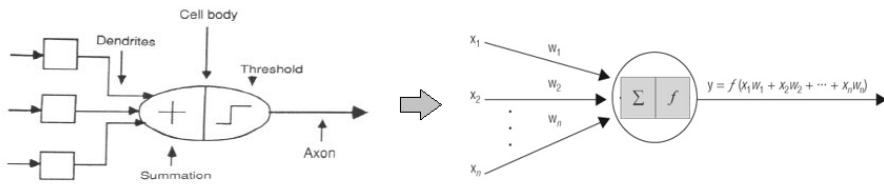
L'apprendimento avviene modificando l'efficacia (riponderazione) delle sinapsi in modo che l'influenza di un neurone su un altro cambi. L'uso ripetuto delle connessioni sinaptiche rafforza le sinapsi.

8.1 Neuroni e reti artificiali - basic concepts

Dalla biologia alle artificial neural networks:

- Soma (corpo cellulare) → Unità di computazione;
- Axon (assone) → Canale di output;
- Dendriti → Canale di input;

- Sinapsi → Pesi;



Hebb scrisse nel 1949:

"When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell. Translating Hebb's concepts to artificial neural networks and artificial neurons, his model can be described as a way of altering the relationships between artificial neurons (also referred to as nodes) and the changes to individual neurons. ... The word weight is used to describe these relationships,..."

8.1.1 Neurone artificiale

Input: vettore a valori reali (x_1, x_2, \dots, x_n)

I pesi sono un vettore a valori reali (w_1, w_2, \dots, w_n) e controllano l'effetto che l'input avrà sull'unità.

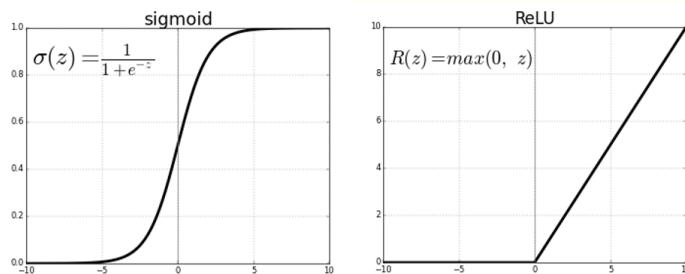
Il corpo computa la combinazione lineare $x_1w_1 + \dots + x_nw_n$ e applica al risultato la funzione f detta funzione di attivazione.

Una funzione di attivazione nota è la sigmoid function: $f(z) = \frac{1}{1+e^{-z}}$.

La Sigmoid ha visto un uso frequente storicamente, ma ora è usata raramente, perché satura e uccide il gradiente.

Un'altra funzione di attivazione semplice è la step function che vale 0 se l'input è sotto un dato threshold e 1 se è sopra (a scalino).

Un'altra funzione di attivazione molto nota e utilizzata è la Rectified Linear Unit (ReLU). L'attivazione ReLU è stata trovata per accelerare notevolmente la convergenza di SGD rispetto alla funzione Sigmoid. L'implementazione è semplice rispetto ad altre funzioni di attivazione che richiedono operazioni complesse.



L'output è quindi un valore reale $y = f(x_1w_1 + \dots + x_nw_n)$

NB: Alla combinazione lineare degli input solitamente viene sommato anche un peso w_0 (denominato bias, considerato collegato a un input fittizio con valore sempre uguale a 1); il bias è utile per "tarare" il punto di lavoro ottimale del neurone.

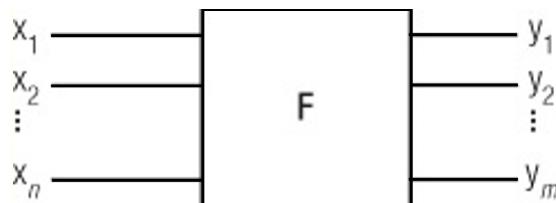
Altre possibili funzioni di attivazione:

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU) [2]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [3]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

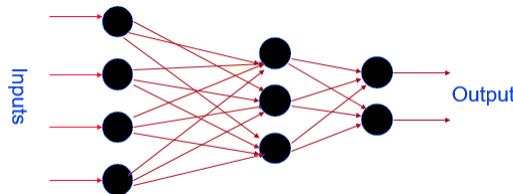
8.1.2 Rete neurale artificiale (ANN)

Le reti neurali sono una collezione di neuroni organizzati in una certa topologia. Input e output sono due vettori di valori reali (x_1, x_2, \dots, x_n) e (y_1, y_2, \dots, y_m).

Una ANN può essere vista come una black box che computa $F : R^n \rightarrow R^m$.



Una Rete Neurale Artificiale è costituita da molti neuroni artificiali che sono collegati tra loro secondo una specifica architettura di rete. L'obiettivo della rete neurale è trasformare gli input in output significativi (unidimensionali o n-dimensionali).



Ogni ANN include la "conoscenza" contenuta nei valori dei pesi di connessione.

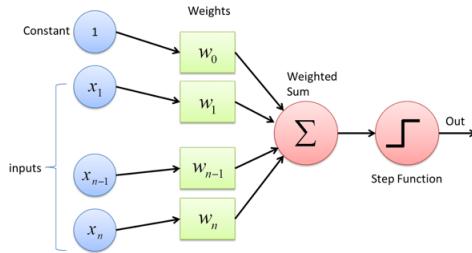
Modificare la conoscenza immagazzinata nella rete in funzione dell'esperienza implica cambiare i valori dei pesi.

Perceptron

Il Perceptron è la rete più semplice, costituita da due soli layer (input e output). Utilizza una funzione di attivazione lineare a soglia (o scalino). Si utilizza come classificatore lineare binario.

Un singolo perceptron è in grado di apprendere solo mapping lineari e pertanto il numero di funzioni approssimabili è piuttosto limitato.

Per superare tale limitazione, si utilizzano reti che hanno almeno 3 layer (di cui uno nascosto).

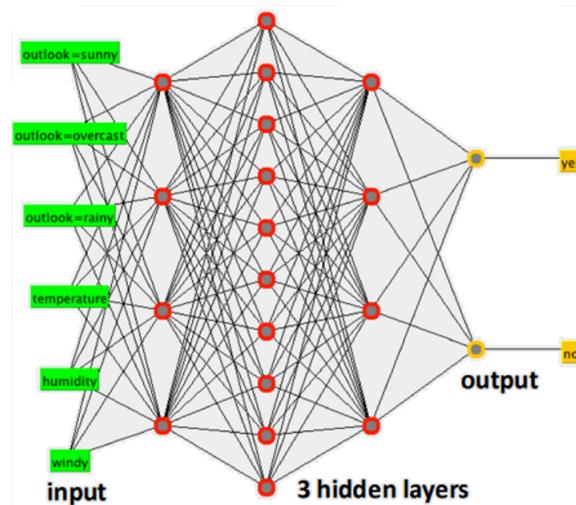


Multiple-Layer Perceptron (MLP)

Un MLP è costituito da:

- Layer di input, layer nascosti e layer di output;
- Ciascuna connessione ha un peso;
- Ciascun nodo computa una somma pesata dei suoi input (+bias) e confronta con un threshold, tipicamente in base alla funzione sigmoid;
- I pesi "contengono" la conoscenza della risoluzione dei problemi.

Esempio:

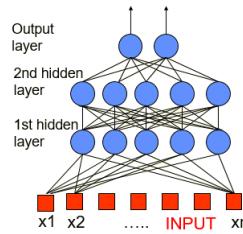


8.1.3 FeedForward

FeedForward significa che le informazioni vengono propagate attraverso i livelli (nascosti) dagli input agli output.

Le connessioni tra i neuroni collegano i neuroni di uno strato con i neuroni dello strato successivo. Non sono consentite connessioni all'indietro e connessioni a neuroni dello stesso livello.

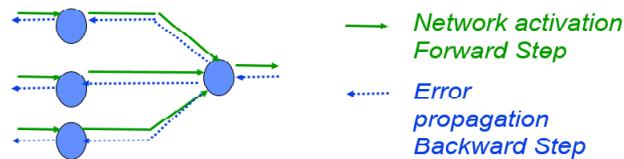
Sia Perceptron che MLP sono ANN feedforward.



8.1.4 Processo di Backpropagation

I pesi vengono inizializzati con valori casuali. Quindi, la Backpropagation consiste nell'applicazione ripetuta dei seguenti due passaggi:

- **Passaggio in avanti:** in questo passaggio la rete viene attivata su un esempio e viene calcolato l'errore di (ogni neurone del) layer di output.
- **Passaggio all'indietro:** in questo passaggio viene utilizzato l'errore di rete per l'aggiornamento dei pesi. A partire dal livello di output, l'errore viene propagato all'indietro attraverso la rete, layer dopo layer. Questo viene fatto calcolando ricorsivamente il gradiente locale di ciascun neurone.



Un'**epoca** corrisponde a un ciclo di Forward + Bakward Propagation, con il corrispondente raffinamento dei pesi. Più epoche, in linea di principio portano a risultati migliori.

Il processo di training è quindi il seguente:

Nella prima fase i valori in input vengono propagati "in avanti" e viene calcolato l'output (feedforward).

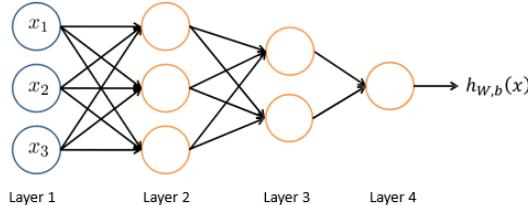
Si utilizzano i pesi dei vari strati w_{ij} , i – indice del nodo di input e j - indice del nodo 'destinazione' .

Viene calcolato l'errore (risultato ottenuto rispetto al valore atteso). Poi viene propagato l'errore all'indietro (backpropagation) per poi riprocedere in avanti andando a aggiustare il valore dei pesi al fine di minimizzare (annullare, se possibile) l'errore medio in base all'errore ottenuto con i pesi precedenti.

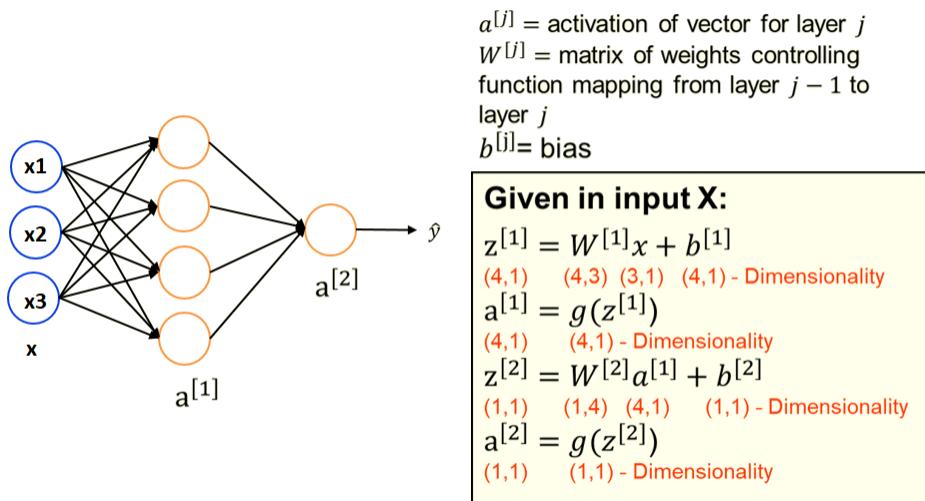
I pesi vengono modificati con il procedimento del gradient descent, applicato alla funzione d'errore (loss function).

8.2 Reti neurali artificiali - rappresentazione

Esempio di rete neurale a più layers:



Rappresentazione più dettagliata:



8.3 Loss Functions

Per la binary classification si utilizza, tipicamente, la **Binary Cross-Entropy Loss**:

$$L(W, b) = -\frac{1}{m} \sum_{i=1}^m -y^{(i)} \log(h_{W,b}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{W,b}(x^{(i)}))$$

Dove m è il numero di esempi nel set di training.

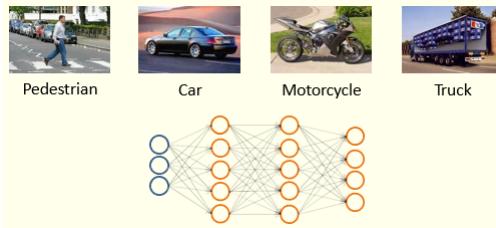
Per la regressione usiamo spesso la funzione di loss: **Mean SquaredError Loss**:

$$L(W, b) = \frac{1}{m} \sum_{i=1}^m (h_{W,b}(x^{(i)}) - y^{(i)})^2$$

Si noti che nel caso della regressione la rete finisce con una singola unità e nessuna f. di attivazione, quindi, l'ultimo layer è puramente lineare e la rete è libera di imparare a predire valori in qualsiasi range.

8.4 Multiclass Classification

Nel ML, la classificazione multiclass o multinomiale è il problema della classificazione delle istanze in una delle tre o più classi (generalizzazione della binary classification).



Si utilizza in questo caso, la one-hot representation: si associa ad ogni classe un indice del vettore etichetta, ad esempio [1,0,0,0] per i pedestrian, [0,1,0,0] per l'auto, e così via.

8.4.1 Softmax Layer

Nella multiclass classification l'ultimo layer della rete neurale tipicamente utilizza la funzione di attivazione SoftMax. Questo significa che la rete produrrà in output una distribuzione di probabilità tra le classi.

Sia $z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}$ Il layer SoftMax è quindi:

$$a^{[i]} = \frac{e^{z^{[i]}}}{\sum_{j=1}^N e^{z_j^{[i]}}}$$

Esempio:

$$z^{[i]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow a^{[i]} = \begin{bmatrix} \frac{e^5}{t} \\ \frac{e^2}{t} \\ \frac{e^{-1}}{t} \\ \frac{e^3}{t} \end{bmatrix} \text{ where } t = (e^5 + e^2 + e^{-1} + e^3) \rightarrow a^{[i]} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \quad \text{The Sum is equal to 1}$$

Per eseguire il training di una rete neurale per un problema di classificazione multiclass si può utilizzare la funzione di costo **Cross-Entropy Loss** (generalizzazione della versione binaria).

La Cross-Entropy è comunemente utilizzata per quantificare la differenza tra due diverse distribuzioni di probabilità.

Tipicamente, la "vera" distribuzione (quella che stiamo cercando di predirre) è espressa in termini di una distribuzione one-hot.

$$L(W, b) = - \sum_{i=1}^m y^{(i)} \log (h_{W,b}(x^{(i)}))$$

Dove m è il numero dei campioni di training, $y^{(i)}$ è la ground truth e $h_{W,b}(x^{(i)})$ è la distribuzione predetta.

In pratica, $y^{(i)}$ è la rappresentazione one-hot della label corretta. Esempio:

$$y^{(i)} = [0 \ 1 \ 0]^T; h_{W,b}(x^{(i)}) = [0.228 \ 0.619 \ 0.153]^T$$

$$L = -(0 * \ln(0.228) + 1 * \ln(0.619) + 0 * \ln(0.153)) = 0.479$$

9 Deep Learning

Ricordiamo dalla definizione presente nell'introduzione che con Deep Learning intendiamo: tecniche di ML basate su ANN costituite da molti layer di neuroni artificiali, che rappresentano una gerarchia di elementi del problema, dove i livelli di più alto livello sono definiti sulla base di quelli di basso livello, a partire dai dati elementari in input.

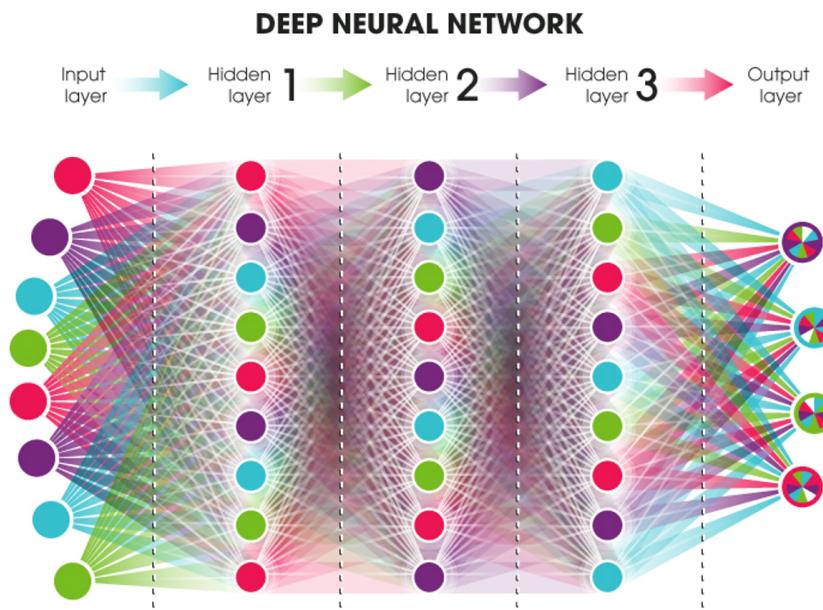
Quando il numero di input, il numero di nodi ed il numero di layer aumenta notevolmente, si parla di Reti Profonde e di Deep Learning. Tali reti ovviamente introducono grossi problemi per la fase di training. Inoltre, sono state proposte delle Reti con architetture particolari, e.g. MLP più complessi che non seguono l'approccio feed-forward.

Da qualche anno però sono stati proposti dei metodi per riuscire ad effettuare il training in modo efficiente ed efficace.

Il successo di tali innovazioni, e la possibilità di disporre di potenza di calcolo, del cloud e di strumenti di sviluppo molto produttivi ed efficaci ha decretato il successo del DL, e più in generale del Machine Learning, e a cascata, dell'Intelligenza Artificiale.

Oggi, spesso quando si utilizza il termine Intelligenza Artificiale, di fatto si intende prevalentemente riferire gli Algoritmi di Apprendimento Profondo, o Deep Learning.

Esempio di un feed forward multilayer perceptron:



Gli input elementari vengono via via aggregati negli strati successivi, a rappresentare aspetti del problema sempre più astratti, fino ad arrivare alla soluzione.

Backpropagation: il training è realizzato in modo incrementale mediante cicli iterativi di calcolo ‘in avanti’, valutazione dell’errore, e ritorno ‘all’indietro’, per correggere i parametri delle unità di calcolo (nodi) contenute nei livelli intermedi; come già visto.

9.1 Tipologie di ANN per il DL

Esistono molte tipologie di Reti Neurali profonde, caratterizzate da un'architettura che va 'oltre' alla tradizionale organizzazione delle reti MLP feed-forward, introducendo schemi diversi che utilizzano connessioni diverse fra i neuroni e i diversi strati della rete.

Le reti profonde più comuni sono:

- CNN - Convolutional Neural Network
- RNN – Recurrent Neural Network
- LSTM - Long short-term memory
- Bi-LSTM,

9.2 Convolutional Neural Networks

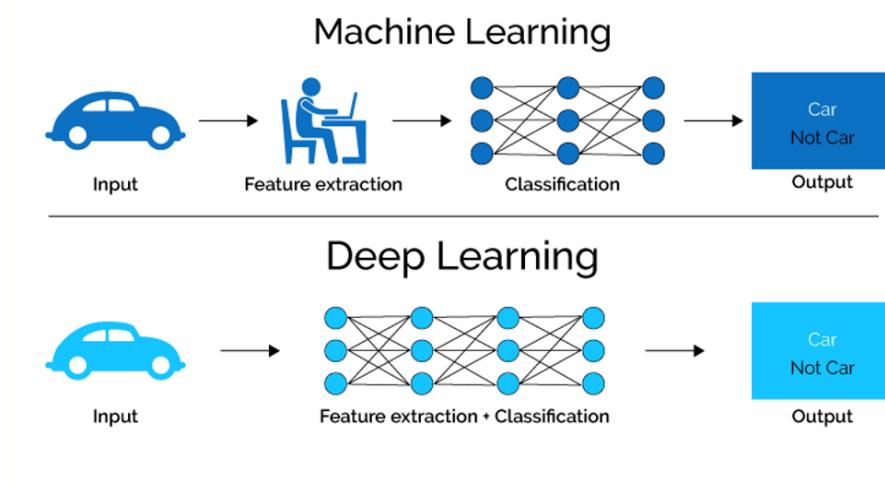
CNN sta per Convolutional Neural Network, tipologia di ANN molto utilizzata nell'ambito della Computer Vision.

Tali reti si basano 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 più basso livello.

L'idea 'vincente' è 'imparare a scoprire le feature' (feature learning) che permettono di eseguire correttamente il task (ad es. riconoscimento di un'immagine). Si alleggerisce così la fase di feature engineering.

Tipicamente quindi si fornisce un input con molti elementi.

Ad esempio: nel caso dei testi, non solo si fornisce in input la struttura del testo ed altre meta-feature, ma anche tutte le parole ed il loro 'significato', nel caso delle immagini tutti i singoli pixel.



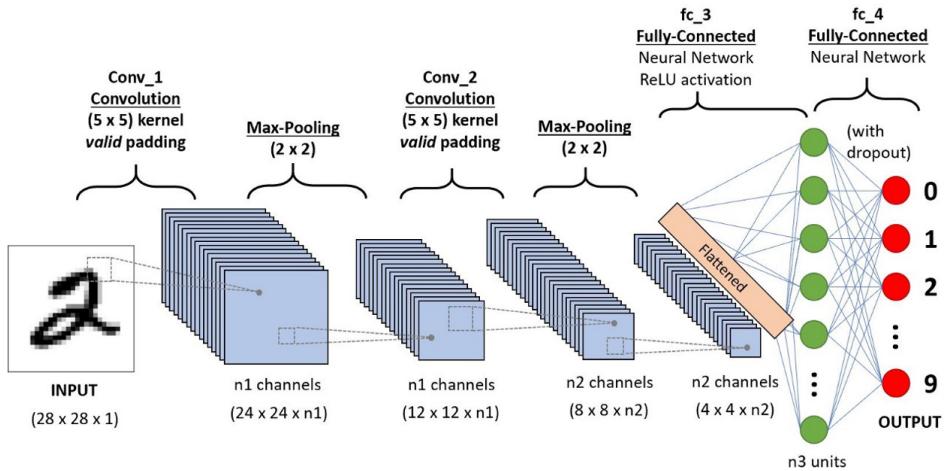
- ML: pochi strati; shallow networks
- DL: molti strati

Le Tecniche di DL aiutano a distinguere fra le feature più o meno significative. Individuano automaticamente le feature rilevanti (pesi maggiori): dimensionality reduction. Il prezzo da pagare è la mole di dati necessaria in input.

9.3 Esempio per la classificazione di immagini

Le reti Convulsive per la Computer Vision utilizzano una successione di diversi tipi di layer, alcuni dei quali denominati **layer convolutivi** in grado di riconoscere all'interno dell'immagine in input delle specifiche configurazioni di forme, colori, tessiture, ecc.

Altri layer (denominati ‘fully connected’) sono di tipo normale, similmente ai layer dei MLP.



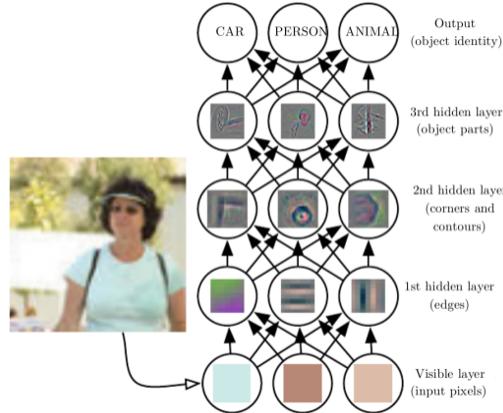
9.3.1 Convolutional layers: Si può pensare che ogni layer convolutorio compia un’operazione di filtraggio dei dati in ingresso al layer stesso. I successivi layer rappresentano quindi una sequenza di filtri che si applicano progressivamente al dato in ingresso (ad es. un’immagine rappresentata da tutti i pixel che la compongono).

Ciascun livello è caratterizzato da un insieme di filtri che riconoscono delle specifiche configurazioni delle feature in input al livello. Il filtro quindi identifica la presenza di una data struttura (pattern) in ingresso, ad esempio analizzando una variazione di colore, la forma di tale variazione, etc.

Quindi, per esempio, al primo layer vengono riconosciuti vari tipi di bordi, ossia zone in cui cambia il livello di colore o luminosità e la relativa forma (ad esempio orizzontale, verticale, diagonale, etc.).

L’immagine viene quindi rappresentata mediante i vari pattern che in essa sono stati riconosciuti.

Viene quindi descritta in una forma ‘compressa’ (‘più astratta’), composta da meno elementi che rappresentano i pattern riconosciuti dal primo layer, che vengono quindi forniti in input al secondo layer., ecc.



Esempio di funzionamento di un filtro convolutivo: Il filtro viene fatto ”scorrere” su tutta l’immagine e quindi riconosce la varie zone dove è presente un certo pattern. Vengono analizzati più pattern, ognuno con un diverso filtro.

FILTRO per Vertical edge detection

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$(3*1+1*1+2*1)+(0*0+5*0+7*0)+$
 $(1*(-1)+8*(-1)+2*(-1))$

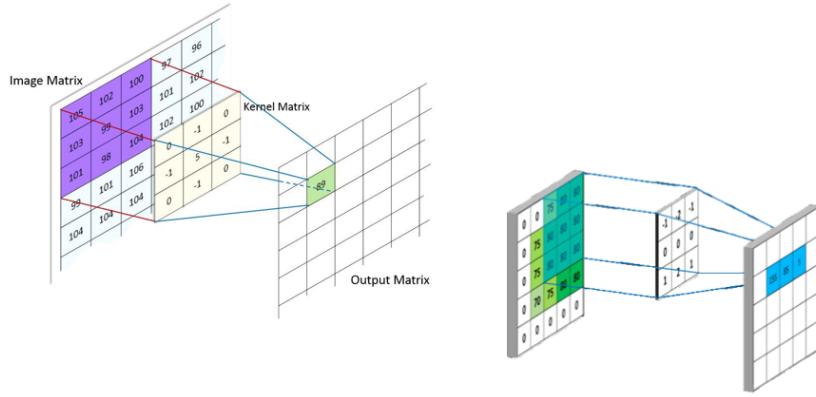
$\begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{matrix}$

$* \quad 3x3 = \quad 4x4$

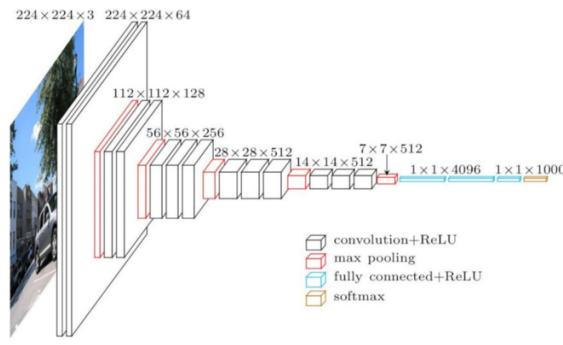
I vari layer convolutivi rimappano l’input in rappresentazioni di dimensioni inferiori, che rappresentano (ai vari livelli gerarchici successivi) dei pattern sempre più articolati/astratti riscontrati nell’input.

I layer successivi a layer convolutivi si comportano più similmente ad una normale MLP. Grazie al meccanismo del BackPropagation, i pesi vengono settati in modo da ‘premiare’ il riconoscimento di certi pattern corrispondenti alle varie classi rispetto alle quali dev’essere classificata l’immagine in input.

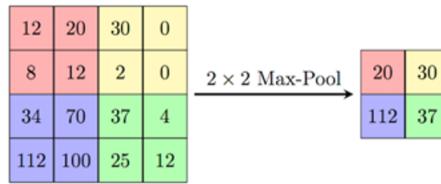
Il Max-pooling layer riduce ulteriormente le dimensioni della matrice, indicando in ciascuno dei suoi elementi la presenza (o meno) nell’immagine di una certa configurazione di elementi riconosciuti in precedenti layer convolutivi.



Esempio di CNN complessa:



9.3.2 Max-Pooling layer: Il Max-pooling layer riduce ulteriormente le dimensioni della matrice, indicando in ciascuno dei suoi elementi la presenza (o meno) nell'immagine di una certa configurazione di elementi riconosciuti in precedenti layer convolutivi. E in successivi strati convolutivi ciò permette di riconoscere ulteriori feature 'più astratte'



9.3.3 Training di una rete profonda

Da un punto di vista generale, si utilizza una procedura di BackPropagation che si utilizza per una Rete Neurale normale. Nel Forward Step si cerca di predire la classe di appartenenza di un'immagine.

In caso di errore, nel Backward step, l'algoritmo corregge i suoi parametri (i pesi dei filtri convolutivi) in modo che all'iterazione successiva la predizione risulti diversa e si avvicini via via a quella corretta. Ciò corrisponde a variare il peso con cui ogni filtro 'percepisce/ri-connosce' certi pattern 'osservati' nei dati, via via nei vari successivi layer. Viene dato maggior o minor peso alla presenza/assenza di certi pattern rilevati negli strati precedenti.

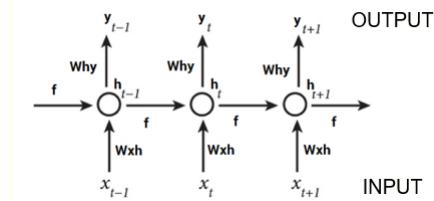
Variando tali parametri (pesi), si privilegia il riconoscimento di particolari combinazioni di pattern che caratterizzano le diverse tipologie di immagini, che quindi vengono riconosciute e classificate correttamente.

9.4 Recurrent ANN:

Nelle reti ricorrenti (RNN) sono previste connessioni di feedback (in genere verso neuroni dello stesso livello, ma anche all'indietro). In tal modo l'elaborazione dell'input operata da un neurone può tener conto anche dei risultati dell'elaborazione di un altro neurone.

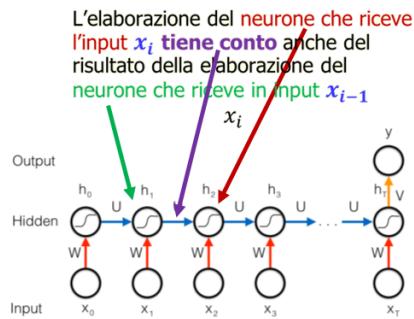
Questo complica notevolmente il flusso delle informazioni e l'addestramento.

D'altro canto queste reti sono più indicate per la gestione di sequenze (es. audio, video, frasi in linguaggio naturale, serie temporali), perché dotate di un effetto memoria (di breve termine) che al tempo rende disponibile l'informazione processata a $t-1$; $t-2$, ecc.



Un particolare tipo di rete ricorrente è LSTM (Long Short Term Memory) e la sua variante BiLSTM (Bidirectional LSTM).

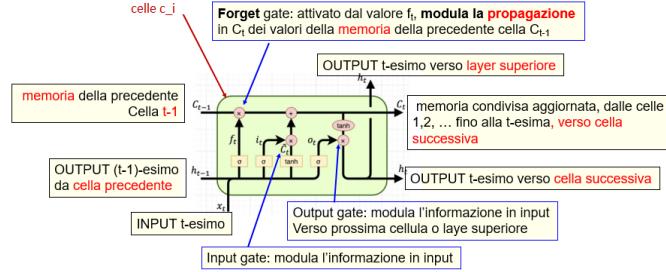
Nelle RNN, ci sono diverse connessioni tra i nodi; ogni nodo viene influenzato dal nodo precedente. Se ho una sequenza in ingresso, non esamino tutti gli elementi in modo indipendente, ma tengo conto di ciò che veniva prima.



9.4.1 LSTM

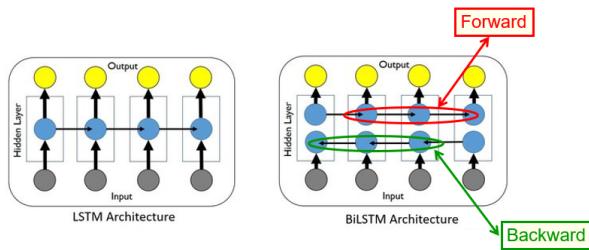
Un particolare tipo di rete ricorrente è LSTM (Long Short Term Memory) e la sua variante BiLSTM (Bidirectional LSTM). La rete LSTM ha capacità di memoria. Molto efficace per flussi di dati, serie temporali, testo in LN, etc.

La rete è costituita da nodi (o celle, C_i) LSTM che contengono dei Gate, in grado di propagare o meno segnali alle celle successive in base a condizioni esterne.



9.4.2 biLSTM

Nella variante BiLSTM (Bidirectional LSTM) il flusso in entrata è analizzato in due direzioni: in avanti e all'indietro.



9.5 Metologia di sviluppo di un sistema ML

Il processo è fondamentalmente di tipo iterativo, guidato da un approccio empirico, in cui la valutazione dei risultati ottenuti può portare alla revisione/affinamenti di scelte precedenti. Le varie fasi/attività tipiche sono le seguenti:

Get Data:

- Definire con precisione gli obiettivi: Regressione (prevedere il futuro di un parametro); Classificazione (classificare la tipologia di evoluzione: normale, anomala, ...).
- Aggiungere o verificare le sorgenti di dati
- Raccogliere, analizzare e taggare i dati

Clean, prepare & manipulate data:

- Noise removal
- Feature Selection/Extraction

Train Model

- Model Selection: definizione dell'algoritmo di ML, scelta di funzione di costo, feature engineering, ottimizzazione hyperparametri, feature scaling e normalization, feature discritization, dataset re-sampling, transforming and combining features.
- Costruzione del modello predittivo mediante training: divisione del dataset in training, test e validation set; risoluzione di overfitting etc. Testing e iterazioni fasi precedenti.

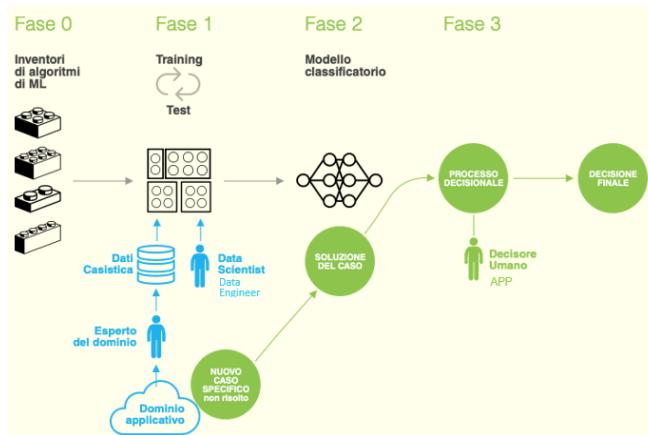
Validation

- Testing del modello e Valutazione

Produzione

- Deployment del modello e decision making tramite il modello predittivo

Workflow utilizzo modelli ML:



9.6 Valutazione della qualità di un sistema ML

Le principali metriche di valutazione della qualità predittiva di un sistema ML sono:

- Accuratezza
- Precision
- Recall
- F1
- Confusion Matrix
- ROC

Si pensi al caso di una classificazione binaria (P o N):

- TP indica il numero di casi positivi (del test set) classificati correttamente, detti True Positive;
- TN indica il numero di casi negativi (del test set) classificati correttamente, detti True Negative;
- FP, indica il numero di casi (del test set) classificati scorrettamente come positivi (ma che sono in realtà negativi), detti Falsi Positivi;
- FN indica il numero di casi (del test set) classificati scorrettamente come negativi (ma che sono in realtà positivi), detti Falsi Negativi.

9.6.1 Accuracy: definita come la frazione di casi previsti (P o N) correttamente, sul totale dei casi di test:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

Anche denominato Classification Accuracy, poiché indica quanto bene è stata fatta la classificazione, relativamente a tutte le classi considerate.

Non è adeguato se le classi (nel caso di molte classi) sono sbilanciate. Infatti se su classi con pochi valori sono impreciso, mentre su classi ‘pesanti’ con molti valori sono preciso, il valore potrebbe essere alto, ma con scarsa capacità di previsione/differenziazione delle varie classi.

9.6.2 Precision: definita come la frazione di casi P previsti correttamente sul totale dei casi (P e N) previsti:

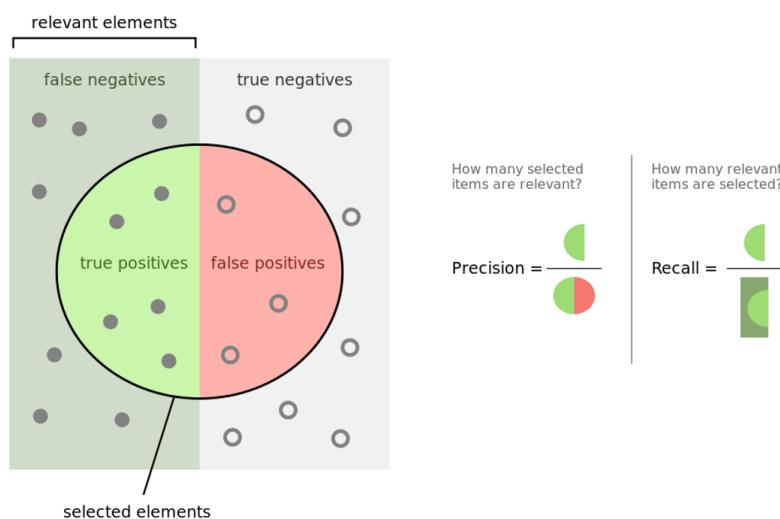
$$Precision = TP / (TP + FP)$$

9.6.3 Recall: frazione di casi P previsti correttamente sul totale dei casi P del test set che sarebbero dovuti essere previsti:

$$Recall = TP / (TP + FN)$$

9.6.4 F1: misura combinata dell’accuratezza del test, partendo dai valori della precision e della recall

$$F1 = \frac{(2 * Precision * Recall)}{(Precision + Recall)}$$



9.6.5 Confusion Matrix: è una matrice 2 x 2 nel caso binario:

La prima riga rappresenta il numero di casi P del test set, suddiviso in:

- (prima colonna): TP, numero di casi P classificati correttamente P
- (seconda colonna): FN, numero di casi P classificati scorrettamente come N

La seconda riga rappresenta il numero di casi N del test set, suddiviso in:

- (prima colonna): FP, numero di casi N classificati scorrettamente P
- (seconda colonna): TN, numero di casi N classificati correttamente come N

Permette di visualizzare quanto bene è stata fatta la classificazione, su quali classi ci sono difficoltà di class.ne, ecc.

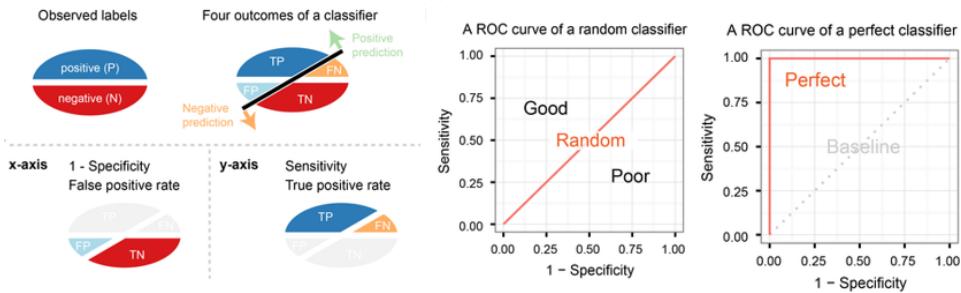
Due esempi:

		Predicted		
		Positive	Negative	
Actual	Actual True	TP	FN	
	Actual False	FP	TN	
				Predicted
				Iris-setosa Iris-versicolor Iris-virginica
	Iris-setosa	100.0 %	0.0 %	0.0 %
	Iris-versicolor	0.0 %	88.7 %	6.4 %
	Iris-virginica	0.0 %	11.3 %	93.6 %
	Σ	50	53	47

9.6.6 Misure dell'errore: tipiche misure dell'errore sono:

Mean Absolute Error (MAE)	$\frac{\sum_{i=1}^N y_i - y'_i }{N}$
Root Mean Squared Error (RMSE)	$\sqrt{\frac{\sum_{i=1}^N (y_i - y'_i)^2}{N}}$
Mean Squared Error (MSE)	$\frac{\sum_{i=1}^N (y_i - y'_i)^2}{N}$
Root Mean Squared Logarithmic Error (RMSLE)	$\sqrt{\frac{\sum_{i=1}^N (\log(y_i + 1) - \log(y'_i + 1))^2}{N}}$
Index of Agreement (IA)	$1 - \frac{\sum_{i=1}^N (y_i - y'_i)^2}{\sum_{i=1}^N (y'_i - \bar{y} + y_i - \bar{y})^2}$

9.6.7 ROC: Esempio:



9.7 Considerazioni finali sulle ANN

- Le reti neurali artificiali sono ispirate ai processi di apprendimento che si svolgono in sistemi biologici.
- I neuroni artificiali e le reti neurali cercano di imitare i meccanismi di lavoro delle loro controparti biologiche.
- L'apprendimento può essere percepito/riformulato come ottimizzazione o minimizzazione dei costi o come processo di approssimazione.
- L'apprendimento neurale biologico avviene dalla modifica della forza sinaptica. Le reti neurali artificiali imparano allo stesso modo, regolando i pesi.
- Le regole di modifica della forza sinaptica per le reti neurali artificiali possono essere derivate applicando vari metodi matematici; come il gradient descent, ecc.
- Le reti neurali possono essere considerate come strumenti di approssimazione per funzioni non lineari (i.e., combinazioni lineari di funzioni di base non lineare), dove i parametri delle reti devono essere trovati applicando metodi di ottimizzazione.
- L'ottimizzazione è svolta in confronto all'approssimazione della misura d'errore.

9.7.1 Aspetti critici sulle ANNs

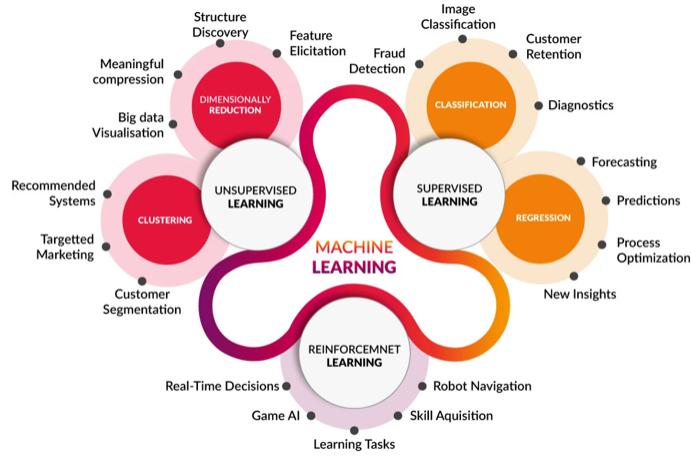
Le relazioni individuali tra le variabili di input e le variabili di uscita non sono sviluppate analizzando, modellando e ingegnerizzando conoscenza; ma in modo che il modello tenda a essere una scatola nera o una tabella di input/uscita senza base analitica.

È un approccio sub-simbolico; Non giustifica/spiega i risultati. È un metodo opaco, non trasparente e spiegabile.

La dimensione del campione deve essere grande. Richiede un sacco di prove ed errori, e quindi il training può risultare difficile e richiedere molto tempo.

L'uso di ANN richiede una buona comprensione del problema e della cooperazione con gli esperti del dominio.

Summary:



10 NLP & Text Mining - Applicazioni DL

10.1 Tecniche di base

L'approccio classico all'NLP è il seguente:

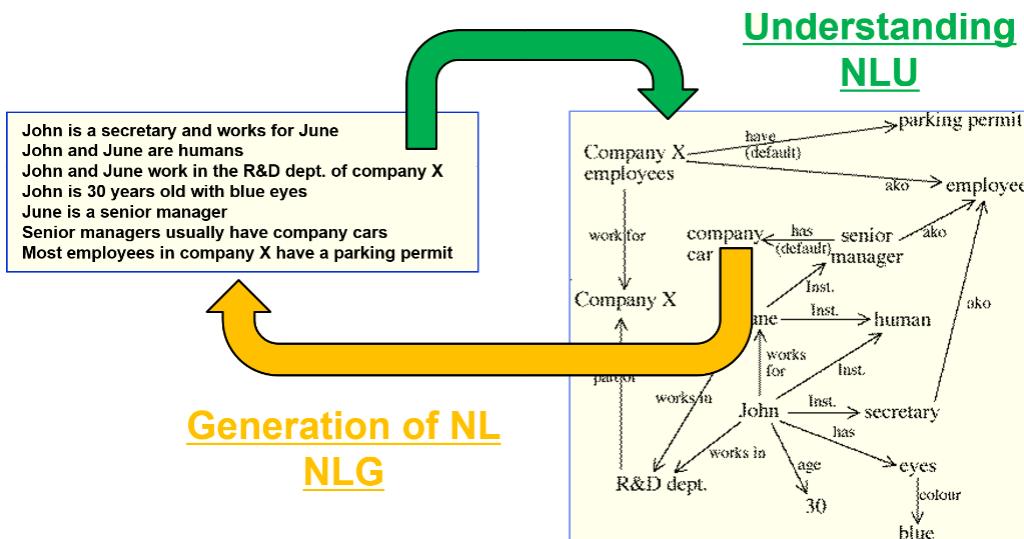
Analizzando contenuti testuali viene costruita una loro rappresentazione interna. Sulla Rappresentazione interna si può operare per risolvere vari compiti specifici. Classicamente venivano applicati tutti i livelli di elaborazione di Chomsky: lessicale, sintattico e semantico. Successivamente si sono adottate tecniche più ‘superficiali’/‘statistiche’, evitando ad esempio l’analisi sintattica e limitandosi all’analisi del discorso (Part-Of-Speech Tagging).

Esempi di elaborazioni su Documenti Web o Social Media, sono estrazione di concetti, valutazione del sentimento, classificazione automatica dei contenuti, automatic summarization, ecc..

Per quanto riguarda il Text Mining: il text mining è il processo di estrazione di conoscenza interessante e non banale da dati di testo non strutturati.

Elaborazioni preliminari sul testo: varie forme di pulizia/normalizzazione dei dati:

- A livello di caratteri – solitamente con la tecnica delle espressioni regolari: Conversione e tutto maiuscolo o minuscolo; Normalizzazione di numeri, simboli; Eliminazione/sostituzione di caratteri speciali (punteggiature, a capo, ..), tag removal, etc.
- A livello di parole: Stemming: le parole vengono sostituite con la loro radice lessicale; Stop-words: vengono eliminate parole ritenute poco rilevanti al fine di capire il significato
- A livello di struttura e lessicale: Tokenizzazione; POS Tagging - Part-of-Speech Tagging, Analisi del discorso.



10.1.1 Tokenization: Processo che analizza il testo e lo suddivide in elementi più piccoli, denominati token. Si identificano gli elementi di interesse, tipicamente le frasi, e nella frasi le parole. Si eliminano elementi non rilevanti quali ad esempio la punteggiatura, i caratteri ‘a capo’, etc.

10.1.2 Part of speech tagging: Processo che assegna a ciascun token la relativa parte-del-discorso, ossia il ruolo grammaticale, come ‘verbo’, ‘aggettivo’, ‘nome’, ‘avverbio’, ‘articolo’, ‘preposizione’,

10.1.3 Stopwords removal: Identificare ed eliminare parole che non contribuiscono significativamente all'analisi del contenuto/significato del testo.

Ha senso se l'analisi del contenuto ha come obiettivo l'identificazione del contenuto concettuale, la cosiddetta topicality.

Topicality di un testo = di cosa si parla in quel testo, che concetti, che argomenti (topic) sono contenuti.

10.1.4 Stemming: Ridurre ad uno stesso termine (la radice lessicale, o stem) le diverse varianti di uno stesso termine. Attenzione che non rimuova suffissi che differenziano i significati: es. universal e university. Si basa sull'assunzione che parole con lo stesso stem si riferiscano allo stesso concetto Molto comune l'Algoritmo di Porter.

10.2 Tecniche di base per la rappresentazione dei significati testuali

Insiemi o vettori di:

- keywords
 - radici lessicali

- termini singoli (Bag of Words - BoW)
- parole composte, key-phrase, POS, patterns
- n-grammi

Tali tecniche di base non affrontano adeguatamente i fenomeni linguistici della polisamia e della sinonimia causando falsi positivi e falsi negativi. Esistono tecniche più sofisticate e potenti:

- LSI/SVI
- Bayes
- WordNet

Finire rivedendo lez Tasso? slides troppo riassuntive