

# ESAME IIA

**Premessa:** ho scritto questi appunti facendo screenshot da altri appunti, facendo gli esami presi dal gruppo telegram (senza risposte corrette) e ragionando sulla teoria. Dunque nulla è sicuro di quello che ho scritto, ho cercato di fare del mio meglio ma errori qua e là possono esserci, per eventuali dubbi chiedere chiarimenti ai professori.

F.T.

## 1. Agenti

### Agente

Qualcosa che percepisce l'ambiente grazie ai sensori e capace di agire su di esso grazie agli attuatori.

Tipologie generali:

- Un Agente razionale per ogni sequenza di percezioni compie l'azione che massimizza il valore atteso della misura delle prestazioni, considerando anche la sua conoscenza pregressa.
- Un Agente autonomo ha un comportamento dipendente dalla sua esperienza.

Tipologie caratteristiche:

- Agenti reattivi semplici : l'azione viene determinata da stato e regole in quell'istante
- Agenti basati su modello : come prima solo che tiene conto della storia delle percezioni
- Agenti con obiettivo: guidati da un obiettivo nella scelta dell'azione, si pianificano le azioni in base al goal
- Agenti con valutazione di utilità : ci sono obiettivi alternativi più facilmente raggiungibili
- Agenti che apprendono: presente componente di apprendimento che produce cambiamenti al programma agente

### Funzione agente

Descrizione matematica che definisce l'azione da compiere per ogni sequenza percettiva. Il programma agente è l'implementazione di questa funzione.

### Percezioni e Azioni

Sequenza percettiva : Storia completa delle percezioni.

La scelta delle azioni è unicamente determinata dalla sequenza percettiva.

### Descrizione PEAS dei problemi

- Performance(obiettivo)
- Environment(ambiente dove si svolgono le azioni)
- Actuators(attuatori che svolgono le azioni)
- Sensors(sensori che percepiscono)

## 2. Parte Algoritmi

### Differenza tra nodo e stato

Gli stati non si ripetono, i nodi sì. Lo stato è una componente del nodo.

In frontiera non inserisco stati ma nodi non esplorati (non per forza negli alberi), ma la frontiera non coincide con la lista dei nodi esplorati in una ricerca che genera un albero di ricerca perché i nodi possono essere tolti dalla frontiera stessa.

Ricerca ad Albero: senza controllare che se i nodi siano già stati esplorati

Ricerca a grafo: si controllano che i nodi siano già stati esplorati

Frontiera: **lista dei nodi** in attesa di essere espansi, cioè le foglie dell'albero di ricerca, viene implementata come una coda. Sulla frontiera si possono eseguire : Pop , Vuota , Inserisci.

### Strategie

- Strategie non informate: BF, DF, DL, ID, UC
- Strategia informate: dette anche ricerca euristica, fanno uso di informazioni riguardo la distanza stimata dalla soluzione.

Valutazione di una strategia:

- completezza : se trova una soluzione
- ottimalità: se trova la soluzione migliore con costo minore
- complessità in tempo: tempo richiesto per trovare la soluzione
- complessità in spazio: memoria richiesta

Per definire queste ultime due:

- $b$  = fattore di ramificazione
- $d$  = profondità del nodo obiettivo più superficiale (depth)
- $m$  = lunghezza massima dei cammini nello spazio degli stati

### Funzionamento algoritmi non informati

- A\* - A a grafo : si espande il nodo partenza, calcolo la  $f$  per i nodi figli ( $f = g+h$  ). Faccio il pop dalla frontiera (cioè quelli espansi) del nodo a  $f$  minore e faccio check sul goal ( *posticipato* : nel senso che prima espando il padre poi verifico se è il goal ), poi genero i figli del nodo estratto e ricalcolo la  $f$ , ora guardo la coda con priorità (in ordine di  $f$ ) di tutti i nodi in frontiera e faccio la pop MA a grafo se facendo la pop nuovamente ritorno al nodo di cui ho già fatto la pop (marcato come esplorato : si marca quando si fa la pop ) non lo visito di nuovo, faccio una nuova estrazione. [se la  $h$  non è consistente ed ammissibile A\* non sarà ottimale]. Se invece trovo un nodo che è già in coda ma non esplorato (non ancora espanso) e ha costo minore, lo sostituisco.
- Depth-First [DF-albero]: prende un nodo e genera i suoi figli, il primo figlio sarà sempre messo in testa alla coda e verrà estratto per primo, così via andando in profondità, se si arriva in fondo si estraе il fratello dell'ultimo e così via a ritroso. *Nessun controllo sugli stati ripetuti*. Infatti se la coda generata in ordine alfabetico si rischia il ciclo.
- Uniform Cost [UC-grafo]: parto dal primo nodo, genero i figli e faccio una coda ordinata in base a  $g(n)$ , estraggo il nodo in testa alla coda e faccio il goal test, genero i suoi figli e li rimetto nella coda ordinata. Se uno dei figli è un nodo già presente in coda lo sostituisco se il nuovo costo fosse minore, se invece non è presente in coda ma è già stato estratto prima allora lo ignoro e non lo rimetto in coda. [= ad A\*]
- Greedy Best First [GBF-grafo]: uguale alla UC ma usa  $h(n)$  invece che  $g(n)$
- Breadth First(ricerca in ampiezza) [BF-Grafo]: si generano i figli, ogni volta che ne genero uno faccio il goal test, successivamente faccio la pop del primo e genero i suoi figli che anch'essi verranno testati e così via finché il goal test non dà positivo

Nelle versioni ad albero come detto prima l'algoritmo è differente unicamente nella mancata presenza di un controllo sugli stati ripetuti ( marcarli come esplorati ).

### Depth-limited (DL)

Ricerca in profondità limitata. Si va in profondità fino ad un certo livello predefinito  $l$ .  
Completa per problemi di cui si conosce un limite superiore per la profondità della soluzione:  
ad esempio **route-finding** [limitata da: **numero di città - 1**]. (es : 85 città allora  $l = 84$ )  
Completo se  $d < l$  con  $d$  profondità della soluzione].

Non ottimale

Complessità in tempo:  $O(b^l)$  *b sono i nodi*  
Complessità in spazio:  $O(b^*l)$

### ID (Iterative deepening-Approfondimento iterativo)

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

Caratteristiche :

- Complessità in tempo:  $O(b^d)$
- Complessità in spazio:  $O(b^*d)$ , vs  $O(b^d)$  del BF vs  $O(b^{d+1})$  di A\*
- completo
- non ottimale
- Miglior compromesso tra BF e DF. Nell'ID, i nodi dell'ultimo livello sono generati una volta, quelli del penultimo 2, del terzultimo 3. . . quelli del primo d volte.

### Ricerca Bidirezionale

In generale sussiste il problema della direzione della ricerca, si ha questa preferenza:

- in avanti quando gli obiettivi sono molti e abbiamo una serie di dati da cui partire
- indietro quando l'obiettivo è chiaramente definito oppure i dati del problema non sono noti e la loro acquisizione può essere guidata dall'obiettivo

Si procede in entrambe le direzioni.

La ricerca bidirezionale diminuisce sempre i costi di ricerca ma non è sempre applicabile.

Complessità in tempo :  $O(b^{d/2})$

Complessità in spazio:  $O(b^{d/2})$  Si potrebbe usare nel problema del labirinto.

### Ottimalità e completezza dei vari algoritmi

| Criterio  | BF              | UC                                      | DF             | DL              | ID              | Bidirez.     |
|-----------|-----------------|---|----------------|-----------------|-----------------|--------------|
| Completa? | Si              | Si <sup>1</sup>                         | No             | Si <sup>3</sup> | Si              | Si           |
| Tempo     | $O(b^d)$        | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(b^m)$       | $O(b^l)$        | $O(b^d)$        | $O(b^{d/2})$ |
| Spazio    | $O(b^d)$        | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(b \cdot m)$ | $O(b \cdot l)$  | $O(b \cdot d)$  | $O(b^{d/2})$ |
| Ottimale? | Si <sup>2</sup> | Si <sup>1</sup>                         | No             | No              | Si <sup>2</sup> | Si           |

<sup>1</sup> Per costi degli archi  $\geq \epsilon > 0$

<sup>2</sup> Se gli operatori hanno tutti lo stesso costo

<sup>3</sup> Per problemi di cui si conosce un limite alla profondità della soluzione (se  $l > d$ )

*per stesso costo si intende :  $g(n) = k * \text{depth}(n)$*

(dunque può essere definito come caso particolare di A\* considerando  $h=0$  e  $g$  uguale alla profondità)

**NB:** DF ad albero non risulta completa, causa possibili cicli, mentre a grafo risulta completa su spazi finiti (infiniti non completa) ma l'occupazione di memoria diventa  $O(b^d)$ .

### Ricerca euristica

#### **Algoritmo Best First**

Usa lo stesso algoritmo di UC ma usando  $f$  per la coda con priorità. La scelta di  $f$  determina la strategia di ricerca: ad ogni passo si sceglie il nodo con la  $f$  migliore (minore se si parla euristica basata sulla distanza dalla soluzione) sulla frontiera.

#### **Algoritmo A**

Un algoritmo A è un algoritmo Best First con una funzione di valutazione del tipo :

$f(n) = g(n) + h(n)$  con  $h(n) \geq 0$  e  $h(goal) = 0$ .  $g(n)$  è il costo del cammino *da radice a n* mentre  $h(n)$  è una stima del costo per arrivare *da n al nodo goal*.

Caratteristiche:

- UC è un caso particolare di A , ha :  $f(n) = g(n)$  [ottimo se costo archi > 0]
- GBF è un caso particolare di A, ha :  $f(n) = h(n)$  [non è ottimo, è greedy]
- $f = -\text{depth}$  diventa un DF, perchè più vai in profondità più  $f$  diminuisce
- è completo, cioè trova sempre la soluzione

#### **Ammissibilità**

Una funzione  $h$  è ammissibile se: *Per ogni nodo il cammino ottimo da esso al goal deve essere  $\geq$  alla h di quel nodo ( cammino ottimo  $\geq h(n)$  ).*

Significa che  $h$  deve essere una distanza in linea d'aria, una SOTTOSTIMA. La distanza in linea d'aria non sovrasta mai, rappresenta il costo minimo in situazioni ideali in cui non ci siano muri da aggirare.

#### **Algoritmo A\***

L'algoritmo A\* è un algoritmo A che ha la funzione euristica  $h$  ammissibile.

Caratteristiche:

- è ottimale e completo
- occupazione di memoria:  $O(b^{d+1})$ , quello in tempo dipende dall'euristica utilizzata
- l'algoritmo è lo stesso usato per UC solo con  $f(n) = g(n) + h(n)$
- non espande alcun nodo con  $f(b) > C^*$  ( $C^*$  è costo minimo/della soluzione ottima, è fissato)
- si cerca la funzione  $h$  più grande tra le ammissibili, così che grazie a  $C^*$  posso non espandere nodi che hanno  $f$  maggiore
- Non ottimizza l'occupazione di memoria
- se applicassi il goal test prima di mettere i nodi in frontiera piuttosto che al momento dell'estrazione allora potrei arrivare prima al goal con un percorso più lungo ed evitando magari quello ottimo che avrei trovato dopo
- la Ricerca in profondità [DF] è un caso speciale di A\* purché  $g$  sia pari all'inverso della profondità (all'aumentare del cammino in profondità il costo diminuisce) e  $h = 0$  (garantisce ammissibilità)
- la componente  $g(n)$  fa sì che si abbandonino i cammini che vanno troppo in profondità

Ricerca ad albero: ammissibile  $\rightarrow$  ottimalità

Ricerca a grafo: consistenza  $\rightarrow$  ottimalità

## **Consistenza/Monotonicità**

Una euristica è consistente se:

1.  $h(goal) = 0$
2. *Consistenza locale*:  $\forall n \mid h(n) \leq c(n, a, n') + h(n')$  dove  $n'$  è un successore di  $n$  e  $c(n, a, n')$  è il costo del cammino  $n \rightarrow n'$  sull'arco  $a \Rightarrow f(n) \leq f(n')$  [f non decresce mai lungo i cammini]

Teorema : monotonicità/consistenza → ammissibilità

Le euristiche monotone/consistenti garantiscono che:

- la soluzione meno costosa venga trovata per prima e quindi sono ottimali *anche nel caso di ricerca a grafo*.
- gli stati vengono espansi in ordine di  $f$  crescente
- si possano trascurare in fase di generazione gli stati che si trovano già nella lista degli esplorati o in frontiera

## **Euristica: Distanza Manhattan**

Senza vincoli/ostacoli è ammissibile.

$$h(x; y) = MD((x; y); (xg; yg)) = |x - xg| + |y - yg|$$

## **Concetto di euristica che domina**

$h_1$  è più informata di  $h_2$ , cioè “domina”  $h_2$ , se  $h_1 \geq h_2$  ( $h_2$  è più sottostima). Un'euristica più informata riduce lo spazio di ricerca, quindi è più efficiente ma molto più costosa da calcolare.  $h^*(n)$  massima informazione è l'oracolo[le domina tutte].

## **Algoritmi evolutivi basati su A\***

### **IDA\***

$A^*$  con approfondimento iterativo. IDA\* combina  $A^*$  con ID: ad ogni iterazione si ricerca in profondità con un limite (cut-off) dato dal valore della funzione  $f$  (e non dalla profondità).

Il limite  $f$ -limit viene aumentato ad ogni iterazione, fino a trovare la soluzione.

Caratteristiche:

- Lo scopo di IDA è ridurre l'occupazione di memoria di  $A^*$
- Completo e Ottimale se le azioni hanno costo costante  $k$  l' $f$ -limit viene aumentato di  $k$
- Occupazione in memoria  $O(b^*d)$  [come ID]

## **RBFS (Best First Ricorsivo)**

Tiene traccia del migliore percorso alternativo ad ogni livello. In caso di fallimento interrompe l'esplorazione quando trova un nodo meno promettente secondo  $f$ . Nel tornare indietro si ricorda il miglior nodo che ha trovato nel sottoalbero esplorato, per poterci eventualmente tornare.

## A\* con memoria limitata

Si cerca di utilizzare al meglio la memoria disponibile: quando la memoria è finita si "dimentica" il nodo peggiore, dopo aver aggiornato il valore del padre. A parità di f si sceglie il nodo migliore più recente e si dimentica il nodo peggiore più vecchio.

## Beam Search

La beam search tiene ad ogni passo solo i k nodi più promettenti, dove k è detto ampiezza del raggio (beam). Fare ciò semplifica la ricerca ma può fare perdere soluzioni. Non è completa e manco ottima.

## Ricerca Locale

### Hill Climbing

Consiste in una ricerca locale greedy, vengono generati i successori e poi valutati. Viene scelto un nodo che migliora la valutazione dello stato attuale (non si tiene traccia degli altri e non ho quindi l'albero di ricerca in memoria).

Tipologie:

- Hill Climbing a salita rapida (migliore)
- Hill Climbing stocastico (uno a caso tra quelli che migliorano)
- Hill Climbing con prima scelta (il primo)

### Tempra simulata

Il valore di T alto tende a favorire scelte del tutto casuali tra migliorative e peggiorative.

Questa tecnica introduce stocasticità, considerando anche mosse peggiorative, al fine di superare i minimi locali. (combina Hill Climbing e Stocasticità).

del tutto casuale perché poco efficiente. L'analogia è col processo di tempra dei metalli: portati a temperature molto elevate e raffreddati gradualmente consentendo di cristallizzare in uno stato a più bassa energia.

**Tempra Simulata** Ad ogni passo si sceglie un successore a caso:

Se migliora lo stato corrente, viene espanso

Se non lo migliora (caso in cui  $\Delta E = f(n') - f(n) < 0$ ), quel nodo viene scelto con probabilità  $p = e^{\Delta E/T}$ , con  $p$  ovviamente  $0 \leq p \leq 1$

(Si genera un numero casuale tra 0 e 1, se questo è  $< p$  il successore viene scelto, altrimenti no)

$p$  è inversamente proporzionale al peggioramento

T (temperatura) decresce al progredire dell'algoritmo, quindi anche  $p$ , secondo un piano definito. Col progredire, rende improbabili le mosse peggiorative.

**Analisi** La probabilità di una mossa in discesa diminuisce col tempo, e l'algoritmo si comporta sempre di più come Hill-Climbing.

Se T viene decrementato abbastanza lentamente, con probabilità tendente ad 1 si raggiunge la soluzione ottimale.

**Analogia:** T corrisponde alla temperatura e  $\Delta E$  alla variazione di energia

**Parametri** Valore iniziale e decremento di T sono parametri.

I valori per T sono determinati sperimentalmente: il valore iniziale di T è tale che per valori medi di  $\Delta E$ ,  $p = e^{\Delta E/T}$  sia circa 0.5

Formula della probabilità:  $p = e^{\Delta E/T}$

- T maggiore  $\Rightarrow p \rightarrow 1 \Rightarrow$  stocasticità totale
- T minore  $\Rightarrow 0 < p < 1 \Rightarrow$  mosse peggiorative improbabili

## Algoritmi per spazi continui > Gradiente

### 5.3.1 Gradient

Se la  $f$  è continua e differenziabile, ad esempio quadratica rispetto il vettore  $x$ , allora il minimo/massimo lo si può cercare utilizzando il **gradiente**, che restituisce la direzione di massima pendenza del punto.

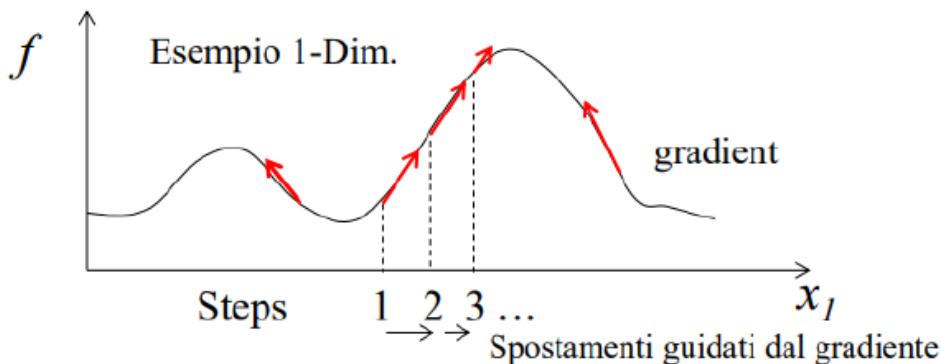
Data  $f$  obiettivo

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

**Hill-Climbing iterativo:**  $x_{new} = x_{old} + \eta \nabla f(x)$  con  $\eta$  dimensione dello step.

Quantifica lo spostamento, senza cercarlo tra gli infiniti possibili successori.

**Nota:** in generale non è sempre necessario il min/max assoluto. Vedremo nel ML.



## Giochi con avversario

Sono ambienti multi-agente competitivi, con avversari. Csp con  $k$  variabili e domini con  $n$  valori, il fattore di diramazione è  $n$  perché è uguale alla dimensione dei domini.

### Algoritmo MIN-MAX

L'albero viene enumerato in profondità.

Tempo :  $(b^m)$

Spazio:  $O(m)$

$m$  è la profondità

## Alfa e Beta

**Algoritmo** Consideriamo un nodo  $v$ : se c'è una scelta migliore sopra, allora quel  $v$  non sarà mai raggiunto. Max passerà da  $\alpha$  piuttosto che finire a  $v$ .

**Implementazione** Si va avanti in profondità fino al livello desiderato e, propagando indietro i valori, si decide se si può abbandonare l'esplorazione nel sotto-albero.

MaxValue e MinValue vengono invocate con due valori di riferimento iniziali

MaxValue =  $\alpha$  (inizialmente  $-\infty$ )

MinValue =  $\beta$  (inizialmente  $+\infty$ )

Rappresentano rispettivamente la migliore alternativa per Max e per Min fino a quel momento

I tagli avvengono quando, nel propagare indietro, si verifica:

$v \geq \beta$  per i nodi Max (taglio  $\beta$ )

$v \leq \alpha$  per i nodi Min (taglio  $\alpha$ )

Alfa riguarda la peggiore per il min e Beta peggiore per il max.

Alfa riguarda la migliore per il max e Beta la migliore per il Min ("fanno da garante" ovvero non espanderanno i nodi in cui non vengono rispettate le condizioni sotto riportate).

Tutti riguardano il valore migliore/peggiore da quel livello in giù (non vale per i superiori).

Caratteristiche :

- I tagli alfa avvengono in corrispondenza dei nodi MIN ( $v \leq \alpha$ ), mentre i beta dei nodi max ( $v \geq \beta$ )
- Può essere esteso a 3 o più giocatori
- Complessità in tempo con *ordinamento ottimale*:  $O(b^{m/2})$ , con  $b$  fattore di diramazione e  $m$  la profondità. Complessità spaziale:  $O(m)$
- senza considerare eventuali tagli, l'algoritmo procede in ampiezza fino al livello di cut off, si va in fondo fino al livello desiderato. Arriva a profondità doppia rispetto a MINMAX
- mosse scelte sono uguali a MINMAX , semplicemente risparmio tempo

Ordinamento ottimale delle mosse: sotto i nodi max le mosse sono ordinate in ordine decrescente e sotto i min in ordine crescente.

Funzionamento: vado nella foglia più a sinistra in fondo, aggiorno il valore di min o di max in base a quale sia l'ultimo nodo (cioè di quel nodo che ha le foglie), poi vado nella foglia a destra e quando ho valutato tutte le foglie di quel nodo aggiorno alfa se era nodo di max (scelta migliore per esso), aggiorno beta se era era nodo di min(scelta migliore per esso).

Dopo aver fatto ciò, a ritroso aggiorno i valori scambiando ogni volta alfa e beta [es: ho (-inf,8) al nodo più in basso, il nodo sopra avrà (8,+inf), quello sopra ancora avrà (-inf,8) e così via. Ognuno di quei nodi però propagherà verso il basso l'intervallo così com'è, senza scambiare i valori. Quando sarò di nuovo alle foglie prendo il valore della foglia più a sinistra poi verifico se devo effettuare un taglio sulle foglie più a destra, se devo taglio se non devo aggiorno il valore di beta se era un nodo min e alfa se era il nodo max, propago la modifica verso l'alto.

### **CSP (problemi di soddisfacimento di vincoli)**

Problema descritto da 3 componenti :

1. X insieme di variabili
2. D insieme di domini (esso ha un insieme di valori possibili per la variabile  $X_i$ )
3. C insieme di vincoli (costituito da [variabili, relazione] )

### **Algoritmo di ricerca con Backtracking a profondità limitata**

Ricerca con backtracking a profondità limitata: controllo anticipato della violazione dei vincoli, si fa backtracking non appena un vincolo è stato violato.

- **Scelta delle variabili:** **MRV(minimum remaining value)** → scegliere la variabile con meno valori legali residui, quella più vincolata. A parità di MRV si usa: **Euristica del grado** → scegliere la variabile coinvolta in più vincoli con le altre variabili (quella più vincolante o di grado maggiore)
- **Scelta dei valori:** **LCV (Least constraining value)** → si sceglie il valore meno vincolante, quello che esclude meno valori possibili tra le variabili.
- **Propagazione dei vincoli:** **FC (Forward Checking)** → una volta assegnato un valore vado a vedere le variabili direttamente collegate nel grafo dei vincoli oppure : **AC(Arc Consistency)** → restringo i valori dei domini delle variabili tenendo conto dei vincoli unari e binari su tutto il grafo (itero finché tutti i nodi e archi sono consistenti) [ ∀ arco verifico che il nodo binario sia soddisfatto]

FC : se istanzio una variabile, es C = 4, allora controllo solo le variabili che la riguardano e restringo i relativi domini e basta, dato che è una verifica locale.

AC: per ogni arco controllo tutti i vincoli binari e restringo i domini.

### **Metodi locali per CSP > Euristica dei conflitti minimi (min-conflicts)**

Questa euristica può essere usata per il problema delle regine e consiste *nel scegliere un valore che crea meno conflitti*.

### **3. Parte Logica**

In questi parte lo stato è una base di conoscenza (KB) a cui porre domande, la quale è rappresentata in un linguaggio espressivo come il PROP (calcolo proposizionale) o il FOL (logica del primo ordine).

#### **KB**

Una KB ha capacità inferenziale, cioè di derivare nuovi fatti da quelli memorizzati. Costituita da un insieme di regole, fatti specifici e generali in un linguaggio simbolico.

#### **Sintassi del PROP**

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

- simbolo  $\rightarrow P \mid Q \mid R \dots$
- formula  $\rightarrow \text{formulaAtomica} \mid \text{formulaComplessa}$
- formulaAtomica  $\rightarrow \text{Ture} \mid \text{False} \mid \text{simbolo}$
- formulaComplessa  $\rightarrow \neg \text{formula} \mid \text{formula} \wedge \text{formula} \mid \text{formula} \vee \text{formula} \mid \text{formula} \Rightarrow \text{formula} \mid \text{formula} \Leftrightarrow \text{formula}$

Possiamo omettere le parentesi assumendo questa precedenza tra gli operatori:

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

#### **Definizioni**

- **Formula soddisfacibile** : se esiste una interpretazione che la rende vera ( o comunque verifica una conseguenza logica)
- **Formula valida (tautologia)** : ci deve essere la conseguenza logica per ogni interpretazione, è vera in tutte le interpretazioni. Oppure: se e solo se  $\neg$ Formula è insoddisfacibile
- **Formula insoddisfacibile** : non deve esistere una interpretazione che la rende vera o che verifichi la conseguenza logica
- **Modello** : assegnamento che soddisfa tutte le clausole
- **Interpretazione** : assegnamento di T/F a tutti i simboli
- **Clausola vuota** : è una contraddizione

#### **Algoritmi di inferenza per calcolo proposizionale (PROP) [Model Checking / soddisficiabilità]**

##### **1. Algoritmo TT-Entails**

Obiettivo: verificare la conseguenza logica, fa Model Checking (enumera tutte le possibili interpretazioni per verificarla).

Enumero tutte le possibili interpretazioni di KB ( $k$  simboli,  $2^k$  possibili interpretazioni). Per ciascuna interpretazione:

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

Tramite tabella di verità: devo soddisfare ogni clausola della KB poiché la KB sia ‘vera’, in queste interpretazioni dette “modelli” voglio verificare la conseguenza logica di  $\alpha$ , per fare ciò devo vedere sulla colonna  $\alpha$  se ho tutti true, se ho anche un solo false allora non è conseguenza logica.

Tramite algoritmo: assegno  $\alpha = T$  e chiamo ricorsivamente assegnando a  $T$  i successivi attributi verificando che la chiamata non restituisca false

## 2. Algoritmo DPLL

Obiettivo: verificare la soddisfabilità di  $KB \models A$  se  $KB \wedge \neg A = \{\}$ .

Usa una enumerazione in profondità di tutte le possibili interpretazioni alla ricerca di un modello.

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

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

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

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

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

Verifico clausola una per una, prima si assegnano i simboli puri poi le clausole unitarie nella KB rimanente, appena una la soddisfo la tolgo dalla KB. Se non ci fossero più puri e clausole unitarie si va in profondità (cioè se ho provato  $P = \text{true}$  e non va allora proverò  $P = \text{false}$ ).

Caratteristiche:

- è completo e termina sempre
- è relativo al problema della soddisfabilità per il calcolo proposizionale ed è decidibile
- complessità esponenziale, enumerazione in profondità
- richiede una forma a clausole, usa l’euristica delle clausole unitarie
- può essere usato per stabilire la conseguenza logica, infatti si aggiunge il negato del simbolo alfa da dimostrare e se l’algoritmo da FAIL perché va in contraddizione allora per il teorema di refutazione è verificata la conseguenza logica

### 3. Algoritmo WalkSAT

Obiettivo: assegnamento che soddisfa tutte le clausole (cioè un modello).

Ad ogni passo sceglie una clausola non ancora soddisfatta. Poi sceglie un simbolo da modificare (flip) con probabilità  $p$  (di solito 0.5) tra una delle seguenti possibilità:

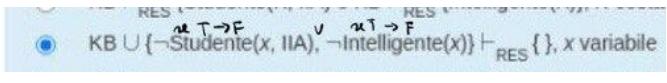
- **Passo casuale** : sceglie un simbolo a caso da flippare (utile per superare il problema dei minimi locali)
- **Passo di ottimizzazione**: sceglie quello che rende più clausole soddisfatte (attenzione: se ci sono più flip che rendono più clausole soddisfatte rispetto a prima si sceglie quello che ne ha di più soddisfatte rispetto a prima, a parità di questi sono tutti corretti)

Risulta incompleto, si arrende dopo un certo numero di flip, non può verificare l'insoddisfacibilità. WalkSAT è molto efficace quando il problema è sottovincolato.

*Il problema della soddisfacibilità è in funzione di  $m/n$ , dove  $m$  è il numero di clausole mentre  $n$  è il numero di simboli.*

*Attenzione: all'esame è stato chiesto di definire questo algoritmo come problema di ricerca, dunque tra varie opzioni trovare quelle che definiscono (stato, funzione di valutazione, stato iniziale, stato successivo e stato obiettivo).*

#### Teoremi, definizioni e proprietà importanti:

- **Teorema di refutazione :**  $\text{KB} \models \alpha \Leftrightarrow \text{KB} \cup \{\neg\alpha\}$  è insoddisfacibile  
esempio : “esiste almeno uno studente intelligente” nego che uno studente possa essere entrambi  
  
NB: Questo teorema vale anche per il FOL.
- **Completezza di un sistema deduttivo** : tutto ciò che è conseguenza logica è anche deducibile  
 $\text{Se } \text{KB} \models A, \text{ allora } \text{KB} \vdash A$
- **Correttezza di un sistema deduttivo** : tutto ciò che è derivabile è anche conseguenza logica  
 $\text{Se } \text{KB} \vdash A, \text{ allora } \text{KB} \models A$
- **Teorema di risoluzione :**  $\text{KB}$  insoddisfacibile  $\Leftrightarrow \text{KB} \vdash_{res} \{\}$  completezza  
(con  $res$  si intende applicazione della regola di risoluzione)
- **Definizione di conseguenza logica**

$$\text{KB} \models \alpha \text{ sse } M(\text{KB}) \subseteq M(\alpha)$$

## 4.11 Regola di risoluzione

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

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

### 4.11.1 Regola di risoluzione in generale per PROP

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

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

## FOL

**Arietà** : numero di parametri della funzione o del predicato.

### 5.3.1 Predicati

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

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

### 5.3.2 Termini

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

### 5.3.3 Formule

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

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

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

$\forall x Mela(x) \Rightarrow Rossa(x)$     x è legata

## Costante di Skolem

Si può sostituire  $\exists x A[x]$  con  $A[k]$  con k costante di skolem. Se  $\exists$  non compare nell'ambito del  $\forall$  è una costante nuova di skolem, altrimenti va introdotta una funzione di skolem:

$\exists x Padre(x, G)$  diventa  $Padre(k, G)$

$\forall x \exists y Padre(x, y)$  diventa  $\forall x Padre(x, p(x))$  e non  $\forall x Padre(x, k)$  altrimenti tutti avrebbero lo stesso padre

## Trasformazione in forma a clausole

Formula a clausole : rappresenta la disgiunzione di letterali

### 5.9.2 Esempio di trasformazione (passo passo)

Esempio di trasformazione in dettaglio per la frase:

"Tutti coloro che amano tutti gli animali sono amati da qualcuno"

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

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

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

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

2. Negazioni all'interno:

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

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

3. Standardizzazione delle variabili: ogni quantificatore una variabile diversa

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

4. Skolemizzazione: eliminazione dei quantificatori esistenziali.

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

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

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

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

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

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

7. Notazione a clausole

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

8. Separazione delle variabili: clausole diverse, variabili diverse

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

## Unificazione

Devo verificare che :  $P(x,y,z) == P(k,w,q)$  tramite sostituzione x/w.

Regole:

- FAIL se non è possibile (nessuna delle risposte fornisce una sostituzione corretta)
- Non si possono fare sostituzioni ricorsive (es:  $x/g(y)$ ,  $y/b$  oppure  $y/b, z/y$  oppure  $x/f(x)$ )
- Le costanti sempre a destra
- Devo trovare l'MGU(most general unifier)
- A sinistra non può comparire una funzione (es:  $G(y)/A$ )

## Metodo di risoluzione FOL: correttezza e completezza

La deduzione per risoluzione è corretta ma non è completa.

Se uso il teorema di refutazione, che è valido per il FOL, aggiungendo quindi il negato di alfa (da dimostrare), e trovo la clausola vuota ho che il metodo di risoluzione è anche completo.

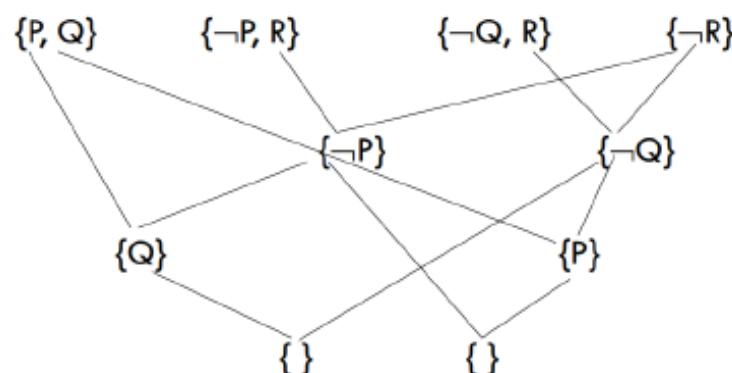
## Dimostrare che una formula del FOL(logica dei predicati) alfa è valida con metodo di risoluzione

Una formula alfa è valida se e solo se not(alfa) è insoddisfacibile (teorema di refutazione).

### Strategia di risoluzione > Strategie di cancellazione :

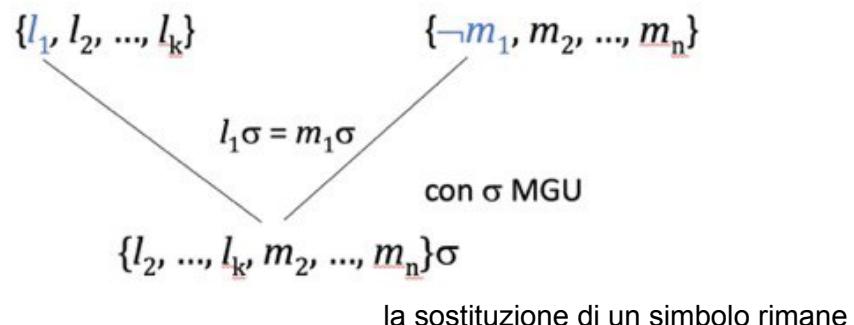
1. **Sussunzione** :  $P(x)$  sussume/implica  $P(A)$ ,  $P(A)$  sussume  $\{P(A), P(B)\}$ . In generale alfa sussume beta se esiste una sostituzione su alfa che porta alfa ad essere un sottoinsieme di beta. Dunque beta può essere eliminata.
2. **Clausole con letterali puri**: Sono i letterali che compaiono sempre senza il negato in tutta la KB, possono essere cancellate.
3. **Tautologie**: Quando in una clausola compare un letterale e il suo negato.

**Strategia di risoluzione > Strategia di restrizione > Risoluzione Unitaria** : faccio la risoluzione in cui una delle due deve essere una clausola unitaria. Se con *Clausole di Horn* (che hanno al massimo un letterale positivo) *risulta completa*, cioè dimostra la soddisficiabilità. La risoluzione dovrebbe essere anche corretta (in generale per la risoluzione deduttiva FOL) .



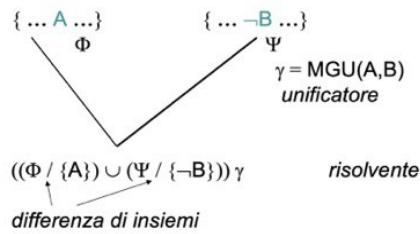
### Strategie di risoluzione > Strategia di ordinamento

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

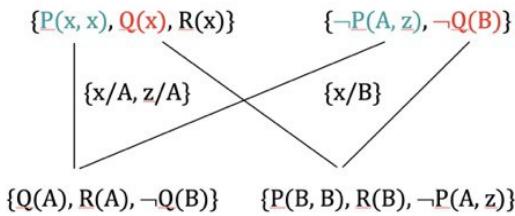


## Refutazione tramite risoluzione per il FOL

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

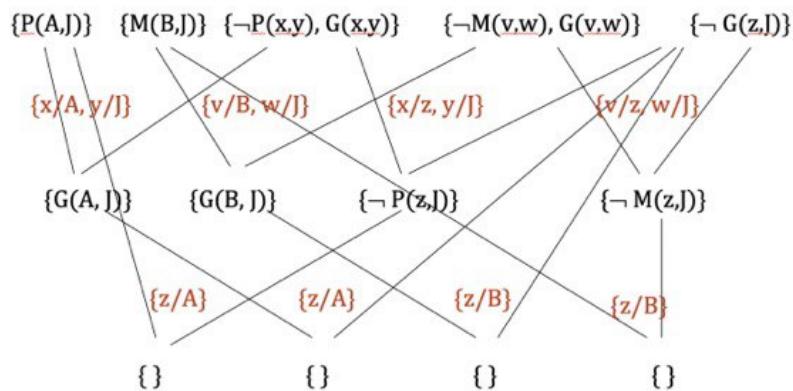


### 5.12.1 Esempio metodo di risoluzione



Grafo di risoluzione

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



Le risposte sono: A, B

//manca parte di PROLOG, chiesta al mio esame

### **3. Parte Machine Learning**

#### **Learning Supervisionato e Non Supervisionato**

##### **2.1 TASK: Supervised Learning**

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

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

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

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

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

Entrambi sono compiti di approssimazione di funzione.

##### **2.2 TASK: UNsupervised Learning**

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

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

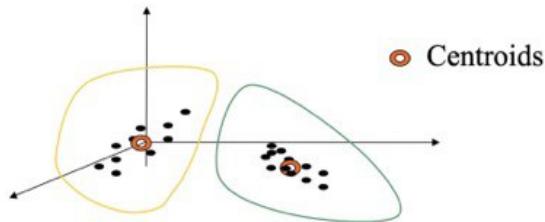


Figure 1: Esempio di clustering

Si può avere in principio underfitting o overfitting per i tutti i modelli supervised.

#### **Concetto di Ipotesi**

- Hypothesis: è una funzione  $h$  che è ritenuta di essere simile a  $f$ . La funzione  $h$  è data in un linguaggio capace di esprimere relazioni tra i dati.
- Hypothesis space: è lo spazio di tutte le ipotesi che possono essere output dell'algoritmo. E dove cerchiamo la funzione  $h$ .

#### **Concept Learning**

Il Concept Learning anche noto come *apprendimento di categoria* è definito come “*la ricerca di attributi che possono essere utilizzati per distinguere modelli dai non modelli di varie categorie*”. I concetti sono le categorie che ci aiutano a classificare oggetti, eventi e idee. In un compito di Concept Learning, una macchina apprende come classificare degli oggetti mostrandole prima una serie di esempi associati alle loro etichette di classe. La macchina semplifica quello che ha osservato "condensandolo" sotto forma di un esempio. Questa versione semplificata di ciò che è stato appreso viene quindi applicata a esempi futuri.

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

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

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

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

Numero di possibili istanze:  $3^2 \cdot 2^2 \cdot 2^2 \cdot 2^2 = 96$

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

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

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

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

### 3.7 Da ordine generale a ordine specifico

Consideriamo due ipotesi

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

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

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

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

#### Lookup table

Non ha bias, incapace di generalizzare. Un unbiased learner non generalizza perché ogni istanza viene classificata + da metà delle ipotesi e - dall'altra metà.

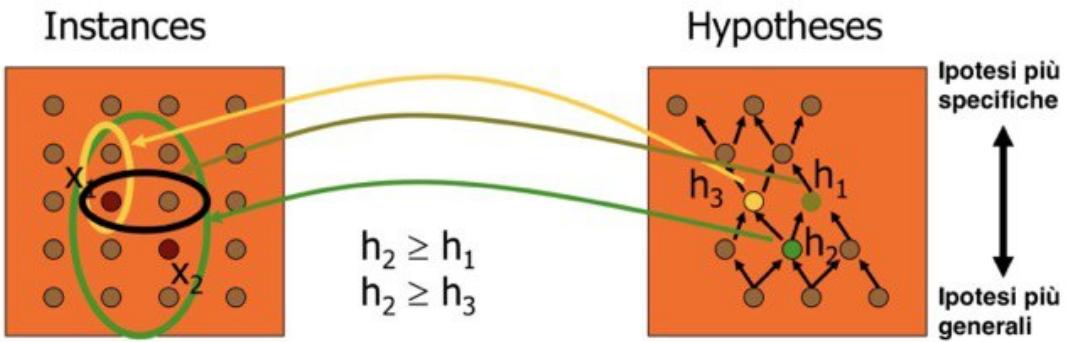


Figure 2: Partial Order

Istanze:

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

Ipotesi:

- $h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ?, ? \rangle$
- $h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ?, ? \rangle$
- $h_3 = \langle \text{Sunny}, ?, ?, ?, \text{Cool}, ? \rangle$

### 3.8 Algoritmo Find-S (Specific)

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

1. Inizializza  $h$  con l'ipotesi più specifica nello spazio delle ipotesi  $H$  ( $h_0 = \langle 0, 0, 0, 0, 0, 0 \rangle$ )
2. Prendo il prossimo esempio di training:
  - Per ogni attributo  $a_i$  dell'ipotesi  $h$ , se  $a_i$  è soddisfatto in  $h$  da  $x$  allora non fare nulla (quindi se l'esempio è negativo non ci sono cambiamenti sull'ipotesi), altrimenti se  $a_i$  NON è soddisfatto e l'esempio è positivo, sostituisce  $a_i$  con il prossimo vincolo più generale soddisfatto da  $x$ . Ripeto il passaggio per ogni esempio positivo.
3. Viene dato in output l'ipotesi  $h$

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

Non so quanto i dati di allenamento siano coerenti perché non classifica gli esempi negativi e non so nemmeno se il sistema di apprendimento converga con il concetto target.  
Lo spazio delle ipotesi è rappresentato come congiunzione di attributi e ciò è molto limitativo.

## **Definizione Version Spaces**

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

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

$$\text{dove } Consistent(h, D) = \forall < x, c(x) > \in D | h(x) = c(x)$$

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

Il VS può essere rappresentato solo dal membro più generale G e dal più specifico S.

Teorema: ogni membro del Version Space si trova tra:

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

## **Candidate Elimination**

L'insieme S è il guardiano dei +, esso ad ogni esempio di training positivo si generalizza per dare + su questo esempio nuovo e sui precedenti, mentre l'insieme G è il guardiano dei -, ad ogni esempio di training negativo rende più specifica la G rimanendo coerente con gli esempi negativi precedenti.  $VS = \{S, G\}$

Conto la maggioranza nel VS e in S e G, se c'è parità allora il CE risponde rejection, sennò + o -.

L'insieme G (general boundary) di un VS è l'insieme dei membri massimamente generali di H consistenti con l'insieme dei dati G [per S è lo stesso ma minimamente].

Non è sempre sufficiente guardare S per vedere se soddisfa le ipotesi: se la soddisfa sicuramente è positivo su tutto essendo che il resto è più generale, se non la soddisfa bisogna vedere il VS e G.

## **Bias induttivo**

Rappresenta un insieme minimo di asserzioni, tramite il quale si fanno deduzioni. Si può anche dire che è l'insieme delle asserzioni che ci permette di trasformare il problema da induttivo a deduttivo.

È necessario ad un sistema per permettere di generalizzare ma non è sufficiente. (CN ma non CS). Bias induttivo più forte si intende più restrittivo.

*BI Find-S > BI Candidate Elimination > BI Linear Model*

Il BI è di 2 tipi [non possono coincidere] :

- language bias [vincolo sulle ipotesi] → spazio di ricerca incompleto
- search bias [vincolo sulla ricerca] → strategia (algoritmo di ricerca) incompleta

Si preferisce quello di ricerca perché preferisco avere tutte le possibili funzioni tra cui quella migliore che affidarmi ad una ricerca completa ma con meno funzioni.

Un sistema di apprendimento che non fa assunzioni preliminari riguardo all'identità del concetto target non ha basi razionali per classificare eventuali istanze non conosciute, dunque un sistema non vincolato non è in grado di generalizzare. Dunque H (spazio delle ipotesi) non può coincidere con l'insieme di tutte le funzioni possibili ma la ricerca deve essere esaustiva.

#### 4.4 Tre (algoritmi) sistemi di apprendimento con Bias differenti

1. Rote Learner: (lookup table) L'apprendimento corrisponde semplicemente alla memorizzazione di ogni esempio di allenamento osservato nella memoria. Le istanze successive vengono classificate osservandole nella memoria. Se l'istanza viene trovata in memoria, viene restituita la classificazione memorizzata. In caso contrario, il sistema rifiuta di classificare la nuova istanza. In poche parole salva gli esempi, classifica  $x$  se e solo se si "accoppiano" con uno degli esempi visti precedentemente.

Non c'è Bias Induttivo → nessuna generalizzazione

2. Candidate Elimination: Le nuove istanze vengono classificate solo nel caso in cui tutti i membri del Version Space corrente concordino la classificazione. In caso contrario, il sistema rifiuta di classificare la nuova istanza.

Bias Induttivo → il concetto target può essere rappresentato nel suo spazio di ipotesi.

3. Find-S: Questo algoritmo, descritto in precedenza, trova l'ipotesi più specifica coerente con gli esempi di addestramento. Quindi utilizza questa ipotesi per classificare tutte le istanze successive. Bias Induttivo → il concetto target può essere rappresentato nel suo spazio delle ipotesi e tutte le istanze sono istanze negative a meno che l'opposto non sia implicato da altre sue conoscenze.

#### Complessità/Flessibilità in fitting

Si indica quello che puoi rappresentare nello spazio delle ipotesi.

La scarsa capacità di generalizzare può dipendere sia da overfitting che da underfitting.

#### Modelli lineari

##### 5.3 Costruzione della funzione via LMS

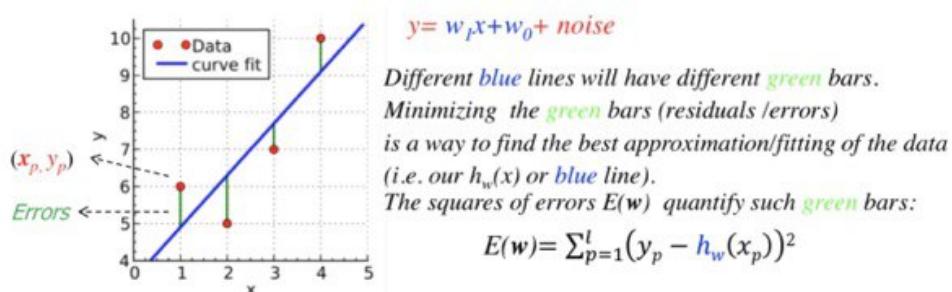
L'algoritmo LeastMeanSquare consiste nel trovare i  $w_i$  per cui l'errore è minimizzato (miglior adattamento dei dati sul training set con  $l$  esempi).

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

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

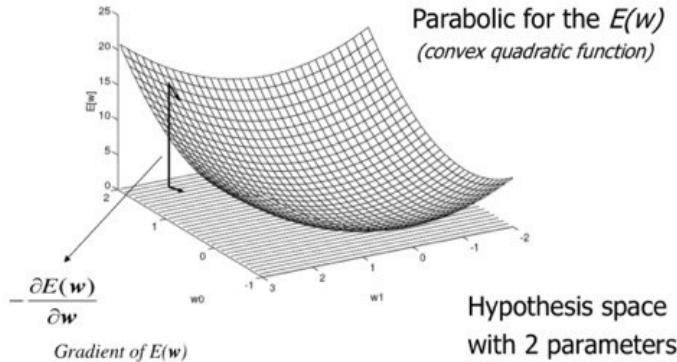
dove  $x_p$  è il p-esimo input e  $y_p$  è il p-esimo output dell'esempio p. Dividendo per l ottengo la media.

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



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

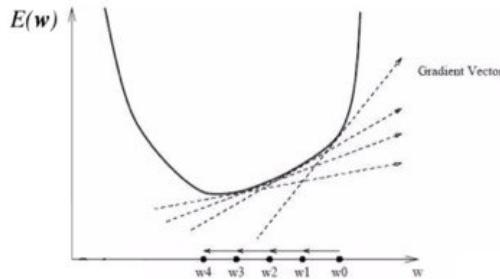
Si cerca il minimo locale della Loss, che si trova dove il gradiente è = 0. Se la funzione perdita è convessa abbiamo la seguente soluzione diretta poiché non abbiamo nessun minimo locale:



### 5.3.1 Gradiente discendente

La derivazione precedente suggerisce la costruzione di un algoritmo iterativo basato su  $\partial E(w)/\partial w_i$ . Il gradiente ci indica la direzione di salita, possiamo spostarci verso il minimo con discesa del gradiente  $\Delta w = -\text{gradiente}E(w)$ . La ricerca locale inizia con il vettore che contiene i valori dei pesi, viene modificato iterativamente per diminuire fino a minimizzare l'errore della funzione.

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



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

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

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

Non è detto che detto che non fermarsi presto assicura un buon learning: nonostante potrebbe essere vero che fermarsi precocemente possa precludere il trovare una buona soluzione, non è detto in generale che continuare possa portarci ad una buona soluzione (compromesso desiderato tra accuratezza e complessità del modello).

La discesa del gradiente può essere usata anche con le LBE.

La discesa del gradiente può essere usata per fare il training di un classificatore.

### Caratteristiche:

- Non è detto che abbiano una flessibilità limitata che permette di evitare l'overfitting.
- Un modello lineare con tutti  $w$  nulli darebbe  $h(x) = 0$  sempre, non è una lookup table.
- Sono lineari nei parametri liberi [ $w$ ]
- Moltiplicare la  $h(x)$  per costanti positive non fa variare la disequazione (Free-Scaling)
- Le  $h(x) = \text{sign}(x * w^T)$  sono modelli per la classificazione e sono a massimo margine
- Le  $h(x)$  realizzano un AND logico, non sempre
- $w_0$  si chiama BIAS o THRESHOLD (se = 0 è nullo, gac.) , è la distanza dalla componente 0
- $h$  realizza una AND se costruendo la tabella essa rispetta l'AND logico
- lineare  $\leq$  polinomiale
- Num. di  $w$  [vettore  $w$  più grande di dimensione, cioè maggiore dimensione dell'input]  $\rightarrow$  VC-Dim alta
- se metto  $x_0 = 1$  allora posso scrivere  $w^T x = x^T w$
- la strategia di ricerca non è completa, essendo una ricerca locale (discesa del gradiente), dipende anche molto da come vengono modificati i  $w$  in base all'algoritmo scelto (la discesa del gradiente è solo uno dei tanti), dunque ha bias di ricerca, cioè una restrizione della strategia di ricerca

### Linear basis expression(LBE):

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

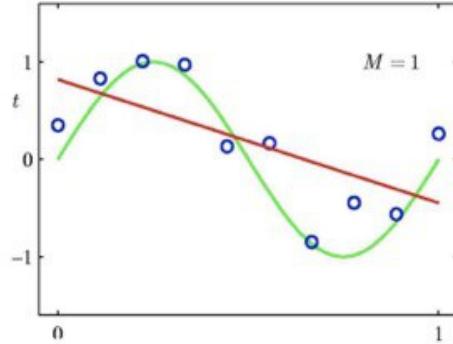
$$h_w(x) = \sum_{k=0}^K w_k \phi_k(x) \quad (13)$$

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

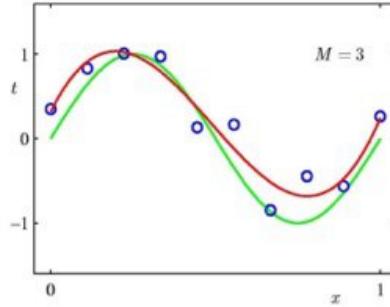
Per lineare nei parametri si intende nei  $w$ . Grazie a questa tecnica posso aumentare la flessibilità del modello gestendo la non linearità nelle variabili di input [ $x$ ] e le pluri dimensioni di  $x$ , quindi usando un classificatore lineare con LBE posso risolvere problemi *non linearmente separabili*. Attenzione che il grado  $x^j$  regola il compromesso tra underfitting e overfitting, anche se in realtà dipende anche dalla dimensione del vettore  $w$  e di  $x$  [dimensione in input], se aumenta si va verso l'overfitting (VC-Dim alta).

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

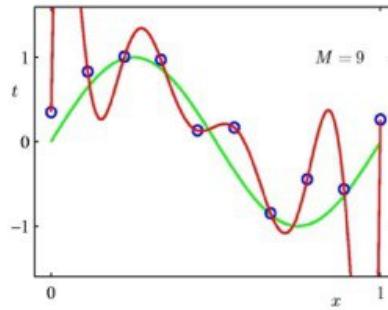
- grado del polinomio = 1, notiamo che è esegue troppo "underfitting".



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



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



### 5.9.2 Tradeoff per la complessità

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

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

## Ridge Regression

La ridge regression è un modello LMS ma regolarizzato, poiché uno non regolarizzato fitta bene ma generalizza male, come si fa? Si aggiunge una penalizzazione alla complessità del modello della Loss, sommando la norma del vettore  $w$ , moltiplicato per un parametro  $\lambda$  (costante) chiamato coefficiente di regolarizzazione, grazie ad esso non devo più modificare il grado dei polinomi o analizzare funzioni più complesse.

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

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

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

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

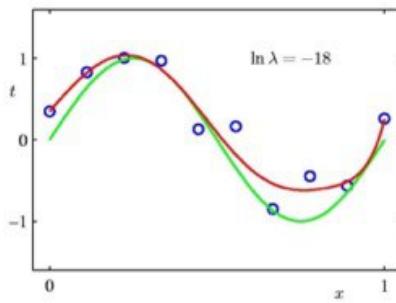


Figure 4: Polinomio di grado 9 con  $\log_e(\lambda) = -18$  quindi  $\lambda$  basso

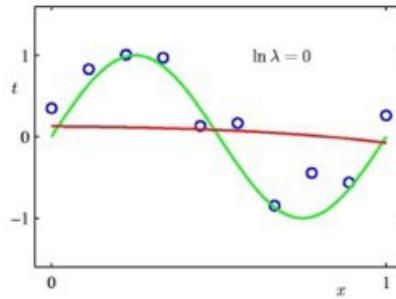


Figure 5: Polinomio di grado 9 con  $\log_e(\lambda) = 0$  quindi  $\lambda$  troppo alto

La regolarizzazione sposta tutto sul bias di ricerca, prima la complessità era tutta sul bias di linguaggio, ovvero vincoli posti sul modello (lineare, quadratico).

Attenzione: la ridge regression con  $\lambda$  può essere applicata solo a LBE polinomiali.

$\lambda$  alto  $\rightarrow$  underfitting  $\rightarrow$  Remp alto  $\rightarrow$  VC-Bound alto  $\rightarrow$  VC-Dim bassa  $\rightarrow$  VC-Confidence bassa

$\lambda$  basso  $\rightarrow$  overfitting  $\rightarrow$  Remp basso  $\rightarrow$  VC-Bound basso  $\rightarrow$  VC-Dim alta  $\rightarrow$  VC-Confidence alta

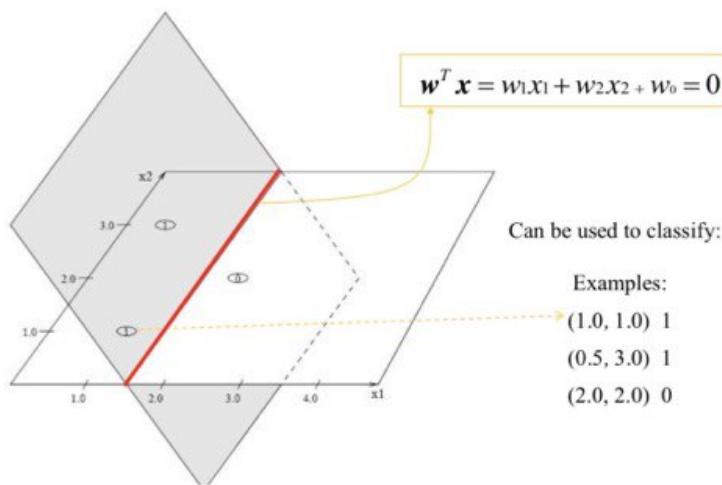
## Classificatore lineare

### 5.12 Classificazione con il modello lineare

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

#### 5.12.1 Visione geometrica dell'iperpiano

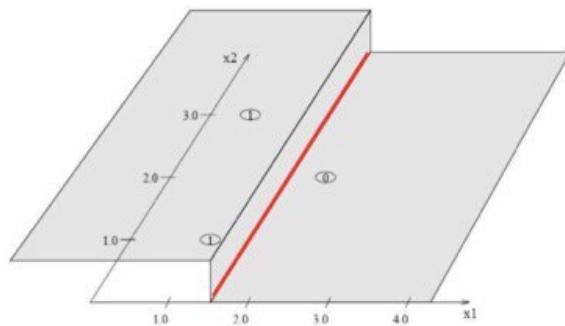
$w^T * x$  (prodotto di vettore riga per vettore colonna) definisce un iperpiano, il Decision Boundary è il luogo geometrico<sup>3</sup> dei punti dove l'iperpiano vale zero e può essere utilizzato per classificare.



Quindi possiamo definire una funzione di classificazione del tipo:

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

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




---

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

*Mind the  $\geq$*

NB: la formula del classificatore lineare è stata chiesta, gli indici della sommatoria possono cambiare

Attenzione: se in un punto il target(output) è 0 un coefficiente  $w$  può essere abbassato, tramite le regole di correzione presenti nel modello lineare.

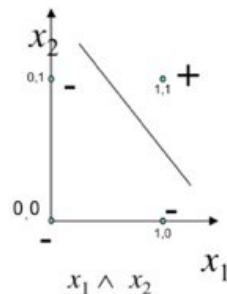
### 5.15 Congiunzioni nel caso di Modello Lineare

Possiamo rappresentare congiunzioni con i modelli lineari, ad esempio:

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

Altro esempio con grafico:

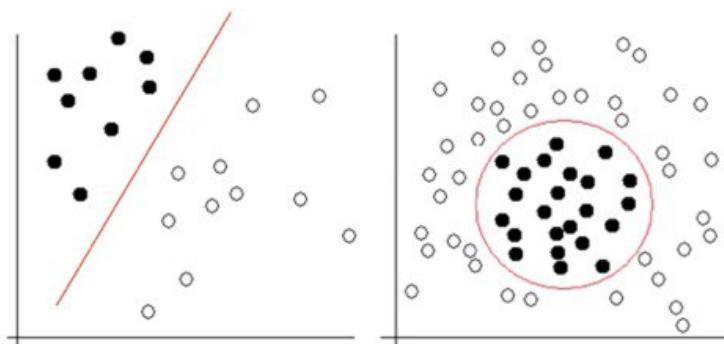
$x_1 \wedge x_2$  cioè  $1x_1 + 1x_2 > 1$



In generale  $w$  può essere appreso per trovare la soluzione.

#### 5.15.1 Limitazioni

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



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

NB: congiunzioni(AND logico)  $\subseteq$  modelli lineari

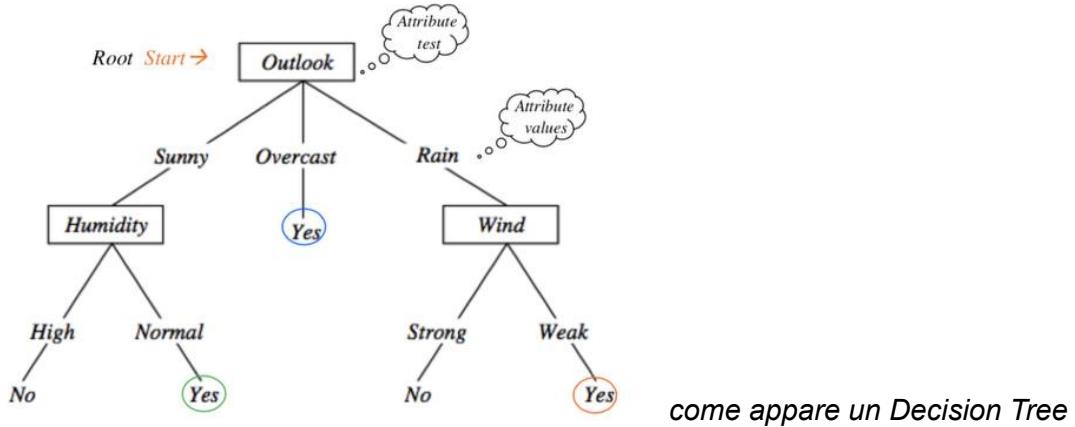
Nel problema di apprendimento per i classificatori lineari si può usare l'algoritmo di discesa del gradiente, calcolando il  $\Delta w$ .

L'iperpiano separatore è detto anche LTU (linear threshold unit): è una soglia/scalino che divide le zone di decisione (ad esempio zone dove vale 0 o vale 1).

#### Decision Tree

Ogni nodo dell'albero rappresenta un test su un attributo e ogni ramo corrisponde alla scelta del valore per quell'attributo, mentre le foglie corrispondono ad una classificazione (yes/no oppure 1/0). Gli alberi permettono di rappresentare anche le disgiunzioni (OR logico).

crescita nodi DT  $\rightarrow$  overfitting  $\rightarrow$  Remp basso & VC-Confidence alta (cresce con l'overfitting). Dunque controllare la crescita dei nodi implica un controllo dell'overfitting: si può controllare con il pruning, che permette di ridurre il numero di nodi (si overfitta poi si taglia).



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

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

```

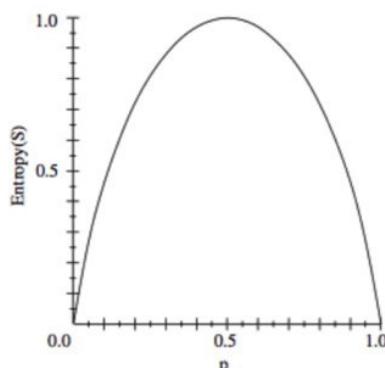
ID3(X, T, Attrs)      X: training examples;
                           T: target attribute
                           Attrs: other attributes, initially all attributes
Create Root node
If all X's are +, return Root with class +
If all X's are -, return Root with class -
If Attrs is empty return Root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

```

**Entropia:** misura l'*impurità* di una collezione di esempi.

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

$$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$



L'entropia è 0 (minima), cioè si hanno collezioni pure, quando o tutti gli esempi sono negativi oppure quando tutti gli esempi sono positivi, mentre è 1 (massima) quando sono 50/50. Dunque l'entropia è:  $0 < Entropy(S) < 1$ .

Formula dell' (Information) **Gain** :

$$Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \left( \frac{|S_v|}{|S|} \cdot Entropy(S_v) \right)$$

[ $S$  insieme iniziale,  $values(A)$  è l'insieme dei possibili valori associabili all'attributo  $A$ ,  $S_v$  è il sottoinsieme degli esempi  $S$  per i quali l'attributo vale  $v$ ]

Il Gain misura la riduzione aspettata dell'entropia causata dal partizionamento degli esempi rispetto all'attributo [ $>$  Gain  $\rightarrow$  efficacia dell'attributo nel classificare i dati di training]. Si cerca di scegliere  $A$  per massimizzare il Gain, l'attributo che discrimina gli esempi che appartengono a classi diverse, dunque per ridurre l'entropia e raggiungere l'omogeneità.

Problema del Gain : favorisce gli attributi con molti valori possibili  $\rightarrow$  introduco Gain Ratio

### 6.6 Gain Ratio

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

dove

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

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

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

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

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

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

Dopo lo splitting il DT rende i casi omogenei: *Split-Information* misura l'entropia di S rispetto ai valori di A, più i dati sono dispersi in modo uniforme più è alto. Classifica anche più casi come positivi? Dipende, in generale no, rende solo i casi più omogenei.

L'algoritmo ID3 seleziona preferibilmente alberi corti rispetto a quelli con molti livelli e quelli che hanno un Information Gain alto vicino alla radice. Il Bias è di linguaggio, lo spazio delle ipotesi è completo.

Problema con alberi di decisione: è l'Overfitting, cioè alberi che si adattano troppo agli esempi di training. Questo porta a comportarsi meglio sul singolo dato ma peggio sull'insieme di dati.

## Validation

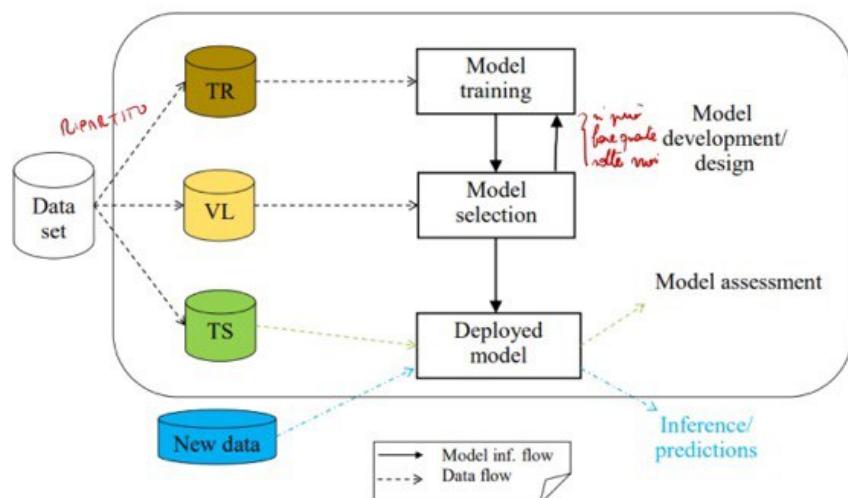
Il problema fondamentale del machine learning è valutare la capacità di generalizzazione.

Training (apprendimento) : si intende il trovare una buona funzione costruendola da uno spazio di funzioni attraverso i dati conosciuti (buona si considera quando ha un basso errore di generalizzazione).

3 fasi:

1. Learning Phase : è il *Model training* si usa il training set per creare varie ipotesi  $h$ , faccio il fitting dei dati, il risultato saranno vari modelli con basso errore di training
2. Predictive Phase : parte della *Model Selection*, si sceglie il miglior modello (o iperparametri) tra quelli generati nella fase precedente
3. Test Phase : questa fase coincide con il *Model Assessment*, ovvero dopo aver scelto un modello finale, si stima il suo errore di previsione (errore di generalizzazione) su nuovi dati di test mai visti prima. Il risultato di questa fase sarà una *stima della capacità predittiva del modello scelto*.

I dati devono essere divisi in 3 parti : TR, VL, TS (ognuno ha obiettivi differenti).



## Meta-Algoritmo

1. Separare TR, VL e TS *migliore (POTES)* *finito apprezzato*
2. Selezionare il miglior  $h_{w,\lambda}()$  cambiando  $\lambda$  *finito apprezzato*
3. Per ciascun  $\lambda$ , allenare il modello cambiando  $w$  che minimizza errore (**fitting TR**). Miglior significa minimo errore su TR (**fitting**).  
*Una volta finito, cambiare  $\lambda$  e riprovare i vari  $w$ .*
4. Selezionare il miglior modello su  $\lambda$  che ha un minor errore su VL
5. Opzionalmente, si può fare fitting su TR+VL (unico training set) col il miglior  $\lambda$  trovato
6. Una volta finito, valutare  $h_{w,\lambda}(x)$  sul TS *(non posso tornare indietro)*

**Iper-Parametro** Un **iper-parametro** è un parametro non appreso, ad esempio  $\lambda$ . Ricercare l'iper-parametro migliore può essere un ciclo su una griglia di possibilità. Può essere automatizzato e parallelizzato (prove indipendenti). Si sceglie il modello migliore su VL, NON si usa TS per model selection.

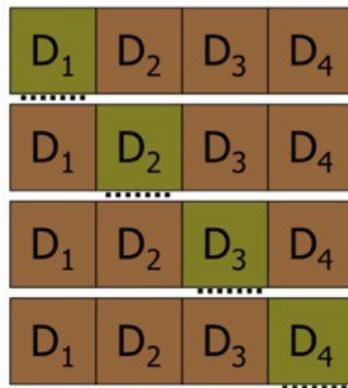
Stimare il fitting non è quanto richiesto e opportuno nella fase di *model selection*. Se si valutasse una media tra errore in training e di validation per la model selection si rischierebbe un modello in overfitting dato che si darebbe troppo peso ai dati.

Il Test Set non dovrà mai essere usato per fare *model selection*, si usa una volta sola poi se lo si riusasse non sarebbe più di test ma servirebbe un nuovo Test Set [e talvolta avere nuovi dati potrebbe essere costoso].

Nella scelta quindi ci si deve basare sull'accuracy sul Validation Set, come se quella del Test Set non esistesse.

Non è detto che se l'errore di training aumenta allora aumenta anche quello in validation e/o test.

#### 7.4 K-fold cross validation



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

K-fold cross validation:

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

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

#### 7.5 Esempio di Model Selection e Assessment

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

#### Grid search

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

## VC + SLT (Statistical Learning Theory)

VC-Dim : è una misura per la complessità dello spazio delle ipotesi H (flessibilità per fissare i dati, come ad esempio il numero di parametri per modelli lineari/grado del polinomio).

- VC-bounds afferma che con una probabilità di  $1 - \delta$  che

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

VC - BOUND

VC - CONFIDENCE

- $R$  : rischio generale
- $\delta$  detto confidenza
- VC-Bound : limite superiore definito da  $R_{emp}$  (rischio empirico: errore di training, è uguale alla Loss) sommato a  $\epsilon$  (VC-Confidence)
- VC-Confidence: dipende in modo inversamente proporzionale da il numero di dati / e da  $\delta$  ( $1/\delta$  indica la probabilità che valga questo bound) e dipende in modo proporzionale da VC (VC-Dim)
- VC-Confidence non dipende da il valore delle  $x$  in input nei modelli lineari perché  $R_{emp} = (y_p - h(x))^2$  dunque non è detto
- Il grado del Kernel polinomiale alza VC-Confidence
- Il grado M di una LBE polinomiale alza la complessità quindi alza VC-Confidence

Relazioni:

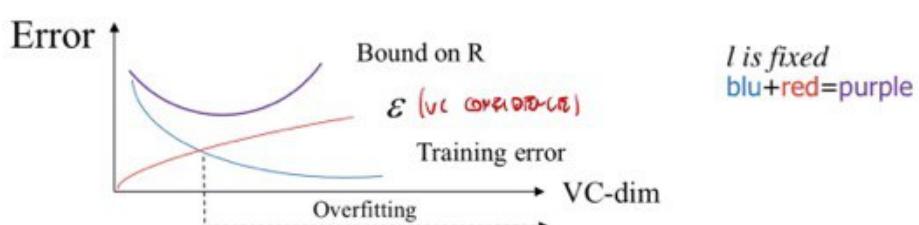
$l$  alto  $\rightarrow R$  basso perché  $R_{emp}$  basso e VC-Confidence bassa

VC-Dim alto  $\rightarrow R_{emp}$  basso ma potrebbe alzare  $R$  (non sempre VC-Dim alto porta ad aumentare il VC-Bound poiché si aumenta il VC-Confidence ma  $R_{emp}$  si abbassa)

VC-Confidence alta  $\rightarrow$  VC-Dim bassa (alta  $\rightarrow$  bassa)

La SLT fonda uno dei principi induttivi sul controllo della complessità.

## 7.9 Structural Risk Minimization

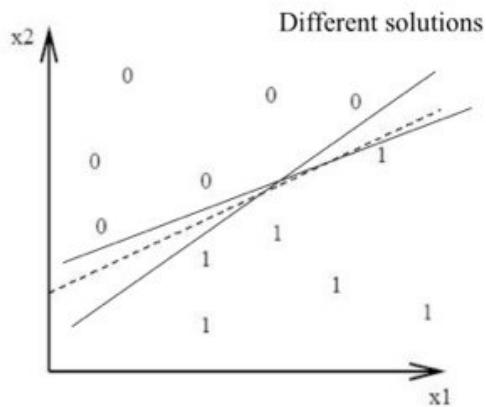


Sfruttiamo il concetto di controllo della complessità del modello, eseguiamo un trade-off tra la complessità del modello e l'accuratezza sul TR set.

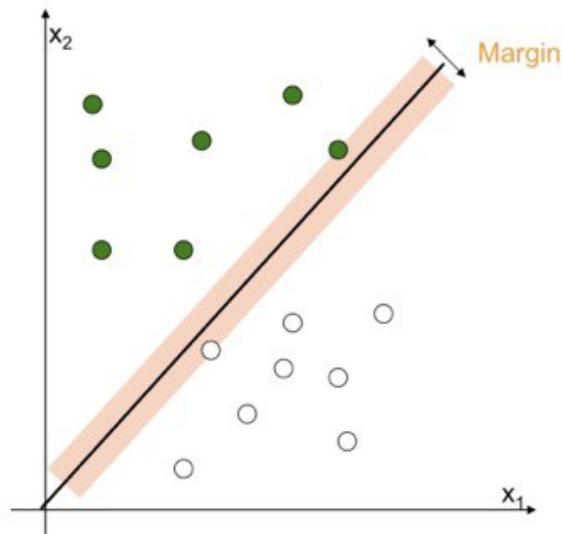
## SVM (Support Vector Machine)

### 8.1 Maximum Margin Classifier (controllo complessità)

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

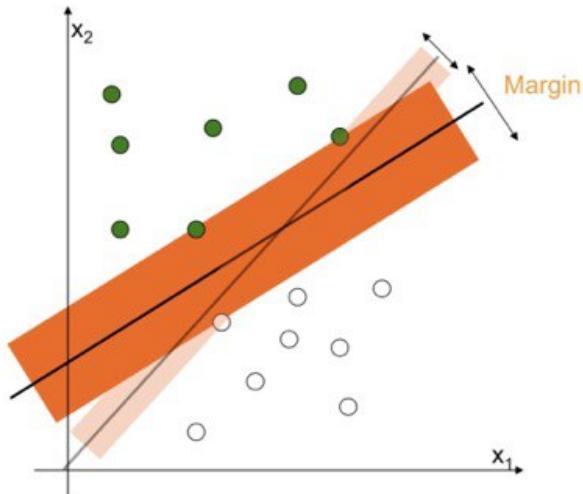


Grazie a SVM abbiamo un criterio per scegliere le soluzioni: minimizzare il rischio strutturale.



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

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



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

## 8.2 Rappresentazione canonica dell'iperpiano e Support Vector

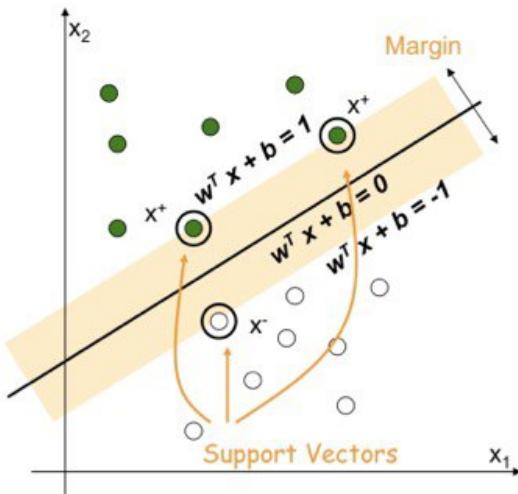


Figure 7: nota:  $w_0 = b$

$$\text{Support Vectors: } x_i : |w^T x_i + b| = 1$$

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

I vettori di supporto sono quei vettori per cui l'iperpiano vale esattamente 1 per i positivi e -1 per i negativi, sono quelli sul bordo del margine.

### Hard Margin

Il problema consiste nel trovare il vettore  $w$  e il valore  $b$  tale che tutti i punti siano classificati correttamente e il margine è massimizzato.

Il margine è  $2/\|w\|$  e sappiamo che  $\|w\|^2 = (w^T w)$  quindi:

- *massimizzare il margine*  $\rightarrow$  *minimizzare*  $\|w\|$   $\rightarrow$  *minimizzare*  $w^T w / 2$
- *margine aumenta*  $\rightarrow$  *VC-Dim diminuisce*  $\rightarrow$  *complessità si abbassa (underfitting)*

[VC-Dim non è più dipendente dalla dimensione dell'input come nel modello lineare  
(aumenta la dim di  $x$  o il numero di parametri  $w \rightarrow$  aumenta VC-Dim)]

A livello pratico il massimo margine si ha quando il margine è a stessa distanza dal punto più vicino dei negativi e più vicino dei positivi.

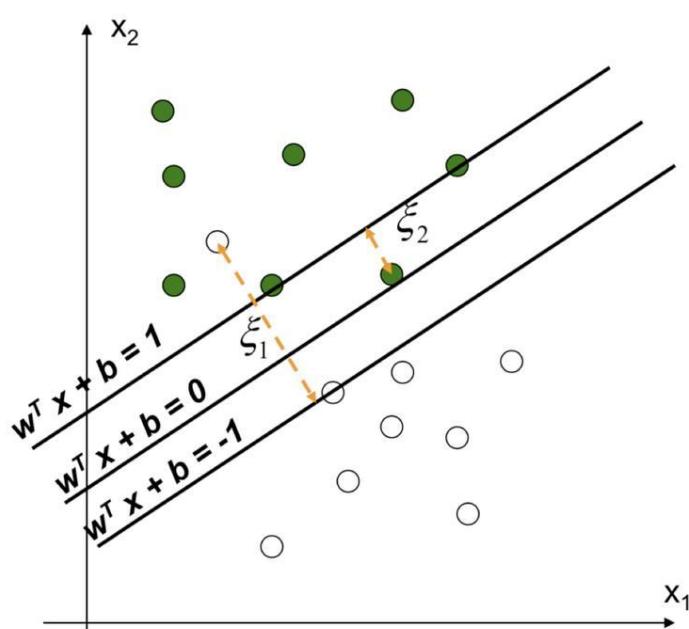
L'iperpiano ottimo è quello che *massimizza il margine* e risolve il problema.

SVM modello primale :

minimizzare  $|w|^2/2$  la funzione obiettivo, è completamente diversa dalla Loss( $w$ ) del modello lineare. La funzione obiettivo non cerca il piano a minimo errore di classificazione in training (questo è svolto dai vincoli che azzerano l'errore di training) ma cerca di minimizzare la complessità.

SVM modello duale:

Esso permette di non avere dipendenza della soluzione *dalla dimensione del feature space*: infatti non dipende dai vettori in generale ma solo da quelli di supporto. L'iperpiano dipende solo dai vettori di supporto.



### Soft Margin

Si introduce perché quasi mai i problemi sono linearmente separabili e anche se lo fossero si rischia che il margine sia piccolissimo. Le  $\xi$  sono dette slack variable, sono la distanza dal vettore di supporto al punto che stiamo analizzando. Esse possono essere indicate indirettamente dall'utente, poiché io scelgo il margine ed esso determina le  $\xi$ . C alto  $\rightarrow \xi$  basse  $\rightarrow$  pochi errori su TR  $\rightarrow$  Remp basso  $\rightarrow$  complessità alta  $\rightarrow$  VC-Confidence alto C viene scelto dall'utente.

Se viene aggiunto un punto nel training set ed è dalla parte giusta dell'iperpiano separatore e oltre il

margine la soluzione non cambia, se invece entra nell'area del margine precedente, la slack variable associata potrà assumere un valore non nullo. Se poi diventa non linearmente separabile la SVM fallisce nel trovare una soluzione che soddisfi i vincoli.

SVM presenta *bias induttivo*: infatti si cerca di massimizzare il margine.

### Differenze tra i due:

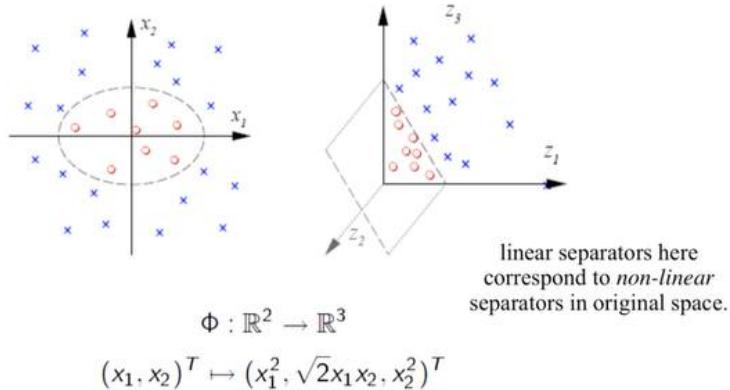
SVM-hard margin  $\rightarrow$  non ammette errori, nonostante ciò l'accuracy in test non è del 100%

SVM-soft margin  $\rightarrow$  ammette errori di classificazione

## 8.7 Mapping per Spazi Dimensionali Ampi

Il compito di SVM è quello di creare un iperpiano che separa linearmente i dati, ma spesso non è possibile dividere linearmente un problema in una certo spazio dimensionale... ed è in questi casi che viene utilizzato il "kernel trick". Cioè spesso i dati non sono linearmente separabili nello spazio dimensionale in input, ma potrebbero essere divisibili in uno spazio dimensionale più grande.

Abbiamo visto cosa succede in casi non lineari: dobbiamo utilizzare efficientemente l'espansione della base via kernel in modo tale da ottenere un approccio flessibile anche per il Supervised Learning non lineare.



Ma questo lo sapevamo già, quando usavamo la funzione  $\phi(x)$  al posto di  $x$ .

$$h_w(x) = \text{sign}\left(\sum_k w_k \phi_k(x)\right)$$

Sappiamo anche che l'utilizzo di spazi ad alta dimensione (funzioni di espansione della base) può essere non conveniente nel calcolo dal punto di vista computazionale e può facilmente portare a un overfitting, nel caso in cui non controlliamo la dimensione dello spazio e la complessità del classificatore (la complessità in questo caso è correlata alla dimensionalità dell'input).

### Kernel - Risolvere problemi linearmente separabili

Grazie alla funzione Kernel posso risolvere problemi non linearmente separabili con SVM: il kernel quindi riduce la dimensione dello spazio input rendendolo linearmente separabile, non riducono la VC-Dim.

La VC-Dim dipende dal margine (significato di regolarizzato) non dalla dimensione dell'input (siccome soft margin è ampio ci possiamo permettere un modello ben regolarizzato), quindi prendiamo LBE ma in un problema regolarizzato. Grazie alla regolarizzazione automatica messa nella funzione obiettivo del SVM riusciamo a tenere a bada la complessità, indipendentemente dalle nuove dimensioni del feature space.

Il kernel permette di non calcolare manco la  $\phi$  e il relativo prodotto scalare, abbiamo degli algoritmi che ci calcolano direttamente il kernel. Se lavoro nel duale la complessità dipende dal numero dei dati non dalla dimensione del feature space.

grado del kernel polinomiale alto → VC-Confidence alta

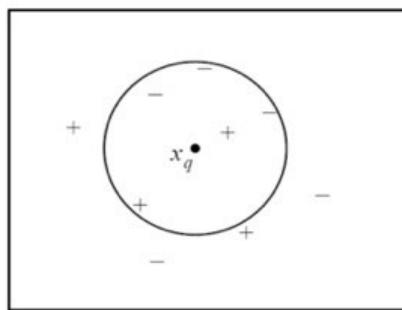
### 8.7.1 Esempi di Kernel

- Lineare:  $K(x_i, x_j) = x_i^T x_j$   
Mapping  $\phi: x \rightarrow \psi(x)$  dove  $\psi(x)$  è  $x$  stesso
- Polinomiale:  $K(x_i, x_j) = (1 + x_i^T x_j)^p$  con  $p$  iper-parametro che indica il grado del polinomio  
Mapping  $\phi: x \rightarrow \psi(x)$  dove  $\psi(x)$  ha dimensione esponenziale rispetto a  $p$
- RBF (radial-basis-function) Gaussiana:  $K(x_i, x_j) = e^{-\|x_i - x_j\|^2 / 2\sigma^2}$  dove  $\sigma$  è un iper-parametro  
Mapping  $\phi: x \rightarrow \psi(x)$  dove  $\psi(x)$  è di dimensione infinita  
(molto potente sul TR ma può portare ad overfitting)

Nel RBF più  $\sigma$  è alta più il cappuccio è alto e stretto, più si classifica in base a se stesso e quindi si ha 0 errore di training, tendendo all'overfitting.

### K-NN

È la versione in cui si "guarda" il comportamento dei K vicini.



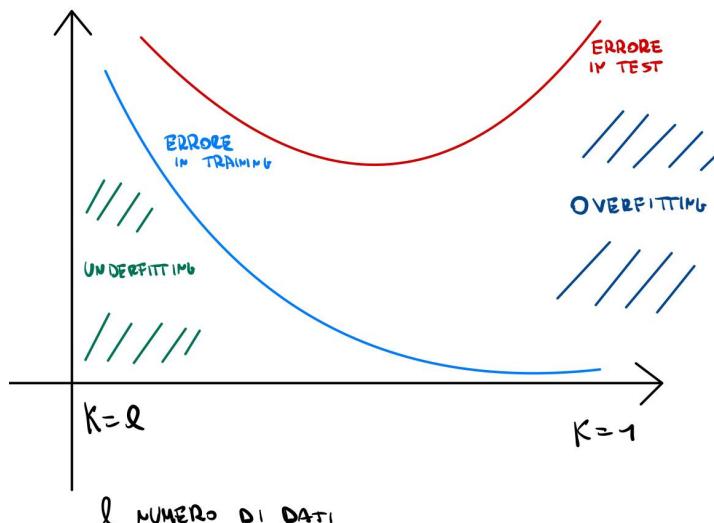
In questo caso 1-NN risponderebbe  $+$  per  $x_q$ , valutando invece con 5-NN viene restituito  $-$  per  $x_q$ . Come succedeva con il polinomio di alto grado che andava in overfitting, anche qui dobbiamo renderlo più "smooth" valutando su un insieme di vicini.

Per questo possiamo "dare un'occhiata" a tot (K) vicini e restituire una media:

$$avg_k(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

dove  $N_k(x)$  è il vicinato di  $x$  che contiene esattamente  $K$  vicini. Se c'è una chiara dominanza di una delle classi nel vicinato di  $x$ , allora è probabile che anche  $x$  appartenga a quella classe. Quindi la regola di classificazione è la maggioranza che vota tra i membri di  $N_k(x)$ .

$$h(x) = 1 \text{ se } avg_k(x) > 0.5 , 0 \text{ altrimenti}$$



K-NN fa parte del Supervised Learning.  
Se la dimensione dell'input diminuisce la capacità di generalizzazione diminuisce, infatti per  $K = 1$  si ha max flessibilità (capacità di generalizzare) ed è prone all'overfitting. Migliora le prestazioni di accuratezza con più dati nel training set.

*Bias induttivo: scegliere il risultato in base alla distanza.*

Quando abbiamo molte variabili in input K-NN spesso fallisce a causa del "curse of dimensionality": quando la dimensione aumenta, il volume dello spazio aumenta in modo tale che i dati disponibili diventano sparsi. Ad esempio la quantità di dati che servono per supportare un risultato spesso cresce esponenzialmente con la dimensione.

Troviamo anche il problema denominato "curse of noisy": se il target dipende da poche altre variabili, potremmo trovarci un "modello simile" con la somiglianza dominata dal gran numero di funzioni irrilevanti.

## 10.1 K-means

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

1. Scegliere k centri di cluster in modo che coincidano con k modelli scelti casualmente (o k punti definiti casualmente all'interno dell'ipervolume contenente il set di pattern)  $c_1 \dots c_k$  con k fissato che sceglio noi.
2. Assegnare ad ogni modello il centro del cluster più vicino (il vincitore) per ogni x calcoliamo

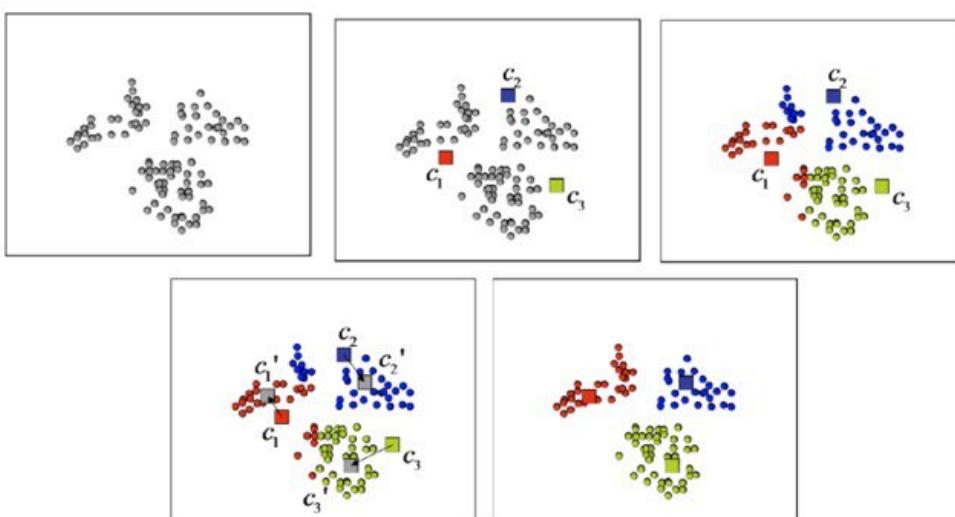
$$i^*(x) = \operatorname{argmin}_i \|x - c_i\|_2^2 = \sum_{j=1}^n (x_j - c_{ij})^2$$

cioè provo tutti i  $c_i$  e quello che ha distanza minima è il vincitore (mi interessa l'indice del vincitore  $i^*$ ). Adesso x appartiene al cluster  $i^*$

3. Ricalcolare i centri del cluster (centroide) utilizzando le nuove appartenenze al cluster corrente

$$c_i = \frac{1}{|\text{num di membri cluster}_i|} \sum_{x_j \in \text{cluster}_i} x_j$$

4. Se non viene soddisfatto un criterio di convergenza, andare al passaggio 2 (criteri come nessuna o minima riassegnazione di schemi a nuovi centri di cluster oppure una minima diminuzione dell'errore quadratico)



### **Algoritmi di training che hanno search bias (strategia di ricerca incompleta)**

- Decision Tree ID3 (ma ha spazio di ipotesi completo)
- Discesa a gradiente per linear model (ha una ricerca locale dunque incompleta e non ha pure bias di linguaggio, dato dalla linearità)
- SVM hard margin per linear model (ha anche linguaggio sempre a causa del modello lineare)

Altri:

- Lookup table (non ha proprio bias)
- Candidate Elimination (ha language bias ma ricerca completa)
- Find S (non lo ha completo, essendo più forte di CE)
- modello lineare senza LBE, non può risolvere problemi non lineari [ha bias di linguaggio], con LBE lo spazio delle ipotesi si avvicina alla completezza [questo bias tende ad essere attenuato] ma non si può togliere del tutto perché rimane la linearità su  $w$

### **Relazioni di complessità**

- modello lineare con più variabili input più complesso rispetto a chi ne ha meno
- DT è più espressivo di un modello con FIND-S poiché può rappresentare anche ipotesi disgiuntive ("V" OR) e non solo congiuntive
- K-NN con K minori hanno più flessibilità/complessità (K= 1 max flessibilità)
- K-NN con K bassi è più flessibile che un modello lineare dato che segue i punti uno ad uno, utilizzando decision boundary non lineari mentre il modello lineare è più rigido
- K-NN è più flessibile che un FIND-S dato che quest'ultimo ha un bias di linguaggio più forte, più rigido
- DT è più flessibile del modello lineare infatti per esempio lo xor

### **Ottimi classificatori per discriminare ad esempio pixel accesi e spenti**

- SVM con kernel polinomiali
- Linear model
- SVM hard margin
- K-NN per opportuno valore di K