



# Riassuntazzo I.A.

Giovanni Sissa

A.A 2018/2019

# Indice

<b>1</b>	<b>Agenti e strategie di ricerca</b>	<b>5</b>
1.1	Algoritmi Costruttivi . . . . .	5
1.1.1	Strategie Blind . . . . .	5
	Breadth first o ricerca in ampiezza . . . . .	5
	Depth first o ricerca in profondità . . . . .	6
1.1.2	Strategie Informate . . . . .	6
	Best First . . . . .	6
	Algoritmo A* . . . . .	7
1.2	Ricerca Locale . . . . .	8
	Struttura dei vicini o neighborhood: . . . . .	8
	Iterative improvement: . . . . .	8
	Criticità di ricerca locale: . . . . .	8
1.3	Meta-euristici . . . . .	9
1.3.1	Ant colony optimization . . . . .	9
1.3.2	Algoritmi genetici . . . . .	9
	Sopravvive il più adatto: . . . . .	9
	Tecniche più usate e varianti . . . . .	10
1.4	Constraint Programming . . . . .	10
1.4.1	CP attraverso un esempio: N queens . . . . .	10
1.4.2	CP o CSP: definizione . . . . .	11
	Albero decisionale: . . . . .	11
1.4.3	Propagazione . . . . .	11
1.4.4	Algoritmi di ricerca di soluzioni in CP: Vincoli a po- steriori . . . . .	12
	Generate and test . . . . .	12
	Standard Backtracking . . . . .	12
1.4.5	Algoritmi di ricerca di soluzioni in CP: Vincoli a Priori . . . . .	12
	Forward checking . . . . .	12
	Look Ahead . . . . .	13
1.4.6	Euristiche in CP . . . . .	13
	Euristiche per la selezione della variabile: . . . . .	13
	Euristiche per la selezione dei valori: . . . . .	14
	Euristiche Statiche: . . . . .	14
	Euristiche dinamiche: . . . . .	14
1.4.7	Algoritmi di ricerca di soluzioni in CP: Tecniche di consistenza . . . . .	14
	Node consistency o consistenza di grado 1: . . . . .	14
	Arc consistency o consistenza di grado 2: . . . . .	14
	Procedimento iterativo: . . . . .	15

	Path consistency o consistenza di grado 3: . . . . .	15
	Teorema Path consistency . . . . .	15
1.4.8	COP . . . . .	15
<b>2</b>	<b>Giochi</b>	<b>16</b>
2.1	Algoritmo min-max . . . . .	16
2.1.1	Labeling . . . . .	16
2.1.2	Labeling di un nodo $< n >$ : algoritmo breadth-first . . . . .	16
2.1.3	Labeling di un nodo $< n >$ : algoritmo depth-first . . . . .	17
2.1.4	Espansione limitata . . . . .	17
	Quanto espando? . . . . .	18
2.1.5	Importante considerazione . . . . .	18
2.2	Tagli alfa-beta . . . . .	18
2.2.1	Algoritmo alpha-beta . . . . .	18
	NOTA: . . . . .	19
2.2.2	Conclusioni . . . . .	19
<b>3</b>	<b>Rappresentazione della conoscenza</b>	<b>20</b>
3.1	Dati e Pattern . . . . .	20
3.2	Tipo di Pattern . . . . .	20
3.2.1	Numerici . . . . .	20
3.2.2	Categorici . . . . .	20
3.2.3	Sequenze . . . . .	20
3.2.4	Altri dati strutturati . . . . .	21
3.3	Rappresentazione di Pattern . . . . .	21
3.3.1	Proteine . . . . .	21
3.3.2	DNA . . . . .	22
3.3.3	Immagini . . . . .	23
	Local binary pattern LBP . . . . .	23
	Extended local binary pattern . . . . .	24
	Filtri e filtri di gabor . . . . .	25
	Focus su convoluzione . . . . .	25
	Informazioni spaziali . . . . .	26
3.3.4	The dissimilarity space . . . . .	26
3.4	Classificazione . . . . .	27
3.4.1	Regressione . . . . .	28
3.4.2	Clustering . . . . .	28
3.5	Riduzione di dimensionalità . . . . .	29

<b>4</b>	<b>Learning</b>	<b>30</b>
4.1	Feature learning (ad esempio deep learning)	30
4.2	Apprendimento	30
4.2.1	Tipi di apprendimento	30
	Supervisionato o Supervised	30
	Non supervisionato o Unsupervised	30
	Semi-supervisionato o Semi-supervised	31
	Altri tipi di apprendimento	31
4.2.2	Esempi di problemi di apprendimento	32
4.2.3	Tecniche di addestramento della rete	33
	Batch	33
	Incrementale	33
	Naturale	33
4.3	Reinforcement Learning	33
4.4	Ottimizzazione di parametri	34
4.4.1	Parametri vs Iperparametri	34
4.5	Valutazione delle prestazioni	35
4.6	Closed set e Open set	36
4.7	Classificatore a soglia	36
4.7.1	Errore di classificazione	36
4.7.2	La scelta della soglia	37
4.7.3	Precision recall	38
4.8	Training, Validation, Test	38
4.9	Convergenza	39
4.10	Overfitting	40
<b>5</b>	<b>Metodi di classificazione</b>	<b>41</b>
5.1	Approccio bayesiano	41
5.1.1	Bayes: parametrico e non-parametrico	42
5.1.2	Bayes parametrico	42
	Distribuzione Normale	42
	Multinormale	43
	Multivariata	43
	Distanza di Mahalanobis	43
	Bayes con Multinormali	44
	Bayes Parametrico riassunto	44
	In pratica	45
5.1.3	Approcci Bayes non parametrici	45
	Parzen Window	46
5.2	Nearest Neighbor (NN)	47
5.2.1	K-NN	47

6	Clustering	49
7	Riduzione di dimensionalità	50
8	Reti neurali e deep learning	51

# 1 Agenti e strategie di ricerca

## 1.1 Algoritmi Costruttivi

Per problemi di una certa complessità una ricerca **esaustiva** non è praticabile. La scelta di quale stato espandere nell'albero di ricerca prende il nome di **strategia**. Esistono strategie informate, dette **euristiche**, o non informate, dette **blid**. Per capire la bontà delle strategie si valutano 4 criteri:

- **Completezza** : la strategia garantisce di trovare una soluzione se esiste?
- **Complessità temporale** : quanto tempo occorre per trovare una soluzione?
- **Complessità spaziale** : quanta memoria occorre per trovare la soluzione?
- **Ottimalità** : la strategia trova la soluzione ottima? quanto è buona la soluzione trovata?

Poiché questi algoritmi di ricerca si occupano di costruire una soluzione aggiungendo componenti a una situazione di partenza in un particolare ordine si dicono **Algoritmi costruttivi**

### 1.1.1 Strategie Blind

#### Breadth first o ricerca in ampiezza

La **profondità** del nodo radice è 0. La profondità di ogni altro nodo è la profondità del padre +1. **Espande sempre i nodi meno profondi dell'albero**. Nel caso peggiore con profondità dell'albero  $d$  e fattore di ramificazione  $b$  il numero massimo di nodi da esplorare è  $b^d$ .

- Svantaggi: grande spreco di memoria e complessità temporale elevata (entrambi  $b^d$  per costo temporale o di memoria) .
- Vantaggio: ritorna sempre il cammino a costo minimo se il costo coincide con la profondità.

## Depth first o ricerca in profondità

**Espande per primi sempre i nodi più profondi.** A parità di profondità sceglie un nodo arbitrario

- Svantaggio: complessità temporale analoga a Breadth first; numero massimo di nodi espansi nel caso peggiore  $b^d$ .
- Vantaggio: modesta occupazione di memoria  $b * d$ .

### 1.1.2 Strategie Informate

Conoscendo il problema o il gioco decido quali rami espandere e quali no. L'intelligenza di un sistema non si misura in termini di capacità di ricerca ma in capacità di utilizzare la conoscenza sul problema per eliminare il pericolo di espansione combinatoria.

L'intelligenza di un sistema sta dunque nella saggia decisione di cosa fare ad ogni step.

Le **Funzioni di valutazione** danno una **stima computazionale dello sforzo** per raggiungere lo stato. Attenzione: il tempo speso a valutare la funzione di valutazione (o Euristiche) per la scelta di quale nodo espandere deve corrispondere ad una riduzione nella dimensione dello spazio esplorato. Trade-off tra tempo per risolvere il problema (livello base) e tempo per scegliere come risolvere il problema (meta-livello).

## Best First

Usa una **funzione di valutazione** che calcola un numero che rappresenta quanto un nodo sia desiderabile relativamente da espandere. In altre parole per ogni figlio mi calcolo un costo e espando solo quello di costo minore. Si segue il miglior candidato e basta.

**Best-first vuol dire scegliere come nodo da espandere quello che sembra più desiderabile.**

**QueuingFn** = inserire i successori in ordine di desiderabilità ovvero in ordine di costo crescente.

**Criticità:** non è detto trovi la soluzione migliore ovvero il cammino migliore per una soluzione. Non tiene cioè conto delle profondità: un nodo goal a profondità 3 ma con costo 4 in un genitore non sarà mai valutato fintanto che ci sono figli con costo minore di 4.

Questo tipo di approccio è anche detto greedy. Ricerca A\* caso particolare di best first.

### Algoritmo A\*

Invece di tenere solo in considerazione il costo relativo di espandere un nodo, considera anche come costo il raggiungimento del nodo dalla radice. Espandiamo quindi i nodi in ordine crescente di:

$$f(n) = g(n) + h'(n) \quad (1)$$

Dove  $g(n)$  è la profondità del nodo e  $h(n)$  è la distanza dal goal ovvero la mia euristica. Combina i vantaggi della ricerca in salita (efficienza) e della ricerca a costo uniforme (ottimalità e completezza).

### Pseudocodice A\*:

- 0) Sia L la lista dei nodi iniziali del problema.
- 1) Sia n di L per cui è minima  $f(n) = g(n) + h'(n)$ . Se L è vuoto fallisci.
- 2) Se n è il goal fermati e return n e la strada fino a lui.
- 3) Altrimenti rimuovi n da L e aggiungi a L tutti i figli di n con label la strada percorsa partendo dal nodo iniziale e ritorna a 2).

Esempio nelle slide da pagina 61. Nota sulle euristiche: si possono definire più euristiche ammissibili.

**Estensione A\*: da alberi a grafi.** Assumere che i problemi siano sempre alberi è semplicistico: in un gioco come quello del filetto si può infatti tornare ad uno stato precedentemente esplorato o scartato. I figli possono contenere cioè nei propri figli un padre o un nonno (loop). Soluzione è mantenere due liste una di nodi chiusi da non ri esaminare e una di nodi aperti ancora da esaminare.

### Pseudocodice A\* in grafi:

- 0) Sia  $L_a$  la lista dei nodi iniziali del problema.
- 1) Sia n di  $L_a$  per cui è minima  $f(n) = g(n) + h'(n)$ . Se  $L_a$  è vuoto fallisci.
- 2) Se n è il goal fermati e return n e la strada fino a lui.



- 3) Altrimenti rimuovi  $n$  da  $L_a$ . Inserisci  $n$  in  $L_c$  (lista dei nodi chiusi) e aggiungi a  $L_a$  tutti i figli di  $n$  con label la strada percorsa partendo dal nodo iniziale. Se un nodo figlio è già in  $L_a$ , non raggiungerlo ma aggiornalo con la strada migliore che lo connette al nodo iniziale. Se un nodo figlio è già in  $L_c$  non aggiungerlo a  $L_a$  ma se il suo costo è migliore aggiorna il costo e il costo dei nodi già espansi che da lui dipendevano.
- 4) Ritorna a 2).

## 1.2 Ricerca Locale

Gli **algoritmi di ricerca locale** partono da una soluzione iniziale e iterativamente cercano di rimpiazzare la soluzione corrente con una migliore in un intorno della soluzione corrente. In questi casi non interessa la strada per raggiungere l'obiettivo.

Si basa sulla esplorazione dei vicini, essi possono migliorare la soluzione corrente mediante modifiche locali.

**Struttura dei vicini o neighborhood:** funzione  $F$  che assegna a ogni soluzione  $s$  dell'insieme delle soluzioni  $S$  un insieme di soluzione  $N(s)$  sottoinsieme di  $S$ . Si capisce che la scelta di  $F$  è fondamentale per la bontà del algoritmo.

La soluzione trovata da una ricerca locale non è detto sia ottima globalmente ma può essere un miglioramento locale. Più largo è il vicinato più è probabile che un massimo o minimo locale sia anche globale e quindi ad un aumento del vicinato aumenta la qualità della soluzione a discapito del costo computazionale.

**Iterative improvement:** l'algoritmo base a cui ci si riferisce di solito. Una soluzione viene generata in modo casuale o mediante un algoritmo costruttivo. Su questa soluzione viene lanciato ricerca locale che prova a migliorarla. Se tra i vicini vi è una soluzione migliore, la si seleziona e si rilancia ricerca locale su di questa, altrimenti è stato trovato un max o min locale. Un esempio di questo algoritmo è di immaginarsi un hill climbing (vedi slide 69 e seguire).

**Criticità di ricerca locale:**

- **Massimi locali:** stati migliori di tutti i vicini ma peggiori di altri stati fuori dal vicinato: l'algoritmo non ci fa uscire da qui.

- **Pianori o altopiani:** zone piatte nello stato di ricerca, tutti i vicini con lo stesso valore. L'algoritmo non sa in che direzione muoversi.
- **Crinali:** zona adiacente migliore ma verso la quale non è possibile muoversi.

Una possibile e semplice soluzione a queste criticità è lanciare più volte l'algoritmo di ricerca locale partendo da soluzioni iniziali diverse. Si salva la soluzione migliore dopo una serie di tentativi (bound dovuto al tempo di CPU o numero di iterazioni).

### 1.3 Meta-euristici

In questa classificazione ricadono l'insieme di algoritmi, tecniche e studi relativi all'applicazione di criteri euristici per risolvere problemi di ottimizzazione.

#### 1.3.1 Ant colony optimization

Solo citati in questo corso, sono ispirati al comportamento di colonie di insetti. Tali insetti mostrano infatti una capacità globale nel trovare il cammino migliore dal cibo alla colonia. Questo ha dato nascita ad algoritmi di ricerca cooperative.

#### 1.3.2 Algoritmi genetici

Sono algoritmi evolutivi ispirati ai modelli delle specie in natura. Principio di selezione naturale: si favoriscono gli individui più adatti ad uno specifico ambiente per sopravvivere e riprodursi. Ogni individuo è una soluzione o cromosoma, con relativo valore della funzione obiettivo detta **fitness**. I tre principali operatori sono: **Selezione; Mutazione; Ricombinazione**.

La criticità sta nel ben rappresentare un gioco o uno stato del problema in un cromosoma e nel bene rappresentare la funzione Fitness.

**Sopravvive il più adatto:** un pool di individui iniziali, solitamente generati randomicamente, vengono fatti riprodurre per formare nuovi individui che ereditano il patrimonio genetico dei genitori. Si introducono mutazioni casuali. Ogni individuo è rappresentato mediante il suo genotipo, il **cromosoma** è una sequenza ordinata di  $L$  **alleli**.

Definizione di una funzione di **fitness**: individui con alta fitness avranno più probabilità di essere selezionati per la riproduzione. Lo scopo dell'agente è selezionare l'individuo con fitness migliore. Nota la fitness potrebbe non

coincidere con la funzione obiettivo.

```
int_population()      // genera una popolazione iniziale random.
for i:=1 to max_gen   // numero massimo di cicli.
  evaluation()        // calcola la fitness per ogni individuo.
  section()           // seleziona N individui per riprodurre.
  crossover()         // ricombinazione genetica tra N/2 coppie.
  mutation()          // aggiunge un fattore randomico in figli.
  replacement()       // scarta gli individui a fitness bassa.
end
return best element
```

### Tecniche più usate e varianti

- Nella **selezione** la tecnica più usata è la funzione di probabilità uniforme sulla fitness:

$$\pi_i = \frac{f(i)}{\sum_{j=1} f(j)}$$

Possono essere adottati tanti altri possibili schemi di soluzione.

- Nel **crossover** lo schema più usato è quello a un punto: si sceglie un punto  $l$  random nel cromosoma tra 1 e  $L$ ; un figlio eredita gli alleli da 1 a  $l-1$  da un genitore e gli altri dall'altro; l'altro figlio il contrario. Varianti sono a due punti o random dai genitori. Bisogna prestare attenzione nel caso di presenza di vincoli.
- Nella **mutazione** si può utilizzare un operatore non uniforme come ad esempio:

$$x'_k = x_k + \Delta(t, x_k)$$

dove  $\Delta$  è un numero casuale tale  $x'_k$  che cada nel intervallo di definizione.

## 1.4 Constraint Programming

Molti problemi di IA possono essere visti come CP. **Obiettivo:** trovare uno stato del problema che soddisfi un dato insieme di vincoli.

### 1.4.1 CP attraverso un esempio: N queens

- Le 8 regine vengono rappresentate con le variabili  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ . Il pedice si riferisce alla colonna occupata, a dominio  $[1, \dots, n]$ .

- l'istanziamento di  $\mathbf{x}_h$  al valore  $\mathbf{k}$  indica che la regina sulla colonna  $h$  sta sulla riga  $k$ .
- Vincoli:

$$\begin{aligned}1 \leq x_i \leq n & \quad \text{per } 1 \leq i \leq n \\x_i \neq x_j & \quad \text{per } 1 \leq i \leq j \leq n \\x_i \neq x_j + (j - 1) & \quad \text{per } 1 \leq i \leq j \leq n \\x_i \neq x_j - (j - 1) & \quad \text{per } 1 \leq i \leq j \leq n\end{aligned}$$

Il primo vincolo si dice unario gli altri binari.

#### 1.4.2 CP o CSP: definizione

Formalmente un CSP (Constraint satisfaction problem) può essere definito su un insieme finito di variabili  $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ , i cui valori appartengono ai domini finiti di definizione  $(\mathbf{D}_1, \mathbf{D}_2, \dots, \mathbf{D}_n)$ , e su un insieme di vincoli. Un vincolo  $c(x_{i1}, x_{i2}, \dots, x_{in})$  tra  $k$  variabili è un sottoinsieme del prodotto cartesiano  $(D_{i1} * D_{i2} * \dots * D_{in})$  che specifica quali valori delle variabili sono compatibili con le altre. Tale sottoinsieme non deve essere definito esplicitamente ma è rappresentato in termini di relazione. Una soluzione prevede l'assegnamento di tutte le variabili. (soluzione ammissibile dal problema se rispetta i vincoli).

**Albero decisionale:** in csp si può far corrispondere ogni problema ad un albero. Ordinando le variabili ogni livello  $i$  corrisponde all'assegnazione del  $i$ -esima variabile. Ogni foglia rappresenta un assegnamento di tutte le variabili e quindi una soluzione ammissibile se tutti i vincoli sono rispettati. La ricerca equivale all'esplorazione di un albero:  $n$  variabili tutte a cardinalità  $d$  il numero di foglie da esplorare è  $d^n$ .

#### 1.4.3 Propagazione

Algoritmi intenti a prevenire fallimenti nella ricerca nell'albero. **Pruning a priori:** utilizzare relazioni tra le variabili (ovvero i vincoli) per ridurre lo spazio di ricerca. Vengono rimossi i rami sicuramente fallimentari limitando il backtracking. **Algoritmi che fanno uso di propagazione sono Forward-checking e (partial and full)Look-ahead, NON fanno uso di propagazione Generate and test (GT) e Standard Backtracking (SB)**

#### 1.4.4 Algoritmi di ricerca di soluzioni in CP: Vincoli a posteriori

##### Generate and test

L'interprete del linguaggio sviluppa a vista un albero decisionale percorrendolo in profondità assegnando valori alle variabili senza preoccuparsi di verificare la consistenza con gli altri vincoli. Solo in un secondo momento l'algoritmo verifica che la soluzione generata soddisfi i vincoli del problema. Esplora tutte le permutazioni delle soluzioni e se la soluzione del problema non soddisfa i vincoli scatta il backtracking. **Inefficienza:** i vincoli sono usati a posteriori, a ricerca già avvenuta.

##### Standard Backtracking

Migliora GT ma vincoli ancora usati a posteriori. Ad ogni istanziazione di una variabile si preoccupa di verificare la coerenza delle variabile appena istanziata con quelle già assegnate. Non si prosegue nella ricerca in un ramo che ai primi livelli già fallisce. **Pruning a posteriori:** lo spazio di ricerca è ridotto ma dopo il tentativo. Problema: in alcune situazioni potrebbe avvenire che l'istanziazione dell'ultima variabile crea problemi. Backtracking molto costoso.

#### 1.4.5 Algoritmi di ricerca di soluzioni in CP: Vincoli a Priori

Come Standard Backtracking ma in più controllano i vincoli tra variabili già legate a variabili ancora da istanziare e anche in gradi diversi (coppie o tuple). La propagazione ha l'effetto di ridurre i domini delle variabili non ancora istanziate.

##### Forward checking

Utilizzato dopo ogni istanziamiento: i vincoli vengono propagati e cioè si eliminano i valori incompatibili con quello appena istanziato dai domini delle variabili ancora libere. Vincente quando le ultime variabili risultano ora poco libere (pochi valori rimasti nei domini): queste risultano molto vincolate e quindi facili da istanziare.

- Se il dominio dell'ultima variabile consiste in un solo valore, assegnamento senza costo computazionale.
- Se un dominio durante la computazione diviene vuoto, il tentativo fallisce senza proseguire in tentativi e Backtracking.

L'idea è che l'assegnazione di un valore ad una variabile ha delle ripercussioni sull'insieme di valori disponibili per le altre. In questo modo i vincoli agiscono forward limitando lo spazio delle soluzioni prima di provare tentativi su di esso.

### Look Ahead

Tecnica più completa per il pruning a priori. Ad ogni istanziazione viene controllata la compatibilità con i vincoli del problema rispetto alle variabili già istanziate in precedenza (come generate and test) e le successive (variabili ancora libere). In più viene sviluppato il **Look Ahead** che controlla l'esistenza, nei domini associati alle variabili ancora libere, di valori compatibili con i vincoli contenenti solo variabili non istanziate. I domini associati a ogni variabile vengono ridotti propagando anche le relazioni contenenti coppie di variabili non istanziate. Viene verificata la possibilità di una futura assegnazione consistente tra le variabili libere (a coppie).

**Partial Look Ahead (PLA):** all'istanziazione di  $x_i$  si ha una propagazione dei vincoli contenenti la variabile appena istanziata nelle variabili non ancora istanziate (da  $x_{i+1}$  in poi). Per ogni variabile  $x_{i+1} \dots x_n$  deve esistere un valore  $k$  per il quale sia possibile trovare almeno un valore compatibile con  $k$  per tutte le variabili rimanenti e "successive". Es slide 117.

**Full Look Ahead (FLA)** il controllo non è fatto solo sulle variabili successive ma anche sulle precedenti non assegnate. Propago in tutte le variabili ancora libere non solo nelle "successive".

#### 1.4.6 Euristiche in CP

Questi algoritmi soffrono di due gradi di libertà: la scelta dell'ordinamento delle variabili e la scelta dell'ordine delle selezioni dei valori all'interno dei domini. Le euristiche agiscono su questi gradi di libertà per garantire soluzioni in tempi più rapidi anche per problemi complessi.

**Euristiche per la selezione della variabile:** determinano quale debba essere la prossima variabile da istanziare. Seguono le due euristiche più usate, che selezionano come la prossima variabile da istanziare la più "difficile".

- **First fail** o Minimum Remaining Values: sceglie la variabile con dominio a cardinalità minore.
- **Most-constrained principle:** sceglie la variabile legata a più vincoli.

**Euristiche per la selezione dei valori:** Determinano quale valore assegnare alla prossima variabile selezionata. Si segue in genere il principio **least constraining principle** che cerca di scegliere prima il valore che si ritiene abbia più probabilità di successo.

**Euristiche Statiche:** determinano l'ordine in cui le variabili (o i valori) vengono scelti prima di iniziare la ricerca e tale ordine rimane invariato. Esempio: nel dominio scelgo valore minimo, assegno come prossima variabile la  $i+1$ .

**Euristiche dinamiche:** scelgo la prossima selezione da effettuare ogni volta che una selezione viene richiesta.

Le euristiche dinamiche sono potenzialmente migliori. La determinazione dell'euristica perfetta è un problema in genere con la stessa complessità del problema di partenza: compromesso.

#### 1.4.7 Algoritmi di ricerca di soluzioni in CP: Tecniche di consistenza

Ridurre il problema originale eliminando dai domini delle variabili i valori che non possono comparire in una soluzione finale. Possono essere applicate staticamente o ad ogni assegnamento come potenti tecniche di propagazione. Tutte le tecniche sono basate sulla rappresentazione di un problema come una rete (grafo) di vincoli. Gli archi possono essere orientati (esempio vincolo  $>$ ) o meno (vincolo  $=$ ).

Per ogni CSP esiste un **constraint graph** i cui nodi rappresentano variabili e gli archi vincoli. Vincoli binari sono archi da una variabile ad un'altra, vincoli unari sono archi entranti e uscenti dallo stesso nodo.

##### Node consistency o consistenza di grado 1:

- Un nodo è consistente se **per ogni** valore  $x_i \in D_i$  il vincolo unario su  $x_i$  è soddisfatto.

##### Arc consistency o consistenza di grado 2:

- Dato un grafo node consistent, un arco  $A(i, j)$  è consistente se **per ogni** valore  $x \in D_i$  esiste almeno un valore  $Y \in D_j$  tale che il vincolo tra  $i$  e  $j$   $P(i, j)$  sia soddisfatto.

**Procedimento iterativo:** La rimozione di alcuni valori dal dominio di una variabile rende necessarie ulteriori verifiche che coinvolgono i vincoli contenenti la variabile stessa. Quindi questo procedimento deve essere ripetuto finché la rete non raggiunge uno stato di QUIESCENZA o stabile. **Vs FLA, AC ha un maggior costo computazionale ma maggior pruning. FLA è detto AC/2**

**Path consistency o consistenza di grado 3:**

- Dato un grafo arc consistent, un cammino tra i nodi  $(i, j, k)$  è path consistente se, **per ogni** valore  $x \in D_i$  e  $Y \in D_j$  (arche node consistenti) esiste un valore  $z \in D_z$  che soddisfa i vincoli  $P(i, k)$  e  $P(k, j)$ .

**Teorema Path consistency**

- **Dato un constraint-graph completo** (ogni nodo connesso ad ogni altro nodo) **se ogni cammino di lunghezza 2 è path-consistente allora l'intera rete è path consistente.**

Un constraint-graph non completo può essere reso completo aggiungendo archi completamente rilassati ( $<$  or  $=$  or  $>$ ) tra due nodi non collegati.

In generale se un grafo a  $n$  variabili è  $k$ -consistente con  $k < n$  allora per trovare la soluzione è necessaria una ricerca nei domini ripuliti. Se ha  $n$  nodi e è  $n$ -consistente allora si può dare una soluzione senza ricerca.

Esiste un algoritmo generale per rendere una rete a  $k$  variabili  $k$ -consistente ma ha una complessità esponenziale in  $k$  e costa cioè come una ricerca nella rete originale.

#### 1.4.8 COP

Abbiamo finora parlato di CSP ovvero di problemi di soddisfacibilità. Si può però estendere il ragionamento a problemi di ottimizzazione con vincoli COP. Un COP è un CSP in cui si aggiunge un obiettivo. **COP è quindi descrivibile formalmente come un CSP in cui l'obiettivo non è trovare una soluzione ma trovare la soluzione ottima.**

Dato un algoritmo in grado di risolvere CSP si può allora utilizzarlo per risolvere COP aggiungendo una variabile che descrive l'obiettivo. Ad ogni soluzione trovata in CSP viene aggiunto un nuovo vincolo che garantisce che ogni nuova soluzione abbia valore obiettivo migliore. Questo procedimento continua finché non sarà più possibile trovare una soluzione migliore. L'ultima trovata sarà la soluzione ottima.



## 2 Giochi

I giochi hanno le seguenti proprietà :

- Due giocatori, **le mosse sono alternate** e le funzioni di utilità complementari: vince e perde.
- Sono giochi con **conoscenza perfetta** in cui i due giocatori hanno la stessa informazione.

Non sono quindi giochi i giochi di carte per esempio, quelli in cui la conoscenza è diversa per ogni giocatore. Si possono affrontare con altre tecniche.

Lo svolgersi del gioco si può interpretare come un albero in cui la radice è la posizione di partenza e le foglie le posizioni finali.

### 2.1 Algoritmo min-max

L'algoritmo minmax è progettato per determinare la strategia ottimale per "max" e suggerirgli di conseguenza la prima mossa migliore da compiere. Per fare questo ipotizza che "min" faccia sempre la scelta per lui più favorevole. Non interessa la strada ma solo la prossima mossa.

#### 2.1.1 Labeling

L'albero delle mosse viene etichettato in questo modo:

**Un nodo con max che deve muovere viene etichettato con il massimo dei labels dei figli. Viceversa per min.**

Deve essere nota una funzione che dato un nodo, stato del gioco, mi dica la sua bontà per min o per max: una funzione obbiettivo. La bontà di tale funzione migliora le prestazioni dell'algoritmo.

#### 2.1.2 Labeling di un nodo $< n >$ : algoritmo breadth-first

- 1) Espandi l'intero albero sotto n.
- 2) Valuta le foglie come vincenti per min o max.
- 3) Seleziona un nodo  $n_i$  senza label i cui figli sono labeled. Se non esiste ritorna il valore assegnamento a n.
- 4) Se  $n_i$  è un nodo in cui muove min assegna ad esso il minimo del valore dei figli, se deve muovere max il massimo. Ritorna a 3.

Complessità spazio temporale per  $b$  fattore di ramificazione e  $d$  livelli:  $\mathbf{b^d}$

### 2.1.3 Labeling di un nodo $\langle n \rangle$ : algoritmo depth-first

- 1) Metti in L i nodi non ancora espansi di  $\langle n \rangle$ .
- 2) Sia  $x$  il primo nodo di L. Se  $x = \langle n \rangle$  e c'è un valore assegnato a esso ritorna tale valore.
- 3) Altrimenti se  $x$  ha un valore assegnato  $V_x$ , sia  $p$  il padre di  $x$  con valore provvisorio  $V_p$ , se  $p$  è un nodo  $\min$   $V_p = \min(V_p, V_x)$ , se  $p$  è un nodo  $\max$   $V_p = \max(V_p, V_x)$ . Rimuovi  $x$  da L e torna a step 2.
- 4) Se  $x$  non è assegnato un valore ed è un nodo foglia, valuta la sua bontà per max min. Lascia  $x$  in L per poter valutare i suoi antenati e vai al 2.
- 5) Se a  $x$  non è stato assegnato un valore e non è un nodo foglia assegna  $V_x = -\inf$  se è un  $\max$   $V - x = +\inf$  viceversa. aggiungi i figli di  $x$  a L **in testa** e ritorna allo step 2.

Complessità temporale  $O(b^d)$  Complessità spaziale  $O(b * d)$

### 2.1.4 Espansione limitata

Espandere tutto l'albero è molto inefficiente. Shannon nel 1949: si guarda in avanti solo per un po' e si valutano le mosse fino ad un nodo non terminale ritenuto di successo. In pratica si applica minimax fino ad una certa profondità. Si introduce una funzione di valutazione di bontà di un nodo  $e(n)$ . La raffinatezza di tale funzione dipende dalle applicazioni: trade off tra ricerca e bontà della funzione.

Algoritmo breadth first diventa:

- 1) Metti in L i nodi non ancora espansi di  $\langle n \rangle$ .
- 2) Sia  $x$  il primo nodo di L. Se  $x = \langle n \rangle$  e c'è un valore assegnato a esso ritorna tale valore.
- 3) Altrimenti se  $x$  ha un valore assegnato  $V_x$ , sia  $p$  il padre di  $x$  con valore provvisorio  $V_p$ , se  $p$  è un nodo  $\min$   $V_p = \min(V_p, V_x)$ , se  $p$  è un nodo  $\max$   $V_p = \max(V_p, V_x)$ . Rimuovi  $x$  da L e torna a step 2.
- 4) Se  $x$  non è assegnato un valore ed è un nodo foglia, **oppure  $x$  è un nodo che decidiamo di non espandere ulteriormente**, valuta  $e(x)$ . Lascia  $x$  in L per poter valutare i suoi antenati e vai al 2).

- 5) Se a  $x$  non è stato assegnato un valore e non è un nodo foglia assegna  $V_x = -inf$  se è un  $\max V - x = +inf$  viceversa. aggiungi i figli di  $x$  a **L in testa** e ritorna allo step 2.

### Quanto espando?

Nota che se  $e(x)$  fosse perfetta basterebbe espandere i figli del nodo di interesse. **Soluzione possibile: espando sempre fino a una certa profondità  $p$ .**

- Problema: mosse più complicate tatticamente dovrebbe essere valutate con più profondità fino alla quiescenza (valori  $e(n)$  che variano poco).

A volte conviene effettuare una ricerca mirata sulla mossa selezionata.

### 2.1.5 Importante considerazione

Per quanto detto fin ora l'algoritmo minimax si basa sul concetto che nessun giocatore sbaglia! Se un giocatore sbaglia devo ricalcolare al volo tutto l'albero!

## 2.2 Tagli alfa-beta

Tecnica per ridurre lo spazio di ricerca all'interno dell'albero. Si  $N$  un nodo all'interno dell'albero, se il giocatore ha una scelta migliore  $M$  a livello del nodo genitore o di qualunque antenato di  $N$ ,  $N$  non sarà mai selezionato. Sia **ALFA** il valore della scelta migliore trovata sulla strada di **MAX** (il più alto) e **BETA** il valore della scelta migliore trovata sulla strada di **MIN** (il più basso), l'algoritmo aggiorna i valori di **ALFA** e **BETA** tagliando quando trova valori peggiori.

- Se un **ALPHA**-value è maggiore o uguale di un **Beta**-value di un nodo discendente: stop alla generazione di discendenti di **beta**.
- Se un **BETA**-value è minore o uguale di un **Alpha**-value di un nodo discendente: stop alla generazione di discendenti di **alpha**.

### 2.2.1 Algoritmo alpha-beta

- 1) Metti in **L** i nodi non ancora espansi di  $\langle n \rangle$ .
- 2) Sia  $x$  il primo nodo di **L**. Se  $x = \langle n \rangle$  e  $c$  è un valore assegnato a esso ritorna tale valore.

- 3) Altrimenti se  $x$  ha un valore assegnato  $V_x$ , sia  $p$  il padre di  $x$ . Se a  $x$  non è assegnato un valore vai al passo 5.
- Determiniamo se  $p$  ed i suoi figli possono essere eliminati dall'albero. Se  $p$  è un nodo min, sia alfa il massimo di tutti i correnti valori assegnati ai fratelli di  $p$  e sei nodi min che sono antenati di  $p$ .
  - Se non ci sono questi valori alfa = -inf.
  - Se  $V_x \leq \text{alfa}$  rimuovi tutti i suoi discendenti da L. (dualmente se  $p$  è un max).
- 4) Se  $p$  non può essere eliminato, sia  $V_p$  il suo valore corrente. Se  $p$  è un nodo min,  $V_p = \min(V_p, V_x)$ , se  $p$  è un nodo max  $V_p = \max(V_p, V_x)$ . Rimuovi  $x$  da L e torna a step 2.
- 5) Se  $x$  non è assegnato un valore ed è un nodo foglia, **oppure  $x$  è un nodo che decidiamo di non espandere ulteriormente**, valuta  $e(x)$ . Lascia  $x$  in L per poter valutare i suoi antenati e vai al 2).
- 6) Se a  $x$  non è stato assegnato un valore e non è un nodo foglia assegna  $V_x = -\text{inf}$  se è un max  $V - x = +\text{inf}$  viceversa. aggiungi i figli di  $x$  a L **in testa** e ritorna allo step 2.

**NOTA:** alfa-beta opera su algoritmo depth-first! Dobbiamo in prima battuta arrivare a una foglia o al massimo livello che possiamo espandere, valutare la funzione di utilità del nodo e propagare il valore.

### 2.2.2 Conclusioni

Ovviamente se valutiamo sempre i nodi peggiori, i nodi valutati successivamente risultano sempre nella linea corrente di ricerca e non c'è nessun taglio. Se valutiamo sempre il nodo migliore i restanti sono sempre tagliati, ma un caso del tutto teorico.

**Nel caso medio con distribuzione casuale dei valori ai nodi, il numero di nodi da esplorare diventa circa  $b^{\frac{3}{4}d}$ .**

## 3 Rappresentazione della conoscenza

### 3.1 Dati e Pattern

I dati sono gli ingrediente fondamentali del machine learning, dove il comportamento non è pre-programmato ma appreso dai dati stessi. Utilizzeremo spesso il termine **Pattern** per riferirci ai dati. Un pattern può essere pensato come **un membro di una classe descritto da valori**: un volto, un carattere scritto a mano, un'impronta digitale, un segnale sonoro, un frammento di testo, l'andamento di un titolo in borsa.

### 3.2 Tipo di Pattern

#### 3.2.1 Numerici

Valori associati a caratteristiche misurabili o conteggi.

- ☐ Tipicamente continui ma anche discreti. In ogni caso soggetti ad ordinamento.
- ☐ Rappresentabili naturalmente come vettori n-dimensionali.
- ☐ Es: estrazione di caratteristiche da segnali audio (feature vectors), una Persona:[altezza, peso, lunghezza piede, ...]

#### 3.2.2 Categorici

Valori associati a caratteristiche qualitative e alla presenza/assenza di una caratteristica (yes/no values).

- ☐ Non semanticamente mappabili in valori numerici.
- ☐ Es: Persona:[sesso, maggiorenne, colore occhi, gruppo sanguigno].
- ☐ Talvolta soggetti ad ordinamento: temperatura bassa, media o alta.
- ☐ Normalmente gestiti da sistemi a regole e alberi di classificazione.

#### 3.2.3 Sequenze

Pattern sequenziali con relazioni spazio o temporali.

- ☐ Uno stream audio corrispondente alla pronuncia di una parola, una frase in linguaggio naturale, un video.

- Spesso di lunghezza variabile.
- La posizione nella sequenza e le relazioni con predecessori e successori sono importanti.
- Critico trattare sequenze con pattern numerici. Allineamento spazio temporale e memoria per tener conto del passato.

### 3.2.4 Altri dati strutturati

Output organizzati in strutture complesse quali alberi e grafi.

- Applicazioni in bioinformatica, processamento del linguaggio naturale, ricognizione di parlato.
- esempio nella traduzione si una frase in linguaggio naturale l'output desiderato è l'insieme di parse tree plausibili.

## 3.3 Rappresentazione di Pattern

### 3.3.1 Proteine

Per poter essere usate in molti sistemi automatici le proteine vengono codificate in uno spazio vettoriale. Le proteine sono sequenze di amino-acidi. Gli amino-acido sono 20.

**Amino acid composition (AS)** conta gli amino acidi contenuti in una proteina normalizzando per la lunghezza della proteina  $N$ .

$$AS(i) = \frac{h(i)}{N} \text{ per ogni } i \in [1...20]$$

Calcolando per ogni  $i$ , ovvero per ognuno dei 20 aminoacidi, ho un vettore lungo 20.

**2-Gram (2G)** rappresenta una proteina con  $20^2$  feature, ogni feature è ottenuta contando il numero di occorrenze di una delle possibili coppie di aminoacidi, normalizzando per la lunghezza dell'aminoacido  $N$ .

$$2G(k) = \left( \frac{h(i, j)}{N} \right) \quad i, j \in [1...20] \quad k = j + 20(i - 1)$$

dove  $h(i, j)$  è il numero di occorrenze di una coppia di amino-acidi  $(i, j)$  in una proteina lunga  $N$ .

**N-Gram (NG)** come 2G, non cerco coppie ma tuple lunghe  $N$ . Dimensione del pattern  $20 * 20 * \dots * 20 = 20^N$ .

**Autocovariance approach (AC)** dato un parametro  $m$  che denota la massima distanza tra due amino-acidi considerati, data una proteina  $P = (p_1, p_2, \dots, p_N)$  e una proprietà fisico-chimica (proprietà dei singoli amino-acidi)  $d$ :

$$AC^d(i) = \begin{cases} h(i)/N & i \in [1..20] \\ \sum_{k=1}^{N-i+20} \frac{(index(p_k, d) - \mu_d) \cdot (index(p_{k+i-20}, d) - \mu_d)}{\sigma_d \cdot (N - i + 20)} & i \in [21..20 + m] \end{cases}$$

$index(p_k, d)$  ritorna il valore della proprietà  $d$  per un amino-acido  $p_k$ , ci sono poi due fattori di normalizzazione cioè media e varianza.  
 $AC \in \mathbb{R}^{20+m}$  Probabilmente le prime  $N$  fetures sono la probabilità di estratte quell'amino-acido perchè sarebbe come fare formulone su steso amino-acido  $p_k$ , sembra simulare il comportamento di un array circolare.

### 3.3.2 DNA

Anche per sequenze di DNA si possono usare le frequenza di occorrenza, in questo caso si parla di occorrenza della singola base azotata o di coppie, definito  $R$  il vettore che memorizza la sequenza abbiamo:

$$\begin{aligned} Nuc &= [R_i] (i=1, 2, 3, \dots, L) \\ F &= [f(A), f(C), f(G), f(T)] \quad \mathbf{f(X) \text{ è la frequenza relativa a X nella sequenza, f(XY) è la frequenza relativa a XY nella sequenza}} \\ F &= [f(AA), f(AC), f(AG), f(AT), \dots, f(TT)] \end{aligned}$$

### Pseudo nucleic acid representation

$$\begin{cases} g_1 = \frac{1}{L-2} \sum_{j=1}^{L-2} \Delta(R_i R_{j+1}, R_{j+2} R_{j+3}) \\ g_2 = \frac{1}{L-3} \sum_{j=1}^{L-3} \Delta(R_i R_{j+1}, R_{j+2} R_{j+3}) \\ g_3 = \frac{1}{L-4} \sum_{j=1}^{L-4} \Delta(R_i R_{j+1}, R_{j+3} R_{j+4}) \\ \dots \\ g_w = \frac{1}{L-w-1} \sum_{j=1}^{L-w-1} \Delta(R_i R_{j+1}, R_{j+w} R_{j+w+1}) \end{cases} \quad (w = L_{min} - 2)$$

$$\Delta(R_i R_{j+1}, R_k R_{k+1}) = \frac{1}{J} \sum_{j=1}^J [V_j(R_i R_{j+1}) - V_j(R_k R_{k+1})]^2$$

Dinucleotide	Physical structures					
	$V_d(RR_{R_{i-1}})$	$V_d(RR_{R_{i+1}})$	$V_d(RR_{R_i})$	$V_d(RR_{R_{i+1}})$	$V_d(RR_{R_{i+2}})$	$V_d(RR_{R_{i+3}})$
AA	0.06	0.50	0.27	1.59	0.11	-0.11
AC	1.50	0.50	0.80	0.13	1.29	1.04
AG	0.78	0.36	0.09	0.68	-0.24	-0.62
AT	1.07	0.22	0.62	-1.02	2.51	1.17
CA	-1.38	-1.36	-0.27	-0.86	-0.62	-1.25
CC	0.06	1.08	0.09	0.56	-0.82	0.24
CG	-1.66	-1.22	-0.44	-0.82	-0.29	-1.39
CT	0.78	0.36	0.09	0.68	-0.24	-0.62
GA	-0.08	0.50	0.27	0.13	-0.39	0.11
GC	-0.08	0.22	1.33	-0.35	0.65	1.39
GG	0.06	1.08	0.09	0.56	-0.82	0.24
GT	1.50	0.50	0.80	0.13	1.29	1.04
TA	-1.23	-2.37	-0.44	-2.24	-1.51	-1.39
TC	-0.08	0.50	0.27	0.13	-0.39	0.11
TG	-1.38	-1.36	-0.27	-0.86	-0.62	-1.25
TT	0.06	0.50	0.27	1.59	0.11	-0.11

$V(a,b)$  è una data proprietà fisica della coppia di basi (vedi la tabella), in tutto si usano 6 proprietà, dunque  $J=6$

**Micro array DNA** Un microarray di DNA (comunemente noto come gene chip, chip a DNA, biochip o matrici ad alta densità) è un insieme di microscopiche sonde di DNA attaccate ad una superficie solida come vetro, plastica, o chip di silicio formanti un array (matrice). Tali array consentono di verificare contemporaneamente la presenza di moltissimi geni all'interno di un campione di DNA (che spesso può rappresentare anche il genoma o il trascrittoma di un organismo).

Un utilizzo tipico è quello di confrontare il profilo di espressione genica di un individuo malato con quello di un individuo sano per individuare quali geni sono coinvolti nella malattia, oppure scegliere il farmaco adatto in base all'espressione genica. Un microarray consente di estrarre l'espressione di oltre 20000 geni, dunque ogni campione di DNA viene descritto da oltre 20000 elementi, la maggior parte non correlati al dato processo di classificazione.

Per ridurre la dimensionalità si utilizzano tecniche di selezione delle features per selezionare solo alcune espressioni geniche, e.g. metodi signal to noise calcolano per ogni gene  $j$  (esempio valido per problema bi classe es malato + non malato -).

$$S(j) = \frac{\mu_+(j) - \mu_-(j)}{\sigma_+(j) - \sigma_-(j)}$$

dove  $\mu_+$  e  $\mu_-$  sono le medie delle classi +1 e -1 per il  $j$ -esimo gene. i geni che danno valori alti saranno correlati alla classe +1, quelli con valori molto negativi alla classe -1.

### 3.3.3 Immagini

**Local binary pattern LBP** The LBP feature vector, in its simplest form, is created in the following manner:

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.



- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

The feature vector can now be processed using the Support vector machine, extreme learning machines, or some other machine-learning algorithm to classify images. Such classifiers can be used for face recognition or texture analysis.

Example	Threshold	Weights																											
<table border="1"><tr><td>6</td><td>5</td><td>2</td></tr><tr><td>7</td><td>6</td><td>1</td></tr><tr><td>9</td><td>8</td><td>7</td></tr></table>	6	5	2	7	6	1	9	8	7	<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td></td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	0	0	1		0	1	1	1	<table border="1"><tr><td>1</td><td>2</td><td>4</td></tr><tr><td>128</td><td></td><td>8</td></tr><tr><td>64</td><td>32</td><td>16</td></tr></table>	1	2	4	128		8	64	32	16
6	5	2																											
7	6	1																											
9	8	7																											
1	0	0																											
1		0																											
1	1	1																											
1	2	4																											
128		8																											
64	32	16																											

Pattern=11110001 LBP=1+16+32+64+128=241

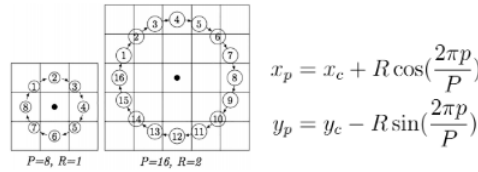
Fig. 1. Original LBP Operator

Formula per il calcolo della LBP centrato in  $(x_c, y_c)$

$$LBP(x_c, y_c) = \sum_{p=0}^7 2^p s(i_p - i_c)$$

$s(x)$  funzione segno dunque valuta uno se  $i_s > i_c$ . Sommare i valori ottenuto dal segno per  $2^p$  serve per rappresentare il vettore binario ottenuto dal pixel  $c$  in decimale.

**Extended local binary pattern** Estensione di LBP per gestire intorni di dimensioni variabile. L'idea è quella di allineare un numero arbitrario di pixel vicini su di un cerchio di raggio invisibile.



dove  $R$  è il raggio di un punto e  $P$  il numero di punti campione. Nota: lungo il raggio ci sono le potenze di 2: come prima estraggo un vettore binario per il centro con i confronti e poi moltiplico e sommo per rappresentarlo in decimale. **Se le coordinate di un punto sul cerchio non corrispondono alle coordinate di un pixel dell'immagine, il punto viene interpolato**

**Filtri e filtri di gabor** L'idea: per estrarre le features eseguire prodotti di convoluzione tra l'immagine e una serie di filtri (detti il banco) e di estrarre dalle immagini ottenute alcuni indicatori di sintesi (media, varianza, momento). Saranno questi a descrivere l'immagine: se per esempio uso 14 filtri e 2 indicatori (media e varianza) avrò un vettore di 28 featurese.

Un insieme di filtri molto usati sono i filtri di **Gabor**, per i quali sono state trovate forti analogie con alcuni meccanismi del sistema visivo umano. Ogni filtro è costituito da una funzione sinusoidale attenuata progressivamente da una gaussiana. Il filtro, costituito da una parte reale e una immaginaria, è regolato da 3 parametri

- la frequenza della sinusoide  $\omega$ .
- l'orientazione della sinusoide  $\theta$  rispetto al piano  $x, y$ .
- l'ampiezza della gaussiana  $\sigma$ .

**Focus su convoluzione Filtro digitale:** Una maschera di pesi che indica come ogni elemento dell'immagine debba essere modificato sulla base del valore dei pixel vicini.

Sia  $F$  un filtro definito su una griglia  $m \times m$  ( $m$  dispari), l'applicazione di  $F$  a un immagine  $I$  nel punto  $[i, j]$  modifica il pixel del punto  $I[i, j]$  come segue:

$$I'[i, j] = \sum_{y=1}^m \sum_{x=1}^m (I[i + \lceil \frac{m}{2} \rceil - y, j + \lceil \frac{m}{2} \rceil - x] \cdot F[y, x])$$

4	-2	1
-1	5	-3
-6	0	4

30	28	32
27	26	10
29	22	18

 $\Rightarrow$ 

$I'[i,j] =$
$18*4 - 22*2 + 29*1 +$
$-10*1 + 26*5 - 27*3 +$
$-32*6 + 28*0 + 30*4$

Tale operazione di media pesata è detta **convoluzione**. Problema: complessità computazionale parecchio elevata: data un immagine  $n \times n$  pixel e un filtro di  $m \times m$  elementi, la convoluzione richiede  $m^2 n^2$  moltiplicazioni e altrettante somme. **Soluzione: lavorare nello spazio delle trasformate di Fourier (trasformata è  $O(n \log(n))$ )**

Siano  $f$  e  $g$  due funzioni la cui convoluzione è indicata da  $f * g$ . Sia  $\mathcal{F}$  l'operatore trasformata di Fourier, sicché  $\mathcal{F}\{f\}$  e  $\mathcal{F}\{g\}$  sono le trasformate di  $f$  e  $g$  rispettivamente. Allora:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$


dove  $\cdot$  denota la moltiplicazione. Si ha anche che:

$$\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$$

Applicando la trasformata inversa  $\mathcal{F}^{-1}$ , si ottiene:

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

**Rappresentazione pattern – immagini – filtri di Gabor**



- Filtri di Gabor: onda sinusoidale modulata da una distribuzione Gaussiana.
- Il filtro di Gabor simmetrico bi-dimensionale ha la seguente forma:

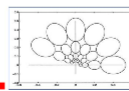
$$g(x, y; \theta, f) = \exp\left\{-\frac{1}{2}\left[\frac{x_\theta^2}{\sigma_x^2} + \frac{y_\theta^2}{\sigma_y^2}\right]\right\} \cdot \cos(2\pi f \cdot x_\theta)$$

dove  $\theta$  è l'orientazione del filtro,  $f$  è la frequenza della sinusoidale,  $\sigma_x$  e  $\sigma_y$  sono le deviazioni standard della Gaussiana negli assi  $x$  e  $y$  rispettivamente e  $[x_\theta, y_\theta]$  sono le coordinate di  $[x, y]$  dopo una rotazione in senso orario degli assi cartesiani di un angolo pari a  $(90^\circ - \theta)$ .

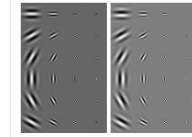
$$\begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} = \begin{bmatrix} \cos(90^\circ - \theta) & \sin(90^\circ - \theta) \\ -\sin(90^\circ - \theta) & \cos(90^\circ - \theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

<http://www.cogsci.nl/pages/gabor-generator.php> permette di rappresentare visivamente i filtri modificando i parametri

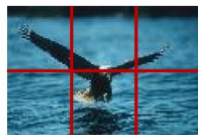
• Per la generazione del banco di filtri di Gabor, una volta definiti i range di interesse, si esegue una discretizzazione (in uno dei tanti modi possibili) dei 3 parametri fondamentali:



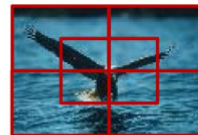
spettro del dominio delle frequenze da parte dei filtri: 6 orientazioni e 4 scale



**Informazioni spaziali** I descrittori appena visti, se calcolati sull'intera immagine, non mantengono informazioni di tipo spaziale. Per arricchire il descrittore con informazioni locali è possibile dividere l'immagine in regioni e calcolare per ciascuna i relativi descrittori. I.E. per divisione a 6 blocchi l'immagine sarà descritta da 6x features rispetto al descrittore per l'intera immagine



Partizionamento dell'immagine in regioni disgiunte



Fuzzy regions: suddivisione in regioni parzialmente sovrapposte

### 3.3.4 The dissimilarity space

Si estraggono una serie di pattern "originali" che verranno utilizzati a mo di base per il dataset. I nuovi pattern sono descritti da una serie di numeri (vettore) dove ogni elemento della serie è la similarità fra il dato pattern e uno dei pattern originali. In questo modo i pattern di base possono essere anche strutture complesse come grafi. In questo caso una serie di distanze tra i grafi sarà la rappresentazione vettoriale dei patterns. Definizione originale:

The dissimilarity space is a vector space in which the dimensions are defined by dissimilarity vectors measuring pairwise dissimilarities between examples and individual objects from the so-called representation set  $R$ . Hence, a dissimilarity representation  $D(X, R)$  is addressed as a data-dependent mapping  $D(\cdot, R) : X \rightarrow \mathbb{R}^n$  from an initial set of objects  $X$  to a dissimilarity space, equipped with the traditional inner product and Euclidean metric. The representation set can be chosen as the complete training set  $T$ , a set of carefully selected or constructed prototypes

Assume  $n$  objects,  $O = o_1, o_2, \dots, o_n$  and an  $n \times n$  dissimilarity matrix  $D := D(O, O)$ .  
 $D_{i,j} = d(o_i, o_j)$  is the dissimilarity between the sensor inputs for objects  $o_i$  and  $o_j$ .  
 $D(O, o_k) := [d(o_1, o_k), \dots, d(o_n, o_k)]^T$  is a feature defined by pairwise dissimilarities to  $o_k$ .  
 $x_i := D(o_i, O) = [d(o_i, o_1), \dots, d(o_i, o_n)]^T$  is a dissimilarity-based representation for  $o_i$ .  
 $X := D$  defines an  $n$ -dimensional vector space in which the dimension  $k$  is defined by  $D(O, o_k)$  and the vector  $x_i$  represents  $o_i$ .

□ **Praticamente i metodi «dissimilarity space» permettono di rappresentare in formato vettoriale patterns complessi anche senza che il «dissimilarity space» sia una metrica.**

Una **distanza** (o **metrica**) su un insieme  $X$  è una funzione

$$d : X \times X \longrightarrow \mathbb{R}$$

che soddisfa le seguenti proprietà per ogni scelta di  $x, y, z$  in  $X$ :

1.  $d(x, y) \geq 0$
2.  $d(x, y) = 0 \iff x = y$
3.  $d(x, y) = d(y, x)$  (simmetria)
4.  $d(x, y) \leq d(x, z) + d(z, y)$  (disuguaglianza triangolare)

La coppia  $(X, d)$  è chiamata spazio metrico.

Uno **spazio pseudometrico**  $(X, d)$  è un insieme  $X$  dotato di una funzione

$$d : X \times X \longrightarrow \mathbb{R}_{\geq 0}$$

chiamata **pseudometrica**, che soddisfa le proprietà seguenti per ogni  $x, y, z$  in  $X$ :

1.  $d(x, x) = 0$ .
2.  $d(x, y) = d(y, x)$  (simmetria)
3.  $d(x, z) \leq d(x, y) + d(y, z)$  (disuguaglianza triangolare)

## 3.4 Classificazione

Classificazione: **assegnare una classe a un pattern.**

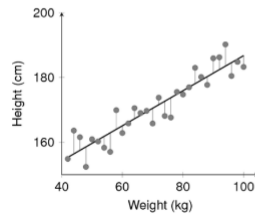
- Necessario apprendere una **funzione** che mappi lo spazio dei **pattern** nello spazio delle **classi**.
- Si usa spesso il termine **riconoscimento**.
- Caso 2 classi **binary classification**, con più **multy-class classification**.

Classe: insieme di pattern aventi proprietà comuni. Es due modi diversi di scrivere A. Il **concetto** di classe è semantico e dipende dall'applicazione: 21 classi per distinguere una lettera alfabeto ita, 2 per distinguere lettera ita da cirillico.

Esempi di problemi di classificazione: Spam detection, Credit card fraud detection, face reonition, Pedestrian classification, medical diagnosys, stock trading.

### 3.4.1 Regressione

- **Regressione:** assegna un **valore continuo** a un pattern.
- Utile per la predizione di valori continui.
- Risolvere un problema di regressione corrisponde ad apprendere una funzione approssimante delle coppie «input, output» date.



Es. stima dell'altezza  
di una persona in  
base al peso

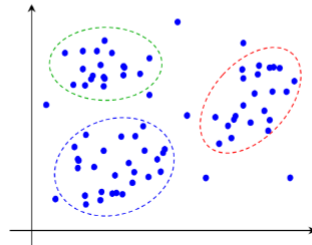
■ **Esempi di problemi di regressione:**

- Stima prezzi vendita appartamenti nel mercato immobiliare
- Stima del rischio per compagnie assicurative
- Predizione energia prodotta da impianto fotovoltaico
- Modelli sanitari di predizione dei costi

Quale tipo di funzione? Con quale grado di regolarità?

### 3.4.2 Clustering

- **Clustering:** individua **gruppi** (cluster) di pattern con caratteristiche simili.
- Le classi del problema non sono note e i pattern non etichettati → la natura non supervisionata del problema lo rende più complesso della classificazione.
- Spesso nemmeno il numero di cluster è noto a priori
- I cluster individuati nell'apprendimento possono essere poi utilizzati come classi.



■ **Esempi di problemi di clustering:**

- Marketing: definizione di gruppi di utenti in base ai consumi
- Genetica: raggruppamento individui sulla base analogie DNA
- Bioinformatica: partizionamento geni in gruppi con simili caratteristiche
- Visione: segmentazione immagini

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.



### 3.5 Riduzione di dimensionalità

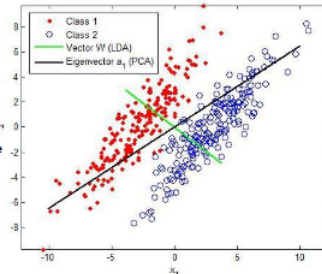
Riduce il numero di dimensione dei pattern in input. Consiste nell'apprendimento di un mapping da  $\mathbb{R}^d$  a  $\mathbb{R}^k$  con  $d < k$ .

L'operazione comporta una perdita di informazione. L'obiettivo è conservare le informazioni importanti, poichè non è detto che tutte le features siano utili (alcune potrebbero portare rumore) eliminandole si può anche migliorare le prestazioni del classificatore. La **definizione formale di importante dipende dall'applicazione**.

Questa procedura è molto utile per rendere trattabili problemi con dimensionalità molto elevate, per scartare informazioni ridondanti e/o instabili, e per visualizzare in 2D o 3D pattern con  $d > 3$ .

Ricordate che una dimensionalità elevata rispetto al numero di pattern porta al cosiddetto curse of dimensionality:

The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. There are multiple phenomena referred to by this name in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.



## 4 Learning

### 4.1 Feature learning (ad esempio deep learning)

Il successo del machine learning dipende dall'efficacia di rappresentazione dei pattern in termini di features. Si seguono due strade: o si crea un algoritmo ad hoc per estrarre le features a partire dai row data o si danno in input al sistema dati grezzi e il sistema estrae autonomamente le features dei pattern.

- La **definizione di features ad-hoc (hand-crafted)** per le diverse applicazioni prende il nome di **feature engineering**. I metodi hand-crafted sono dunque quei metodi che data un immagine o un oggetto **estraggono un pattern che rappresenta il dato**. Il sistema ha in input i dati come l'umano li vuole rappresentare. Ad esempio per il riconoscimento di oggetti esistono numerosi descrittori di forma, colore e tessitura che possiamo utilizzare per convertire immagini in vettori numerici.
- **Representation Learning o features learning** fa tutto il sistema: la rete deep decide come rappresentare il dato a partire dai dati grezzi, estraendo da essi le features in maniera autonoma. Gran parte delle tecniche di deep learning operano in questo modo esempio CNN (convolutional neural networks).

### 4.2 Apprendimento

#### 4.2.1 Tipi di apprendimento

##### Supervisionato o Supervised

Sono note le classi dei pattern per l'addestramento. Per ogni dato so a che classe appartiene, il **Training set è etichettato** (labeled). Situazione tipica nella classificazione, regressione e in alcune tecniche di riduzione della dimensionalità.

##### Non supervisionato o Unsupervised

Non sono note le classi dei pattern utilizzati per l'addestramento. **Il training set non è etichettato**. Situazione tipica: clustering ovvero trovare dei label per un insieme di pattern, e nella maggior parte di riduzione di dimensionalità.

### Semi-supervisionato o Semi-supervised

**Il training set è parzialmente etichettato.** Non tutti i pattern hanno label, magari troppo costoso farli classificare da un esperto umano. Si usa il core etichettato come base per tutti i pattern. La distribuzione dei pattern non etichettati può aiutare a ottimizzare la regola di classificazione.

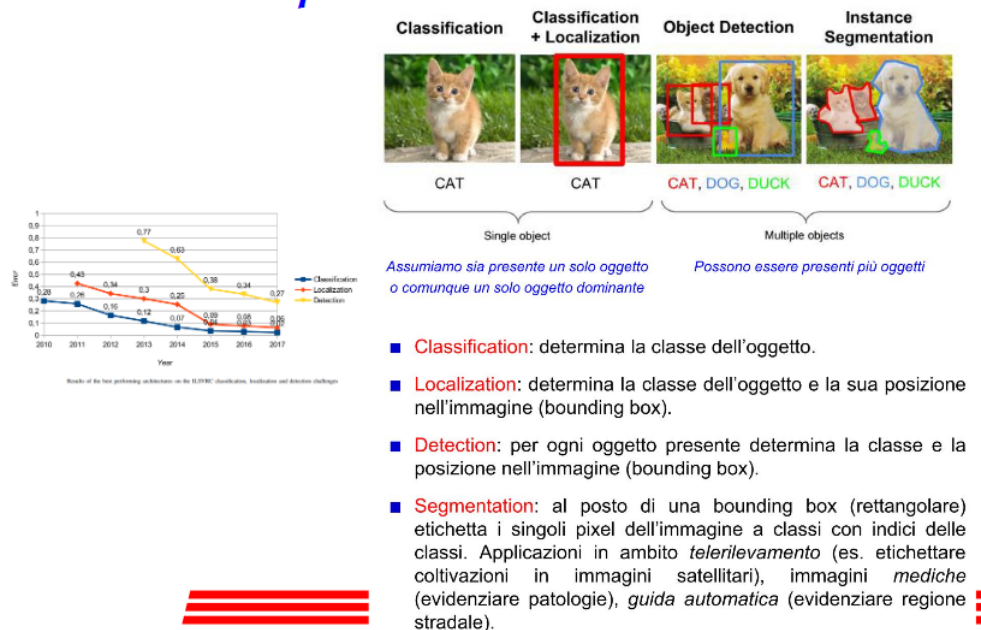
### Altri tipi di apprendimento

- **Multilabel:** un pattern viene assegnato a più classi, per esempio un articolo può essere di gossip, calcio, news contemporaneamente. Esistono algoritmi ad hoc per gestire problemi multi-label. Vedi dispense per esempi di applicazioni.
- **Active learning:** caso particolare di semi-supervised, ad ogni iterazione vengono scelti alcuni patterns senza label e fatti etichettare. Il sistema sceglie dunque quali pattern avere come labeled da un umano. Il sistema sceglie dunque autonomamente i patterns più utili o i più difficili ed un esperto umano fornisce il label. Il sistema ora sa nuovi pattern prima difficili. Il numero di patterns da far etichettare all'umano dipende dal costo del processo che il sistema compie per selezionare i pattern più utili.



## 4.2.2 Esempi di problemi di apprendimento

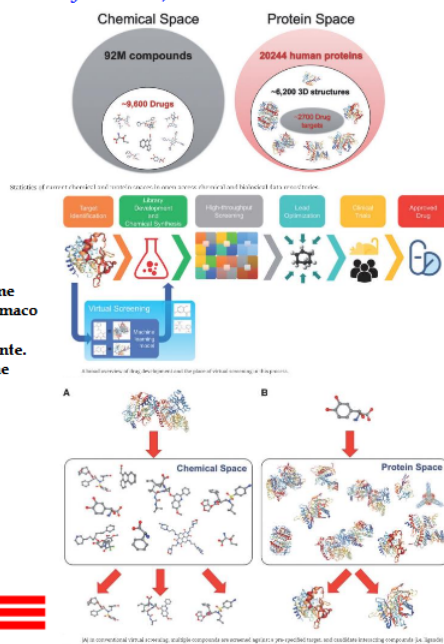
## Esempio in ambito computer vision



## Esempio in ambito biomedico da :Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases»

- A drug is an approved [by Food and Drug Administration (FDA), for example] bioactive compound that acts on protein targets to cure/decelerate a specific disease or to promote the health of a living being.
- A target protein (or just a target) is a naturally occurring biomolecule of an organism that is bound by a ligand and has its function modulated, which results in a physiological change in the body of the organism.
- A ligand is a molecular structure that physically binds another molecular structure and modulates its function.
- A compound is a chemical structure that is formed by the combination of two or more atoms that are connected by chemical bonds.

Obiettivo di un farmaco è modificare il comportamento di una proteina, si descrive sia il farmaco che la proteina con un diverso feature vector e si classifica tale coppia come «Interagiscono» vs «non-interagiscono»: ideale trovare farmaco che interagisce solo su quella proteina e non su altre (per minimizzare effetti collaterali), purtroppo succede raramente. Inoltre le malattie più complesse sono legate a più proteine mal gestite dal corpo umano.



### 4.2.3 Tecniche di addestramento della rete

#### Batch

L'addestramento è fatto una sola volta alla creazione della rete su un training set dato. Una volta terminato il training la rete passa in working mode e non è più in grado di apprendere ulteriormente. Attualmente il maggior numero di sistemi di machine learning opera in questo modo. Generalmente approccio supervisionato.

#### Incrementale

A seguito dell'addestramento iniziale, sono possibili ulteriori sessioni di addestramento. Ogni  $N$  tempo ri addestro la rete. Buona pratica è copiare la rete, lasciarne una on line nel mentre ri addestro al seconda, fare lo switch al termine dell'addestramento. Scenari: cicli di batch e working mode, i batch dopo il primo sono generalmente in unsupervised training. Rischio: catastrofic fogetting (la rete dimentica tutto ciò che ha imparato).

#### Naturale

Addestramento continuo, per tutta la vita. Addestramento attivo in work-mode: ogni volta che arriva un pattern si setta i parametri. Coesistenza di approccio supervisionato e non supervisionato. Human-like learning: inizialmente un piccolo pool di istruzioni (il labeling dei genitori durante l'infanzia) arricchito da grosso ammontare di esperienze non supervisionate.

## 4.3 Reinforcement Learning

**Apprendere un comportamento:** l'obiettivo è apprendere un comportamento a partire da esperienze passate. Un agente esegue azioni che modificano l'ambiente provocando un passaggio di stato. Quando l'agente ottiene risultati positivi riceve un reward che però può essere temporalmente ritardato rispetto all'azione, o alla sequenza di azioni che lo hanno determinato. L'obiettivo è apprendere le azioni ottimali in ciascuno stato, in modo da massimizzare la somma dei reward ottenuti nel lungo periodo.

Nella pratica non è facile ottenere esempi che siano allo stesso tempo corretti e rappresentativi di tutte le situazioni in cui l'agente deve agire. Pertanto il classico approccio supervisionato non è facilmente applicabile. Si può

però far operare l'agente in una realtà virtuale per potergli far fare migliaia di simulazioni al giorno.

Q learning è uno degli approcci più usati. La sua estensione deep, ovvero Deep Reinforcement Learning è alla base di Google DeepMind.

## 4.4 Ottimizzazione di parametri

In generale un algoritmo di machine learning è regolato da un insieme di parametri  $\Theta$  (esempio i pesi delle connessioni in una rete neurale). L'apprendimento consiste nel determinare il valore ottimo di  $\Theta^*$  di questi parametri.

Dato un training set  $Train$  e un insieme di parametri  $\Theta$ , la funzione obbiettivo  $f(Train, \Theta)$  può indicare:

- l'ottimalità della soluzione, da massimizzare

$$\Theta^* = \operatorname{argmax}_{\Theta} f(Train, \Theta)$$

- l'errore o la perdita (loss-function) da minimizzare

$$\Theta^* = \operatorname{argmin}_{\Theta} f(Train, \Theta)$$

Nella pratica si divide il data-set in due: una parte comporrà il train set, l'altra il test set. Solo il train set viene utilizzato per ottimizzare i parametri. Il test set viene utilizzato in un secondo momento per capire la bontà del mio sistema. Si dice pertanto che il test set è blind, il nostro modello non può vederlo.

$f(Train, \Theta)$  può essere ottimizzata in due modi:

- **Esplicitamente** con metodi che operano a partire dalla sua definizione matematica. Per esempio calcolando le derivate parziali rispetto ai parametri (gradiente) porlo uguale a zero e risolvere nei parametri.
- **Implicitamente** utilizzando un euristica che modifichi i parametri (come un algoritmo genetico).

### 4.4.1 Parametri vs Iperparametri

Molti algoritmi oltre all'ottimizzazione di parametri  $\Theta$  necessitano la definizione di iperparametri  $H$ . Questi iperparametri sono scelti prima dell'apprendimento vero e proprio e sono scelti da chi va a definire l'agente o l'intelligenza. In generale gli iperparametri  $H$  definiscono com'è fatto l'algoritmo, i parametri sono i  $\Theta$  che dovranno essere ottimizzati con l'addestramento.

Esempi di iperparametri sono: il numero di neuroni in una rete neurale, il numero di vincoli  $k$  in un classificatore  $k - nn$ , il grado di un polinomi utilizzato in regressione, il tipo di loss-function.

## 4.5 Valutazione delle prestazioni

Sono possibili diverse tecniche:

- Utilizzare direttamente la funzione obbiettivo per classificare le prestazioni. Si preferisce però in generale una misura più direttamente legata alla semantica del problema.
- In un problema di classificazione, l'**accuratezza** di classificazione è la percentuale di pattern correttamente classificati. L'errore di classificazione è il complemento.

$$accuratezza = \frac{\text{pattern correttamente classificati}}{\text{pattern classificati}}$$

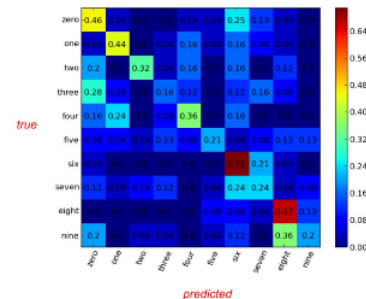
$$errore = 1 - accuratezza$$

- Nei problemi di **regressione** si valuta in genere RMSE (Root Mean Squared Error, scarto quadratico medio, radice della varianza).

$$RMSE = \sqrt{\frac{1}{N} \sum_{j=1 \dots N} (pred_i - true_i)^2}$$

- Nei problemi di **classificazione** (multiclasse) molto utile è la matrice di confusione per capire come sono distribuiti gli errori. Nel caso ideale vorremmo la matrice diagonale. Una tecnica per migliorare le prestazioni è aggiungere informazione: per le fragole oltre colore e forma anche la trama della pelle.

- Sulle righe le classi **true** sulla colonne le classi **predicted**.
- Una cella (r,c) riporta la percentuale di casi in cui il sistema ha predetto di classe **c** un pattern di classe vera **r**.
- Idealmente la matrice dovrebbe essere diagonale. Valori elevati (fuori diagonale) indicano concentrazioni di errori.



## 4.6 Closed set e Open set

Nel caso più semplice (e più comune nei benchmark di machine learning) si assume che il pattern da classificare appartenga ad una delle classi note per esempio uomini, donne. Questo è detto **closed set**. In molti casi reali invece i pattern da classificare possono appartenere a una delle classi note o a nessun di queste **open set**. Es classificatore di frutta mele, banane, pere riceve in input kiwi o ciliegie.

Due soluzioni

- Si aggiunge una classe fittizia "il resto del mondo" e si aggiungono al training set i cosiddetti "esempi negativi".
- Si consente al sistema di non assegnare il pattern. A tal fine si definisce una soglia e si assegna il pattern alla classe solo quando la probabilità superiore alla soglia.

## 4.7 Classificatore a soglia

Consideriamo un problema di classificazione binario dove le due classi corrispondono a esempi positivi (vera classe) e negativi (classe fittizia resto del mondo). È un problema a closed set, per esempio Face detection: se nell'immagine c'è un volto positivo altrimenti negativo. Possiamo trasformare questo problema in uno a Open set: consideriamo solo la classe positiva e un sistema con soglia  $t$  in grado di calcolare la probabilità  $p$  che un'immagine appartenga alla classe. Il pattern viene assegnato alla classe solo se  $p > t$ .

### 4.7.1 Errore di classificazione

Dati  $N$  pattern da classificare, il risultato di ciascuno dei tentativi del classificatore può essere:

- **True Positive (TP)** pattern positivo assegnato positivi.
- **True Negative (TN)** pattern negativo assegnato ai negativi.
- **False Positive (FP)** pattern negativo assegnato ai positivi.
- **False Negative (FN)** pattern positivo assegnato ai negativi.

Le frequenze dei due tipi di errore sono:

$$\text{False positive rate} = \frac{FP}{N_p}$$

$$\text{False negative rate} = \frac{FN}{N_n}$$

Entrambi mi dicono che percentuale di errore c'è tra tutti i pattern assegnati positivi ( $N_p$ ) e negativi ( $N_n$ ).

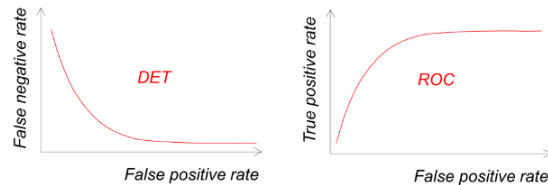
#### 4.7.2 La scelta della soglia

Soglie restrittive (elevate) riducono i False positive alzando i False negative, viceversa soglie tolleranti riducono i Falsi negati a discapito dei falsi positivi.

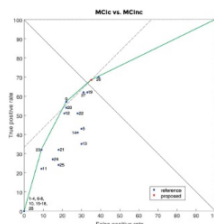


Altre rappresentazioni:

- Le due curve possono essere «condensate» in una curva **DET** (Detection Error Tradeoff) che nasconde la soglia. Piuttosto usata è anche la rappresentazione **ROC** (Receiver Operating Characteristic) che in ordinata riporta True positive invece di False negative (ROC è ribaltata verticalmente rispetto a DET).



L'area sottostante alla curva ROC (AUC, acronimo dei termini inglesi "Area Under the Curve") è una misura di accuratezza (singolo valore facilmente utilizzabile per confrontare algoritmi di classificazione). Tanto maggiore è l'area sotto la curva (cioè tanto più la curva si avvicina al vertice del grafico) tanto maggiore è il potere discriminante del test.



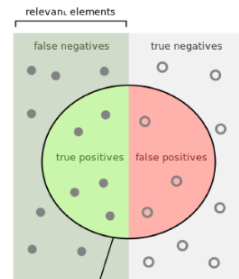
**Soglia**

Prediction of Mild Cognitive Impairment (MCI). conversion (MCIc) or not-conversion (MCInc) to Alzheimer Disease (within 18 months of follow up).

Come si legge la ROC curve (plot "verde")? Ammettendo un Falso Positive Rate di 10% (al 10% delle persone che non si ammaleranno verrà predetta erroneamente la malattia) avremmo un True positive Rate di 35%. Sembrano risultati bassi (è metodo basato solo su risonanza magnetica) ma già utili per estrarre set di persone sulle quali fare tests per nuovi farmaci, i.e. il TPR ci permette di selezionare un buon numero di persone che si ammaleranno.

- Un tipico esempio di sistemi con soglia sono i **sistemi biometrici** di **verifica** di identità:
  - Questa immagine dell'iride è del soggetto Q ? Solo se score è maggiore di una soglia
  - False positive e False negative in questo caso prendono il nome di **False Match** e **False Non-Match**.

### 4.7.3 Precision recall



**Precision** indica quanto è accurato il sistema.

Che percentuale di documenti selezionati è pertinente?

**Recall** quanto è selettivo.

Di tutti i documenti pertinenti quanti ne sono stati selezionati?

**Problema:** dato un set di articoli estrarre solo quelli il cui argomento viene selezionato, e.g. Estrarre tutti gli articoli relativi al terremoto di Messina del 1908

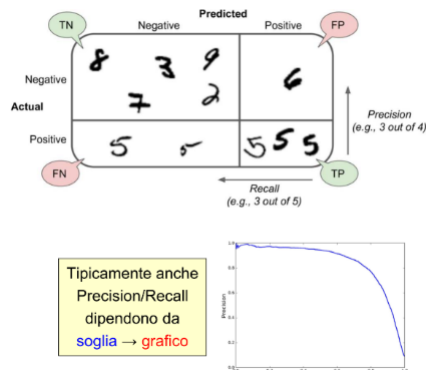
$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Attenzione ai problemi di classificazione con cardinalità classi molto sbilanciate:

- Es. in un problema di classificazione binaria, se una delle due classi è rara, un classificatore «dummy» che non fa predice mai potrebbe raggiungere un'accuratezza di classificazione vicina al 100%. Meglio usare Precision/Recall in questo caso

**Matrice di confusione** è un'altra rappresentazione grafica per comprendere precision recall, come già visto in valutazione delle prestazioni. Nell'esempio il classificatore binario ha come classe positiva il digit 5 e negativa tutti gli altri.



## 4.8 Training, Validation, Test

- **Training Set o Train** è l'insieme dei pattern su cui il sistema si addestra, trovando il valore ottimo dei parametri  $\Theta$ .
- **Validation Set o Valid** è l'insieme di pattern su cui tarare gli iperparametri  $H$ .
- **Test Set o Test** è l'insieme dei pattern su cui valutare le prestazioni finali del sistema.

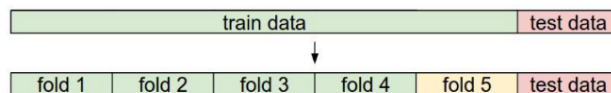
MAI SETTARE IPERPARAMETRI SU TEST SET pena OVER-FITTING (sovrastima delle prestazioni, vedi poi).

MAI USARE TEST SET SE NON PER LA VALIDAZIONE FINALE.

MAI USARLO PER EFFETTUARE SCELTE INERENTI IL MODELLO.

Nei benchmark di machine learning la suddivisione dei pattern è spesso predefinita per rendere confrontabili i risultati. Se non fosse così si procede come di seguito. Spiegazione con esempio, dati 12000 pattern:

- **Set disgiunti:** 10000 Train, 1000 Valid, 1000 Test.
- **K-fold Cross-Validation:** 2000 Test, K = 5 partizioni (fold) da 2000 pattern. Si esegue cinque volte il training scegliendo uno dei fold come Valid e i 4 rimanenti come Train. La prestazione finale è la media/mediana delle 5 prestazioni (sul Test).

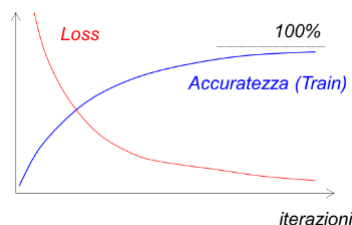


- **Leave-one-out:** caso estremo di cross-validation dove i fold hanno dimensione 1. Visto il costo computazionale, si utilizza quando i pattern sono pochi (es. < 100).

## 4.9 Convergenza

Primo obiettivo da perseguire durante il training: convergenza sul Training Set. In un classificatore a addestramento iterativo si ha convergenza quando:

- Il Loss (output della loss function) ha un andamento decrescente.
- L'accuratezza ha un andamento crescente.



Se il Loss non decresce o oscilla significativamente il sistema non converge: il metodo di ottimizzazione non è efficace, gli iperparametri sono fuori range, il learning è inadeguato, ci sono errori di implementazione ecc.



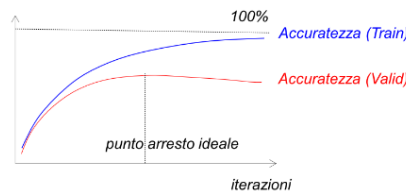
Se il loss decresce ma l'accuratezza non cresce, probabilmente è stata scelta una loss non adeguata.

Se l'accuratezza non si avvicina al 100% sul Train i gradi di libertà del classificatore non sono sufficienti per gestire la complessità del problema, oppure features non adatte a descrivere patterns, poichi patterns, il problema è molto complesso.

## 4.10 Overfitting

Una volta ottenuta la convergenza su Train si vuole massimizzare l'accuratezza su Valid. Il nostro scopo è infatti di creare un sistema in grado di generalizzare ovvero di trasportare la sua capacità acquisita sul Train nel Valid. Dagli esempi imparati comportarsi bene con esempi non noti.

**Pericolo:** se i gradi di libertà sono eccessivi, si raggiunge elevata accuratezza sul Train ma non sul Valid. Il sistema è scarso a generalizzare. In questo caso si parla di **overfitting di train**, bravo in quello che conosce scarso nell'ignoto. Questa situazione si verifica spesso se il Train è di piccole dimensioni.



Nei processi di addestramento iterativo, tipicamente dopo un certo numero di iterazioni, l'accuratezza su Valid non aumenta (può diminuire) a causa dell'overfitting. Monitorando l'andamenti su ouò interrompere l'addestramento nel punto ideale.

**I gradi di libertà non devono essere eccessivi ma adeguati alla complessità del problema.** Buona norma è partire con pochi gradi di libertà e via via aumentarli monitorando accuratezza su Train e Valid.

### ■ ■ ■ ■ Nella pratica

- Non utilizzate approcci di Machine Learning per problemi sui quali **non avete a disposizione sufficienti esempi** per il Training e il Test.
- Collezionare esempi (ed **etichettarli**) può richiedere **ingenti sforzi**, a meno che non siate in grado di reperire i pattern in rete, e/o non possiate pagare qualcuno per collezionarli/etichettarli al posto vostro (es. **Crowdsourcing** via **Amazon Mechanical Turk** per ImageNet).  
<https://www.mturk.com/>
- Collezione pattern **rappresentativi** del problema da risolvere e distribuiteli adeguatamente tra Train, Valid e Test.
  - evitate di concentrarvi solo su casi troppo semplici (ad esempio **rimuovendo** iterativamente i pattern che il vostro approccio non riesce a gestire).

Amazon Mechanical Turk (Mturk) è un servizio internet di crowdsourcing che permette ai programmatori informatici (conosciuti come requester) di coordinare l'uso di intelligenza umana per eseguire compiti. I Requester possono pubblicare obiettivi conosciuti come HIT (Human Intelligence Tasks), come identificare gli artisti in un cd musicale, le migliori fotografie di un negozio, la scrittura delle descrizioni di un prodotto.

- **Automatizzate** «subito» al meglio le procedure di valutazione delle prestazioni, le eseguite molte volte ... e alla fine avrete risparmiato un sacco di tempo.
  - **Confrontate** le prestazioni del vostro sistema solo con altri addestrati sullo stesso dataset e con lo stesso protocollo.
  - **Attenzione all'affidabilità statistica** dei risultati su set di piccole dimensioni. **Intervalli di confidenza** e **simulazioni su più Run** (al variare delle condizioni iniziali) possono aiutarvi.
- «Su più run» intende lanciare più volte la simulazione, e.g. lanciare k volte il n-fold cross validation
- Infine (**ma estremamente importante**): scrivete **codice** strutturato, ordinato, eseguite debug incrementale e unit testing. Gli algoritmi di Machine Learning non sono «**esatti**» e trovare bug nel codice può essere molto difficile!

## 5 Metodi di classificazione

### 5.1 Approccio bayesiano

Il problema è posto in termini probabilistici. Se tutte le distribuzioni il gioco sono note l'approccio Bayesiano costituisce la migliore regola di classificazione possibile: **soluzione ottima**.

- Sia  $V$  uno spazio  $d$ - dimensionale e  $W = w_1, w_2 \dots w_s$  un insieme di  $s$  classi disgiunte costituite da elementi di  $V$ .
- $\forall x \in V$  e  $\forall w_i \in W$ , indichiamo con  $p(x|w_i)$  la probabilità condizionale (o condizionata) di  $x$  data  $w_i$  ovvero la densità di probabilità che il prossimo pattern sia  $x$  sotto l'ipotesi che la sua classe di appartenenza sia  $w_i$ .
- $\forall w_i \in W$ ,  $p(w_i)$  è la **probabilità a priori** ovvero la probabilità indipendente dall'osservazione che il prossimo pattern da classificare sia di classe  $w_i$ . Un approccio semplice è le occorrenza nel train.
- $\forall x \in V$  indichiamo con  $p(x)$  la **densità a posteriori** di  $x$  ovvero la densità di probabilità che il prossimo pattern da classificare sia  $x$ .

$$p(x) = \sum_{i=1}^s p(x|w_i) \cdot p(w_i)$$

- $\forall x \in V$  e  $\forall w_i \in W$  indichiamo con  $p(w_i|x)$  la **probabilità a posteriori** di  $w_i$  dato  $x$  ovvero la probabilità che avendo osservato il pattern  $x$  la sua classe di appartenenza sia  $w_i$ . Questa va definita mediante vari approcci ad esempio NEAREST NEIGHBORHOOD.

Per il teorema di Bayes:

$$p(w_i|x) = \frac{p(x|w_i) \cdot p(w_i)}{p(x)}$$

Regola di classificazione di Bayes, si assegna a  $x$  la classe  $b$ :

$$b = \operatorname{argmax}_{i=1 \dots s} \{p(w_i|x)\}$$

Massimizzare la probabilità a posteriori significa massimizzare la densità di probabilità condizionale tenendo comunque conto della probabilità delle classi. La regola si dimostra ottima in quanto minimizza la probabilità di errore di classificazione.

### 5.1.1 Bayes: parametrico e non-parametrico

Mentre la stima delle probabilità a priori è abbastanza semplice (se non si hanno elementi si possono ipotizzare classi equiprobabili), la conoscenza delle densità condizionali  $p(x|w_i)$  è possibile solo in teoria. Nella pratica si seguono due soluzioni.

**Approccio Parametrico** si fanno ipotesi sulla forma delle distribuzioni a priori (es distribuzione multinormale) senza guardare i dati e si apprendono i parametri fondamentali (vettore medio, matrice covarianza) dal training set.

**Approccio non Parametrico** si apprendono le distribuzioni dal training set (es metodo di Parzen Window). Dai dati tiro fuori un **modello di distribuzione a posteriori**.

### 5.1.2 Bayes parametrico

Si utilizza quando, oltre ad avere una ragionevole certezza (o speranza) che la forma della distribuzione sia adeguata, la dimensione del training set non è sufficiente per una buona stima delle densità. Questo approccio è caratterizzato da un minor numero di gradi di libertà rispetto all'approccio non parametrico, e quindi rischia meno l'overfitting soprattutto in dataset piccoli.

### Distribuzione Normale

■ La densità di probabilità della distribuzione normale ( $d = 1$ ) è:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

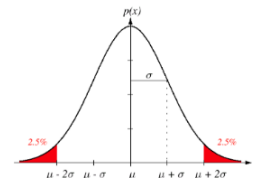
dove  $\mu$  è il **valor medio** è  $\sigma$  la **deviazione standard** (o **scarto quadratico medio**) e il suo quadrato  $\sigma^2$  la **varianza**.

Solo il 5% circa del "volume" è esterno all'intervallo  $[\mu - 2\sigma, \mu + 2\sigma]$ .

Solitamente si assume che la distribuzione valga 0 a distanze maggiori di  $3\sigma$  dal valore medio.

$$\sigma_X^2 = \frac{\sum_i (x_i - \mu_X)^2}{n},$$

dove  $\mu_X = \frac{\sum_i x_i}{n}$  è la **media aritmetica** di  $X$ .



## Multinormale

- La densità di probabilità nella distribuzione multinormale ( $d > 1$ )

è:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{-\frac{1}{2} (\mathbf{x}-\mu)^t \Sigma^{-1} (\mathbf{x}-\mu)}$$

Sia data una popolazione di  $n$  elementi su cui sono rilevati  $k$  caratteri quantitativi  $X_i$ . Cioè ogni  $X_i$  con  $i = 1, \dots, k$  è un vettore di  $n$  elementi, indicati con  $x_{ik}$  con  $k = 1, \dots, n$ . L'elemento  $x_{ik}$  rappresenta quindi la modalità dell' $i$ -esima unità statistica rispetto al carattere  $X_i$ . La matrice delle covarianze ha dimensione  $k \times k$  e ogni elemento è definito come

$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n (x_{ki} - \mu_i)(x_{kj} - \mu_j),$$

dove  $\mu_i$  indica la media del carattere  $X_i$ .

dove  $\mu = [\mu_1, \mu_2, \dots, \mu_d]$  è il **vettore medio** e  $\Sigma = [\sigma_{ij}]$  la **matrice di covarianza** ( $d \times d$ ).

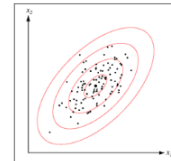
- Si assume che i vettori siano di tipo «colonna». L'apice  $t$  (trasposto) li trasforma in righe.
- $|\Sigma|$  e  $\Sigma^{-1}$  sono rispettivamente il determinante e l'inversa di  $\Sigma$ .
- La matrice di covarianza è sempre simmetrica e definita positiva, pertanto ammette inversa. Essendo simmetrica il numero di **parametri** che la definisce è  $d \cdot (d+1)/2$ .
- Gli elementi diagonali  $\sigma_{ij}$  sono le varianze dei rispettivi  $x_i$  (ovvero  $\sigma_i^2$ ); gli elementi non diagonali  $\sigma_{ij}$  sono le covarianze tra  $x_i$  e  $x_j$ :
  - se  $x_i$  e  $x_j$  sono statisticamente indipendenti  $\sigma_{ij} = 0$
  - se  $x_i$  e  $x_j$  sono correlati positivamente  $\sigma_{ij} > 0$
  - se  $x_i$  e  $x_j$  sono correlati negativamente  $\sigma_{ij} < 0$

## Multivariata

Per  $d = 2$  la forma della distribuzione è quella di un'ellisse.

- $\mu = [\mu_1, \mu_2]$  controlla la posizione del centro.
- $\sigma_{11}$  e  $\sigma_{22}$  determinano l'allungamento sui due assi dell'ellisse.
- $\sigma_{12} = \sigma_{21}$  controlla la rotazione dell'ellisse rispetto agli assi cartesiani.
  - se  $= 0$  (matrice di covarianza **diagonale**), la distribuzione multinormale è definita come prodotto di  $d$  normali monodimensionali. In tal caso gli assi dell'ellisse sono paralleli agli assi cartesiani
  - Se  $> 0$  (come nel caso della figura)  $x_1$  e  $x_2$  sono positivamente correlate (quando aumenta  $x_1$  aumenta anche  $x_2$ ).
  - Se  $< 0$   $x_1$  e  $x_2$  sono negativamente correlate (quando aumenta  $x_1$  cala  $x_2$ ).
- Gli assi dell'ellisse sono paralleli agli autovettori di  $\Sigma$ .

[https://it.wikipedia.org/wiki/Autovettore\\_e\\_autovalore](https://it.wikipedia.org/wiki/Autovettore_e_autovalore)



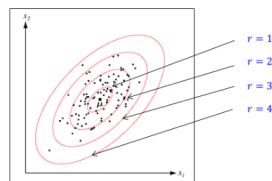
Le diverse ellissi individuano luoghi di punti a densità costante

## Distanza di Mahalanobis

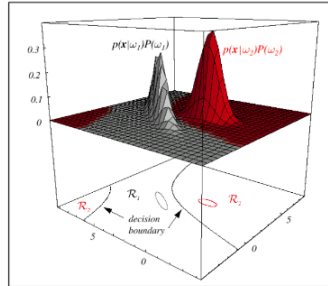
- La distanza di Mahalanobis  $r$  tra  $\mathbf{x}$  e  $\mu$ , definita dall'equazione:

$$r^2 = (\mathbf{x} - \mu)^t \Sigma^{-1} (\mathbf{x} - \mu)$$

definisce i bordi a densità costante in una distribuzione multinormale. Tale distanza viene spesso utilizzata in sostituzione della **distanza euclidea**, essendo in grado di "pesare" le diverse componenti tenendo conto dei relativi **spazi di variazione** e della loro **correlazione**.



## Bayes con Multinormali



- Nell'esempio sono visualizzate le **densità condizionali** di 2 classi di pattern (distribuiti con distribuzione normale 2-dimensionale) **corrette** sulla base delle rispettive **probabilità a priori**.
- La classificazione è eseguita utilizzando la regola **Bayesiana**. Lo **spazio è suddiviso** in regioni non connesse. Nel caso specifico  $\mathcal{R}_2$  è costituita da due componenti disgiunte.

Iper nel senso che la dimensione può anche essere  $> 3$

- Un **decision boundary** o **decision surface** (**superficie decisionale**) è una zona di confine tra regioni che il classificatore associa a classi diverse. Sul boundary la classificazione è ambigua.
- Le superfici decisionali possono assumere forme diverse. Nel caso specifico si tratta di **due iperbolici**. In generale:
  - Se le 2 matrici di covarianza sono **uguali tra loro**: la superficie decisionale è un **iper-piano**.
  - Se le 2 matrici di covarianza sono **arbitrarie**: la superficie decisionale è un **iper-quadratica**.

Un problema di definire linearmente separabile se esiste iperpiano che permette di separare i patterns che appartengono alle diverse classi

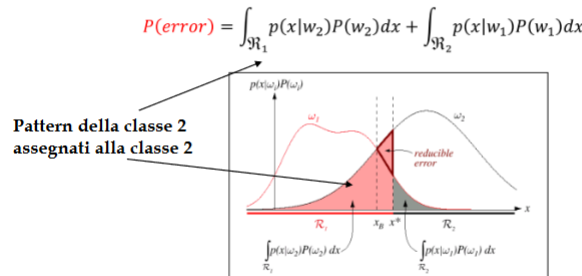
**Bayes Parametrico riassunto** Grandissimo vantaggio classificatore di bayes: da un valore output probabilistico che può essere usato come confidenza.

Un classificatore assegna quindi un pattern a una classe con una certa certezza (o confidenza) che può essere usata per scartare pattern in applicazioni open-set con soglia o costruire un multi classificatore. Se non si è interessati alla confidenza basta non dividere per  $p(x)$ :

$$b = \operatorname{argmax}_{i=1\dots s} \{p(x|w_i) \cdot p(w_i)\}$$

Il classificatore minimizza l'errore di classificazione, ad esempio nel caso 2 classi e  $d = 1$ .

- La regola si dimostra **ottima** in quanto minimizza l'**errore di classificazione**. Ad esempio nel caso di 2 classi e  $d = 1$ :



La soglia ottima è  $x_B$  che minimizza errore,  $x^*$  invece ottiene performance peggiore, l'errore aggiuntivo è quello chiamato nella figura «reducible error»

## In pratica

Spesso si azzarda ipotesi di normalità delle densità senza verifiche sperimentali, ciò porta a cattivi risultati di classificazione. Pertanto, dato un training set significativo deve essere innanzitutto valutata la rispondenza alla normalità. Ciò può essere fatto in due modi:

- In modo formale, esempio test statistico di Malkovich.
- In modo empirico, esempio visualizzando in vari modi i dati o gli istogrammi sulle diverse componenti e confrontandoli con le diverse curve teoriche.

Un volta provata una normalità bisogna stimare a partire dai dati il vettore medio e la matrice di covarianza. Le probabilità a priori possono essere poste tutte uguali o estratte dalla percentuale di campioni che nel training set appartengono alle diverse classi. Ogni nuovo pattern da classificare è una delle possibili classi in accordo con le regole di Bayes nella quale media e covarianza sono ora note.

### 5.1.3 Approcci Bayes non parametrici

Tecnologia usata ancora oggi. Non vengono fatte ipotesi sulle distribuzioni dei pattern e le densità di probabilità sono stime dirette del training set. Il problema della stima accurata delle densità è ritenuto da molti un problema più complesso della classificazione, perché risolvere sotto problema più complesso dell'originale? In genere la stima di densità è affrontabile per dimensionalità ridotte ( $\mathbb{R}^3$ ) ed esplode alla crescita delle dimensioni, volumi vuoti iniziano a essere enormi e i pattern sparsi.

**Parzen Window**

Si parte dalla densità di distribuzione binomiale. Dati  $n$  pattern indipendenti, la probabilità che  $k$  di questi cadano nella regione  $\mathfrak{R}$  che ho scelto è:

$$P_k = \binom{n}{k} p^k (1-p)^{n-k}$$

Nota in media il numero di pattern che cadono nella regione tende alla media della binomiale  $k \simeq E[P_k]$  il cui valore medio  $k \simeq E[P_k] = n \cdot p$  e quindi

$$p = k/n$$

Calcoliamo ora il parametro  $p$  come la probabilità che un pattern  $x$  cada all'interno di  $\mathfrak{R}$  e assumendo che la regione sia piccola di volume  $V$  e che  $p(\cdot)$  non vari significativamente al suo interno:

$$p = \int_{\mathfrak{R}} p(x') d(x') \simeq p(x) \cdot V$$

Ma allora:

$$p(x) = \frac{p}{V} = \frac{k}{n \cdot V}$$

Si sceglie come regione  $\mathfrak{R}$  un ipercubo in generale  $d$  - *dimensionale*, chiamato Finestra o Window definito dalla funzione:

$$\varphi(u) = \begin{cases} 1 & |u_j| \leq 1/2 \quad j = 1 \dots d \\ 0 & \text{altrimenti} \end{cases}$$

Dato un generico ipercubo centrato in  $x$  e avente lato  $h_n$  (e quindi volume  $V_n = h_n^d$ ), contiamo il numero di pattern al suo interno:

$$k_n = \sum_{i=1}^n \varphi\left(\frac{x_i - x}{h_n}\right)$$

sostituendo in  $k_n$  si ottiene:

$$p_n(x) = \frac{1}{n \cdot V_n} \sum_{i=1}^n \varphi\left(\frac{x_i - x}{h_n}\right) \quad \text{dove} \quad V_n = h_n^d$$

**Abbiamo così stimato la probabilità che  $x$  caschi in  $n$  ovvero la probabilità di osservare  $x$  sapendo che cascherà in  $n$   $p(x|n)$ .**

Ovviamente, soprattutto nel caso in cui numero ridotto di pattern, la dimensione della finestra (e quindi del lato) influiscono sul risultato: se la finestra è piccola la stima risulta piuttosto "rumorosa", molto attratta dai campioni e statisticamente instabile; se la finestra è grande la stima è più stabile ma piuttosto vaga e sfuocata. Si dimostra che per ottenere convergenza la dimensione della finestra deve essere calcolata tenendo conto del numero di campioni del training set:  $V_n = \frac{V_1}{\sqrt{n}}$  dove  $V_1$  o  $h_1$  sono iperparametri.

**Nella pratica** invece di ipercubi, come funzioni per le finestre si utilizzano **kernel function** più soft in cui ogni pattern contribuisce alla stima di densità di un intorno di  $x$  in accordo alla distanza da  $x$ . In questo modo si ottengono superfici decisionali molto più regolari. Le kernel function devono essere funzioni di densità, un approccio può essere una multinormale a media nulla e matrice di covarianza unitaria.

## 5.2 Nearest Neighbor (NN)

Invece di derivare dai dati la distribuzione condizionale delle classi per usare Bayes, questo classificatore cerca in modo pragmatico di massimizzare direttamente la probabilità a posteriori, infatti se il nuovo pattern  $x'$  è molto vicino a un pattern noto  $x$  allora è lecito pensare che  $p(w_i|x') \approx p(w_i|x)$

Data una matrice  $dist(\cdot)$  nello spazio multidimensionale (es. distanza euclidea), il classificatore assegna al nuovo pattern  $x'$  la stessa classe dell'elemento del TS (training set) più vicino.

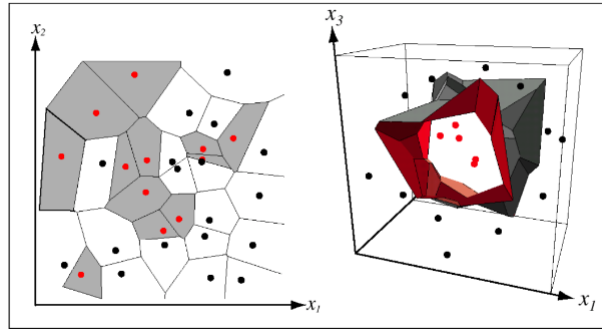
$$dist(x', x) = \min_{x \in TS} \{dist(x', x_i)\}$$

Si può dimostrare che nell'ipotesi di TS popolato da infiniti campioni, la probabilità  $P$  di errore nell'approccio NN non è mai peggiore (superiore) al doppio dell'errore possibile con Bayes. In pratica però questo non significa che l'approccio Bayesiano fornisca sempre risultati migliori di NN, se la stima di densità condizionale è poco accurata il classificatore di Bayes può andare peggio.

### 5.2.1 K-NN

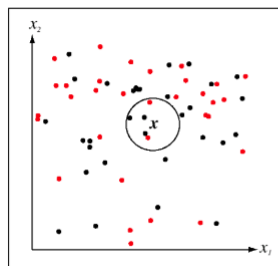
La regola NN produce una partizione dello spazio nota come tassellazione di Voronoi ovvero ogni elemento  $x_i \in TS$  determina un tassello all'interno del quale i pattern saranno assegnati alla classe  $x_i$ .





Regola piuttosto radicale: basta che un elemento non sia ben etichettato o affidabile, **Outlier**, affinché tutti i pattern nelle sue vicinanze siano etichettati male.

Un modo più robusto è quello di allargare il vicinato a  $k$  vicini: determino i  $k$  elementi più vicini al nuovo pattern  $x'$  da classificare ( $k$  è un iperparametro); ogni pattern tra i  $k$  vicini vota per la propria classe di appartenenza come classe da assegnare al nuovo pattern. La classe che ottiene più voti viene assegnata a  $x'$ .



nella figura il classificatore **5-NN**, assegna **x** alla classe "nera" in quanto quest'ultima ha ricevuto 3 voti su 5.

*Nel caso di 2 classi è bene scegliere  $k$  dispari per evitare pareggi.*

Per TS infiniti  $k$ -NN si dimostra meglio di 1-NN (solito NN) e all'aumentare di  $k$  converge all'errore bayesiano. Nella pratica (TS limitati) aumentare  $k$  significa estendere l'ipersfera di ricerca andando a sondare le probabilità a posteriori lontano dal punto di interesse; il valore ottimale di  $k$  (solitamente  $< 10$ ) va stimato su un validation set separato. Estrarre confidenza da  $k$ -NN, chiamiamo  $v_i$  i voti che il pattern ha ottenuto dalla classe  $i$ , siano  $s$  in tutto le classi:

$$\text{confidenza} = \left[ v_1/k, \dots, v_s/k \right]$$

il nuovo pattern appartiene alla classe  $i$  con probabilità  $v_i/k$ . In TS a numerosità elevate  $k$ -NN diventa problematico: necessario memorizzare tutti i pattern del TS, per ogni classificazione necessario calcolare la distanza da tutti i pattern del TS e ordinare le distanze per ottenere le più piccole. Quando

l'efficienza è importante è consigliabile indicizzare i dati attraverso strutture dati spaziali (es. kd-tree) che consente di individuare i vicini senza ricerca esaustiva. Sono strutture dati che consentono di trovare i vicini in tempo logaritmico, ovviamente in cambio c'è un maggiore costo ogni volta che si inserisce un nuovo pattern nel train.

## 6 Clustering

## 7 Riduzione di dimensionalità

## 8 Reti neurali e deep learning