Università degli Studi di Verona

# Dispensa del corso di Teorie e Tecniche del Riconoscimento

*Sebastiano Fregnan*

2019-2020

# Indice

# 1 | Introduzione

Il corso tratterà in particolare le tematiche

- Teoria della decisione di Bayes;
- Stima dei parametri e metodi non parametrici;
- Classificatori lineari, non lineari e funzioni discriminanti;
- Trasformazioni lineari e metodo di Fisher, estrazione e selezione delle feature, Principal Component Analysis;
- Misture di Gaussiane e algoritmi di Expectation-Maximization;
- Metodi generativi e discriminativi;
- Metodi Kernel e Support Vector Machine;
- Hidden Markov Models;
- Reti neurali artificiali;
- Pattern recognition per l'analisi ed il riconoscimento in immagini e video*;
- Metodi di classificazione non supervisionata (clustering)*.

* se ci sarà tempo. Ci approcceremo ai problemi di pattern recognition utilizzando l'intelligenza artificiale ed in particolare il machine learning.

## 1.1 Immagini al computer

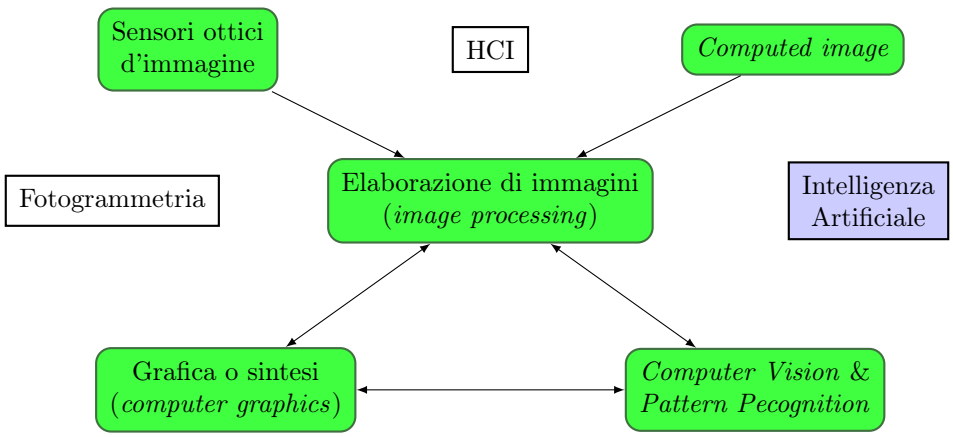



Figura 1.1: Interazioni tra le materie riguardanti le immagini al computer

I macroargomenti sono

**Sensori:** ottici (DSLR con CMOS o CCD), termici, multispettrali, 3D, a eventi, ...

**Computed image:** immagini ottenute studiando la fisica di un certo soggetto ed estrapolandone i dati usando un algoritmo. Esempi possono essere le immagini ottenute con una TAC o mediante raggi-X;

**Elaborazione:** utilizzo di filtri per il miglioramento, restauro, trasformazione dell'immagine e l'estrapolazione di informazioni da esse. I primi albori risalgono alla II guerra mondiale, poi proseguono negli anni '60 per le immagini da satellite. Viene usata in funzione alla Computer Vision o Graphics (per eliminare aliasing e sampling);

**Grafica:** algoritmi che sintetizzano (producono) un immagine;

**Computer Vision:** algoritmi che analizzano una o più immagini estrapolandone il contenuto, in maniera automatica. Informazioni sono geometria e dinamiche del mondo 3D, emulando l'interpretazione umana dell'immagine;

**Pattern Recognition:** un livello superiore alla Computer Vision, che cerca di estrarre concetti e significati dai contenuti di un immagine. Richiede una fase di pre-elaborazione per ridurre rumore e ridondanza delle misure e l'utilizzo della conoscenza delle proprietà statistiche e strutturali delle misure. Usata per

- Estrazione e selezione delle features;
- Classificazione (statistica o simbolica, comunque supervised);
- Cluster Analysis (unsupervised);

**Human Computer Interaction:** legati alla psicologia, come un uomo può interfacciarsi con un computer o un robot (COBOT) attraverso immagini, suoni e dispositivi di I/O;

**Fotogrammetria:** si preoccupa della misurazione di scene attraverso marker, in maniera semiautomatica;

**Intelligenza Artificiale:** approccio molto generale e ampio per seguire gli algoritmi sopra descritti.

## 1.2 Apprendimento

Diamo a "machine learning" un significato più informatico, mentre a "pattern recognition" una connotazione ingegneristica, anche se in pratica sono la stessa cosa. Definiamo invece degli altri termini chiave

***Supervised Learning***

Dato un set di coppie $(x, y)$ con $y = f(x)$ per una certa funzione $f$, l'obiettivo è trovare una funzione $\hat{f}$ che approssima $f$.

***Unsupervised Learning***

Dato un set di dati l'obiettivo è trovare una funzione $f$ che da una rappresentazione compatta del set.

***Reinforcement Learning***

Dato un set di triple $(x, a, r)$ tali che $r = f(a \mid x)$ è il premio (reward) con $a$ detta azione (etichetta), l'obiettivo è trovare mapping $x \to a$ tale che il reward $r$ sia massimo.

***Artificial Intelligence***

Materia con definizioni divergenti, in generale si impone di permettere ad un agente artificiale di

**pensare umanamente** , emulando quindi cognizione e psicologia umana degli eventi basandosi sugli studi di neuroscienza;

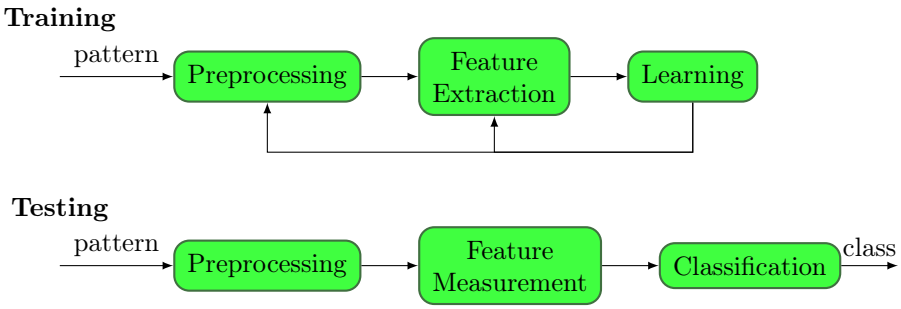
**pensare razionalmente** , quindi derivare ciò che accade mediante ragionamenti logici e regole metematiche;

**agire razionalmente** , quindi decidere cosa fare logicamente, massimizzando il risultato finale e non necessariamente pensando;

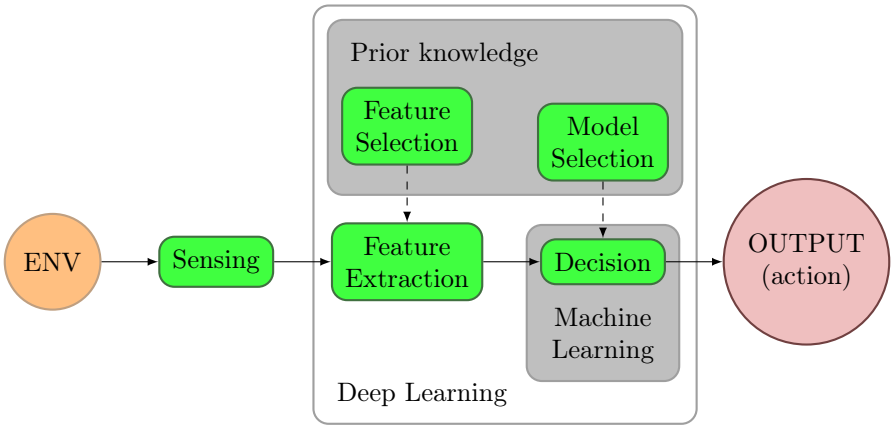
**agire umanamente** , comportandosi come farebbe un umano, possibilmente provando emozioni ed empatia (test di Turing, che non è riproducibile né costruttivo e non ha fondamenti matematici).

## 1.3 Modello di Pattern Recognition statistico

Si divide nelle fasi di ***training*** e ***testing***, in cui inzialmente la macchina si migliora nello svolgimento di un certo compito fino a raggiungere un certo livello di accuratezza su dati di training e successivamente tale livello di apprendimento viene valutato su nuovi dati. Mostriamo uno schema del funzionamento




e possiamo vedere tutto il procedimento di un sistema di pattern recognition come




Vediamo a grandi linee ognuno di questi blocchi, per poi studiarne nel dettaglio le caratteristiche.

### 1.3.1 Raccolta dati

Inizialmente, i dati devono essere collezionati in numero "sufficiente" (abbastanza campioni per ogni classe) e "rappresentativo" (più campioni differenti per ogni classe), ma questo dipende fortemente dalla "architettura" del classificatore.

### 1.3.2 Scelta delle feature

Dai dati vanno estrapolate le ***feature***, che sono proprietà o caratteristiche individuali misurabili di un fenomeno che viene osservato. Le feature vengono poste "affiancate" in un vettore detto appunto ***pattern*** e per essere "buone", esse devono essere

- *poche*, solo quelle rilevanti;
- *semplici* da ottenere, possibilmente in "real-time";
- *invarianti* a trasformazioni irrilevanti (rotazione, shift, rumori, errori, ...);
- *indipendenti* tra loro, altrimenti alcune sono eliminabili;
- *discriminative*, che mi permettono di classificare.

**Riconoscimento caratteri manoscritti**

Per identificare caratteri scritti nero su bianco a mano si potrebbero scegliere le feature:

- numero totale di pixel neri (invariante alla rotazione ma poco discriminante, ignora la forma);
- rapporto altezza/lunghezza (discrimina la forma allungata o allargata, ad esempio differenzia la "b" dalla "w" e dalla "i");

Il processo di scelta delle feature è in realtà più strutturato e si può dividere in 3 passaggi principali:

1. **feature extraction**, il processo di trasformazione del dato grezzo in valori misurabili utili alla modellazione del problema;
2. **feature transformation**, il processo di combinazione delle feature esistenti per migliorare le prestazioni della modellazione attuale;
3. **feature selection**, il processo di selezione di un sottoinsieme di feature rilevanti per la decisione.

### 1.3.3 Scelta del modello

Il ***modello*** per la classificazione è una struttura logica che rientra in certe regole di classificazione, che successivamente, nella fase di test, il classificatore userà per stimare il grado di appartenenza di un oggetto a una o più classi sulla base del vettore di feature che lo caratterizza. In questo merito, per il modello è necessario decidere:

- *tipo* (base matematica da usare);
- *parametri* (variabili indipendenti);
- *dimensionalilt`a* (complessità e intricatezza);
- *procedura di apprendimento* (funzione di costo, algoritmo di ottimizzazione);
- *strategia di validazione* (stima della bontà).

Il modello serve a dare un'***astrazione*** della classe che rappresenta e il classificatore si occupa di identificare le feature che meglio astraggono tale classe, sapendo che non necessariamente si sarà in grado di raggiungere la migliore astrazione possibile.

### 1.3.4 Addestramento

Dopo aver definito come costruire il modello comincia la fase di ***training***, nella quale il classificatore utilizza i dati a disposizione (***training set***) per costruire il modello con le caratteristiche sopra citate, cercando di evincere le regole che governano il fenomeno.

**Overfitting** Bisogna essere accorti rispetto al modello che scegliamo e a quanto permettere al classificatore di imparare dal training set, poiché raggiunta una certa "quota di apprendimento" esso andrà infatti a cercare di essere il più simile possibile al training set e quindi non sarà più buono per astrarre la classe e per classificare correttamente i dati di testing: questo è il problema di ***overfitting*** e non può essere risolto a priori, infatti sarà cura del progettista del classificatore capire in che modo evitarlo, in base ai dati iniziali e al tipo di modello scelto.

**Addestramento supervisionato (classificazione)** In tutto quello che abbiamo visto fino ad ora abbiamo dato per scontato che le classi fossero imposte a priori, etichettando ogni elemento del training set con la sua classe di appartenenza. Ora resta da capire se esiste un algoritmo di training di questo tipo che riesce a trovare la soluzione ottimale, in tempo utile, sempre, e se esso anche è sufficientemente scalabile per un numero maggiore di classi, mantenendo la condizione che la soluzione sia semplice.

**Addestramento non supervisionato (clustering)** Un approccio alternativo è invece quello di non etichettare il training set, così da imporre al sistema di evincere i ***cluster*** (gruppi) "naturali" all'interno del training set, sulla base della "similarità" dei pattern. Ovviamente questo approccio è intrisecamente più difficile della classificazione, dove *cluster naturali* sono gruppi di pattern che sono più simili tra loro e dove la *similarità* è una metrica che misura la distanza tra pattern.

**Addestramento con rinforzo (learning with a critic)** Viste le complicanze del clustering, cerchiamo una via di mezzo con la classificazione. In questo caso, non vengono fornite informazioni sulla categoria esatta, ma viene dato un giudizio sulla correttezza della classificazione, modificando di conseguenza la strategia di addestramento.

### 1.3.5 Valutazione

Infine, viene misurata la capacità di generalizzare del classificatore usando dei dati mai visti prima (***testing set***); riuscire a non avere errori però non significa aver raggiunto il classificatore ottimale, potrebbe infatti esserci overfitting.

**Set** La scelta della dimensione del training set e del testing set da creare usando la totalità dei dati risulta quindi essere cruciale, e infatti esistono alcune differenti strategie:

- *holdout*, suddivisione arbitraria (es. 50%/50% oppure 80%/20%). Tutte le altre strategie si basano su questa;
- *averaged holdout*, si effettuano più partizioni holdout e, dopo aver fatto il training ed il testing, si fa la media dei risultati ottenuti, trovando il risultato indipendente dalla partizione scelta;

- *leave-one-out*, si permuta ogni possibile partizione holdout dove il testing set è composto da un solo elemento, infine si compie la media. Questo meccanismo è abbastanza oneroso computazionalmente, si preferisce usare il prossimo;
- *leave-K-out*, come per il leave-one-out, ma questa volta si usano $K$ elementi.

In ogni caso, il fine ultimo di partizionare i dati è prevenire l'overfitting, andando a far convergere a 0 l'errore sul training set e cercando di evitare la risalita del testing set, sintomo dell'overfitting appunto.

**Variabilità** Durante l'apprendimento, il classificatore deve tenere conto di due importanti aspetti delle classi:

- *inter-class similarity*, ovvero casi in cui due classi differenti hanno elementi che sembrano essere simili (es. il 4 e il 7 manoscritti);
- *intra-class variability*, ovvero i casi in cui elementi della stessa classe sembrano di due classi diverse (es. il 7 manoscritto con o senza la barra al centro).

**Fairness** Sulla base di come il dataset è stato realizzato, è possibile che il classificatore evinca correlazioni erronee tra l'elemento e la sua classe di appartenenza e che quindi venga valutato ingiustamente (*unfairly*), cosa che ovviamente va evitata (es. l'assegnazione di un mutuo avviene mediante studi sui clienti precedenti che possibilmente si basano su inferenze oggigiorno non più valide, come l'età o il sesso).

**"No free lunch" theorem** "Nessun classificatore è innatamente superiore rispetto ad un altro", poiché dal punto di vista della generalizzazione, ogni algoritmo di apprendimento è valido. Se a priori non facciamo assunzioni rispetto alla processo di classificazione, anche la scelta casuale di una classe può essere un approccio valido, e non si può dimostrare che in generale esiste la superiorità di un algoritmo, poiché non ci sono ragioni indipendenti dal contesto del problema che favoriscono un metodo di apprendimento rispetto ad un altro. Il fatto che poi un algoritmo, in uno specifico caso, sia migliore rispetto ad un altro risiede solamente nello studio a priori dei dati e dalla natura del problema (es. le reti neurali non sono l'algoritmo definitivo, spesso possono essere invece la soluzione meno favorita).

**Rasoio di Occam** Durante l'apprendimento, spesso avviene che, per un certo input, il modello che fino a poco prima era ottimo deve essere aggiustato di conseguenza; è evidente quindi che un modello non può ritenersi perfetto finché l'apprendimento non ha incontrato ogni caso possibile. Evidentemente quindi, nella creazione del modello impattiamo sempre un ***bias induttivo***, che ci impedisce di proiettare con successo quello che alla fine sarà il modello ottimale, aggiungendo di volta in volta nuove assunzioni. Il ***rasoio di Occam*** è un principio secondo il quale quando ci si pongono davanti più ipotesi, è sempre meglio scegliere quella con meno assunzioni (es. se un modello è caratterizzato da un polinomio di grado 3 oppure uno di grado 9, è meglio scegliere quello di grado 3, poiché l'altro potrebbe risultare essere un caso di overfitting).

# 2 | Teoria di Bayes

Un problema di classificazione non è diverso dalla regressione: dato $x$ incognito, stimiamo il relativo valore $t$ (continuo nella regressione, discreto nella classificazione [sono etichette]). Trovati $x$ e stimato $t$, la stima della ***probabilità congiunta*** $p(x \cap t)$ è un problema di *inferenza*, nel quale si cerca di massimizzare questa funzione di densità di probabilità raffinando la stima di $t$.

Sia dato lo ***stato di natura*** $\omega$ ignoto, che può essere una qualsiasi classe $\omega_j$, per ora usiamo due classi $\omega_1$ e $\omega_2$; sapendo la ***probabilità a priori*** $P(\omega_j)$ di ogni classe (quindi senza nessuna misurazione, detta anche ***prior***), ad esempio $P(\omega_1) = 0.7$ e $P(\omega_2) = 0.3$, posso stimare $\omega$ in base alla *regola di decisione*

"scelgo la classe $\omega_j$ con probabilità $P(\omega_j)$ maggiore".

Se invece oltre al prior avessi anche una ***misurazione*** $x$, cioè una variabile aleatoria dipendente da $\omega_j$, otterrei una Funzione di Densità di Probabilità (PDF) $p(x \mid \omega_j)$ detta ***likelihood***, o *Funzione di Densità di Probabilità Classe-Condizionale (CCPDF)*, ovvero la probabilità di avere la misurazione $x$ sapendo che lo stato di natura è $\omega_j$.[1]

Fissata $x$, più è alta $p(x \mid \omega_j)$ più è *verosimile* (*likely*) che $\omega_j$ sia lo stato "giusto" da scegliere (in pratica significa che la misurazione $x$ è molto legata allo stato $\omega_j$).

## 2.1 Teoria di decisione di Bayes

Per decidere quale sia ($\omega_1$ oppure $\omega_2$) lo stato di natura $\omega$, possiamo ora fruttare la likelihood della misurazione $x$ per migliorare la stima, ed è quindi necessaria una formula che esprima la probabilità di $\omega_j$ anche in base alla bontà della misurazione.

**Formula di Bayes.** Note $P(\omega_j)$ e $p(x \mid \omega_j)$, la decisione dello stato di natura $\omega$ diventa

$$p(\omega_j \cap x) = P(\omega_j \mid x)p(x) = p(x \mid \omega_j)P(\omega_j)$$

e quindi

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j)P(\omega_j)}{p(x)}.$$

Definiamo la ***probabilità a posteriori*** $P(\omega_j \mid x)$ (quindi dopo la misurazione, detta anche ***posterior***) e utilizziamo l'***evidenza*** $p(x) = \sum_{j=1}^{2} p(x \mid \omega_j)P(\omega_j)$ della misurazione (quindi la probabilità complessiva di avere quella misurazione $x$) per normalizzare il numeratore e limitare $P(\omega_j \mid x)$ in un range $[0, 1]$, così da assicurare $\sum_{j=1}^{2} P(\omega_j \mid x) = 1$.

Sfruttando questa formula introduciamo la *regola di decisione di Bayes*:

"scelgo la classe $\omega_i$ con posterior $P(\omega_i \mid x)$ maggiore".

Con questa regola sorge la necessità di esprimere l'errore di aver scelto una classe piuttosto che l'altra (si poteva fare anche con per la regola precedente, ma non sarebbe stato molto utile; ora però che abbiamo la misurazione che influisce sulla probabilità, diventa necessario). Definiamo l'errore data la misurazione $x$ come

$$P(error \mid x) = \begin{cases} P(\omega_1 \mid x) & \text{se decido } \omega_2 \\ P(\omega_2 \mid x) & \text{se decido } \omega_1 \end{cases}$$

quindi l'errore di una scelta è semplicemente il posterior dell'altra classe. Detto questo, la probabilità complessiva di compiere un errore è

$$P(error) = \int_{-\infty}^{+\infty} P(error \cap x)\,\mathrm{d}x = \int_{-\infty}^{+\infty} P(error \mid x)p(x)\,\mathrm{d}x$$

e notiamo che la regola di decisione di Bayes minimizza l'errore quando è nota $x$, infatti viene sempre scelto il posterior con probabilità maggiore, lasciando come errore sempre il valore minore.

[1] usiamo $p(\cdot)$ per indicare funzioni di densità di probabilità (cioè con variabile aleatoria nel continuo), mentre $P(\cdot)$ per indicare funzioni di massa di probabilità (cioè con variabile aleatoria nel discreto)

### 2.1.1 Classificazione con feature

Ora estendiamo la misurazione $x$ a un vettore $\mathbf{x}$ di $d$ misurazioni dette ***feature*** e lo stato di natura $\omega$ ad un numero di $c$ stati detti ***categorie***; inoltre, conseguentemente alla categoria decisa, compiamo un'***azione*** $\alpha_i$ presa da un insieme di $a$ azioni.

Visto che ora decidere quale stato $\omega_j$ sia quello giusto avrà delle conseguenze a causa dell'azione compiuta, definiamo una ***funzione di costo*** $\lambda(\alpha_i \,|\, \omega_j)$, che descrive il costo (o anche *perdita*) del compiere l'azione $\alpha_i$ quando lo stato deciso è $\omega_j$ (in pratica è una versione potenziata della probabilità di errore).

La formula di Bayes in ogni caso non cambia (se non che ora abbiamo $c$ classi)

$$P(\omega_j \,|\, x) = \frac{p(x \,|\, \omega_j)P(\omega_j)}{p(x)} \qquad \text{posterior per } w_j$$

$$p(x) = \sum_{j=1}^{c} p(x \,|\, \omega_j)P(\omega_j) \qquad \text{evidenza di } x$$

però ora, data la natura incerta di $\omega_j$ a causa dell'aleatorietà di $\mathbf{x}$, avremo una *perdita attesa* (o ***rischio condizionale***) a seguito del compimento $\alpha_i$, che sarà

$$R(\alpha_i \,|\, \mathbf{x}) = \sum_{j=1}^{c} \lambda(\alpha_i \,|\, \omega_j)P(\omega_j \,|\, \mathbf{x})$$

e quindi avremo un rischio di errore per ogni azione compiuta data la misurazione $\mathbf{x}$. Come viene scelta l'azione $\alpha_i$ diventa un problema cruciale ed è quindi necessario definire propriamente una ***regola di decisione***: descriviamo tale regola mediante l'utilizzo di una ***funzione di decisione*** $\alpha(\mathbf{x})$, la quale sceglierà secondo una certa logica l'azione $\alpha_i$ da compiere in base a $\mathbf{x}$. Detto questo, definiamo il ***rischio complessivo*** dovuto all'utilizzo di una specifica $\alpha(\mathbf{x})$ come

$$R = \mathbb{E}\Big[R\big(\alpha(\mathbf{x}) \,|\, \mathbf{x}\big)\Big] = \int_{\Omega} R\big(\alpha(\mathbf{x}) \,|\, \mathbf{x}\big)\, p(\mathbf{x})\, \mathrm{d}\mathbf{x}$$

e chiaramente, scegliendo $\alpha(\mathbf{x})$ in modo che essa ritorni per ogni $\mathbf{x}$ l'azione $\alpha_i$ con il rischio $R(\alpha_i \,|\, \mathbf{x})$ minore, il rischio complessivo $R$ sarà sempre il minore possibile (precisiamo che l'integrale è definito nello spazio $\Omega$ delle feature, che solitamente è $\mathbb{R}$).

La ***regola di decisione di Bayes*** vista prima, che adesso rappresenta formalmente la funzione $\alpha(\mathbf{x})$ di nostra scelta, viene quindi estesa per minimizzare il rischio (non più il posterior) e seguirà i seguenti step:

1. calcola $R(\alpha_i \,|\, \mathbf{x})$ per ogni $i = 1 \dots a$;
2. scegli $\alpha_i$ dove $i = \min_i R(\alpha_i \,|\, \mathbf{x})$.

In questo modo viene sempre scelta l'azione che è meno rischiosa da compiere; il rischio complessivo risultante prende il nome di ***rischio di Bayes*** $R^*$ ed è in assoluto la migliore performance che può essere raggiunta.

***Valore atteso (expected value)***

Per il caso continuo, il valore atteso (o ***media*** o ***speranza***) di una variabile casuale $X$ di dominio $\Omega$ che segue la funzione di densità di probabilità $p(x)$ è definito come

$$\mathbb{E}[X] = \int_{\Omega} x\, p(x)\, \mathrm{d}x = \mu$$

Solitamente $\Omega = \mathbb{R}$ e quindi l'integrale risulta essere $\int_{-\infty}^{+\infty}$. Un particolare caso è la ***varianza***

$$\mathbb{E}[(X-\mu)^2] = \int_{\Omega} (X-\mu)^2\, p(x)\, \mathrm{d}x = \mathbb{E}[X^2] - \big(E[X]\big)^2 = \sigma^2$$

### 2.1.2 Classificazione a 2 categorie

Torniamo al problema con due stati di natura $\omega_1$ e $\omega_2$, e scriviamo in forma compatta il costo usando $\lambda_{ij} = \lambda(\alpha_i \,|\, \omega_j)$; in questo particolare caso dato lo stato $\omega_i$ l'azione corretta da compiere è $\alpha_i$. Per quanto detto sopra, il rischio condizionale di ogni azione sarà dato da

$$R(\alpha_1 \,|\, \mathbf{x}) = \lambda_{11}P(\omega_1 \,|\, \mathbf{x}) + \lambda_{12}P(\omega_2 \,|\, \mathbf{x})$$
$$R(\alpha_2 \,|\, \mathbf{x}) = \lambda_{21}P(\omega_1 \,|\, \mathbf{x}) + \lambda_{22}P(\omega_2 \,|\, \mathbf{x})$$

e dobbiamo ora esprimere la regola di decisione di rischio minimo $\alpha(\mathbf{x})$. Ci sono svariati modi equivalenti di esprimere questa regola ma alla base possiamo dire di decidere $\omega_1$ se $R(\alpha_1 \mid \mathbf{x}) < R(\alpha_2 \mid \mathbf{x})$ e $\omega_2$ altrimenti; riscriviamo ora questa regola in vari termini, espandendo via via le definizioni date finora:

$$\begin{aligned} R(\alpha_1 \mid \mathbf{x}) &< R(\alpha_2 \mid \mathbf{x}) && \text{tramite la regola fondamentale} \\ (\lambda_{21} - \lambda_{11})P(\omega_1 \mid \mathbf{x}) &> (\lambda_{12} - \lambda_{22})P(\omega_2 \mid \mathbf{x}) && \text{in termini di posterior} \\ (\lambda_{21} - \lambda_{11})p(\mathbf{x} \mid \omega_1)P(\omega_1) &> (\lambda_{12} - \lambda_{22})p(\mathbf{x} \mid \omega_2)P(\omega_2) && \text{in termini di prior} \end{aligned}$$

ma la più interessante di tutte è in termini di ***likelihood ratio*** $\frac{p(\mathbf{x} \mid \omega_1)}{p(\mathbf{x} \mid \omega_2)}$

$$\frac{p(\mathbf{x} \mid \omega_1)}{p(\mathbf{x} \mid \omega_2)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \frac{P(\omega_2)}{P(\omega_1)}$$

in cui si pone il focus sulla dipendenza da $\mathbf{x}$ delle densità di probabilità $p(\mathbf{x} \mid \omega_j)$. Dato che il termine destro e composto di valori tutti costanti, la scelta di $\omega_1$ sarà indipendente dalla misurazione $\mathbf{x}$ e avverrà solo se il rapporto di likelihood supera la soglia a destra.

## 2.2 Classificazione Minimum-Error-Rate

In un problema di classificazione solitamente si ha un'azione $\alpha_i$ per ogni stato $\omega_j$ e si sceglie l'azione tale che $i = j$. Introduciamo quindi una funzione di costo detta ***simmetrica*** o ***di perdita 0-1***, definita come

$$\lambda(\alpha_i, \omega_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

la quale valuta tutti gli errori come ugualmente errati (hanno tutti costo 1). Il rischio corrispondente a questa funzione di costo è esattamente la *probabilità media di errore*, dato che

$$\begin{aligned} R(\alpha_i \mid x) &= \sum_{j=1}^{c} \lambda(\alpha_i \mid \omega_j) P(\omega_j \mid x) \\ &= \sum_{i \neq j}^{c} P(\omega_j \mid x) \\ &= 1 - P(\omega_i \mid x) \end{aligned}$$

e il rischio complessivo in questo caso devo massimizzare $P(\omega_i \mid x)$, quindi per il *Minimum-Error-Rate*

"scelgo la classe $\omega_i$ con posterior $P(\omega_i \mid x)$ maggiore"

che è la regola di decisione di Bayes che abbiamo visto all'inizio.

## 2.3 Classificatori e Funzioni discriminanti

Il problema di classificazione può essere quindi diviso in una parte di *inferenza* (per addestrare il modello $p(\omega_j \mid \mathbf{x})$ usando i dati) e una di *decisione* (per scegliere la classe usando i posterior). In modo alternativo si può usare una ***funzione discriminante*** per passare direttamente dall'input $\mathbf{x}$ alla classe decisa. Quindi si distinguono tre approcci:

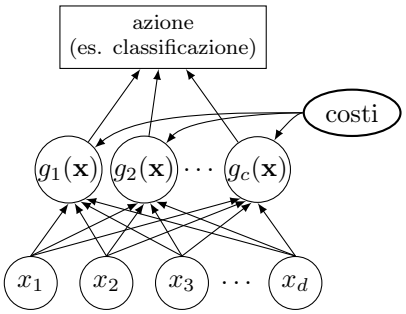
- inferire le likelihood $p(\mathbf{x} \mid \omega_j)$ e i prior $P(\omega_j)$ per ogni classe e usare la formula di Bayes per trovare i posterior $P(\omega_j \mid \mathbf{x})$, per poi determinare la classe tramite la teoria di decisione [oppure modellare direttamente la probabilità congiunta $p(\omega_j \cap \mathbf{x})$ per poi normalizzarla e ottenere subito la posterior $P(\omega_j)$, usando ***modelli generativi***; richiedono training set esaustivi ma permettono il fine-tuning delle variabili in gioco];
- inferire direttamente la posterior $P(\omega_j)$ e determinare la classe tramite la teoria di decisione, usando ***modelli discriminativi***, che per problemi di classificazione sono più efficienti delle precedenti;
- trovare una ***funzione discriminante*** $f(\mathbf{x})$ che mappa direttamente l'input $\mathbf{x}$ ad una classe, che in pratica divide lo spazio degli input con delle superfici di separazione.

Spesso però stimare la posterior è utile, in quanto

- minimizza il rischio $R$ quando la funzione di costo $\lambda(\alpha_i, \omega_j)$ cambia nel tempo;
- dato che abbiamo direttamente la posterior, vengono compensati i prior sbilanciati delle classi quando il training set non è uniforme (succede spesso nei casi reali di avere più esempi di una certa classe piuttosto che di un'altra);
- quando un problema ha più misurazioni indipendenti tra loro, è possibile risolvere i singoli sottoproblemi di ogni misurazione e poi fondere insieme i risultati (***classificazione naive Bayes***).

## 2.4 Funzioni discriminanti

Con questo appoccio cambiamo punto di vista, e anziché calcolare le probabilità singolarmente, cerco di ottenere una funzione ***classificatore*** che data in input l'osservazione decida la classe alla quale assegnarla. In pratica quindi, un classificatore è visto come un set di ***funzioni discriminanti*** $g_i(\mathbf{x})$ dove $i = 1 \dots c$, ognuna delle quali delinea tramite dei ***confini di decisione*** una porzione dello spazio delle osservazioni per la classe $\omega_i$, detta ***superfice di separazione*** $\mathcal{R}_i$ (o di *decisione*).




Con questa definizione un classificatore è una rete di $c$ funzioni discriminanti $g_i(\mathbf{x})$ che assegna un vettore di feature $\mathbf{x}$ alla classe $\omega_i$ se $g_i(\mathbf{x}) > g_j(\mathbf{x})$ per ogni $j \neq i$, ovvero la classe la cui funzione discriminante ha valore maggiore rispetto a tutte le altre.

Notiamo che le classificazioni viste finora possono essere mappate 1:1 con questo modello, infatti impostando $g_i(\mathbf{x}) = -R(\alpha_i \,|\, \mathbf{x})$ avremo un minimizzatore di rischio generico, mentre impostando $g_i(\mathbf{x}) = P(\omega_i \,|\, \mathbf{x})$ avremo la massimizzazione del Minimum-Error-Rate. In generale, data una famiglia di funzioni $g_i(\mathbf{x})$ ne esistono infinite equivalenti, ad esempio per $f$ funzione crescente monotona abbiamo $g_i(\mathbf{x}) \equiv f(g_i(\mathbf{x}))$; questo risulta molto conveniente quando analiticamente la funzione $g_i$ è semplice ma computazionalmente è intrattabile (ad esempio quando $g_i(\mathbf{x}) = e^{\mathbf{x}}$).

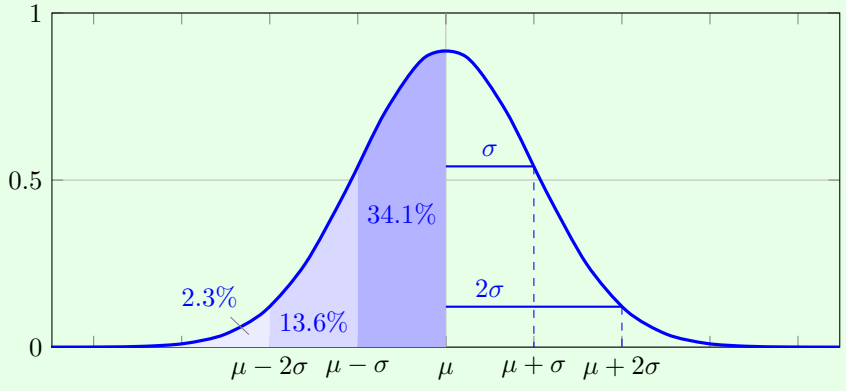
### 2.4.1 Densità normale

Per ora abbiamo definito le funzioni discriminanti esattamente come le prior delle classi o le densità condizionali, ma in generale possiamo utilizzare funzioni più complesse: ricorriamo alla ***densità normale*** o ***Gaussiana***, la quale è analiticamente trattabile e, tramite il ***teorema del limite centrale***, permette, sotto alcune condizioni (che vedremo in seguito), di modellare in maniera ottimale ogni caso reale, visto che in generale la distribuzione della somma di $d$ variabili aleatorie indipendenti tende appunto alla distribuzione normale. Inoltre, la densità normale è l'unica funzione che ritorna sè stessa dopo la trasformata di Fourier, e quindi è ottimale per la localizzazione *precisa* nel tempo oppure in frequenza (ma per il principio di indeterminatezza, non in entrambe contemporaneamente).

***Densità normale univariata***

Definiamo formalmente la ***densità normale univariata*** $N(\mu, \sigma^2)$, di *media* $\mu$ e varianza $\sigma^2$, come la funzione di probabilità

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



Come da definizione, possiamo verificare che la media è $\mu$ usando la definizione di media nel continuo

$$\mathbb{E}[x] = \int_{-\infty}^{+\infty} x\, p(x)\, \mathrm{d}x \doteq \mu$$

e possiamo verificare che la varianza è $\sigma^2$ usando la definizione di varianza nel continuo

$$\mathbb{E}\big[(x-\mu)^2\big] = \int_{-\infty}^{+\infty} (x-\mu)^2\, p(x)\, \mathrm{d}x \doteq \sigma^2$$

Fissata media e varianza, la densità normale è quella dotata di massima entropia: l'entropia misura l'incertezza di una distribuzione o la quantità di informazione necessaria in media per descrivere la variabile aleatoria associata, ed è data da

$$H\big(p(x)\big) = -\int p(x) \ln p(x)\, \mathrm{d}x$$

***Densità normale multivariata***

Analogamente, definiamo formalmente la ***densità normale multivariata*** $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ a $d$ dimensioni

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\Big\{-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big\}$$

dove $\mathbf{x}$ è il vettore delle $d$ variabili, $\boldsymbol{\mu}$ è il vettore di media delle $d$ variabili e $\boldsymbol{\Sigma}$ è la matrice $d \times d$ della covarianza, calcolabile analiticamente come

$$\boldsymbol{\Sigma} = \mathbb{E}\big[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\top\big] = \int_{-\infty}^{+\infty} (\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\top p(\mathbf{x})\, \mathrm{d}\mathbf{x}$$

mentre elemento per elemento

$$\sigma_{ij} = \mathbb{E}\big[(x_i-\mu_i)(x_j-\mu_j)\big] = \int_{-\infty}^{+\infty} (x_i-\mu_i)(x_j-\mu_j) p(\mathbf{x})\, \mathrm{d}x$$

In generale, la matrice di covarianza $\boldsymbol{\Sigma}$ ci permette di calcolare la dispersione dei dati su ogni superficie dello spazio e ha delle caratteristiche interessanti:

- è simmetrica e semidefinita positiva (cioè $\det(\boldsymbol{\Sigma}) \geq 0$);
- tutti i $\sigma_{ij}$ sono la covarianza tra $x_i$ ed $x_j$ e quindi tutti i $\sigma_{ii}$ sono la varianza $\sigma_i^2$ di $x_i$. Se $x_i$ ed $x_j$ sono statisticamente indipendenti allora $\sigma_{ij} = 0$, poiché il cambiamento di una non modifica l'altra;
- se $\sigma_{ij} = 0\ \forall i \neq j$ allora $p(\mathbf{x})$ risulta essere il prodotto delle densità normali univariate di ogni variabile che compone $\mathbf{x}$;
- possiamo rimodellare la densità secondo le nostre necessità: data una densità $p(\mathbf{x}) \approx N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ e una matrice $A$ di trasformazione $d \times k$, definiamo $\mathbf{y} = A^\top \mathbf{x}$ e otteniamo che

  $$p(\mathbf{y}) \approx N(A^\top \boldsymbol{\mu}, A^\top \boldsymbol{\Sigma} A)$$

  - un caso particolare di questa trasformazione è l'utilizzo di un vettore $\mathbf{a}$ $d \times 1$, che ci fa ottenere $y = \mathbf{a}^\top \mathbf{x}$, ovvero uno scalare che rappresenta la proiezione di $\mathbf{x}$ su una linea in direzione di $\mathbf{a}$; in questo modo $\mathbf{a}^\top \boldsymbol{\Sigma} \mathbf{a}$ rappresenta la varianza di $\mathbf{x}$ su $\mathbf{a}$;
  - un altro caso particolare è la ***trasformazione sbiancante*** (*whitening transform*), ovvero quando dopo l'applicazione di una certa matrice $A_W$ si ottiene una distribuzione per $\mathbf{y}$ con matrice di covarianza $\mathbb{I}$, il che rende praticamente l'input $\mathbf{y}$ rumore bianco, dato che ogni variabile ha varianza 1 e non è correlata con le altre ($\sigma_{ij} = 0\ \forall i \neq j$). Questa trasformazione è possibile per infinite matrici $A_W$, ma in particolare descriviamo tre approcci standard

    **sbiancamento di Mahalanobis o ZCA:** dove viene scelta $A_W = \boldsymbol{\Sigma}^{-1/2}$;

    **sbiancamento di Cholesky:** dove viene usata la decomposizione di Cholesky di $\boldsymbol{\Sigma}^{-1}$;

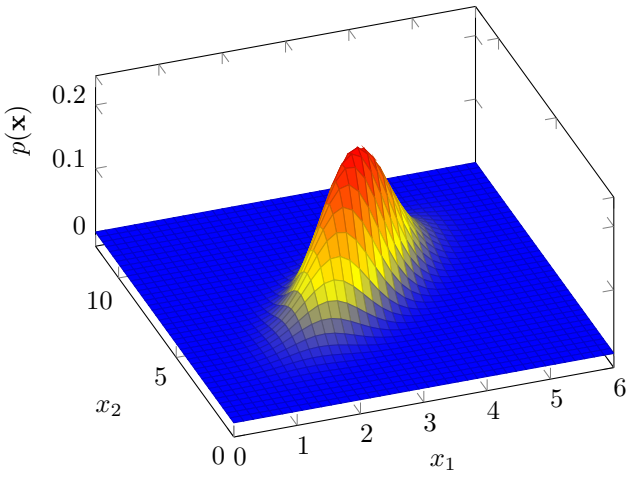
**sbiancamento PCA:** dove viene usata la Principal Component Analysis di $\mathbf{\Sigma}$, ovvero abbiamo $A_W = \Phi\Lambda^{-1/2}$, dove $\Phi$ sono gli autovettori ortonormali di $\mathbf{\Sigma}$ in colonna mentre $\Lambda$ è la matrice diagonale dei corrispondenti autovalori.

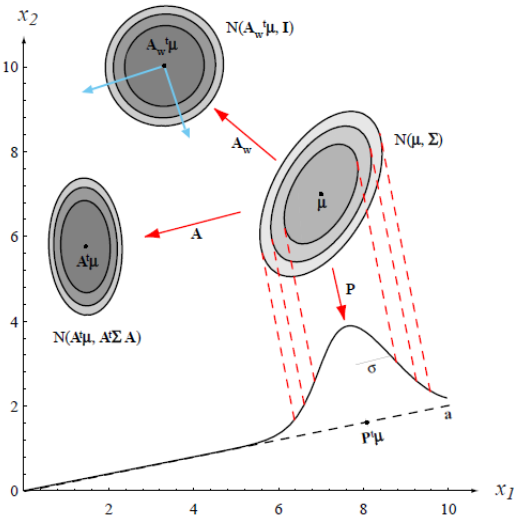
Con questa trasformazione abbiamo anche un preciso significato geometrico: la distribuzione $d$-dimensionale $N(\boldsymbol{\mu}, \mathbf{\Sigma})$, i cui iperellissoidi sono disposti sugli assi identificati dalle colonne di $\Phi$ e di lunghezza $\Lambda$, viene trasformata in una nuova distribuzione $d$-dimensionale $N(A_W^\top \boldsymbol{\mu}, A_W^\top \mathbf{\Sigma} A_W)$, in cui gli assi principali degli iperellissoidi sono i versori canonici, ovvero $e_i \ \forall i \in [1, d]$, che sono di lunghezza ovviamente 1 (la nuova matrice di covarianza è appunto $\mathbb{I}$); gli iperellissoidi di una distribuzione gaussiana (ipercampana $d$-dimensionale) sono quei luoghi geometrici dei punti tali che rimane costante la ***distanza di Mahalanobis***

$$D_M(x) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

che è la distanza di $\mathbf{x}$ da $\boldsymbol{\mu}$ (informalmente, sono tutte le "fette" di campana che si ottengono sezionandola con un piano parallelo al piano $x_1 x_2$).

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \boldsymbol{\mu} = \begin{bmatrix} 3 \\ 6 \end{bmatrix} \quad \mathbf{\Sigma} = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 2 \end{bmatrix}$$




### 2.4.2 Funzioni discriminanti con le gaussiane normali

Ora che abbiamo lo strumento delle gaussiane, descriviamo un classificatore in cui le funzioni discriminanti sono nella forma

$$g_i(\mathbf{x}) = p(\mathbf{x} \,|\, \omega_i) P(\omega_i) \equiv \ln\big(p(\mathbf{x} \,|\, \omega_i) P(\omega_i)\big) = \ln p(\mathbf{x} \,|\, \omega_i) + \ln P(\omega_i)$$

e sostituiamo la densità $p(\mathbf{x} \,|\, \omega_i)$ con una gaussiana multivariata, ottenendo

$$\begin{aligned} g_i(\mathbf{x}) &= \ln\Big(\tfrac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma}_i)}} \exp\big\{-\tfrac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\big\}\Big) + \ln P(\omega_i) \\ &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln\det(\mathbf{\Sigma}_i) + \ln P(\omega_i) \\ &\equiv -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\ln\det(\mathbf{\Sigma}_i) + \ln P(\omega_i) \end{aligned}$$

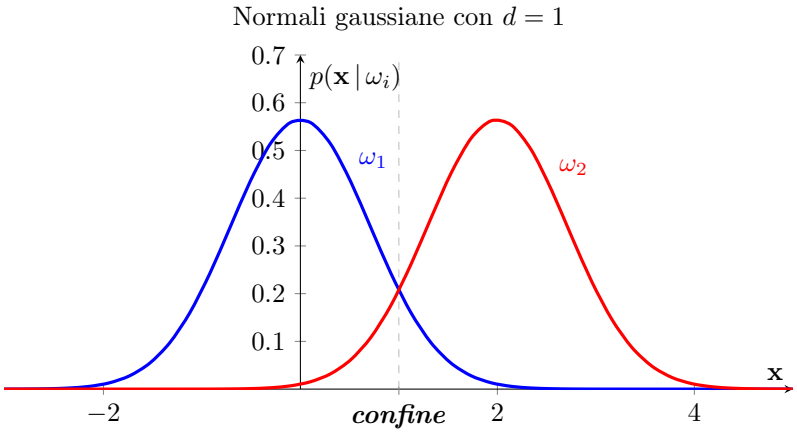
Ora consideriamo alcuni casi interessanti:

**varianza costante:** tutte le feature sono statisticamente indipendenti e ogni classe ha la stessa varianza $\sigma^2$, quindi $\sigma_{ij} = 0 \ \forall i \neq j$ e $\sigma_{ij} = \sigma^2 \ \forall i = j$, ovvero $\mathbf{\Sigma}_i = \sigma^2 \mathbb{I} \ \forall i$. In questo caso possiamo semplificare la formula vista sopra eliminando $-\frac{1}{2}\ln\det(\mathbf{\Sigma}_i)$ poiché sempre costante e compattando il prodotto

matriciale, eliminando successivamente $-\frac{1}{2\sigma^2}\mathbf{x}^\top\mathbf{x}$ poiché anch'esso sempre costante

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^\top\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\ln\det(\boldsymbol{\Sigma}_i) + \ln P(\omega_i) \\ &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^\top(\sigma^2)^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) + \ln P(\omega_i) \\ &= -\frac{1}{2\sigma^2}(\mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_i^\top\mathbf{x} + \boldsymbol{\mu}_i^\top\boldsymbol{\mu}_i) + \ln P(\omega_i) \\ &\equiv \frac{1}{\sigma^2}\boldsymbol{\mu}_i^\top\mathbf{x} - \frac{1}{2\sigma^2}\boldsymbol{\mu}_i^\top\boldsymbol{\mu}_i + \ln P(\omega_i) \\ &= \mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0} \end{aligned}$$

dove $\mathbf{w}_i = \frac{1}{\sigma^2}\boldsymbol{\mu}_i$ e $\mathrm{w}_{i0} = -\frac{1}{2\sigma^2}\boldsymbol{\mu}_i^\top\boldsymbol{\mu}_i + \ln P(\omega_i)$ rappresenta la ***soglia*** (o ***bias***) per l'$i$-esima classe.
Notiamo che la forma $\mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0}$ è lineare, di conseguenza un classificatore che usa funzioni discriminanti lineari è detto ***linear machine***.




Come intuitivamente possiamo vedere dal grafico, i confini di decisione sono dati dalle intersezioni delle campane, ovvero quando $g_i(\mathbf{x}) = g_j(\mathbf{x})$, e semplificando si ottiene

$$\mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0} = \mathbf{w}_j^\top\mathbf{x} + \mathrm{w}_{j0}$$

$$\frac{1}{\sigma^2}\boldsymbol{\mu}_i^\top\mathbf{x} - \frac{1}{2\sigma^2}\boldsymbol{\mu}_i^\top\boldsymbol{\mu}_i + \ln P(\omega_i) = \frac{1}{\sigma^2}\boldsymbol{\mu}_j^\top\mathbf{x} - \frac{1}{2\sigma^2}\boldsymbol{\mu}_j^\top\boldsymbol{\mu}_j + \ln P(\omega_j)$$

$$(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_i^\top\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^\top\boldsymbol{\mu}_j) + \sigma^2\big(\ln P(\omega_i) - \ln P(\omega_j)\big) = 0$$

$$(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_i^\top\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^\top\boldsymbol{\mu}_i + \boldsymbol{\mu}_i^\top\boldsymbol{\mu}_j - \boldsymbol{\mu}_j^\top\boldsymbol{\mu}_j) + \sigma^2\ln\frac{P(\omega_i)}{P(\omega_j)} = 0$$

$$(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top\left(\mathbf{x} - \left(\frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \frac{\sigma^2}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2}\ln\frac{P(\omega_i)}{P(\omega_j)}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\right)\right) = 0$$

$$\mathbf{w}^\top(\mathbf{x} - \mathbf{x}_0) = 0$$

("non vedi che è una retta" - cit. ma questa volta è un iperpiano) dove

$$\mathbf{w} = \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \qquad \mathbf{x}_0 = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \frac{\sigma^2}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2}\ln\frac{P(\omega_i)}{P(\omega_j)}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$$

Notiamo che se $\sigma^2 \ll \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2$ allora il confine è esattamente a metà tra le due gaussiane in quanto insensibile ai prior poiché la seconda parte di $\mathbf{x}_0$ è ignorabile; analogamente, se i due prior sono uguali, ovvero se $P(\omega_i) = P(\omega_j)$ otteniamo lo stesso comportamento, che determina un ***classificatore a distanza minima***.

Con queste funzioni discriminanti quindi abbiamo un comportamento interessante: le linee di confine sono date da un iperpiano $\mathbf{w}^\top(\mathbf{x} - \mathbf{x}_0) = 0$ ortogonale a $\mathbf{w}$ e passante per $\mathbf{x}_0$, e a parità di varianza il prior maggiore determina la classificazione (anche abbastanza intuitivamente, *assegna $\boldsymbol{x}$ alla classe la cui media $\boldsymbol{\mu}_i$ è più vicina*).

**covarianza costante:** le feature non sono statisticamente indipendenti e ogni classe ha una propria varianza, $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}\ \forall i$. Possiamo nuovamente semplificare la formula vista sopra eliminando $-\frac{1}{2}\ln\det(\boldsymbol{\Sigma}_i)$ poiché sempre costante e compattando il prodotto matriciale, eliminando successivamente $-\frac{1}{2}\mathbf{x}^\top\boldsymbol{\Sigma}^{-1}\mathbf{x}$ poiché anch'esso sempre costante

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^\top\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\ln\det(\boldsymbol{\Sigma}_i) + \ln P(\omega_i) \\ &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) + \ln P(\omega_i) \\ &= -\frac{1}{2}(\mathbf{x}^\top\boldsymbol{\Sigma}^{-1}\mathbf{x} - 2\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i) + \ln P(\omega_i) \\ &= \mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0} \end{aligned}$$

dove $\mathbf{w}_i = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i$ e $\mathrm{w}_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \ln P(\omega_i)$ rappresenta al solito la *soglia* per l'$i$-esima classe. Nuovamente la forma $\mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0}$ è lineare e quindi nuovamente i confini di decisione saranno iperpiani.

Se le regioni di decisione $\mathcal{R}_i$ e $\mathcal{R}_j$ sono contigue allora il confine tra esse è dato nuovamente da $g_i(\mathbf{x}) = g_j(\mathbf{x})$, ovvero quando

$$\mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0} = \mathbf{w}_j^\top\mathbf{x} + \mathrm{w}_{j0}$$

$$\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \ln P(\omega_i) = \boldsymbol{\mu}_j^\top\boldsymbol{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_j^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j + \ln P(\omega_j)$$

$$(\boldsymbol{\mu}_i^\top - \boldsymbol{\mu}_j^\top)\boldsymbol{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j) + \ln\frac{P(\omega_i)}{P(\omega_j)} = 0$$

$$\big(\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\big)^\top\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i + \boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j - \boldsymbol{\mu}_j^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j) + \ln\frac{P(\omega_i)}{P(\omega_j)} = 0$$

$$\big(\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\big)^\top\left(\mathbf{x} - \left(\frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \frac{1}{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)}\ln\frac{P(\omega_i)}{P(\omega_j)}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\right)\right) = 0$$

$$\mathbf{w}^\top(\mathbf{x} - \mathbf{x}_0) = 0$$

dove

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) \qquad \mathbf{x}_0 = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \frac{1}{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)}\ln\frac{P(\omega_i)}{P(\omega_j)}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$$

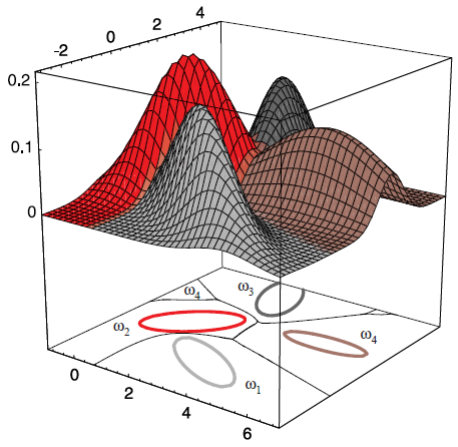
Le linee di confine sono date da un iperpiano $\mathbf{w}^\top(\mathbf{x} - \mathbf{x}_0) = 0$ ma ora non sono più ortogonali a $\mathbf{w}$ poiché compare $\boldsymbol{\Sigma}^{-1}$, pur rimanendo passanti per $\mathbf{x}_0$; nuovamente, quando i prior sono uguali allora $\mathbf{x}_0$ si trova in mezzo alle medie $\boldsymbol{\mu}_i$ e $\boldsymbol{\mu}_j$.

**covarianza arbitraria:** quando non possiamo semplificare in nessun modo come fatto sopra siamo nel caso di matrici $\boldsymbol{\Sigma}_i$ tutte analiticamente diverse tra loro e quindi dobbiamo studiare la formula così com'è, ma possiamo comunque ricondurci a una forma lineare

$$g_i(\mathbf{x}) = \mathbf{x}^\top\mathbf{W}_i\mathbf{x} + \mathbf{w}_i^\top\mathbf{x} + \mathrm{w}_{i0}$$

dove

$$\mathbf{W}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1} \qquad \mathbf{w}_i = \boldsymbol{\Sigma}_i^{-1}\mu_i \qquad \mathrm{w}_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\ln\det(\boldsymbol{\Sigma}_i) + \ln P(\omega_i)$$



# 3 | Metodi parametrici

## 3.1 Stima dei parametri

Abbiamo visto in precedenza che per costruire un classificatore Bayesiano (che utilizza la regola di decisione di Bayes) è necessario conoscere le prior $P(\omega_i)$ e le posterior $p(\mathbf{x} \mid \omega_i)$. Siccome le performance del classificatore dipendono fortemente da questi parametri è necessario conoscerli con una bontà molto alta, ma nei casi reali non si conosco praticamente mai tutte queste informazioni: spesso si ha semplicemente una vaga *conoscenza del problema*, dalla quale stimare delle prior "decenti", e si possiedono dei *pattern rappresentativi* del problema, detti ***training data***, usati per addestrare il classificatore, ma che spesso non sono in numero sufficiente. La parte più complessa rimane la stima delle posterior, per la quale possiamo adottare due strategie:

- stima della funzione sconosciuta $p(\mathbf{x} \mid \omega_i)$, ovvero capire qual è la natura del fenomeno che studiamo;
- stima dei parametri sconosciuti della funzione conosciuta $p(\mathbf{x} \mid \omega_i)$, ovvero assumere che il fenomeno si comporti secondo un certo andamento, ad esempio $p(\mathbf{x} \mid \omega_i) \approx N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, e stimarne i parametri, nell'esempio il vettore $\theta_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$.

Questo secondo punto è complesso ma comunque più semplice rispetto al primo e rappresenta un problema classico della statistica. L'iter è il seguente:

1. stima dei parametri dai training data;
2. costruzione di un classificatore Bayesiano su tali stime.

## 3.2 Stima dei prior

Dato un insieme $D$ di $n$ dati di training etichettati (ovvero per ogni campione nel set conosco la classe $\omega_i$ a cui appartiene), le varie prior sono calcolabili come $P(\omega_i) = \frac{n_i}{n}$ dove $n_i$ è il numero di campioni etichettati con la classe $\omega_i$. Questo è un approccio di ***supervised learning*** in quanto è necessario che un operatore umano etichetti i campioni.

## 3.3 Stima dei posterior

Come detto in precedenza, assumiamo che la funzione per $p(\mathbf{x} \mid \omega_i)$ sia nota e quindi dato un insieme $D$ di $n$ dati di training etichettati ci concentriamo sulla stima dei parametri. Possiamo adottare due strategie, computazionalmente differenti ma qualitativamente simili.

### 3.3.1 Maximum Likelyhood

Assume che i parametri siano fissati ma sconosciuti e quindi la migliore stima dei loro valori e quella che massimizza le probabilità di riottenere i dati di training. Ponendo in ipotesi (molto forte) che i pattern nel set $D$ siano *indipendenti e identicamente distribuiti* (o anche ***i.i.d***), estraiamo la probabilità congiunta e, dato che i pattern sono indipendenti, avremo la produttoria

$$p(D \mid \boldsymbol{\theta}) = \prod_{k=1}^{n} p(\mathbf{x}_k \mid \boldsymbol{\theta})$$

dove $\mathbf{x}_k$ è il $k$-esimo pattern. Chiamiamo $p(D \mid \boldsymbol{\theta})$ la ***likelihood*** di $\boldsymbol{\theta}$ (fissato ma sconosciuto) rispetto al set di pattern $D$ e la stima di Maximum Likelihood di $\boldsymbol{\theta}$ è, per definizione, il valore $\hat{\boldsymbol{\theta}}$ che massimizza $p(D \mid \boldsymbol{\theta})$.

Analiticamente, è più semplice lavorare con la ***log-likelihood*** $l(\boldsymbol{\theta})$, in quanto la produttoria diventa una sommatoria

$$l(\boldsymbol{\theta}) \equiv \ln p(D \mid \boldsymbol{\theta}) = \sum_{k=1}^{n} \ln p(\mathbf{x}_k \mid \boldsymbol{\theta})$$

quindi, dati $p$ parametri da stimare, esplicitiamo il vettore $\boldsymbol{\theta}$ ed il suo operatore gradiente

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} \qquad \nabla_{\boldsymbol{\theta}} \equiv \begin{bmatrix} \frac{\partial}{\theta_1} \\ \vdots \\ \frac{\partial}{\theta_p} \end{bmatrix}$$

per poi passare allo svolgimento dei calcoli: lo scopo è ottenere $\hat{\boldsymbol{\theta}}$ che massimizza $p(D \,|\, \boldsymbol{\theta})$, ma equivalentemente possiamo usare la sua forma logaritmica, quindi cerchiamo

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} l(\boldsymbol{\theta})$$

e per farlo, come al solito, poniamo il gradiente della funzione a 0, quindi $\hat{\boldsymbol{\theta}}$ sarà il valore di $\boldsymbol{\theta}$ tale per cui

$$\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = 0$$

facendo attenzione a controllare poi che il valore trovato sia un massimo *globale*, e quindi non massimo locale, un flesso o peggio ancora un minimo.

**Caso gaussiano ($\boldsymbol{\mu}$ ignota)** Assumiamo $p(\mathbf{x} \,|\, \omega_i) \approx N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ e assumiamo che solo la media $\boldsymbol{\mu}$ sia sconosciuta. Consideriamo il punto campione $\mathbf{x}_k$, riscriviamo

$$\ln p(\mathbf{x}_k \,|\, \boldsymbol{\mu}) = -\frac{1}{2}(\mathbf{x}_k - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu}) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln\det(\boldsymbol{\Sigma})$$

e finalmente applichiamo il gradiente, che cancella le varie costanti

$$\nabla_{\boldsymbol{\mu}} \ln p(\mathbf{x}_k \,|\, \boldsymbol{\mu}) = \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})$$

Ora abbiamo tutto il necessario per risolvere l'equazione della Maximum Likelihood: dato che l'unico parametro da stimare è $\boldsymbol{\mu}$ avremo che $\boldsymbol{\theta} \equiv \boldsymbol{\mu}$ e quindi cerchiamo il valore $\hat{\boldsymbol{\mu}}$ tale da soddisfare $\nabla_{\boldsymbol{\mu}} l(\boldsymbol{\mu}) = 0$. Svolgiamo i calcoli

$$\nabla_{\boldsymbol{\mu}} l(\hat{\boldsymbol{\mu}}) = \sum_{k=1}^{n} \nabla_{\boldsymbol{\mu}} \ln p(\mathbf{x}_k \,|\, \hat{\boldsymbol{\mu}}) = \sum_{k=1}^{n} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}) = 0 \qquad \Longrightarrow \qquad \hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}_k$$

Come possiamo apprezzare, il valore medio ottimale seguendo l'approccio di Maximum Likeihood è proprio la media dei campioni nel training set, ovvero l'operazione più intuitiva da adottare. Questo valore $\hat{\boldsymbol{\mu}}$ viene detto ***sample mean*** e viene a volte scritto come $\hat{\boldsymbol{\mu}}_n$ a indicare che tale media è dipendente dal numero dei campioni.

**Caso gaussiano ($\boldsymbol{\mu}$ e $\boldsymbol{\Sigma}$ ignote)** Assumiamo nuovamente $p(\mathbf{x} \,|\, \omega_i) \approx N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ ma ora sia la media $\boldsymbol{\mu}$ che la covarianza $\boldsymbol{\Sigma}$ sono sconosciute, come nei casi reali d'altronde. Consideriamo i casi per complessità crescente:

**univariata:** $\boldsymbol{\theta} = (\mu, \sigma^2)$ e in questo caso abbiamo

$$\ln p(x_k \,|\, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2}(x_k - \mu)^2 - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln\sigma^2$$

la cui derivata è

$$\nabla_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \ln p(x_k \,|\, \boldsymbol{\theta}) = \begin{bmatrix} \frac{1}{\sigma^2}(x_k - \mu) \\ -\frac{1}{2\sigma^2} + \frac{(x_k-\mu)^2}{2(\sigma^2)^2} \end{bmatrix}$$

e quando posta a 0 per la Maximum Likelihood in modo da trovare $\hat{\mu}$ e $\hat{\sigma}^2$ ci da due equazioni

$$\sum_{k=1}^{n} \frac{1}{\hat{\sigma}^2}(x_k - \hat{\mu}) = 0 \qquad \Longrightarrow \qquad \hat{\mu} = \frac{1}{n}\sum_{k=1}^{n} x_k$$

$$\sum_{k=1}^{n} -\frac{1}{2\hat{\sigma}^2} + \frac{(x_k-\mu)^2}{2(\hat{\sigma}^2)^2} = 0 \qquad \Longrightarrow \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n}(x_k - \hat{\mu})^2$$

Notiamo con piacere come anche la formula della covarianza seguendo l'appoccio di Maximum Likelihood sia proprio la sua classica definizione in statistica.

**multivariata:** $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ e in questo caso, in modo analogo, otteniamo

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}_k \qquad \hat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{k=1}^{n}(\mathbf{x}_k - \hat{\boldsymbol{\mu}})(\mathbf{x}_k - \hat{\boldsymbol{\mu}})^\top$$

**Bias ed errori** È necessario prestare attenzione al fatto che la stima della covarianza ottenuta ha un *bias*, ovvero uno sbilanciamento, infatti

$$\mathbb{E}\left[\frac{1}{n}\sum_{k=1}^{n}(x_k - \hat{\mu})^2\right] = \frac{n-1}{n}\sigma^2 \neq \sigma^2$$

e quindi per ottenere la ***sample covariance*** è necessario aggiustare la stima

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \hat{\mu})^2 \qquad \mathbf{C} = \frac{1}{n-1}\sum_{k=1}^{n}(\mathbf{x}_k - \hat{\boldsymbol{\mu}})(\mathbf{x}_k - \hat{\boldsymbol{\mu}})^\top$$

Ovviamente al crescere di $n$ il bias diventa infimo, ma è bene notare che abbiamo *due* matrici di covarianza tra le quali scegliere, $\hat{\boldsymbol{\Sigma}}$ oppure $\mathbf{C}$, ma nessuna di esse è quella "corretta", sono semplicemente diverse. Quando un classificatore è bilanciato (cioè senza bias) per ogni distribuzione, allora si dice *assolutamente bilanciato*, mentre se tende ad essere bilanciato al crescere dei campioni allora si dice *asintoticamente bilanciato*; è evidente che $\hat{\boldsymbol{\Sigma}} = \frac{n-1}{n}\mathbf{C}$ e quindi quando usiamo $\hat{\boldsymbol{\Sigma}}$ il classificatore rientrerà nella seconda definizione.

Un classificatore Bayesiano che adotta l'approccio di Maximum Likelihood darà ottimi risultati, a patto che il modello stimato sia accurato; in caso contrario, la classificazione sarà pessima a causa dell'errore nel modello. Non abbiamo mai parlato di *confidenza* o *affidabilità* della stima dei parametri effettuata e inoltre nel caso in cui volessimo in qualche modo alzarla, aggiungendo più campioni nel training set, dovremmo ricalcolare tutti i parametri daccapo. Questo significa che se avessimo un training set che cresce nel tempo, dovremmo svolgere numerosi raffinamenti, ad un costo spesso importante.

### 3.3.2 Approccio Bayesiano

Assume che i parametri siano variabili aleatorie aventi determinate prior. A questo punto vengono usati i dati di training come osservazioni per calcolare le posterior, raffinando tale stima al crescere del numero dei campioni utilizzati: questo processo è detto ***Bayesian learning***. In questo caso usiamo il training set $D$ per trasformare la prior $p(\boldsymbol{\theta})$ nella posterior $p(\boldsymbol{\theta} \mid D)$, e quindi durante la classificazione useremo la posterior $P(\omega_i \mid \mathbf{x}, D)$ poiché appunto la variabile $\mathbf{x}$ ora segue una distribuzione che si basa su parametri aleatori $\boldsymbol{\theta}$ basati su $D$.

Assumiamo che $D = D_1 \cup \cdots \cup D_c$ dove $c$ è il numero di classi e che $D_i$ dia informazioni rilevanti solo su $\omega_i$; assumiamo inoltre che le prior delle classi siano note poiché facilmente calcolabili e che non dipendano dal training set, esprimibile come $P(\omega_i \mid D) = P(\omega_i)$. Applicando la formula di Bayes otteniamo

$$\begin{aligned} P(\omega_i \mid \mathbf{x}, D) &= \frac{p(\mathbf{x} \mid w_i, D)\, P(\omega_i \mid D)}{\sum_{j=1}^{c} p(\mathbf{x} \mid w_j, D)\, P(\omega_j \mid D)} \\ &= \frac{p(\mathbf{x} \mid w_i, D_i)\, P(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x} \mid w_j, D_j)\, P(\omega_j)} \end{aligned}$$

A questo punto ci siamo ridotti a $c$ problemi del tipo: usa il set $D$, composto di campioni estratti indipendentemente secondo la distribuzione $p(\mathbf{x})$, per determinare $p(\mathbf{x} \mid D)$. Siccome $p(\mathbf{x})$ è ignota, possiamo al massimo assumerne la forma parametrica, i cui parametri $\boldsymbol{\theta}$ sono però ignoti; ciò significa che $p(\mathbf{x} \mid \boldsymbol{\theta})$ è completamente nota. Qualsiasi informazione (ottenuta manualmente) riguardo $\boldsymbol{\theta}$ prima della misurazione dei campioni è contenuta in $p(\boldsymbol{\theta})$ mentre dopo aver osservato il set $D$ avremo $p(\boldsymbol{\theta} \mid D)$, una distribuzione che speriamo essere centrata attorno ai veri valori di $\boldsymbol{\theta}$. Al solito abbiamo

$$\begin{aligned} p(\mathbf{x} \mid D) &= \int p(\mathbf{x}, \boldsymbol{\theta} \mid D)\, \mathrm{d}\mathbf{x} \\ &= \int p(\mathbf{x} \mid \boldsymbol{\theta}, D)\, p(\boldsymbol{\theta} \mid D)\, \mathrm{d}\boldsymbol{\theta} \\ &= \int p(\mathbf{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid D)\, \mathrm{d}\boldsymbol{\theta} \end{aligned}$$

# 4 | Metodi non-parametrici

Nei metodi parametrici troviamo assumevamo di conoscere la forma della distribuzione (spesso abbiamo usato la gaussiana) e ci prefiggevamo l'obiettivo di stimarne i parametri (nel nostro caso, la media e la matrice di covarianza). Nei problemi di riconoscimento però non abbiamo mai a che fare con funzioni di densità *unimodali* (ovvero con un solo massimo, come la gaussiana) ma quasi esclusivamente con funzioni multimodali, che ovviamente non possono essere espresse appieno con l'approssimazione una singola gaussiana. I metodi non-parametrici invece non assumono nulla e non fanno altro che stimare la forma della distribuzione direttamente dai dati, senza un modello noto a priori; per completezza citiamo anche i metodi ***semi-parametrici***, nei quali si assume che esista una famiglia molto ampia di funzioni di densità (un esempio sono le reti neurali).

Il problema alla base è riuscire a stimare la quantità

$$\hat{p}_i(\mathbf{x}) \approx p(\mathbf{x} \,|\, \omega_i)$$

così da utilizzarla al solito in un classificatore Bayesiano. Come abbiamo detto prima, le funzioni di densità sono praticamente sempre multimodali, e quindi andremo ad approssimarle componendo tra loro funzioni semplici (dette ***funzioni potenziali***) estratte dai campioni.

## 4.1 Funzioni potenziali

Sia $\gamma(\mathbf{x}, \mathbf{y}_j)$ una funzione potenziale per un campione generico $\mathbf{y}_j$ della classe $i$ secondo la distribuzione $\mathbf{x}$; possiamo scrivere che

$$\hat{p}_i(\mathbf{x}) = \frac{1}{N_i} \sum_{j=1}^{N_i} \gamma(\mathbf{x}, \mathbf{y}_j)$$

dove $N_i$ è il numero di campioni della classe $i$. Affinché la stima sia buona dobbiamo trovare delle adeguate funzioni $\gamma$, le quali devono soddisfare

- $\gamma(\mathbf{x}, \mathbf{y}_j) \geq 0$ e continua;
- $\arg\max_{\mathbf{x}} \gamma(\mathbf{x}, \mathbf{y}_j) = \mathbf{y}_j$, ovvero $\gamma(\mathbf{x}, \mathbf{y}_j)$ deve essere massima per $\mathbf{x} = \mathbf{y}_j$;
- $\gamma(\mathbf{x}, \mathbf{y}_1) \cong \gamma(\mathbf{x}, \mathbf{y}_2)$ se $|\mathbf{y}_2 - \mathbf{y}_1| < \varepsilon$, ovvero se i due vettori $\mathbf{y}_1$ e $\mathbf{y}_2$ sono abbastanza vicini (in questo modo si garantisce la continuità di $\hat{p}$);
- $\gamma(\mathbf{x}, \mathbf{y}_j) \cong 0$ se $\mathbf{x}$ è molto lontana da $\mathbf{y}_j$ (di conseguenza alla precedente);
- $\int_{-\infty}^{+\infty} \gamma(\mathbf{x}, \mathbf{y}_j)\, \mathrm{d}\mathbf{x} = 1$, ovvero non deve avere volume infinito.

Da ora useremo come notazione semplicemente $\gamma(\mathbf{u})$, dove in riferimento a $\gamma(\mathbf{x}, \mathbf{y}_j)$ avremo invece $\mathbf{u} = \mathbf{x} - \mathbf{y}_j$, più compatto ed egualmente espressivo per le funzioni che andremo ad usare. Esempi di funzioni (unidimensionali ma estendibili a più dimensioni) che soddisfano le proprietà sopra sono

**rettangolo:** $\gamma(\mathbf{u}) = \begin{cases} \frac{1}{2} & \text{se } \|\mathbf{u}\| \leq \frac{1}{2} \\ 0 & \text{altrimenti} \end{cases}$;

**triangolo:** $\gamma(\mathbf{u}) = \begin{cases} 1 - \|\mathbf{u}\| & \text{se } \|\mathbf{u}\| \leq 1 \\ 0 & \text{altrimenti} \end{cases}$;

**gaussiana:** $\gamma(\mathbf{u}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\|\mathbf{u}\|^2}{2}}$;

**esponenziale decrescente:** $\gamma(\mathbf{u}) = \frac{1}{2} e^{-\|\mathbf{u}\|}$;

**distibuzione di Cauchy:** $\gamma(\mathbf{u}) = \frac{1}{\pi(1+\|\mathbf{u}\|^2)}$;

**funzione di tipo** sin**:** $\gamma(\mathbf{u}) = \frac{1}{2\pi} \left\| \frac{\sin(\frac{\mathbf{u}}{2})}{\frac{\mathbf{u}}{2}} \right\|^2$.

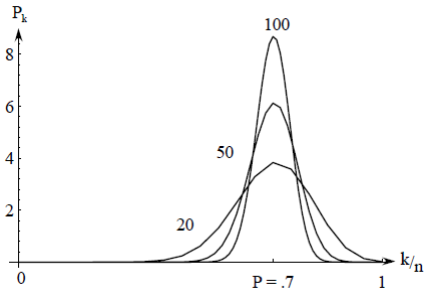
Con un abuso di notazione, consideriamo ora $p(\mathbf{x})$ come la densità di una singola classe, per poi alla fine ritornare alla usuale notazione $p_i(\mathbf{x})$ per considerarle nell'insieme: l'idea alla base è che la probabilità $P$ che un campione $x$ rientri in una regione $\mathcal{R}$ è al solito

$$P = \int_{\mathcal{R}} p(\mathbf{x})\, \mathrm{d}\mathbf{x}$$

e quindi possiamo sfruttare questo per stimare $p(\mathbf{x})$ tramite $\hat{p}(\mathbf{x})$ a partire da $P$ (che in pratica è $p(\mathbf{x})$ in versione *smoothed*). Supponiamo di avere $n$ campioni $\mathbf{y}_j$ indipendenti e identicamente distribuiti secondo $p(\mathbf{x})$; la probabilità $P_k$ che esattamente $k$ campioni rientrino in una regione $\mathcal{R}$ (che per precisazione non è l'intero spazio dei campioni) è

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

il cui valore atteso per $k$ è $\mathbb{E}[k] = nP$ (questa è la *legge binomiale*, nella quale notiamo $P^k$ probabilità di avere $k$ campioni in $\mathcal{R}$, $P^{n-k}$ probabilità che tutti i restanti campioni non siano in $\mathcal{R}$ e il binomiale, a indicare che la disposizione dei campioni è ininfluente).




Una buona stima di questa probabilità è quindi $P = k/n$, e se assumiamo $p(\mathbf{x})$ sia continua e $\mathcal{R}$ talmente piccola da vedere praticamente invariata $p(\mathbf{x})$ allora possiamo scrivere

$$P = \int_{\mathcal{R}} p(\mathbf{x})\,\mathrm{d}\mathbf{x} \cong p(\mathbf{x})V$$

dove $V$ è il volume di $\mathcal{R}$; in questo modo otteniamo $p(\mathbf{x}) \cong \hat{p}(\mathbf{x}) = \frac{k/n}{V}$. Studiamo i casi al limite per $n$ fissato

- $\lim_{V\to 0, k=0} p(\mathbf{x}) = 0$, ovvero non ci sono campioni in $\mathcal{R}$;
- $\lim_{V\to 0, k\neq 0} p(\mathbf{x}) = \infty$, ovvero per quanto piccolo il volume, c'è sempre un campione, quindi ci sono infiniti campioni e la probabilità è infinita.

In pratica, dato che il numero di campioni è sempre limitato, non possiamo considerare volumi $V$ infinitesimi e per questo avremo una certa varianza nella stima di $k/n$ (e conseguentemente dovremo mediare le densità ottenute con tali stime per ottenere $\hat{p}(\mathbf{x})$ nella sua forma finale). Supponendo di avere un numero illimitato di campioni a disposizione, consideriamo $\mathcal{R}$ come una regione crescente $\mathcal{R}_1, \mathcal{R}_2, \ldots$ in modo che man mano contenga rispettivamente $1, 2, \ldots$ campioni e sempre il campione $\mathbf{x}$; sia $V_n$ il volume di $\mathcal{R}_n$ (ovvero la regione che contiene tutti i campioni), $k_n$ il numero di campioni in $\mathcal{R}_n$ e $\hat{p}_n(\mathbf{x})$ la stima $n$-esima: otteniamo

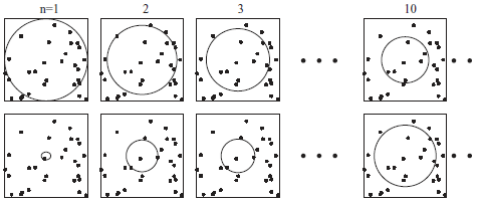
$$\hat{p}_n(\mathbf{x}) = \frac{k_n/n}{V_n}$$

e, dato che vogliamo farla convergere a $p(\mathbf{x})$, dobbiamo avere

- $\lim_{n\to\infty} V_n = 0$;
- $\lim_{n\to\infty} k_n = \infty$;
- $\lim_{n\to\infty} k_n/n = 0$.

Infine, per soddisfare queste condizioni possiamo adottare due metodi

- fissare $k_n$ proporzionale ad $n$, per esempio usando $k_n = \sqrt{n}$, e aumentare il volume $V_n$ finché non include $k_n$ vicini di $\mathbf{x}$. Questo è il metodo denominato ***$k_n$-nearest-neighbor*** o ***KNN***;
- rimpicciolire man mano il volume definendolo inverso proporzionale ad $n$, per esempio usando $V_n = \frac{1}{\sqrt{n}}$, e dimostrando che $\lim_{n\to\infty} \hat{p}_n(\mathbf{x}) = p(\mathbf{x})$. Questo è il metodo denominato ***Parzen windows***.



## 4.2 Parzen windows

Sempre considerando una sola classe, assumiamo $\mathcal{R}_n$ un ipercubo $d$-dimensionale centrato in $\mathbf{x}$ di lato lungo $h_n$, e quindi di volume $V_n = h_n^d$, e vogliamo calcolare il numero $k_n$ di campioni al suo interno. Definiamo la funzione potenziale

$$\gamma(\mathbf{u}) = \begin{cases} 1 & \text{se } |u_i| \leq \frac{1}{2} \quad \forall i = 1, \dots, d \\ 0 & \text{altrimenti} \end{cases}$$

che rappresenta un ipercubo unitario centrato nell'origine e che sarà la nostra ***funzione finestra*** (da qui il nome); da questo notiamo che $\gamma\big(\frac{\mathbf{x}-\mathbf{y}_j}{h_n}\big) = 1$ se $\mathbf{y}_j$ si trova in $\mathcal{R}_n$ (ovvero nel volume $V_n$ centrato in $\mathbf{x}$), ed è 0 altrimenti. Per trovare il numero $k_n$ di campioni in $V_n$ quindi basta semplicemente contarli con

$$k_n = \sum_{j=1}^{n} \gamma\Big(\frac{\mathbf{x} - \mathbf{y}_j}{h_n}\Big)$$

e la stima $\hat{p}_n(\mathbf{x})$ viene ottenuta sostituendo quanto appena trovato

$$\hat{p}_n(\mathbf{x}) = \frac{k_n/n}{V_n} = \frac{1}{n} \sum_{j=1}^{n} \frac{1}{V_n} \gamma\Big(\frac{\mathbf{x} - \mathbf{y}_j}{h_n}\Big)$$

Ovviamente la stima diventa sempre più accurata al crescere del numero di campioni, come richiesto dalle condizioni del metodo Parzen windows, quindi possiamo raffinare la stima scrivendo

$$\hat{p}_n(\mathbf{x}) = \lim_{n \to \infty} \frac{1}{n} \sum_{j=1}^{n} \frac{1}{V_n} \gamma\Big(\frac{\mathbf{x} - \mathbf{y}_j}{h_n}\Big)$$

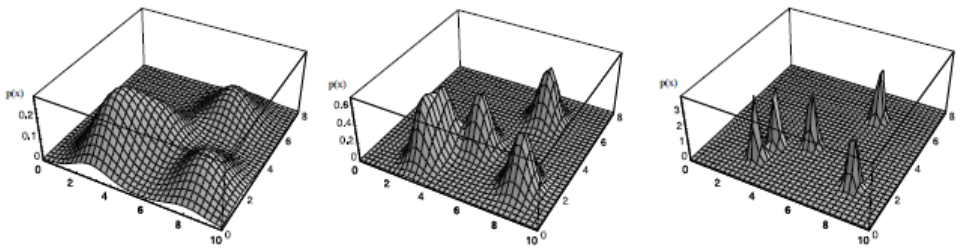
Abbiamo finalmente raggiunto il punto d'incontro, ora possiamo usare questa formula (che ricordiamo usare come funzione potenziale un ipercubo di lato $h_n$) per la singola classe e quindi ne diamo la notazione

$$\hat{p}_i(\mathbf{x}) = \lim_{N_i \to \infty} \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{1}{h_{N_i}^d} \gamma\Big(\frac{\mathbf{x} - \mathbf{y}_j}{h_{N_i}}\Big)$$

dove la $i$ indica al solito la classe $\omega_i$ e invece $n$ viene riscritta come $N_i$, ovvero i campioni appartenenti a $\omega_i$.

Ovviamente la scelta della funzione finestra $\gamma(\mathbf{x})$ e di conseguenza quella della lunghezza $h_{N_i}$ dell'ipercubo determinano la bontà della stima, infatti per $h_{N_i}$ grandi avremo una bassa risoluzione ma per $h_{N_i}$ piccoli avremo una grande variabilità statistica; adottiamo quindi l'euristica $h_{N_i} = \frac{h_1}{N_i}$, dove $h_0$ è un parametro manuale sul quale dovremo fare svariati tentativi per trovare il valore migliore.

**Caso gaussiano** Quando scegliamo $\gamma(\mathbf{x}, \mathbf{y}_j) \approx N(\mathbf{y}_j, \sigma^2 \mathbb{I})$ stiamo usando una campana circolare come funzione finestra; in questo contesto, le funzioni potenziali gaussiane vengono dette *funzioni di Specht*. Per forzare $\sigma^2$ è necessario trasformare le feature del training set in modo che abbiano tutte varianza $\sigma^2$ (per questo usiamo le trasformate sbiancanti, che trasformano $\mathbf{\Sigma} = \mathbb{I}$, e quindi dobbiamo inoltre scalare opportunamente per $\sigma^2$).



Possiamo notare 5 funzioni finestra gaussiane che si modellano man mano, per $\sigma^2 \to 0$, esattamente attorno ai 5 punti usati per l'esempio; nella prima figura sono molto smooth attorno ai punti e identificano solo 3 classi, ma al diminuire della covarianza diventano sempre più precise, fino a raggiungere l'overfitting.

Notiamo che $\sigma^2$ funge da parametro di smoothing (prima era $h_{N_i}$) e va impostato con parsimonia, poiché per $\sigma^2 \to \infty$ la campana sarà così panciuta da dare $\hat{p}(\mathbf{x})$ costante, mentre per $\sigma^2 \to 0$ avremo $\hat{p}(\mathbf{x})$ somma di impulsi di Dirac (overfitting); inoltre, impostare un valore casuale potrebbe sovrapporre troppo le curve. Una

buona euristica per la scelta di $\sigma^2$ è prendere gli $L$ campioni $\mathbf{y}_j$ più vicini al generico campione $\mathbf{y}_i$ e calcolarne la media delle distanze, ovvero

$$\sigma = \frac{1}{L}\sum_{j=1}^{L} \|\mathbf{y}_j - \mathbf{y}_i\|$$

ma qui abbiamo introdotto un nuovo parametro $L$, il numero di vicini, che spesso viene impostato a $L = 0.05\,n$.

Ora vogliamo al solito arrivare alle funzioni discriminanti $g_i(\mathbf{x})$: a partire da $\hat{p}_i(\mathbf{x})$, sviluppando i calcoli

$$\begin{aligned}\hat{p}_i(\mathbf{x}) &= \frac{1}{N_i}\sum_{j=1}^{N_i}\gamma(\mathbf{x},\mathbf{y}_j) = \frac{1}{N_i}\sum_{j=1}^{N_i}\left(\frac{1}{\sqrt{(2\pi)^d\sigma^2}}\exp\left\{-\tfrac{1}{2}\frac{(\mathbf{x}-\mathbf{y}_j)^\top(\mathbf{x}-\mathbf{y}_j)}{\sigma^2}\right\}\right)\\ &= \frac{1}{N_i\sqrt{(2\pi)^d\sigma^2}}e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}}\sum_{j=1}^{N_i}e^{\frac{\mathbf{x}^\top\mathbf{y}_j}{\sigma^2}}\underbrace{e^{-\frac{\|\mathbf{y}_j\|^2}{2\sigma^2}}}_{\text{noto}\,\doteq c_j}\end{aligned}$$

e ora possiamo sostituire $e^{\frac{\mathbf{x}^\top\mathbf{y}_j}{\sigma^2}}$ con lo sviluppo di Taylor $e^{\mathbf{x}} = 1 + \mathbf{x} + \frac{\mathbf{x}^2}{2!} + \frac{\mathbf{x}^3}{3!} + \dots$ nell'equazione

$$\begin{aligned}\hat{p}_i(\mathbf{x}) &= \frac{1}{N_i\sqrt{(2\pi)^d\sigma^2}}e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}}\sum_{j=1}^{N_i}c_j\sum_{k=0}^{r}\frac{\left(\frac{\mathbf{x}^\top\mathbf{y}_j}{\sigma^2}\right)^k}{k!}\\ &= \frac{1}{N_i\sqrt{(2\pi)^d\sigma^2}}e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}}\sum_{j=1}^{N_i}c_j\sum_{k=0}^{r}\frac{1}{\sigma^{2k}k!}(\mathbf{x}^\top\mathbf{y}_j)^k\end{aligned}$$

dove $r$ è la bontà dell'approssimazione di $e^{\mathbf{x}}$. Finalmente possiamo trovare la formula per le funzioni discriminanti (viene già presentata senza i termini comuni)

$$g_i(\mathbf{x}) = \hat{p}_i(\mathbf{x}\,|\,\omega_i)\,\hat{P}(\omega_i) = \frac{1}{N_i}\sum_{j=1}^{N_i}c_{j,i}\sum_{k=0}^{r}\frac{1}{\sigma^{2k}k!}(\mathbf{x}^\top\mathbf{y}_{j,i})^k\,\hat{p}(\omega_i)$$

dove identifichiamo $\mathbf{y}_{j,i}$ con una $i$ per indicare che è il campione $j$-esimo della classe $\omega_i$ (anche $c_{j,i} = e^{-\frac{\|\mathbf{y}_{j,i}\|^2}{2\sigma^2}}$ ha una $i$ per lo stesso motivo). Impostando $r = 1$ abbiamo una formula lineare facilmente calcolabile, con un certo margine di bontà ovviamente; alternativamente abbiamo le seguenti considerazioni:

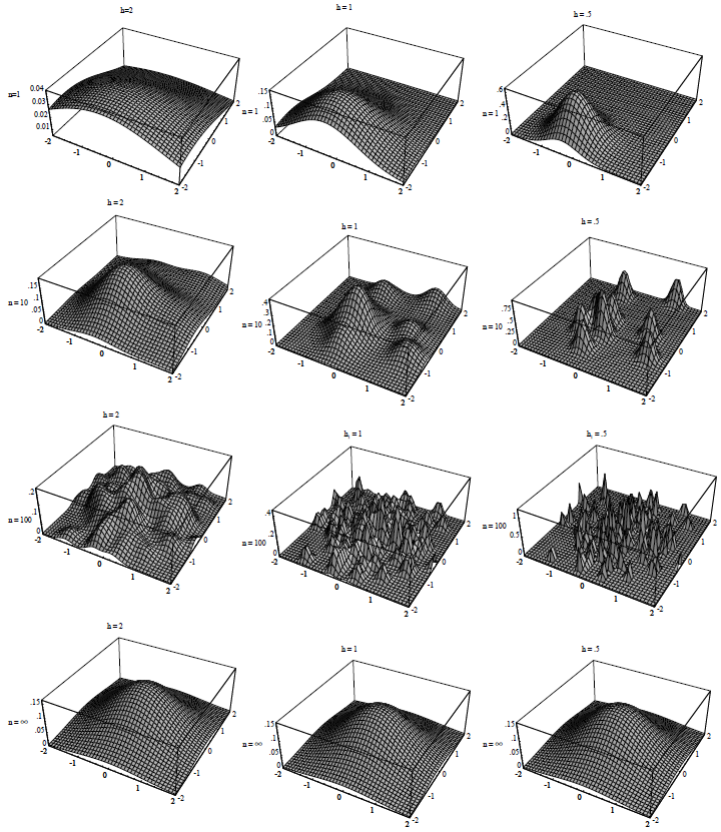
- quando $\sigma \to \infty$ basta una $r$ piccola (in pratica è il ***metodo dei prototipi***);
- quando $\sigma \to 0$ è necessaria una $r$ grande (la funzione discriminante diventa una sommatoria di impulsi di Dirac e in pratica ci riconduciamo al ***KNN***).

## 4.3 Metodo dei prototipi

Un caso particolare del metodo Parzen windows con finestra gaussiana è l'utilizzo di una $\sigma^2$ molto grande, che rende le campane talmente larghe da essere considerabili costanti. Estremizzando questo pensiero, quando $\sigma^2 \to \infty$ possiamo considerare le campane tutte costanti e quindi mantenere solo il loro punto centrale, ovvero il baricentro di quella classe; questo approccio, che prende il nome di ***metodo dei prototipi***, risulta essere molto efficiente quando si usa un classificatore basato su distanza minima (anche detto ***MDC***) per trovare le regioni di decisione. Supponendo che i campioni di una certa classe si concentrino strettamente intorno ad un pattern rappresentativo per la classe stessa, considerando il baricentro $\boldsymbol{m}_i$ di ogni classe e $D(\cdot,\cdot)$ una metrica nello spazio delle features, la regola di decisione applicata dal MDC è scegliere la classe $\omega_i$ se $D(\mathbf{x},\boldsymbol{m}_i) = \min_{j\neq i} D(\mathbf{x},\boldsymbol{m}_j)$, ovvero assegnare $\mathbf{x}$ alla classe con baricentro più vicino. Trovare le funzioni discriminanti risulta essere altrettanto semplice: date due classi $\omega_i$ e $\omega_j$

1. si determinano i baricentri $\boldsymbol{m}_i$ ed $\boldsymbol{m}_j$;
2. si determina la retta $r_{ij}(\mathbf{x})$ che congiunge i due baricentri;
3. si calcola l'iperpiano $g_{ij}(\mathbf{x})$ perpendicolare alla retta $r_{ij}(\mathbf{x})$: tale iperpiano è sbilanciato verso $\boldsymbol{m}_i$ oppure $\boldsymbol{m}_j$ in base ai prior delle due classi (come al solito, più $P(\omega_i)$ è basso più l'iperpiano si avvicina a $\boldsymbol{m}_i$).

I vantaggi immediati di questo metodo sono la semplicità è la bassa complessità spaziale, ma con lo svantaggio che anche un singolo campione mal posizionato (che magari nemmeno esiste nella realtà fisica) potrebbe sbilanciare completamente il baricentro e portare a confini di decisione molto articolati.



## 4.4 k-nearest-neighbors

Come già introdotto nel metodo Parzen windows e raffinato concettualmente nel caso gaussiano, parliamo ora del ***KNN***. Possiamo vedere questo metodo come quello dei prototipi, in cui ogni campione rappresenta un prototipo a se stante e quindi è un baricentro. A causa della fitta presenza di prototipi, scegliere solamente il più vicino sarebbe troppo superficiale, e quindi si prendono i primi $k$ più vicini e si decide la classe in base a quelle tra quei campioni. Formalmente, dato un punto $\mathbf{x}$, se ne considera un intorno $U(\mathbf{x})$ contenente $k$ campioni (in pratica si calcola $D(\mathbf{x}, \mathbf{y}_j)$ per ogni $\mathbf{y}_j$ e si prendono i $k$ campioni con distanza più piccola); a questo punto la scelta della classe si può caratterizzare in vari modi

- *massima frequenza*: si calcola l'istogramma delle classi dei $k$ campioni e si sceglie per $\mathbf{x}$ quella più frequente (in pratica è la moda dei $k$ campioni);
- *distanza dai baricentri*: si trovano i baricentri delle classi dei $k$ campioni e si sceglie per $\mathbf{x}$ la classe del baricentro più vicino a $k$ (praticamente è il metodo dei prototipi applicato sul sottoinsieme $U(\mathbf{x})$ di campioni). Notiamo che per $k \to N$ ci riconduciamo al metodo dei prototipi, in quanto l'intorno $U(\mathbf{x})$ contiene tutti i punti.

In caso di pareggio si possono adottare svariate strategie, come scegliere arbitrariamente, scegliere la classe che ha il baricentro più vicino, scegliere la classe che ha più campioni, ....

Per applicare questo metodo è essenziale scegliere bene la metrica $D(\cdot, \cdot)$ e il numero di punti $k$, che solitamente viene impostato come il numero dispari più vicino a $\sqrt{N}$. Bisogna considerare anche implicazioni di complessità spaziale, ovvero necessitiamo di memorizzare tutti gli $N$ campioni in memoria ma ne usiamo solo $k$, e quindi questo metodo è indicato quando $N$ è basso. Infine le feature dei dati di training vanno sempre normalizzate e le superfici di decisione sono non-lineari.

Il metodo KNN è distante dalle funzioni di probabilità $p_i(\mathbf{x})$ rispetto ai metodi visti prima, ma possiamo fare comunque delle considerazioni a riguardo: al solito, le prior vengono calcolate con $\hat{P}(\mathbf{x}) = \frac{N_i}{N}$, mentre ricordando la formula ottenuta in precedenza, le prior vengono calcolate con $\hat{p}_i(\mathbf{x} \,|\, \omega_i) = \frac{k_i / N_i}{V}$ (ovvero, ripetiamolo, il rapporto tra il numero $k_i$ di campioni appartenenti a $\omega_i$ rispetto ai $N_i$ campioni totali per $\omega_i$, contenuti nel volume $V$). Per la regola di decisione di Bayes, scelgo la classe $\omega_i$ quando $p(\mathbf{x} \,|\, \omega_i)\, P(\omega_i) > p(\mathbf{x} \,|\, \omega_j)\, P(\omega_j)$, cioè

$$\hat{p}(\mathbf{x} \,|\, \omega_i)\, \hat{P}(\omega_i) > \hat{p}(\mathbf{x} \,|\, \omega_j)\, \hat{P}(\omega_j) \quad \Longrightarrow \quad \frac{1}{V}\frac{k_i}{N_i}\frac{N_i}{N} > \frac{1}{V}\frac{k_j}{N_j}\frac{N_j}{N} \quad \Longrightarrow \quad k_i > k_j$$

che è proprio la regola che adottiamo, e quindi il metodo KNN ha fondamenta probabilistiche consistenti.

# 5 | Funzioni discriminanti lineari

[manca tutto]

# 6 | Estrazione e selezione delle feature

## 6.1 Trasformazioni lineari

[MANCA ROBA]

Spesso i campioni nel training set di classi diverse non sono ben separati spazialmente come ci si aspetterebbe, con conseguenti basse performance del classificatore. Per ovviare a questo problema applichiamo ai campioni di training $\mathbf{x}$ una trasformazione lineare $\boldsymbol{W}$, in modo da migliorare la distribuzione spaziale: vogliamo che il nuovo training set

$$\mathbf{y} = \boldsymbol{W}^\top \mathbf{x}$$

abbiamo campioni ben separati interset (ovvero tra classi diverse) e ben compatti intraset (ovvero della stessa classe), e quindi è necessario studiare come costruire al meglio $\boldsymbol{W}$.

Data una distribuzione statistica di campioni (quindi a nuvola di punti, e non lamellare o a spirale) di due classi $\omega_i$ e $\omega_j$, composte da rispettivamente $N_i$ ed $N_j$ campioni $\mathbf{x}_q^{(i)}$ e $\mathbf{x}_p^{(j)}$, definiamo ***distanza interset*** $S_{ij}$ la media tra tutte le possibili distanze tra due campioni *di classi diverse*

$$S_{ij} = \frac{1}{N_i N_j} \sum_{q=1}^{N_i} \sum_{p=1}^{N_j} d^2\left(\mathbf{x}_q^{(i)}, \mathbf{x}_p^{(j)}\right)$$

mentre ***distanza intraset*** $R_i$ la media tra tutte le possibili distanze tra due campioni *della stessa classe*

$$R_i = \frac{1}{N_i(N_i-1)} \sum_{q=1}^{N_i} \sum_{p=1}^{N_i} d^2\left(\mathbf{x}_q^{(i)}, \mathbf{x}_p^{(i)}\right)$$

dove $d$ è una funzione di distanza. Come abbiamo detto, una buona trasformazione $\boldsymbol{W}$ vuole massimizzare $S_{ij}$ e minimizzare $R_i$ ed $R_j$ (gli indici ai pedici vanno generalizzati nel caso di più classi), e quindi possiamo porre come obiettivo

$$\text{minimizzare } Q(\mathbf{y}) = \frac{R_i + R_j}{S_{ij}}$$

che essendo una funzione non lineare è complicata da risolvere. La funzione di distanza $d$ va scelta con parsimonia:

- non possiamo usare quella euclidea perché peserebbe molto i punti più lontani, quindi cerchiamo una distanza che tende a *saturare* man mano che cresce;
- le *distanze di saturazione* però ha lo svantaggio di introdurre un punto di discontinuità (ad esempio, $y = \max(x, 50)$ è ha un punto di discontinuità in $x = 50$) e sono quindi difficilmente derivabili;
- la distanza dei ***raccordi continui*** invece sembra fare al caso nostro, ed è una delle soluzioni più adottate: praticamente parte come la distanza euclidea ma viene smorzata ad un certo punto

  $$d^*(\mathbf{x}, \mathbf{y}) = 1 - e^{-\frac{1}{2}\frac{d^2(\mathbf{x},\mathbf{y})}{D^2}}$$

  dove $D$ è un parametro a scelta e $d$ la distanza euclidea (notiamo con piacere l'estrema somiglianza del secondo termine ad una campana gaussiana univariata di media nulla e varianza $D^2$).

Fatte queste considerazioni, entriamo nel merito della matrice $\boldsymbol{W}$: se abbiamo $\dim(\mathbf{x}) = n$ e vogliamo $\dim(\mathbf{y}) = m$ (solitamente si sceglie $m < n$ per rimpicciolire la dimensione del training set senza però ridurre corposamente l'informazione), quindi avremo $\dim(\boldsymbol{W}) = n \times m$. Per trovare i valori della matrice $\boldsymbol{W}$ possiamo considerare diversi casi, di complessità crescente:

***W* diagonale $n \times n$:** una versione semplificata, nellla quale non riduciamo lo spazio. Definiamo

$$\sigma_k^2 = \frac{1}{M-1} \sum_{r=1}^{M} \left( \left[\mathbf{x}_r^{(\ell)}\right]_k - \left[\bar{\mathbf{x}}^{(\ell)}\right]_k \right)^2$$

come la varianza campione della $k$-esima feature del training set $\mathbf{x}^{(\ell)}$, dove $M = N_i + N_j$ e $\ell = i, j$ se voglio $R_i + R_j$ minimo (quindi considero tutti i campioni uniti) mentre $M = N_i$ e $\ell = i$ se voglio solo $R_i$ minimo (la notazione $[\,\cdot\,]_k$ indica la $k$-esima componente del vettore e $\bar{\mathbf{x}}^{(\ell)}$ è la media dei campioni $\bar{\mathbf{x}}_r^{(\ell)}$).

Ora possiamo procedere in due modi:

- impostando il *vincolo a perimetro costante* (o di minima Lagrangiana, vedere il metodo dei moltiplicatori di Lagrange), ovvero $R_i + R_j$ minimo e $\sum_k w_{kk} = 1$ (questo per impedire allo spazio di arrivo di non esplodere ed essere contenuto in quello di partenza) possiamo definire $\boldsymbol{W}$ usando

$$w_{kk} = \frac{1}{\sigma_k^2 \sum_{h=1}^{n} \frac{1}{\sigma_h^2}}$$

  Con questa particolare definizione (ce ne sono tante altre, per esercizio dimostrare questa soddisfa il vincolo sopra) viene esaltata l'affidabilità delle singole feature, infatti varianze $\sigma_k^2$ piccole, che indicano l'affidabilità della misura della $k$-esima feature, sono pesate maggiormente (grazie alla frazione) e poste in evidenza rispetto a feature meno affidabili, aventi $\sigma_k^2$ grandi.

- impostando il *vincolo a volume costante* invece, ovvero $R_i + R_j$ minimo e $\prod_k w_{kk} = 1$ possiamo definire $\boldsymbol{W}$ usando

$$w_{kk} = \frac{1}{\sigma_k} \sqrt[n]{\prod_{h=1}^{n} \sigma_h}$$

  e anche qui pesiamo maggiormente le feature più affidabili (con la deviazione standard questa volta).

***W* arbitraria $n \times m$:** ora abbiamo bisogno di due vincoli, $R_i + R_j$ minimo e $R_i + R_j + S_{ij}$ costante. Definiamo due matrici $\boldsymbol{B}$ e $\boldsymbol{C}$, di dimensione $n \times n$, sulla base dei parametri di intraset e interset

$$b_{kh} = \frac{1}{N_i N_j} \sum_{q=1}^{N_i} \sum_{p=1}^{N_j} \Big( \big[\mathbf{x}_q^{(i)}\big]_k - \big[\mathbf{x}_p^{(j)}\big]_k \Big) \Big( \big[\mathbf{x}_q^{(i)}\big]_h - \big[\mathbf{x}_p^{(j)}\big]_h \Big)$$

$$c_{kh} = \frac{1}{N(N-1)} \sum_{q=1}^{N} \sum_{p=1}^{N} \Big( \big[\mathbf{x}_q^{(\ell)}\big]_k - \big[\mathbf{x}_p^{(\ell)}\big]_k \Big) \Big( \big[\mathbf{x}_q^{(\ell)}\big]_h - \big[\mathbf{x}_p^{(\ell)}\big]_h \Big)$$

dove nuovamente $N = N_i + N_j$ ed $\ell = i, j$. Ora diagonalizziamo la matrice $\boldsymbol{BC}^{-1}$ e ordiniamo le coppie autovalore-autovettore in ordine di autovalore decrescente: in questo modo costruiamo $\boldsymbol{W}$ matrice $n \times n$

$$\lambda_1 > \lambda_2 > \cdots > \lambda_n \qquad\qquad \boldsymbol{W} = \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_n \end{bmatrix}$$

che più essere ridotta ad una matrice $n \times m$ togliendo $n - m$ autovettori da destra; così facendo, riduciamo lo spazio delle feature, togliendo quelle che hanno un impatto minore. L'autovettore $\boldsymbol{u}_1$ è detto ***direzione di Fisher*** e permette di avere la proiezione lungo l'asse che da massima distanza interset $S_{ij}$ e minime distanze intraset $R_i$ e $R_j$.

Come nota finale, potevamo scegliere anche i vincoli $S_{ij}$ massimo e $R_i + R_j + S_{ij}$ costante, ma la trasformazione in questo caso comporta calcoli più onerosi, e quindi non viene usata.

## 6.2 Trasformata di Fisher

Con quanto detto sopra, possiamo ridurre dimensionalmente uno spazio $d$-dimensionale in uno 1-dimensionale, proiettando i campioni su una retta, così da ridurre *notevolmente* la computazione e, spesso, rendere un problema intrattabile computazionalmente semplice; la riduzione è talmente drastica che se le classi erano già ben separate prima, dopo la trasformazione, tipicamente, non lo saranno più, e quindi dobbiamo trovare la direzione migliore sulla quale proiettare i campioni.

Dato un insieme di $N$ campioni $d$-dimensionali $\mathbf{x}_1, \ldots, \mathbf{x}_N$ appartenenti a due classi $\omega_i$ e $\omega_j$, aventi rispettivamente $N_i$ ed $N_j$ campioni l'una, si vuole cercare la trasformazione $\boldsymbol{w}$ $n \times 1$ tale da generare i campioni scalari $y_1, \ldots, y_N$ tramite

$$y_k = \boldsymbol{w}^\top \mathbf{x}_k$$

Dato che ci interessa solo la direzione e non l'ampiezza, possiamo imporre $\|\boldsymbol{w}\| = 1$, così da semplificare anche i procedimenti successivi.

Nell'ipotesi che le distribuzioni siano statistiche (al solito, ellissi e non spirali o lamelle), procediamo usando come misura di separazione la distanza tra le medie dei campioni, ovvero $\boldsymbol{\mu}_i$ e $\boldsymbol{\mu}_j$ per lo spazio $d$-dimensionale ed $\mu_i$ e $\mu_j$ per quello 1-dimensionale. Definiamo il ***discriminante lineare di Fisher*** come la funzione lineare $\boldsymbol{w}^\top \mathbf{x}$ per la quale la funzione $J(\boldsymbol{w})$ è massima, dove

$$J(\boldsymbol{w}) = \frac{(\mu_i - \mu_j)^2}{s_i^2 + s_j^2} \qquad \text{con} \qquad s_i^2 = \sum_{k=1}^{N_i} \big(y_k^{(i)} - \mu_i\big)^2$$

dove $s_i^2$ e $s_j^2$ sono le ***dispersioni*** (o ***scatter***) dei campioni nelle rispettive classi, che vogliamo tenere piccole, per mantenere i campioni compatti. Per esplicitare $J(\boldsymbol{w})$ secondo $\boldsymbol{w}$, partiamo dal denominatore e definiamo delle *matrici di dispersione*

$$\boldsymbol{S}_i = \sum_{k=1}^{N_i} \big(\mathbf{x}_k^{(i)} - \boldsymbol{\mu}_i\big)\big(\mathbf{x}_k^{(i)} - \boldsymbol{\mu}_i\big)^\top$$

poi chiamiamo $\boldsymbol{S}_\mathrm{W} = \boldsymbol{S}_i + \boldsymbol{S}_j$ la matrice di covarianza intraset (*within-class*); da questo segue che

$$s_i^2 = \sum_{k=1}^{N_i} \big(y_k^{(i)} - \mu_i\big)^2 = \sum_{k=1}^{N_i} \big(\boldsymbol{w}^\top \mathbf{x}_k^{(i)} - \boldsymbol{w}^\top \boldsymbol{\mu}_i\big)^2 = \sum_{k=1}^{N_i} \Big(\boldsymbol{w}^\top \big(\mathbf{x}_k^{(i)} - \boldsymbol{\mu}_i\big)\Big)^2 = \sum_{k=1}^{N_i} \boldsymbol{w}^\top \big(\mathbf{x}_k^{(i)} - \boldsymbol{\mu}_i\big)\big(\mathbf{x}_k^{(i)} - \boldsymbol{\mu}_i\big)^\top \boldsymbol{w} = \boldsymbol{w}^\top \boldsymbol{S}_i \boldsymbol{w}$$

e quindi

$$s_i^2 + s_j^2 = \boldsymbol{w}^\top \boldsymbol{S}_i \boldsymbol{w} + \boldsymbol{w}^\top \boldsymbol{S}_j \boldsymbol{w} = \boldsymbol{w}^\top \boldsymbol{S}_\mathrm{W} \boldsymbol{w}$$

Passiamo al numeratore e chiamiamo $\boldsymbol{S}_\mathrm{B} = (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top$ la matrice di covarianza interset (*between-class*); da questo segue che

$$(\mu_i - \mu_j)^2 = \big(\boldsymbol{w}^\top \boldsymbol{\mu}_i - \boldsymbol{w}^\top \boldsymbol{\mu}_j\big)^2 = \big(\boldsymbol{w}^\top (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)\big)^2 = \boldsymbol{w}^\top (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\top \boldsymbol{w} = \boldsymbol{w}^\top \boldsymbol{S}_\mathrm{B} \boldsymbol{w}$$

e finalmente possiamo scrivere

$$J(\boldsymbol{w}) = \frac{(\mu_i - \mu_j)^2}{s_i^2 + s_j^2} = \frac{\boldsymbol{w}^\top \boldsymbol{S}_\mathrm{B} \boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{S}_\mathrm{W} \boldsymbol{w}}$$

Dato che vogliamo massimizzare $J(\boldsymbol{w})$, come al solito ne poniamo la derivata nulla, e otteniamo il $\boldsymbol{w}$ che risolve il nostro problema

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0 \quad \Longrightarrow \quad \boldsymbol{w} \propto \boldsymbol{S}_\mathrm{W}^{-1} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$$

## 6.3 Metodo PCA

Dalla trasformazione di Fisher notiamo la potenza dell'utilizzo degli autovettori per cambiare la dimensionalità dei campioni. Con il metodo PCA (o delle *componenti principali*, noto in origine come *trasformata di Hotelling* per i vettori casuali e successivamente come *trasformata di Karhunen-Loève* per i segnali continui) possiamo estrarre le direzioni sulle quali si espande la distribuzione (ovvero la direzione della varianza massima) e sfruttarli per decorrelare le feature dei campioni.

Data una popolazione di vettori $n \times 1$ di variabili casuali $\mathbf{x} = \{\mathbf{x}_1^\top, \mathbf{x}_2^\top, \dots\}$, di media $\boldsymbol{\mu}_\mathbf{x} = \mathbb{E}[\mathbf{x}]$ e covarianza $\boldsymbol{\Sigma}_\mathbf{x} = \mathbb{E}\big[(\mathbf{x} - \boldsymbol{\mu}_\mathbf{x})(\mathbf{x} - \boldsymbol{\mu}_\mathbf{x})^\top\big]$, se $\mathbf{x}$ ha cardinalità finita $M$ possiamo scrivere

$$\boldsymbol{\mu}_\mathbf{x} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i \qquad \boldsymbol{\Sigma}_\mathbf{x} = \frac{1}{M} \sum_{i=1}^{M} (\mathbf{x}_i - \boldsymbol{\mu}_\mathbf{x})(\mathbf{x}_i - \boldsymbol{\mu}_\mathbf{x})^\top = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i \mathbf{x}_i^\top - \boldsymbol{\mu}_x \boldsymbol{\mu}_x^\top$$

Siccome $\boldsymbol{\Sigma}_\mathbf{x}$ è simmetrica possiamo sempre scomporla spettralmente per ottenere i suoi autovettori $\boldsymbol{u}_i$ ed autovalori $\lambda_i$. Ponendo gli autovalori in ordine decrescente $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, sia $\boldsymbol{U} = [\boldsymbol{u}_1^\top, \dots, \boldsymbol{u}_n^\top]^\top$ la matrice modale di $\boldsymbol{\Sigma}_\mathbf{x}$, e tramite questa otteniamo una nuova popolazione

$$\mathbf{y} = \boldsymbol{U}(\mathbf{x} - \boldsymbol{\mu}_\mathbf{x})$$

che avrà media $\boldsymbol{\mu}_\mathbf{y} = 0$ e covarianza $\boldsymbol{\Sigma}_\mathbf{y} = \boldsymbol{U} \boldsymbol{\Sigma}_\mathbf{x} \boldsymbol{U}^\top = diag(\lambda_1, \dots, \lambda_n)$, ovvero avrà tutte le feature decorrelate. Dato che $\boldsymbol{U}$ è modale, e quindi composta di vettori ortonormali, allora $\boldsymbol{U}^{-1} = \boldsymbol{U}^\top$: questo ci permette di eseguire facilmente la computazione su $\mathbf{y}$ e poi di tornare indietro su $\mathbf{x}$ trasformando i risultati ottenuti.

Si potrebbe pensare, per ridurre la dimensionalità dei campioni, di considerare solo i primi $k$ autovettori, quelli relativi agli autovalori più grandi: per farlo, costruiamo $\boldsymbol{U}_k$ matrice $k \times n$, che ci darà campioni $\mathbf{y}$ di dimensione $k \times 1$. Ovviamente, applicare la trasformazione inversa per tornare a $\mathbf{x}$ è impossibile ora, poiché $\boldsymbol{U}_k$ non è quadrata e quindi non è invertibile. In ogni caso, manteniamo la formula $\hat{\mathbf{x}} = \boldsymbol{U}_k^\top \mathbf{y} + \boldsymbol{\mu}_\mathbf{x}$ per ricostruire $\mathbf{x}$ con un margine di errore, che ovviamente dobbiamo mantenere il più basso possibile: provate gioia signore e signori poiché la trasformata di Hotelling è già quella che minimizza il MSE (errore quadratico medio) dei campioni $\hat{\mathbf{x}}$

$$MSE_{\hat{\mathbf{x}}} = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{k} \lambda_i = \sum_{i=k+1}^{n} \lambda_i$$

**Confronto PCA vs. Trasformata di Fisher** Dopo l'introduzione sulle trasformate lineari, che ci hanno aperto le prospettive verso sottodimensionaggi spaziali, le due tecniche proposte sono a prima vista abbastanza simili, ma intrinsicamente diverse: la *trasformata di Fisher* permette di massimizzare/minimizzare parametri intra/inter-set e ha bisogno di conoscere le classi dei campioni nel training set, quindi è una tecnica molto potente nei casi *supervised* (cioè quando si hanno le etichette dei campioni); la *PCA* invece utilizza proprietà intrinseche dei campioni e non necessità di conoscerne la classe, potendo offrire una tecnica ottima (ma purtroppo inferiore rispetto a Fisher, poiché non sempre massimizzare la varianza garantisce la separazione dei dati) nei casi ***unsupervised*** (cioè quando i campioni non hanno un'etichetta).

## 6.4 Curse of dimensionality

Nella pratica, i training set sono ben lontani dall'essere ideali: pochi campioni, poche/molte feature, feature non selezionabili (cioè queste ho e queste devo usare) oppure non indipendenti (cioè correlate tra loro) ...; oltre a queste problematiche, bisogna tenere conto anche della complessità computazionale del sistema, che con le premesse date non sembra essere il colpo di grazia.

Quando il numero di campioni non è sufficiente potremmo incombere nel fenomeno dell'***underfitting***; per evitare questo problema, dobbiamo ricorrere spesso a pesanti modifiche del training set:

- ridurre la dimensionalità delle feature;
- combinare in modo furbo le feature (***feature extraction***);
- stimare meglio la matrice di covarianza a partire da una stima precedente;
- impostare una soglia alla matrice di covarianza o imporre (pesantemente) che sia diagonale, forzando l'indipendenza delle feature, che potrebbero però essere correlate fra loro.

# 7 | Support Vector Machines

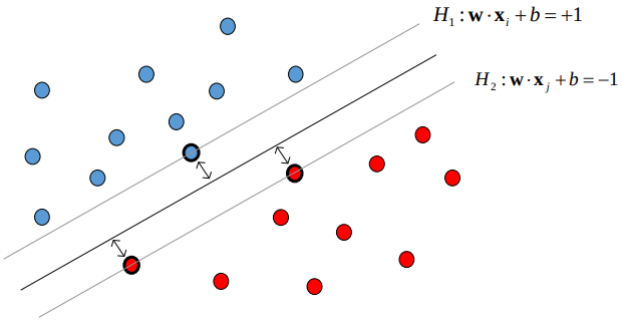
Fino ad ora abbiamo visto solo classificatori Bayesiani, i quali usavano funzioni discriminanti lineari (abbiamo visto solo quelle). Abbiamo anche visto come semplificare gli spazi delle feature, sottodimensionando i campioni del training set tramite delle trasformazioni lineari e semplificando quindi la complessità computazionale dei successivi riconoscimenti. Vediamo ora un nuovo tipo di classificatore, che recentemente (anni '50) ha portato molta innovazione grazie al cambio di paradigma che ha adottato.

## 7.1 SVM lineari

Dato uno spazio delle feature che divide i campioni su due porzioni linearmente separabili, definiamo le SVM lineari come classificatori binari che dividono lo spazio utilizzando al solito un iperpiano, decidendo la classe per $\mathbf{x}_i$ usando

$$y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 1$$

dove $y_i = \{-1, 1\}$ è l'etichetta del campione $\mathbf{x}_i$, $\boldsymbol{w}$ sono i pesi per i campioni e $b = -\boldsymbol{w}^\top x_{i0}$ è la soglia esplicitata. Notiamo che il particolare dominio di $y_i$ ci permette di descrivere il classificatore con una singola formula, cosa che sarà utilissima più avanti. L'approccio è infatti quello di massimizzare il ***margine*** delle classi, ovvero la più piccola distanza tra il punto più vicino di una classe alla linea di separazione.




I campioni $\mathbf{x}_i$ e $\mathbf{x}_j$ che stanno sulle rette $H_1$ ed $H_2$ sono detti i ***support vector*** della classe di cui fanno parte e la loro distanza dalla linea di separazione è $d^+$ e $d^-$ (che ovviamente sono uguali poiché la linea si pone esattamente in mezzo). Possiamo dimostrare (usando la formula della distanza retta-punto) che si ha

$$d^+ = d^+ = \frac{1}{\|\boldsymbol{w}\|}$$

e dato che vogliamo massimizzare il margine $d = d^+ + d^- = \frac{2}{\|\boldsymbol{w}\|}$, il problema si riduce a

$$\text{minimizzare} \quad \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} \quad \text{con i vincoli} \quad y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 1 \quad \forall i = 1 \dots N$$

Utilizziamo il metodo dei moltiplicatori di Lagrange per ognuno dei vincoli e cambiamo la formula in

$$L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i \big(y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) - 1\big)$$

dove $\alpha_i \geq 0$ sono i moltiplicatori di Lagrange; qui massimizziamo per i termini $\boldsymbol{w}$ e $b$ e minimizziamo per i termini $\alpha_i$, per trovare i punti di sella

$$\frac{\partial}{\partial \boldsymbol{w}} L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0 \quad \Longrightarrow \quad \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial}{\partial b} L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0 \quad \Longrightarrow \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\frac{\partial}{\partial \alpha_i} L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0 \quad \Longrightarrow \quad y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) = 1 \quad \forall i = 1 \dots N \quad \text{(sono i vincoli iniziali)}$$

e quindi la soluzione $\boldsymbol{w}$ è data dalla combinazione di tutti quei campioni $\mathbf{x}_i$ che hanno $\alpha_i \neq 0$, ovvero quei campioni che chiamiamo ***support vector***; sostituiamo quanto appena trovato nell'equazione e otteniamone la *rappresentazione duale*

$$L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \qquad \text{con} \quad \alpha_i \geq 0 \quad \text{e} \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

che andremo a massimizzare, così da ottenere i moltiplicatori $\alpha_i$ (il problema è quadratico e impiega $O(N^3)$, quindi con un numero di campioni elevato il gioco si fa duro; al contrario, quando il numero di feature $n < N$ conviene minimizzare direttamente $\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$, che ha costo $O(n^3)$).

Trovato il vettore $\boldsymbol{\alpha}$ possiamo calcolare $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$ e $b$, per il quale sarebbe idealmente sufficiente utilizzare un qualsiasi support vector ($\alpha_i \neq 0$) attraverso $b = y_i - \boldsymbol{w}^\top \mathbf{x}_i$, ma che a causa della precisione dei numeri a virgola mobile e dei troncamenti sarebbe meglio calcolare con $b = \frac{1}{N_\mathcal{S}} \sum_{i \in \mathcal{S}} (y_i - \boldsymbol{w}^\top \mathbf{x}_i)$, ovvero tramite la media dei bias calcolati da ogni support vector (i cui indici sono contenuti nell'insieme $\mathcal{S}$). Ora che abbiamo tutti gli ingredienti possiamo finalmente classificare un qualsiasi campione $\mathbf{x}$ utilizzando

$$y_\mathbf{x} = \boldsymbol{w}^\top \mathbf{x} + b$$

che idealmente (utilizziamo la formula ideale $b = y_i - \boldsymbol{w}^\top \mathbf{x}_i$) rappresenta proprio l'equazione della retta passante per un punto, proprio un support vector

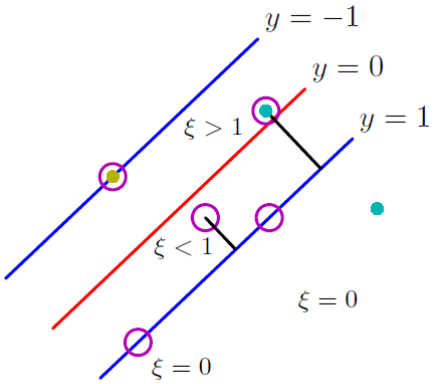
$$y_\mathbf{x} = \boldsymbol{w}^\top \mathbf{x} + (y_i - \boldsymbol{w}^\top \mathbf{x}_i) \quad \Longrightarrow \quad (y_\mathbf{x} - y_i) = \boldsymbol{w}^\top (\mathbf{x} - \mathbf{x}_i)$$

## 7.2 Classi non-linearmente separabili

Ovviamente l'assunzione di confini di separazione lineari è abbastanza importante, infatti quasi mai è così, e l'ottimizzazione quadratica descritta sopra non riesce a convergere in questi casi. Facendo una piccola modifica però possiamo gestire anche questo caso

$$y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$$

dove $\xi_i$ sono ***variabili slack*** che indicano, una per campione, la distanza che esse hanno oltre la linea dei loro support vector (se invece sono interni allora $\xi_i = 0$). Con questa descrizione, data una linea di separazione $\boldsymbol{w}$, tutti i campioni con $\xi_i > 1$ verranno classificati male, ovvero come appartenenti all'altra classe, e quindi con questo metodo permetteremo alla SVM di commettere errori di classificazione, con il vantaggio però di mantenere la semplicità dell'iperpiano.




Il nuovo obiettivo diventa quindi

$$\text{minimizzare} \quad \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{N} \xi_i \quad \text{con i vincoli} \quad y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{e} \quad \xi_i \geq 0 \quad \forall i = 1 \ldots N$$

dove $C$ è un costo definito manualmente in base alla sensibilità agli errori che vogliamo dare al classificatore. Al solito, massimizziamo

$$L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \big(y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) - 1 + \xi_i\big) - \sum_{i=1}^{N} \mu_i \xi_i$$

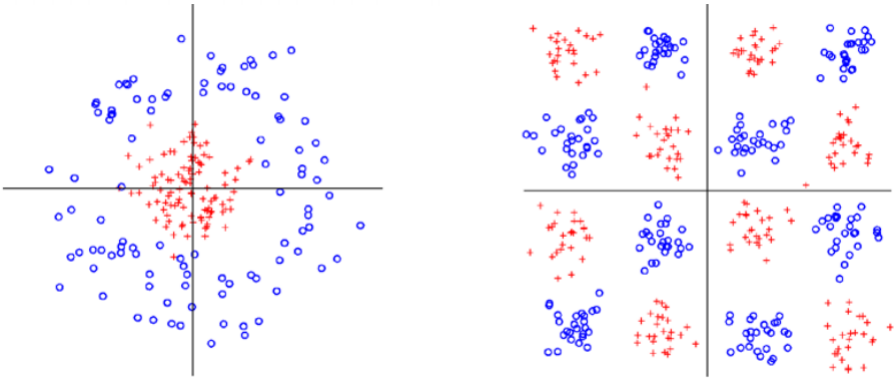
dove $\alpha_i$ e $\mu_i$ sono moltiplicatori di Lagrange. Sostituiamo quanto ottenuto dalle derivate rispetto a $\boldsymbol{w}$, $b$ e $\boldsymbol{\xi}$, in modo da ottenere sorprendentemente lo stesso risultato precedente, ma con delle condizioni leggermente differenti

$$L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \qquad \text{con} \quad 0 \leq \alpha_i \leq C \quad \text{e} \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

A questo punto, come prima, andiamo semplicemente ad estrarre $\boldsymbol{\alpha}$, e poi $\boldsymbol{w}$, $b$ e $\boldsymbol{\xi}$, così da infine poter ottenere la stessa formula di classificazione precedente.

## 7.3 Kernel trick

Come detto sopra, nei casi reali le classi potrebbero non essere separabili mediante semplici linee ma tramite curve: quando non vogliamo esplicitamente introdurre errori (come facevamo sopra con le $\xi_i$), si fa uso di appositi ***mapping*** $\phi$, ovvero funzioni che trasportano i campioni in spazi, solitamente di dimensione maggiore, nei quali è possibile una separazione lineare.



Nel caso delle distribuzioni a toroide qui sopra ad esempio, il mapping

$$\phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}$$

permette di separare in uno spazio tridimensionale cubico le due classi, i cui punti si trovano negli angoli opposti della diagonale interna. Ovviamente il problema dei mapping è la loro difficoltà computazionale a causa dell'aumento di dimensionalità (potrebbero essere necessari mapping infinito-dimensionali a volte) e che soprattutto non c'è una procedura standard per trovarli, e quindi bisogna fare affidamento sull'intuito.

Con il concetto di mapping appena introdotto andiamo a sostituire nelle formule precedenti tutti gli $\mathbf{x}_i$ con $\phi(\mathbf{x}_i)$, in quanto possiamo vedere ogni campione come già mappato nel nuovo spazio (quando il mapping non è necessario basta definire $\phi : \mathbf{x} \mapsto \mathbf{x}$). Notiamo però che gli $\mathbf{x}_i$ non compaiono mai da soli ma sempre in coppie $\mathbf{x}_i^\top \mathbf{x}_j = \mathbf{x}_i \cdot \mathbf{x}_j$, che quindi mappiamo in $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$, e notiamo un altro dettaglio: dato che appunto sono sempre in coppia, a noi non interessa tanto il mappare i singoli campioni, quanto invece mappare le coppie $(\mathbf{x}_i, \mathbf{x}_j)$, e quindi introduciamo il concetto di ***kernel***, ovvero una matrice $K(\mathbf{x}_i, \mathbf{x}_j)$ che rappresenta il prodotto scalare dei mapping tra ogni $\mathbf{x}_i$ e $\mathbf{x}_j$.

***Kernel***

Dato il mapping $\phi$, la funzione $K(\mathbf{x}_i, \mathbf{x}_j)$ è un kernel se viene soddisfatta la condizione di Mercer, ovvero se la matrice $\boldsymbol{G} = [K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j}$ di Gram è semidefinita positiva (e quindi ha solo autovalori $\lambda_i \geq 0$).

$$\boldsymbol{G} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K(\mathbf{x}_1, \mathbf{x}_j) & \cdots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots & & \vdots \\ K(\mathbf{x}_i, \mathbf{x}_1) & \cdots & K(\mathbf{x}_i, \mathbf{x}_j) & \cdots & K(\mathbf{x}_i, \mathbf{x}_N) \\ \vdots & & \vdots & \ddots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & \cdots & K(\mathbf{x}_N, \mathbf{x}_j) & \cdots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

Per come l'abbiamo definito, siccome non viene mai calcolato esplicitamente (le funzioni dei mapping si possono combinare e semplificare spesso), un kernel può essere visto come un modulo a se stante rappresentante uno specifico problema e utilizzato in un algoritmo general purpose. Alcuni kernel sono canonici, ad esempio

**Lineare:** $K(\boldsymbol{p}, \boldsymbol{q}) = \boldsymbol{p} \cdot \boldsymbol{q}$, usato quando llo spazio delle feature è enorme;

**Polinomiale:** $K(\boldsymbol{p}, \boldsymbol{q}) = (\boldsymbol{p} \cdot \boldsymbol{q} + 1)^c$, ha problematiche di cancellazione catastrofica vicino a 0 e $\infty$, aggiunge un parametro $c$ per il grado;

**Radial Basis Functions (RBF):** $K(\boldsymbol{p}, \boldsymbol{q}) = e^{-\frac{\|\boldsymbol{p}-\boldsymbol{q}\|^2}{2\sigma^2}}$, solitamente è la scelta migliore, aggiunge un parametro $\sigma$ per l'ampiezza della campana;

**Sigmoide:** $K(\boldsymbol{p}, \boldsymbol{q}) = \tanh\big(a(\boldsymbol{p} \cdot \boldsymbol{q}) + b\big)$, derivato dalle reti neurali e solitamente non usato, aggiunge due parametri $a$ e $b$ che in alcuni casi non permettono di soddisfare la condizione di Mercer.

## 7.4 Cookbook

Con queste nozioni e tecniche possiamo stilare i passaggi da compiere per ottenere un buon classificatore, unendo gli ultimi due metodi visti ($\boldsymbol{\xi}$ e kernel):

1. rescaling dei dati;
2. utilizzo del kernel RBF (potrebbe dare overfitting, a volte è meglio un kernel polinomiale con grado alto);
3. usare la cross-validation per scegliere i migliori valori per i parametri $C$ e $\sigma$ (è time consuming, si potrebbe procedere gerarchicamente, solitamente si usano valori esponenziali nei range $C = 2^{[-5 \dots 15]}$ e $\sigma = 2^{[-15 \dots 5]}$);
4. usare tali valori per fare training sul training set;
5. testare il test set.

## 7.5 SVM vs. ANN

Una tecnologia precedente alle SVM sono le Artificial Neural Network (basate su percettroni), tornate in pompa magna recentemente ma che all'epoca non vissero battaglie sul campo a causa dei seguenti problemi:

- possono cadere in molteplici minimi locali (le SVM invece hanno una singola soluzione, e pure globale);
- hanno complessità dipendente dalla dimensionalità dei campioni (le SVM no grazie ai kernel);
- adottano una minimizzazione del rischio empirica (le SVM strutturale, più robusta e precisa);
- sono prone all'overfitting (più delle SVM);
- spesso perdono a confronto con le SVM (parliamo di ANN classiche, le DNN sono un'altra story bro);
- sono più lente e imprevedibili;
- trovano più a fatica relazioni tra i campioni;
- necessitano di più informazioni riguardanti lo specifico problema.

Sembra che non abbia senso usare le ANN da sta disfatta di elenco, il plot twist è che le ANN possono lavorare in modalità unsupervised, mentre le SVM *si basano intrinsicamente* sulle etichette e quindi possono lavorare solamente in modalità supervised (ci sono delle tecniche che permettono anche la modalità semi-supervised, ma lasciamo stare). Inoltre per le SVM va scelta con cura la funzione kernel e i suoi parametri, il training non è incrementale (quindi non è adatto per situazioni online ed ogni volta che si aggiungono campioni va rifatto tutto il training daccapo), senza contare che non esiste una versione ad $M$ classi, ed in questi casi (praticamente tutti i casi reali) è necessario invece costruire $M$ classificatori binari, ognuno che discrimini una singola classe rispetto a tutte le altre.

# 8 | Artificial Neural Networks

Ancora prima delle SVM la ricerca si soffermò sulla simulazione dei processi biologici di approccio ai problemi, ovvero *algoritmi genetici*, *simulated annealing* e infine ***reti neurali artificiali***, nelle quali si cerca di copiare il funzionamento dei neuroni che formano il cervello. Il neurone ha un comportamento molto semplice: da un nucleo centrale si diramano i dendriti, che ricevono segnali dagli altri neuroni, e l'assone, che manda un segnale ad un prossimo neurone; il nucleo esegue praticamente una somma pesata dei segnali in ingresso e decide se emettere un segnale d'uscita in base a processi biologici tanto tanto difficili, che qui sono resi da una funzione.

## 8.1 Multilayer Perceptron Networks

Il modello più immediato per simulare un neurone è quello del ***percettrone***, che dati $n$ input $I_i$, ne computa una somma pesata usando $w_i$ e applica una soglia $t$; infine viene utilizzata una ***funzione di attivazione*** $h$ *non-lineare*, per trasmettere al prossimo percettrone

$$O = h\left(\sum_{i=1}^{n} w_i I_i - t\right) = h(\boldsymbol{W}^\top \boldsymbol{I} - t) \qquad \text{con} \qquad h : \mathbb{R} \to [0,1] \qquad x \mapsto \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases} \quad \text{(gradino di Heaviside)}$$

Per omogeneizzare i parametri possiamo pensare a $t$ come ottenuto da un altro input *costante*, ovvero $t = w_{i0} I_0$, così da poter riscrivere tutto come $O = h\big(\sum_{i=0}^{n} w_i I_i\big)$; da ora in poi tratteremo le formule in questi termini, omettendo la $t$, o alle volte ignorandola direttamente.

**Funzioni di attivazione** In altri casi le funzioni di attivazione sono definite su tutto $\mathbb{R}$, come ad esempio

**identità**
$h(x) = x$

**soglia**
$$h(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ -1 & \text{altrimenti} \end{cases}$$

**sigmoide**
$h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

**tangente iperbolica**
$h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

**gaussiana**
$h(x) = e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)}$

**Rectified Linear Unit (ReLU)**
$h(x) = \max\{0, x\}$

**soft computation (Softplus)**
$h(x) = \log(1 + e^x)$

Notiamo che l'argomento di $h$ non è altro che il solito piano separatore lineare che abbiamo usato fino ad ora, e quindi le reti neurali non sono nulla di troppo distante da ciò che abbiamo visto. La parte interessante comincia quando di dispongono i neuroni a strati e connessi con i successivi: in questo modo si divide la rete a ***livelli***, che permettono di aumentare l'astrazione degli input, in modalità ***feed-forward*** (connessioni tutte acicliche) o ***recurrent*** (almeno una connessione ciclica), ma ci concentriamo solo sulla prima. Posti in parallelo $k$ neuroni su un livello, essi tracciano $k$ linee di separazione, le quali definiscono una porzione di spazio anche molto complessa. La potenza espressiva sta qui: funzioni di attivazione, numero di livelli e di neuroni in essi.

**Teoremi e limiti** Ovviamente la domanda è: ma se basta un livello di neuroni già per tracciare confini complessi, è sufficiente per qualsiasi complessità del problema? Meh... Per la classificazione a due classi feed-forward (ovvero $n$ input, un livello nascosto di $k$ neuroni ed un solo neurone di output, che mappa 0 e 1 alle due classi) ci sono tre teoremi:

**Teorema 1 (Huang & Lipmann '88):** reti neurali feed-forward con un solo livello nascosto riescono ad approssimare quasi tutte le superfici di separazione tra due classi (proprio tutte se invece di un solo neurone per l'ultimo livello ne mettiamo di più);

**Teorema 2 (Gibson & Lowan '90):** alcune regioni dello spazio delle feature non sono approssimabili da un solo livello (punti di incrocio tra due linee di separazione ad esempio);

**Teorema 3 (Lipmann '87):** reti neurali con almeno due livelli possono approssimare qualsiasi regione nello spazio (la precisione della regione dipende dal numero di neuroni nel livello ovviamente).

Come avrete certamente capito signori del pubblico, non ci sono numeri per sta roba, è tutta euristica, sta a voi capire come pesare gli input, quanti hidden layers usare e di quanti neuroni... Empirismo caput mundi.

**Training** Entriamo nel vivo dei formalismi: dato un training set $T = \{(\mathbf{x}_1, y_1) \dots (\mathbf{x}_N, y_N)\}$ l'obiettivo è aggiustare i pesi sulla base di questi campioni in modo da massimizzare l'accuratezza (almeno su questi campioni). Definiamo $\tilde{y}_i = f_{\boldsymbol{W}}(\mathbf{x}_i)$ l'uscita della rete per l'input $\mathbf{x}_i$ sottoposto ai pesi $\boldsymbol{W}$; una volta fissatane la topologia (quanti input, quanti layer e di quanti neuroni, quanti output), la matrice $\boldsymbol{W}$ identifica completamente la rete neurale. Ora comincia l'aggiustamento: voglio minimizzare l'errore dato dai pesi $\boldsymbol{W}$, formalmente detto ***LOSS***, che può essere dato in molte forme (MSE, MAE, RMSE, ...) ma che per semplicità trattiamo con il MSE

$$E(\boldsymbol{W}) = \frac{1}{2}\sum_{i=1}^{N}\big(y_i - \tilde{y}_i\big)^2$$

in cui $\frac{1}{2}$ è lì solo per semplificare l'esponente (spoilerz!); l'obiettivo è trovare $\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} E(\boldsymbol{W})$, e indovina un po', come al solito, serve il gradiente. Siccome siamo ben lontani dal mondo idilliaco che è $\mathbb{R}$, dobbiamo usare il gradiente discreto (chiaramente è un'equazione alle differenze), e iterativamente aggiustare $\boldsymbol{W}^{(k+1)}$ usando il gradiente locale (impostiamo $\boldsymbol{W}^{(0)}$ casualmente, tanto la convergenza verso un minimo è garantita)

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} + \Delta\boldsymbol{W}^{(k)} \qquad \text{dove} \qquad \Delta\boldsymbol{W}^{(k)} = -\eta\frac{\partial E}{\partial \boldsymbol{W}}\bigg|_{\boldsymbol{W}=\boldsymbol{W}^{(k)}}$$

dove $\eta$ è detto ***learning rate*** e rappresenta la velocità di convergenza verso la soluzione.

Con questa tecnica, detta ***gradient descent***, attraversiamo lo spazio dei pesi usando un percorso che minimizza l'errore cercando il punto di minimo globale (spesso si trovano minimi locali, ma c'è comunque un modo per disincagliarsi). L'aggiornamento può avvenire in

**approccio batch:** le modifiche ai pesi vengono effettuate dopo aver presentato alla rete un insieme di campioni (tutti o piccoli gruppetti, i *batch* appunto, questa seconda opzione è più usata ora) e l'errore viene valutato su tale insieme. L'idea è di fare poche modifiche ma sostanziali;

**approccio on-line:** le modifiche vengono fatte ad ogni campione presentato, valutando l'errore caso per caso. L'idea è di fare tante piccole modifiche, esplorando molto lo spazio dei pesi.

Ogni volta che tutti i campioni vengono presentati alla rete segniamo la fine di un'***epoca***, che rappresenta l'unità di misura temporale per l'addestramento; a questo punto si ricomincia daccapo, andando di epoca in epoca finché l'errore non sarà sotto ad una soglia scelta a piacere.

**Brutte bestie del gradient descent**

- impostare bene $\eta$: quando è troppo piccolo training ci metterà una vita, quando è troppo grande quasi certamente continueremo a saltellare attorno al punto di minimo senza raggiungerlo mai;
- minimi locali: in base a $\boldsymbol{W}^{(0)}$ possiamo incappare in minimi locali anziché globali, e quindi introduciamo il concetto di ***momento*** riscrivendo

  $$\Delta\boldsymbol{W}^{(k)} = -\eta\frac{\partial E}{\partial \boldsymbol{W}}\bigg|_{\boldsymbol{W}=\boldsymbol{W}^{(k)}} + \alpha\big(\boldsymbol{W}^{(k)} - \boldsymbol{W}^{(k-1)}\big)$$

  dove $0 < \alpha < 1$ è il ***parametro di momento***, che in pratica descrive l'*inerzia* del sistema. Altre tecniche sono l'utilizzo di simulated annealing, algoritmi genetici o la Reactive Tabu Search;
- punti di plateau: le varie $h$ potrebbero avere zone di saturazione, che nella derivata si traducono in zone quasi nulle, nelle quali l'iterazione è talmente piccola da rendere l'apprendimento microscopico (circolo vizioso). Possiamo risolvere bene o male impostando $\boldsymbol{W}^{(0)}$ con pesi molto piccoli, così almeno inizialmente le $h$ non saturano;
- gradiente: calcolare il gradiente di una matrice $\boldsymbol{W}$ ha complessità quadratica, e quindi diventa onerosa da computare al crescere delle dimensione della rete. Questo si sistema con la *back-propagation.*

**Notazione matriciale** Nel layer $L^{(i)}$, composto da $q$ percettroni, il percettrone $L_j^{(i)}$ riceve un input $\mathbf{x}_{p\times 1}^{(i-1)}$ e produce un output scalare $x_j^{(i)}$ usando il vettore di pesi $\boldsymbol{w}_j^{(i)}$ (dimensione $p \times 1$) e la funzione di attivazione $h_j^{(i)}$; in modo compatto

$$L_j^{(i)} : x_j^{(i)} = h_j^{(i)}\big(\boldsymbol{w}_j^{(i)} \cdot \mathbf{x}^{(i-1)}\big)$$

Dato che questo calcolo deve essere fatto per ognuno dei $q$ percettroni, poniamo affiancati in colonna tutti i vettori dei pesi nella matrice $\boldsymbol{W}_{p\times q}^{(i)}$ e produciamo in output direttamente un vettore $\mathbf{x}_{q\times 1}^{(i)}$, usando per tutti la stessa funzione di attivazione $h^{(i)}$, così

$$L^{(i)} : \mathbf{x}^{(i)} = h^{(i)}\big(\boldsymbol{W}^{(i)} \cdot \mathbf{x}^{(i-1)}\big)$$

Infine, indichiamo con $\mathbf{x}_{n\times 1}^{(0)}$ il vettore degli input e $\tilde{\mathbf{y}}_{m\times 1}$ il vettore degli output e $\boldsymbol{a}^{(i)} = \boldsymbol{W}^{(i)} \cdot \mathbf{x}^{(i-1)}$.

**Esempio**

In questo modo, una rete neurale con un hidden layer (segnamo con $^{(O)}$ e operazioni del layer di output) può essere espressa analiticamente come

$$\tilde{\mathbf{y}}_{m\times 1} = h^{(O)}\Big(\boldsymbol{W}^{(O)}_{p_1\times m}\cdot \mathbf{x}^{(1)}_{p_1\times 1}\Big) = h^{(O)}\Big(\boldsymbol{W}^{(O)}_{p_1\times m}\cdot h^{(1)}\big(\boldsymbol{W}^{(1)}_{n\times p_1}\cdot \mathbf{x}^{(0)}_{n\times 1}\big)\Big)$$

mentre una rete neurale che discrimina 2 classi, con ingressi a 5 feature, 3 hidden layer di dimensione rispettivamente 20, 50 e 10, avrà forma

$$\tilde{\mathbf{y}}_{2\times 1} = h^{(O)}\left(\boldsymbol{W}^{(O)}_{20\times 2}\cdot h^{(3)}\left(\boldsymbol{W}^{(3)}_{50\times 20}\cdot h^{(2)}\Big(\boldsymbol{W}^{(2)}_{10\times 50}\cdot h^{(1)}\big(\boldsymbol{W}^{(1)}_{5\times 10}\cdot \mathbf{x}^{(0)}_{5\times 1}\big)\Big)\right)\right)$$

(i pedici indicano le dimensioni e non vengono mai usati nella notazione, qui servono da esempio)

**Back-propagation** Questa tecnica di apprendimento si basa sul gradient descent e si divide in due fasi:

1. *forward phase*: viene presentato un campione (o un batch) alla rete, se ne determina l'uscita e si calcola l'errore (finora tutto uguale a prima) *relativo di ogni peso di ogni percettrone del layer di output* (novità!);
2. *backward phase*: in base agli errori relativi dell'$i$-esimo hidden layer si calcolano quelli del layer precedente, tornando fino al primo hidden layer. In questo modo andiamo ad aggiustare progressivamente i pesi *in base ai loro contributi nei vari percettroni*, impiegando tempo lineare (ora vediamo come) nel numero dei pesi, molto meglio di calcolare tutto il gradiente!

Quindi, per calcolare l'influenza di ogni peso sull'errore, udite udite, si usa di nuovo la derivata, ma più in piccolo: dopo aver applicato in input il campione $\mathbf{x}^{(0)}$,

- partiamo dall'ultimo layer $L^{(O)}$ e, applicando la chain rule, abbiamo

$$\frac{\partial E}{\partial w^{(O)}_{jk}} = \frac{\partial E}{\partial a^{(O)}_k}\frac{\partial a^{(O)}_k}{\partial w^{(O)}_{jk}} = \frac{\partial E}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial a^{(O)}_k}\frac{\partial a^{(O)}_k}{\partial w^{(O)}_{jk}}$$

  dove $j$ è l'indice per i percettroni del layer $L^{(O)}$ e $k$ l'indice delle feature. Ora sì che si ragiona: per un singolo campione, la funzione di LOSS è $E = \frac{1}{2}\sum_{i=1}^{n}\big(y_k - \tilde{y}_k\big)^2 = \frac{1}{2}\|\mathbf{y}-\tilde{\mathbf{y}}\|^2$, mentre ricordiamo che $\boldsymbol{a}^{(i)} = \boldsymbol{W}^{(i)}\cdot \mathbf{x}^{(i-1)}$ e che $\tilde{\mathbf{y}} = h^{(O)}(\boldsymbol{a})$, quindi andiamo a espandere le derivate

$$\frac{\partial E}{\partial \tilde{y}_k} = y_k - \tilde{y}_k \qquad \frac{\partial \tilde{y}_k}{\partial a^{(O)}_k} = \frac{\partial\Big(h^{(O)}\big(a^{(O)}_k\big)\Big)}{\partial a^{(O)}_k} = h^{(O)\prime}\big(a^{(O)}_k\big) \qquad \frac{\partial a^{(O)}_k}{\partial w^{(O)}_{jk}} = \frac{\partial\big(\boldsymbol{w}^{(O)}_k\cdot \mathbf{x}^{(O-1)}\big)}{\partial w^{(O)}_{jk}} = x^{(O-1)}_j$$

  e finalmente, definendo $\delta^{(O)}_k$ il ***contributo*** del $k$-esimo percettrone nell'ultimo layer, otteniamo

$$\boxed{\frac{\partial E}{\partial w^{(O)}_{jk}} = \delta^{(O)}_k x^{(O-1)}_j} \qquad \text{con} \qquad \boxed{\delta^{(O)}_k = \frac{\partial E}{\partial a^{(O)}_k} = (y_k - \tilde{y}_k)\,h^{(O)\prime}\big(a^{(O)}_k\big)}$$

- per tutti gli altri layer $L^{(i)}$, sempre applicando la chain rule abbiamo

$$\frac{\partial E}{\partial w^{(i)}_{jk}} = \frac{\partial E}{\partial a^{(i)}_k}\frac{\partial a^{(i)}_k}{\partial w^{(i)}_{jk}} = \left(\sum_\ell \frac{\partial E}{\partial a^{(i+1)}_\ell}\frac{\partial a^{(i+1)}_\ell}{\partial x^{(i)}_k}\frac{\partial x^{(i)}_k}{\partial a^{(i)}_k}\right)\frac{\partial a^{(i)}_k}{\partial w^{(i)}_{jk}}$$

  in cui abbiamo tre nuove derivate da valutare

$$\frac{\partial E}{\partial a^{(i+1)}_\ell} = \text{ricorsivo!} \qquad \frac{\partial a^{(i+1)}_\ell}{\partial x^{(i)}_k} = \frac{\partial\big(\boldsymbol{w}^{(i+1)}_\ell\cdot \mathbf{x}^{(i)}\big)}{\partial x^{(i)}_k} = w^{(i+1)}_{k\ell} \qquad \frac{\partial x^{(i)}_k}{\partial a^{(i)}_k} = \frac{\partial\Big(h^{(i)}\big(a^{(i)}_k\big)\Big)}{\partial a^{(i)}_k} = h^{(i)\prime}\big(a^{(i)}_k\big)$$

  e quindi, definendo anche $\delta^{(i)}_\ell$ (contributo dell'$\ell$-esimo percettrone nell'$i$-esimo layer), otteniamo

$$\boxed{\frac{\partial E}{\partial w^{(i)}_{jk}} = \delta^{(i)}_\ell x^{(i-1)}_j} \qquad \text{con} \qquad \boxed{\delta^{(i)}_\ell = \frac{\partial E}{\partial a^{(i)}_\ell} = \left(\sum_\ell \delta^{(i+1)}_\ell\, w^{(i+1)}_{k\ell}\right)h^{(i)\prime}\big(a^{(i)}_k\big)}$$

Ottimo, ora manca solo il passo di aggiornamento, e il gioco è fatto ($p$ e $q$ sono gli indici dei percettroni di ingresso ed uscita)

$$\Delta \boldsymbol{W}^{(i)}_{pq} = -\eta\,\delta^{(i)}_q x^{(i-1)}_p$$

**Training/Testing sets** Ok abbiamo imparato ad imparare, ma cosa impariamo? Dove sono i miei campioni? Eh, c'è penuria... Nei casi reali non abbiamo a disposizione abbondanti campioni, perché potrebbe essere costoso oppure complicato ottenerli, e dobbiamo usarli al meglio. Abbiamo due tecniche

**metodo Resubstitution:** in cui l'intero set viene usato sia per il training che per il testing. Ovviamente avremo un errore molto basso (un limite inferiore infatti) poiché sentirsi chiedere solamente ciò che si ha studiato è una vittoria facile.

**metodi di Cross Validation:** separiamo i dati in *training* e *testing* set, applicando poi un approccio tra

**holdout:** partizionamento casuale in due sottoinsiemi disgiunti (non necessariamente esaustivo, solitamente però si divide 80/20 o ancora meglio 90/10, estremo il 95/5);

**averaged holdout:** media dei risultati di molteplici holdout su insiemi ogni volta diversi;

**leave one-out:** tutti i campioni formano il training set a parte uno, che è l'unico a far parte del testing set, poi si va di round robin fino a tornare al primo elemento usato e si mediano i risultati;

**leave k-out:** come il leave one-out ma si dividono i campioni in $S$ insiemi distinti, si fa il training con $S-1$ di essi e quello restante viene usato per il test, poi round robin sull'insieme di test e si mediano i risultati.

**Quando usare le reti neurali** Abbiamo visto che sono uno strumento molto potente, ma non sempre sono il top della gamma, infatti abbiamo bisogno di

- istanze attributi-classe: è necessaria una fase di preprocessing in cui gli attributi vanno scalati nel range $[0, 1]$ e i valori discreti diventano booleani;
- un training set molto corposo;
- tanto tanto tempo per l'addestramento;
- fregarcene della forma del confine di separazione, poiché sarebbe impossibile da capire per un umano.
- hardware dedicato (GPU o AVX nelle CPU) per i calcoli paralleli, altrimenti è davvero lentissimo;

Inoltre non sappiamo ancora come strutturare la topologia deterministicamente e dobbiamo proseguire a tentativi, sapendo che con troppi pochi neuroni non abbiamo generalizzazione (underfitting) e al contrario con troppi facciamo solo memorizzazione (overfitting).

## 8.2 Self-Organizing Feature Map

Quando non abbiamo le etichette dei campioni sorge un problema, presto risolto da questo tipo di rete, la quale ha layer con neuroni *competitivi* che si organizzano in ***cluster*** durante il training man mano che evidenziano caratteristiche nei campioni. Durante la fase di test, dopo la presentazione di un campione, solo uno tra i vari cluster avrà un'attività maggiore rispetto agli altri, e quella sarà la sua classe (ogni cluster viene etichettato con un nome in modo da indicare le classi scoperte). Anche qui abbiamo gli stessi svantaggi delle reti neurali classiche, ovvero servono tanti campioni, hanno un training lungo e i cluster identificati non sono sempre facilmente interpretabili.

## 8.3 Reti di Hopfield

Un altro approccio all'unsupervised learning, qui abbiamo una rete ricorrente in cui i neuroni sono completamente connessi e la rete impara a memorizzare i campioni che gli vengono proposti (quindi non è tanto capace di generalizzare, ma comunque è utilissima per quei casi in cui dobbiamo ripulire rumore o ricostruire un campione danneggiato).

## 8.4 Generative Adversarial Networks

Qui dobbiamo stare attenti perché ci fottono tutti: in pratica abbiamo un discriminatore e un generatore, il primo impara a riconoscere qualcosa (tipo i quadri di Monet), il secondo prova ad imitarli e li sottopone al discriminatore, il quale cerca di capire se è legit o un fake; entrambi imparano da questo e si migliorano nel tempo, finché il generatore non riesce a fregare sempre il discriminatore, poiché ha imparato a disegnare come Monet (questa qua è una figata fra).

# 9 | Unsupervised Learning

Bello bello sapere le cose ma non possiamo sempre avere le pappe pronte. Etichettare i campioni è molto costoso, time consuming e non sempre possibile da fare a mano (immaginiamo un training set con milioni di campioni). Lasciamo quindi che sia la macchina a produrre etichette in modo automatico, fidandoci di quello che produce, e poi usando tali etichette per fare training su un riconoscitore (potremmo poi chiudere il ciclo e, dopo il riconoscimento, dire all'etichettatore i risultati, in modo che possa raffinarsi, e via così). Con queste tecniche riusciamo anche a seguire il trend dei campioni, ad esempio se questi cambiano caratteristiche nel tempo, e adattare le etichette di conseguenza, caratterizzando i campioni per quelli che sono i *cluster naturali*.

***Clustering***

Dato un insieme di campioni, l'operazione di clustering crea dei gruppi (***cluster***) in modo tale che i campioni in un cluster siano più simili tra loro rispetto a quelli di altri cluster. In pratica, si raggruppano i campioni secondo il loro ***discriminante naturale***.

Dato un set di campioni $\mathbf{x}_i$, una distanza inter-oggetto $D(\mathbf{x}_i, \mathbf{x}_j)$ e un numero $K$ di cluster da trovare (questo è opzionale, ma senza il clustering sarebbe molto più difficile, possiamo immaginarlo come il lavoro che ha svolto Darwin), l'obiettivo è trovare la funzione di partizione $P(\mathbf{x})$ tale che $P(\mathbf{x}_i) = P(\mathbf{x}_j)$ se e solo se $\mathbf{x}_i$ ed $\mathbf{x}_j$ fanno parte dello stesso cluster. Le distanze possono essere le più svariate, ad esempio la *distanza euclidea* $D(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$, che però soffre di forte sbilanciamento quando un componente è molto diverso dagli altri, o anche la *distanza di Hamming* $D(\mathbf{x}, \mathbf{x}') = \sum_d \mathbb{1}(x_d \neq x'_d)$, in cui la distanza è il numero di componenti con valori diversi.

La creazione dei cluster può seguire diverse filosofie, vedremo un algoritmo per ognuna di queste:

**metodi di partitioning:** dividono i campioni in $K$ cluster minimizzando una cost function (es. *$K$-means*);

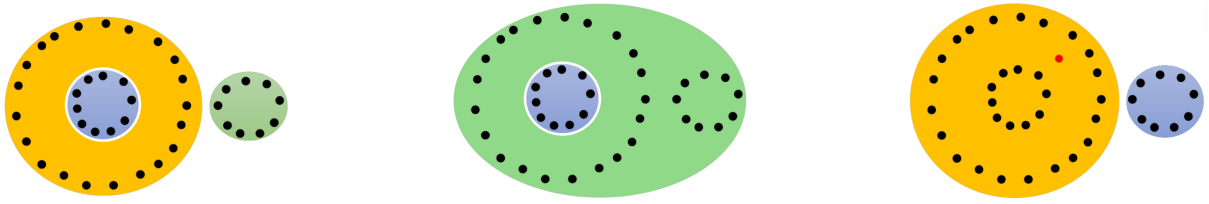
**metodi density-based:** localizzano regioni di alta e bassa densità di campioni e tracciano i confini di conseguenza (es. *mean-shift*);

**metodi gerarchici:** creano un dendrogramma (struttura ad albero) di cluster dove i nuovi cluster sono creati raggruppando o suddividendo (merge/split) i cluster precedentemente definiti (es. *Linkage*).

## 9.1 Clustering gerarchico

Con questa tecnica possiamo utilizzare due approcci:

**approccio agglomerativo:** è bottom-up, inizialmente ogni campione fa parte del suo cluster e poi questi vengono iterativamente uniti a coppie in cluster più grandi in base alle distanze inter-cluster (da definire in un qualche modo), fino a raggiungere il numero $K$ di cluster desiderato. Qui abbiamo l'algoritmo di ***Single Linkage Clustering***, dove la distanza inter-cluster è $D(c_1, c_2) = \min_{\mathbf{x} \in c_1, \mathbf{y} \in c_2} D(\mathbf{x}, \mathbf{y})$; questo algoritmo è deterministico ma sensibile agli outlier poiché la distanza è valutata tra ogni coppia possibile dei campioni nei due cluster (nelle figure sotto vediamo un clustering con $K = 3$, uno con $K = 2$, che a pelle ha qualcosa che non va, e infine un altro con $K = 2$ ma con un outlier, che sbilancia il clustering e lo rende "più ragionevole"). Altri algoritmi usano distanze sogliate per mitigare il contributo degli outlier oppure valutano i centroidi dei cluster per misurare le distanze, ma non sono gerarchici;



**approccio divisivo:** è top-down, inizialmente ogni campione è dentro lo stesso cluster e poi si dividono mano mano in cluster più piccoli.

## 9.2 Clustering con $K$-means

Tra i metodi di partitioning spicca per la sua semplicità l'algoritmo $K$-means, il quale considera i centroidi $\boldsymbol{\mu}_k$ dei cluster: usiamo la ***misura di distorsione***

$$J = \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

in cui $r_{ik} \in \{0,1\}$ è una variabile che indica l'assegnazione del campione $\mathbf{x}_i$ al cluster il cui centroide è $\boldsymbol{\mu}_k$ (1 indica assegnato, 0 indica non assegnato, questo prende il nome di *1-of-K coding scheme*). L'obiettivo ora è minimizzare $J$ con i migliori valori per $r_{ik}$ e $\boldsymbol{\mu}_k$, in modo da ottenere i centroidi che meglio rappresentano i $K$ cluster formati dai campioni a loro più vicini; per farlo procediamo iterativamente:

1. scegliere valori casuali per ogni $\boldsymbol{\mu}_k$;
2. minimizzare $J$ rispetto a $r_{ik}$ fissando $\boldsymbol{\mu}_k$: in pratica si assegna $r_{ik} = 1$ se il $k$-esimo centroide è il più vicino all'$i$-esimo campione, ovvero
$$r_{ik} = \begin{cases} 1 & \text{se } k = \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{altrimenti} \end{cases}$$
3. minimizzare $J$ rispetto a $\boldsymbol{\mu}_k$ fissando $r_{ik}$: in pratica facciamo la media dei campioni che sono stati assegnati al $k$-esimo cluster. Questa procedura, oltre ad essere perfettamente logica, ha anche evidenza analitica: minimizzare rispetto a $\boldsymbol{\mu}_k$ significa porre come al solito $\frac{\partial J}{\partial \boldsymbol{\mu}_k} = 0$, e quindi
$$\frac{\partial J}{\partial \boldsymbol{\mu}_k} = \frac{\partial}{\partial \boldsymbol{\mu}_k}\left(\sum_{i=1}^{N}\sum_{j=1}^{K} r_{ik}\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2\right) = 2\sum_{i=1}^{N} r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0 \qquad \Longrightarrow \qquad \boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} r_{ik}}$$
4. tornare a 2. e ripetere fino alla convergenza (oppure per un numero fissato di volte).

A causa dell'assegnamento casuale iniziale, $K$-means non è deterministico e nemmeno ottimale, è sensibile allo scaling dei punti e agli outlier e trova solo cluster sferici.

## 9.3 Clustering con Expectation Maximization

Adottiamo una nuova filosofia: per ogni cluster usiamo una distribuzione gaussiana e le combiniamo in modo da creare una densità di probabilità detta ***mistura di gaussiane***

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k N(\mathbf{x} \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

dove $0 \leq \pi_k \leq 1$ sono ***coefficienti di mixing*** tali che $\sum_{k=1}^{K} \pi_k = 1$. Costruiamo ora una ***variabile latente*** $K$-dimensionale $\mathbf{z}$ con 1-of-$K$ coding scheme su $z_k$ (ovvero $\mathbf{z} = [0, \cdots, 0, 1, 0, \cdots, 0]$ dove solo il $k$-esimo elemento è 1, e quindi la condizione $\sum_{k=1}^{K} z_k = 1$ è soddisfatta e $\mathbf{z}$ è una variabile casuale); notiamo che $\mathbf{z}$ ha al massimo $K$ configurazioni, in cui ognuna "accende" un solo elemento. Cerchiamo ora di incastrarla con $p(\mathbf{x})$, poiché ci verrà utile dopo: definiamo

$$p(z_k = 1) = \pi_k$$

e, sfruttando il fatto che alla fine $\mathbf{z}$ è un selettore del $k$-esimo elemento (quindi elevando a potenza rimane solo $\pi_k$, mentre tutti gli altri valori $(\pi_{k'})^{z_{k'}} = 1$ e nella produttoria sono ininfluenti) scriviamo $p(\mathbf{z})$ in questa forma poco ortodossa

$$p(\mathbf{z}) = \prod_{k=1}^{K} (\pi_k)^{z_k}$$

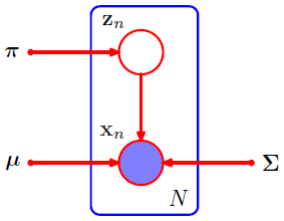
Sempre sfruttando la rappresentazione di $\mathbf{z}$, risolviamo $p(\mathbf{x} \,|\, \mathbf{z})$, infatti $p(\mathbf{x} \,|\, z_k = 1) = N(\mathbf{x} \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ e quindi, sempre in forma poco ortodossa

$$p(\mathbf{x} \,|\, \mathbf{z}) = \prod_{k=1}^{K} N(\mathbf{x} \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

A questo punto definiamo la joint distribution $p(\mathbf{x}, \mathbf{z})$ in termini di $p(\mathbf{z})$ e $p(\mathbf{x} \mid \mathbf{z})$, e notiamo che la definizione di $p(\mathbf{x})$ è rimasta inalterata, infatti

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z}) = \sum_{\mathbf{z}} \prod_{k=1}^{K} (\pi_k)^{z_k} \prod_{k=1}^{K} N(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k} = \sum_{k=1}^{K} \pi_k N(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Ottimo, ora, dati $N$ campioni $\mathbf{x}_i$, per ognuno di esso abbiamo una variabile latente $\mathbf{z}_i$, che rappresenta l'appartenenza ad uno dei $K$ cluster (è praticamente l'etichetta per $\mathbf{x}_i$). Tutta la riscrittura di $p(\mathbf{x})$ non è stata inutile quindi, poiché quello che abbiamo ora è che una qualsiasi distribuzione $p(\mathbf{x}_i)$ sottostà alla variabile $\mathbf{z}_i$. Introduciamo un'altra notazione, che rappresenta la ***responsabilità*** che il $k$-esimo componente ha nel spiegare il campione $\mathbf{x}$ (indipendente e identicamente distribuito)




$$\gamma(z_k) = p(z_k = 1 \mid \mathbf{x}) = \frac{p(z_k = 1)\, p(\mathbf{x} \mid z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)\, p(\mathbf{x} \mid z_j = 1)} = \frac{\pi_k N(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j N(\mathbf{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

(nella figura vediamo la ***plate notation***, in cui vediamo che la variabile latente, in bianco, influisce sul campione, in blu, e tutto questo viene ripetuto $N$ volte, per ogni campione)

Andiamo ora a massimizzare la (log-)likelihood: definiamo $\boldsymbol{\pi}_{K\times 1}$, $\boldsymbol{\mu}_{N\times D}$ e $\boldsymbol{\Sigma}_{D\times D}$ le matrici dei parametri, $\boldsymbol{X}_{N\times D}$ quella dei campioni e $\boldsymbol{Z}_{N\times K}$ quella delle variabili latenti; scriviamo la funzione della log-likelihood

$$\ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \ln \left( \sum_{k=1}^{K} \pi_k N(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

Bene, ora dobbiamo trovare massimizzarla, per farlo (plot twist!) dobbiamo calcolare le derivate e porle a 0, ma abbiamo il vincolo $\sum_{k=1}^{K} \pi_k = 1$, quindi dobbiamo usare i moltiplicatori di Lagrange e massimizzare invece

$$L(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left( \sum_{k=1}^{K} \pi_k - 1 \right)$$

Nessun problema, andiamo per step

- deriviamo rispetto a $\boldsymbol{\mu}_k$ e poniamo a 0

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\sum_{i=1}^{N} \frac{\pi_k N(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0 \quad \Longrightarrow \quad \boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik}) \mathbf{x}_i}{\sum_{i=1}^{N} \gamma(z_{ik})}$$

  che guarda caso è molto molto simile al risultato dell'algoritmo $K$-means, strano eh?

- deriviamo anche rispetto a $\boldsymbol{\Sigma}_k$ e poniamo a 0, ottenendo (anche qui molto simile, molto bello)

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0 \qquad \Longrightarrow \qquad \boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik}) (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^{\top}}{\sum_{i=1}^{N} \gamma(z_{ik})}$$

- ottimo, ora fissiamo i $\boldsymbol{\mu}_k$ e $\boldsymbol{\Sigma}_k$ appena trovati e deriviamo rispetto a $\pi_k$ ponendo a 0

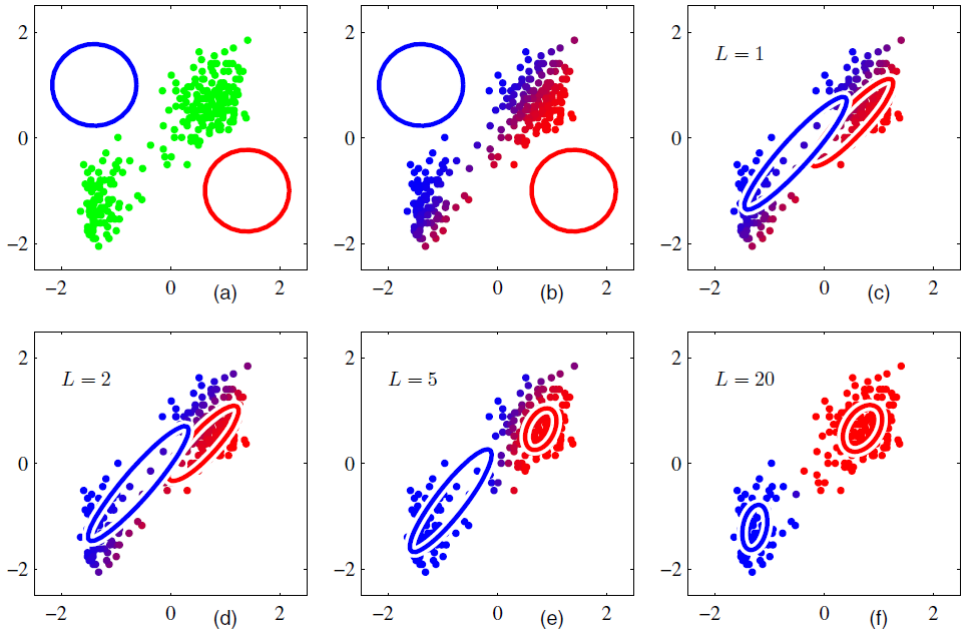
$$\frac{\partial}{\partial \pi_k} \ln p(\boldsymbol{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \frac{N(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda = 0 \qquad \Longrightarrow \qquad \pi_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik})}{N}$$

Notare che il risultato ottenuto (*molto* simile al $K$-means) non è in forma chiusa poiché i vari $\gamma(z_{ik})$ vanno calcolati in base ai $z_{ik}$, che ci devono essere forniti in qualche modo (ricordiamo che gli $\mathbf{z}_i$ sono vettori che rappresentano le etichette, che non abbiamo, ma che dovevamo introdurre per avere una notazione che le rappresentasse). Dato che appunto dobbiamo trovare queste etichette $\mathbf{z}_k$ possiamo procedere con un approccio iterativo di ***Expectation-Maximization*** o *EM*, ovvero

1. scegliere i valori iniziali per ogni $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ e $\pi_k$;
2. ***step E***: usare i parametri per valutare le responsabilità $\gamma(z_{ik})$;
3. ***step M***: usare le responsabilità per valutare i parametri $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ e $\pi_k$;

4. ripetere da 2. fino a convergenza o raggiunto un limite massimo di iterazioni.

Notiamo che nuovamente tutto è *estremamente* simile a $K$-means, e dato che EM è molto lento poiché assegna i campioni ai cluster in maniera *soft* (infatti ne descrive l'appartenenza tramite probabilità, come si vede in figura sotto, mentre $K$-means ha assegnamenti *hard*), possiamo usare $K$-means per fare preprocessing dei campioni e ottenere i valori iniziali, così da avere una stima di partenza più accurata e convergere più rapidamente.




**Dissertazioni su EM** Per capire appieno il nome di questo approccio riproponiamo l'equazione della log-likelihood in termini di posterior e seguiamo la vita di $\boldsymbol{Z}$,

$$\ln p(\boldsymbol{X} \mid \boldsymbol{\theta}) = \ln \sum_{\mathbf{z}} p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta})$$

dove $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$; avendo il dataset completo $\{\boldsymbol{X}, \boldsymbol{Z}\}$ è semplice calcolare $p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta})$, infatti basta calcolare $\boldsymbol{\theta}$ con uno dei tanti metodi di supervised learning che abbiamo visto, ma siccome la maledetta $\boldsymbol{Z}$ ci manca dobbiamo scoprirla in qualche modo. La nostra conoscenza su $\boldsymbol{Z}$ è data unicamente dalla posterior $p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\theta})$, quindi abbiamo bisogno dei parametri $\boldsymbol{\theta}$, ma per ottenere i parametri dobbiamo essere in supervised learning e quindi abbiamo bisogno di $\boldsymbol{Z}$. Procediamo iterativamente quindi: a partire da un $\boldsymbol{\theta}_0$, stimiamo l'aspettativa (*expectation*) della posterior $p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\theta}_0)$ e la usiamo per massimizzare (*maximization*) la likelihood $\mathcal{Q}$ in caso di dataset completo (non so il perché di questa formula per $\mathcal{Q}$, è un dogma)

$$\boldsymbol{\theta}^{new} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) \qquad \text{dove} \qquad \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = \sum_{\mathbf{z}} p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\theta}^{old}) \ln p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta})$$

A questo punto basta impostare $\boldsymbol{\theta}^{old} \leftarrow \boldsymbol{\theta}^{new}$ e ricominciare daccapo, fino alla convergenza. Ora l'algoritmo EM dovrebbe avere un po' più di senso dal punto di vista teorico (ti piacerebbe eh).

## 9.4 Clustering con mean-shift

Con questo approccio invece, dato l'insieme dei campioni, andiamo a studiare le ***mode*** dei campioni tramite *finestre* di dimensioni scelte. Dati i campioni $\mathbf{x}_i$, andiamo a posizionare un ***centro di massa*** casualmente nello spazio dei campioni; questo punto segna il centro di una funzione finestra (tipo Parzen windows) che racchiude al suo interno alcuni campioni. Tutti i campioni nella finestra vengono mediati, così da ottenere un nuovo centro di massa, e si ricomincia; in questo modo il centro di massa si sposterà verso il punto più denso nello spazio dei campioni, ovvero verso la moda dei campioni.

Più formalmente, data una funzione kernel $K(\mathbf{x})$ (ad esempio la gaussiana $K(\mathbf{x}) = e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}}$) e l'insieme di vicinato $N(\mathbf{x})$ (ad esempio l'insieme dei campioni che stanno al massimo a distanza $r$ da $\mathbf{x}$), l'algoritmo ***mean-shift*** esegue

1. scelta di $K$ centri di massa $\boldsymbol{\mu}_k$ (di solito casualmente);

2. per ogni $\boldsymbol{\mu}_k$, calcolo $m(\boldsymbol{\mu}_k)$ e reimposto $\boldsymbol{\mu}_k \leftarrow m(\boldsymbol{\mu}_k)$, dove

$$m(\boldsymbol{\mu}_k) = \frac{\sum_{\mathbf{x}_i \in N(\boldsymbol{\mu}_k)} K(\mathbf{x}_i - \boldsymbol{\mu}_k)\mathbf{x}_i}{\sum_{\mathbf{x}_i \in N(\boldsymbol{\mu}_k)} K(\mathbf{x}_i - \boldsymbol{\mu}_k)}$$

3. riprendo da 2. e continuo fino alla convergenza.

Questo algoritmo è molto semplice (e guarda caso molto simile a $K$-means, tutti della stessa razza in pratica) ma richiede che siano definiti il kernel $K(\mathbf{x})$ e il vicinato $N(\mathbf{x})$, che ovviamente vanno a influire sulla bontà del clustering.

## 9.5 Model selection

Tutte le tecniche viste fino ad ora avevano bisogno di conoscere il numero di cluster $K$ da cercare, ma come facciamo quando non lo conosciamo? Anche qui la letteratura si spreca, ma in generale adottiamo un processo esaustivo detto di ***model selection***, nel quale partiamo con $K = 1$ e incrementiamo via via $K$, eseguendo ogni volta il clustering e confrontandolo con quelli precedenti tramite misure varie (di solito si usano le misure inter/intra-cluster, come abbiamo già fatto con la trasformata di Fisher) per stimare la bontà di questo clustering rispetto agli altri, e ovviamente si sceglie il migliore (nell'esempio di misure inter/intra-cluster, scegliamo il clustering con il $K$ che ci da misura inter-cluster maggiore e intra-cluster minore, così da avere cluster compatti e ben separati).

# 10 | Dati sequenziali

Finora tutte le tecniche che abbiamo visto avevano bisogno di un dataset già fornito e soprattutto formato di campioni indipendenti e identicamente distribuiti. Come facciamo quando i dati ci arrivano a scaglioni, come nel caso di elettrocardiogrammi, segnali audio, andamenti di mercato, e in generale tutto ciò che accade real-time? Nulla di quello che abbiamo visto può essere adeguato, poiché il learning era fatto a priori. Intanto rilassiamo le condizioni: mettiamo per un attimo da parte l'essere real-time e immaginiamo di aver già collezionato tutti i campioni, che saranno collocati in un certo ordine nel tempo; tali dataset prendono il nome di ***time series*** (o di *dati sequenziali*).

## 10.1 Processi di Markov

Sono processi stocastici (quindi eventi che variano in base a una famiglia di variabili casuali) che soddisfano la *proprietà di Markov* (cioè la "memorylessness"). In pratica ci permettono di esprimere tutti quegli eventi che di cui si può prevedere il futuro usando semplicemente lo stato presente, senza avere uno storico; in altre parole, le posterior dello stato presente, di quelli passati e futuri sono indipendenti. Dal punto di vista formale, un processo di Markov è una struttura di $N$ stati $s_i$ caratterizzata da passi discreti $t$; in questa struttura ci sposteremo di stato in stato, mantenendo la notazione $q_t$ per indicare lo stato corrente al tempo $t$. Inizialmente, la probabilità di partire da un particolare stato è

$$P(q_1 = s_i) = \pi_i \qquad \text{dove} \qquad \pi_i \geq 0 \quad , \quad \sum_{i=1}^{N} \pi_i = 1$$

ovviamente i valori $\pi_i$ devono essere forniti oppure scoperti, ma ci arriveremo. Facciamo partire il processo e, giunti allo stato $q_t$, dobbiamo passare allo stato $q_{t+1}$: ogni stato $s_i$ ha una certa probabilità di passare ad un qualsiasi altro stato $s_j$ che però dipende solamente da $s_i$, infatti come abbiamo detto prima per la

***Proprietà di Markov***

$$P(q_{t+1} = s_j \,|\, q_t = s_i, q_{t-1} = s'_i, \ldots, q_1 = s_i^{(n)}) = P(q_{t+1} = s_j \,|\, q_t = s_i)$$

e quindi avremo una matrice $T_{N \times N}$ di transizione *tempo invariante*, in cui ogni cella descrive la probabilità di passare da uno stato all'altro

$$t_{ij} = P(q_{t+1} = s_j \,|\, q_t = s_i)$$

In generale quindi, definiamo un processo di Markov come

***Modello di Markov***

Un modello (probabilistico) di Markov (del primo ordine, ovvero che tiene conto di un solo stato prima di $q_{t+1}$) è una tripla $\lambda = \langle \mathcal{S}, \Pi, T \rangle$ che descrive un certo processo di Markov, in cui

- $\mathcal{S}$ è l'insieme degli $N$ stati del processo, che devono essere tutti ***espliciti*** (so quanti sono) e ***osservabili*** (posso catturarli univocamente tramite dei sensori);
- $\Pi = \{\pi_i = P(q_1 = s_i)\}$ è la distribuzione delle probabilità iniziali;
- $T = \{t_{ij} = P(q_{t+1} = s_j \,|\, q_t = s_i)\}$ è la ***matrice di transizione*** $N \times N$ degli stati.

Altre volte viene semplicemente indicato con la tupla $\lambda = \langle \Pi, T \rangle$ poiché il numero di stati è evincibile dalle dimensioni di $T$ ed il loro nome non è rilevante.

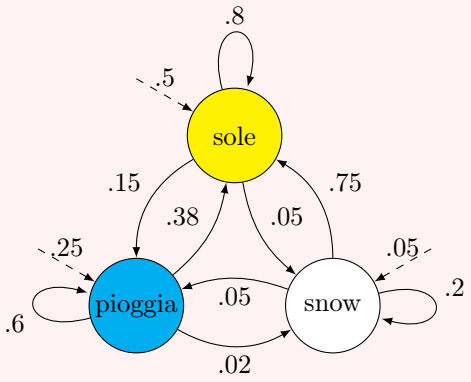
**Esempio**

Dato un processo di Markov con $N = 3$ che descrive il meteo abbiamo gli stati

$$\mathcal{S} = \{s_1, s_2, s_3\} = \{sole, pioggia, neve\}$$

le seguenti probabilità iniziali (perché magari siamo in autunno)

$$\begin{cases} P(q_1 = s_1) = .7 \\ P(q_1 = s_2) = .25 \\ P(q_1 = s_3) = .05 \end{cases} \implies \Pi = \begin{bmatrix} .5 & .25 & .05 \end{bmatrix}$$




e le seguenti probabilità di transizione

$$\begin{cases} P(q_{t+1} = s_1 \,|\, q_t = s_1) = .8 \\ P(q_{t+1} = s_2 \,|\, q_t = s_1) = .15 \\ P(q_{t+1} = s_3 \,|\, q_t = s_1) = .05 \end{cases} \quad \begin{cases} P(q_{t+1} = s_1 \,|\, q_t = s_2) = .38 \\ P(q_{t+1} = s_2 \,|\, q_t = s_2) = .6 \\ P(q_{t+1} = s_3 \,|\, q_t = s_2) = .02 \end{cases} \quad \begin{cases} P(q_{t+1} = s_1 \,|\, q_t = s_3) = .75 \\ P(q_{t+1} = s_2 \,|\, q_t = s_3) = .05 \\ P(q_{t+1} = s_3 \,|\, q_t = s_3) = .2 \end{cases}$$

dalle quali possiamo estrarre la matrice di transizione

$$A = \begin{bmatrix} .8 & .15 & .05 \\ .38 & .6 & .02 \\ .75 & .05 & .2 \end{bmatrix}$$

A questo punto si effettuano delle misurazioni meteo per 6 giorni e ci viene fornita la sequenza di stati

$$\mathcal{Q} = \{sole, pioggia, pioggia, pioggia, neve, neve\}$$

per la quale ci viene chiesto di calcolare la probabilità; in base alla matrice di transizione e alle probabilità iniziali (tutto noto) andiamo semplicemente a calcolare

$$P_{\mathcal{Q}} = P(sole)\, P(pioggia \,|\, sole)\, P(pioggia \,|\, pioggia)\, P(pioggia \,|\, pioggia)\, P(neve \,|\, pioggia)\, P(neve \,|\, neve)$$

e otteniamo $P_{\mathcal{Q}} = 0.0001512$.

Continuiamo a parlare delle sequenze: quando non ho i parametri $\Pi$ e $T$ posso usare le sequenze per stimarli con una certa bontà (data dal numero di sequenze fornite e dalla loro lunghezza), mentre quando i parametri sono noti posso dare la probabilità che una certa sequenza avvenga.

Soffermiamoci su questo punto analiticamente: noti $\Pi$ e $T$ e data $\mathcal{Q} = \{q_t, \dots, q_1\}$ una sequenza desiderata, per calcolarne la probabilità usiamo la chain rule della teoria di Bayes

$$P(\mathcal{Q}) = P(q_t, \dots, q_1) = P(q_t \,|\, q_{t-1}, \dots, q_1)\, P(q_{t-1}, \dots, q_1) = \cdots = \prod_{i=1}^{t} P(q_i | q_1, \dots, q_{i-1})$$

e dato che vale la proprietà di Markov possiamo scrivere

$$P(\mathcal{Q}) = P(q_t, \dots, q_1) = P(q_1) \prod_{i=2}^{t} P(q_i | q_{i-1})$$

che è esattamente quello che abbiamo fatto prima nell'esempio.

In alcuni casi potremmo voler sapere la probabilità che, data una qualsiasi sequenza, il suo stato finale $q_T$ sia esattamente uno in particolare $s_i$; per fare questo calcolo dovremmo prima trovare tutte le sequenze che hanno $q_T = s_i$, calcolarne le probabilità e poi sommarle. Ovviamente questo calcolo è un brute force a complessità esponenziale $O(N^T)$ che non possiamo permetterci, quindi adottiamo una tecnica più furba: definiamo il formalismo $p_t(j) = P(q_t = s_j)$ e induttivamente

$$p_1(j) = \begin{cases} 1 & \text{se } q_1 = s_j \\ 0 & \text{altrimenti} \end{cases} \qquad p_{t+1}(j) = P(q_{t+1} = s_j) = \sum_{i=1}^{N} P(q_{t+1} = s_j, q_t = s_i)$$

quindi da qui possiamo scrivere

$$p_{t+1}(j) = \sum_{i=1}^{N} P(q_{t+1} = s_j, q_t = s_i) = \sum_{i=1}^{N} P(q_{t+1} = s_j \mid q_t = s_i)\, P(q_t = s_i) = \sum_{i=1}^{N} t_{ij} p_t(i)$$

che possiamo usare per calcolare iterativamente $p_T(i) = P(q_T = s_i)$ in tempo $O(T\,N^2)$, molto meglio.

## 10.2 Hidden Markov Models

Bello bello tutto ma spesso non possiamo fare affidamento sulle dirette osservazioni, magari perché non abbiamo i sensori per quel particolare evento (ad esempio non ho modo di misurare le onde gravitazionali direttamente), oppure perché non abbiamo accesso diretto all'evento (ad esempio il semaforo non è nell'inquadratura della telecamera e quindi non so che colore ci sia). In questi casi dobbiamo fare affidamento sulle misure della *dinamica del sistema*, che non ci danno accesso diretto allo stato ma ne sono dipendenti (ad esempio, se non ho modo di controllare il meteo ma vedo che la gente gira con gli ombrelli aperti molto probabilmente sta piovendo, oppure se vedo macchine passare attraverso un incrocio molto probabilmente c'è verde). Con queste misure però non abbiamo certezza dello stato e non sempre possiamo fare inferenza tramite esse (di che colore è il semaforo se non ci sono macchine all'incrocio per dirmelo?), e quindi diciamo che il modello di Markov ha *stati nascosti*.

***Hidden Markov Model***

Un modello (probabilistico) di Markov (del primo ordine) a stati nascosti è una quintupla $\lambda = \langle \mathcal{S}, \Pi, T, \mathcal{V}, E \rangle$ che descrive un certo processo di Markov a stati nascosti, in cui

- $\mathcal{S}$ è l'insieme degli $N$ stati del processo, che devono essere tutti ***espliciti*** (so quanti sono) e ma non necessariamente ***osservabili*** (posso evincerli tramite osservazioni di fenomeni a loro correlati);
- $\Pi = \{\pi_i = P(q_1 = s_i)\}$ è la distribuzione delle probabilità iniziali;
- $T = \{t_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)\}$ è la matrice di transizione $N \times N$ degli stati;
- $\mathcal{V}$ è l'insieme degli $M$ simboli di osservazione $v_k$ che caratterizzano gli stati nascosti;
- $E = \{e_{ik} = P(o_t = v_k \mid q_t = s_i)\}$ è la ***matrice di emissione*** $N \times M$ che indica la probabilità di emissione del simbolo $v_k$ quando lo stato del sistema è $s_i$.

A questo punto oltre alle sequenze di stati $\mathcal{Q} = \{q_1, \ldots, q_T\}$ dobbiamo considerare anche delle sequenze di osservazioni $\mathcal{O} = \{o_1, \ldots, o_T\}$: possiamo usare quest'ultima per stimare la prima e viceversa, oppure entrambe per stimare il modello, ma ora vediamo tutto.

Perfetto, ora che abbiamo tutta la notazione che ci serve possiamo andare a sfruttare questo modello a nostro vantaggio: abbiamo tre principali problemi da poter risolvere:

1. **Evaluation (o Likelihood):** dato un HMM $\lambda$ noto e una sequenza di osservazioni $\mathcal{O}$, calcolare $P(\mathcal{O} \mid \lambda)$ con la ***procedura forward***;
2. **Deconding:** dato un HMM $\lambda$ noto e una sequenza di osservazioni $\mathcal{O}$, calcolare la più probabile sequenza di stati $\mathcal{S}$ che ha generato $\mathcal{O}$ con la ***procedura di Viterbi***;
3. **Training:** data una sequenza di osservazioni $\mathcal{O}$, determinare il miglior HMM $\lambda$ che può averla generata, ovvero il modello che massimizza $P(\mathcal{O} \mid \lambda)$, con la ***procedura di Baum-Welch***.

**Evaluation** la probabilità di ottenere esattamente le osservazioni $\mathcal{O}$ per forza bruta è intrattabile, quindi troviamo un modo furbo: date le osservazioni $\mathcal{O}$, definiamo la probabilità

$$\alpha_t(i) = P(o_1, \ldots, o_t, q_t = s_i \mid \lambda)$$

come la probabilità, dopo aver visto le prime $t$ osservazioni, di essere finiti nello stato $s_i$. Possiamo ridefinire quanto fatto ricorsivamente

$$\begin{aligned} \alpha_1(i) &= P(o_1, q_1 = s_i) \\ &= P(q_1 = s_i)\, P(o_1 \mid q_1 = s_i) \\ &= \pi_i\, e_{i,1} \end{aligned} \qquad \begin{aligned} \alpha_{t+1}(j) &= P(o_1, \ldots, o_{t+1}, q_{t+1} = s_j \mid \lambda) \\ &= \cdots \\ &= \textstyle\sum_{i=1}^{N} a_{ij}\, \alpha_t(i)\, e_{j,(t+1)} \end{aligned}$$

e finalmente otteniamo il primo importante risultato

$$P(o_1, \ldots, o_T) = \sum_{i=1}^{N} \alpha_T(i)$$

Ottimo, però siamo solo a metà strada: le $\alpha_t(j)$ sono chiamate ***variabili forward*** poiché considerano in avanti le osservazioni. Possiamo definire anche $\beta_t(j)$ le ***variabili backward*** come fatto prima, quindi

$$\beta_t(j) = P(o_{t+1}, \dots, o_T, q_t = s_j \mid \lambda) = \cdots = \sum_{i=1}^{N} a_{ij}\, \beta_{t+1}(i)\, e_{j,(t+1)}$$

e finalmente arrivare alla soluzione del problema originale: dato un qualsiasi $t$

$$P(\mathcal{O} \mid \lambda) = \sum_{j=1}^{N} \alpha_t(j)\, \beta_t(j) \qquad = \sum_{j=1}^{N} \beta_0(j)$$

**Decoding** qui ci chiediamo, data la sequenza di osservazioni $\mathcal{O}$ quale sia quella sequenza di stati ottenuta da $\arg\max_{\mathcal{Q}} P(\mathcal{Q} \mid \mathcal{O})$. Anche qui introduciamo una variabile

$$\delta_t(i) = \max_{\mathcal{Q}} P(q_1, \dots, q_{t-1}, q_t = s_i, o_1, \dots, o_t)$$

che indica la massima probabilità dei cammini di lunghezza $t-1$ che finisco nello stato $s_i$ e producono come output $\mathcal{O}$. Adesso usiamo l'***algoritmo di Viterbi*** (programmazione dinamica) per massimizzare $P(\mathcal{Q} \mid \mathcal{O}, \lambda)$.

**Training** qui vogliamo stimare i parametri $\lambda$ del HMM. Di solito si usa la maximum likelihood

$$\lambda^* = \arg\max_{\lambda} P(\mathcal{O} \mid \lambda)$$

ma si possono anche usare altre stime, come

$$\lambda^* = \max_{\lambda} P(\lambda \mid \mathcal{O})$$

Qui usiamo la stima di Baum-Welch, che è una tecnica di ottimizzazione gradient descent, quindi dai... (vatti a vedere l'algoritmo da solo perché io non te lo scrivo. Ah, c'è un certo Rabiner che ha scritto un paper, la Bibbia degli HMM dicono, su come calcolare efficientemente un paio di probabilità che servono nell'algoritmo).

## 10.3 Problemi aperti

I problemi relativi agli HMM sono i soliti: quanti stati? che topologia? La figata degli HMM è che sono modelli generativi, infatti dato il modello io posso generare le sequenze di osservazioni e stati, ma posso renderlo anche discriminativo?

Per il numero $k$ di stati posso andare iterativamente ad aumentarne il numero, ma più stati ho più diventa complesso il sistema, quindi possiamo usare una funzione di costo $C(k)$ per trovare un compromesso

$$k_{best} = \arg\max_{k} \big( LogLikelihood(\mathbf{y} \mid \boldsymbol{\theta}_k) - C(k) \big)$$

queste tecniche (BIC, MDL, MML, AIC, ...) prendono il nome di ***penalized likehood***.

Per la topologia invece abbiamo dei modelli standard da seguire,

- ergodic HMM: tutti gli stati sono collegati con tutti gli altri (non hanno vincoli);
- circular HMM: gli stati si collegano tra loro in modo da permettere transizioni solo in un certo senso (sono molto indicate per la shape recognition);
- left-to-right HMM: gli stati si collegano tra loro in modo da permettere transizioni solo in un certo verso (utili per lo speech recognition).