

Fondamenti di machine learning

Riassunto del corso

Creato da:
Davide Zampieri

Indice

1	Introduzione	1
1.1	Sistemi di riconoscimento	1
1.1.1	Raccolta dei dati	2
1.1.2	Scelta delle features	2
1.1.3	Selezione del modello	2
1.1.4	Addestramento del modello	2
1.1.5	Valutazione del modello	3
1.2	Course of dimensionality	4
1.3	Approcci per la pattern recognition	4
1.4	Classificatore di Bayes	4
1.5	Probability Density Functions	5
1.5.1	K-Nearest Neighbor	5
1.5.2	Reti neurali	5
1.6	Combinazioni di classificatori	6
2	Teoria della decisione di Bayes	7
2.1	Introduzione	7
2.1.1	Regola di decisione di Bayes	8
2.2	Funzione discriminante	8
2.3	Funzione Gaussiana	9
2.3.1	Densità normale univariata	9
2.3.2	Densità normale multivariata	10
2.4	Funzioni discriminanti - densità normale	11
2.4.1	Primo caso: $\Sigma_i = \sigma^2 I$	11
2.4.2	Secondo caso: $\Sigma_i = \Sigma$	12
2.4.3	Terzo caso: Σ_i arbitraria	12
3	Estrazione delle features	14
3.1	Principal Component Analysis	14
3.1.1	Passi formali della PCA	15
3.1.2	Procedura di feature extraction con PCA	16
3.1.3	Considerazioni su PCA	16
3.1.4	Esempio in MATLAB	16
3.1.5	Da PCA alle eigenfaces	18
3.1.6	Problemi delle eigenfaces	18
3.1.7	Algoritmo per le eigenfaces	18
3.2	Linear Discriminant Analysis	18
3.2.1	LDA nel caso di due classi	19

3.2.2	LDA nel caso di C classi	21
3.2.3	Esempio in MATLAB	21
3.2.4	Fisherfaces: algoritmo ed esempio in MATLAB	23
3.3	Bag of words	26
4	Learning	27
4.1	Stima parametrica	27
4.1.1	Approccio maximum likelihood (caso mono-modale)	27
4.1.2	Approccio maximum likelihood (caso multi-modale)	28
4.1.3	K-means	29
4.1.4	Algoritmo EM	30
4.2	Stima non parametrica	31
4.2.1	Parzen Windows	31
4.2.2	Meanshift	32
4.2.3	Clustering con meanshift	34
4.3	Tracking (stima di densità dinamiche)	34
4.3.1	Inseguimento	35
4.3.2	Particle filtering	35
4.3.3	Algoritmo di Condensation	36
4.3.4	Campionamento delle particelle	37
4.3.5	Multi-object tracking	37
5	Classificatori generativi	38
5.1	Processi di Markov	38
5.1.1	Modelli di Markov	38
5.1.2	Inferenze	39
5.1.3	Limiti dei modelli Markoviani	40
5.2	Hidden Markov Models	40
5.2.1	Problemi chiave degli HMM	41
5.2.2	Procedura di forward	41
5.2.3	Algoritmo di Viterbi	42
5.2.4	Stima dei parametri di un HMM	44
5.2.5	Algoritmo di Baum-Welch	45
5.3	Questioni aperte	45
5.3.1	Selezione dell'ordine del modello	46
5.3.2	Estensione dei modelli standard	47
5.3.3	Inserimento di comportamenti discriminativi	47
5.4	Sommario	48
6	Classificatori discriminativi	50
6.1	Introduzione	50
6.2	Funzioni discriminanti lineari	50
6.2.1	Training	51
6.2.2	Tecnica del gradiente discendente	52
6.2.3	Funzioni discriminanti lineari generalizzate	53
6.3	Support Vector Machines	54
6.3.1	Addestramento di una SVM	54
6.3.2	SVM lineari	55
6.3.3	Margini soft	56
6.3.4	SVM non lineari	57

<i>INDICE</i>	iii
6.3.5 Kernel trick	58
6.3.6 Selezione del kernel	59
6.3.7 Sommario	59
7 Clustering	61
7.1 Introduzione	61
7.2 Classi di approcci	61
7.3 Clustering partizionale	63
7.3.1 Clustering sequenziale	63
7.3.2 Center-based clustering	64
7.3.3 Model-based clustering	64
7.4 Clustering gerarchico	65
7.5 Validazione	65
Credits	67

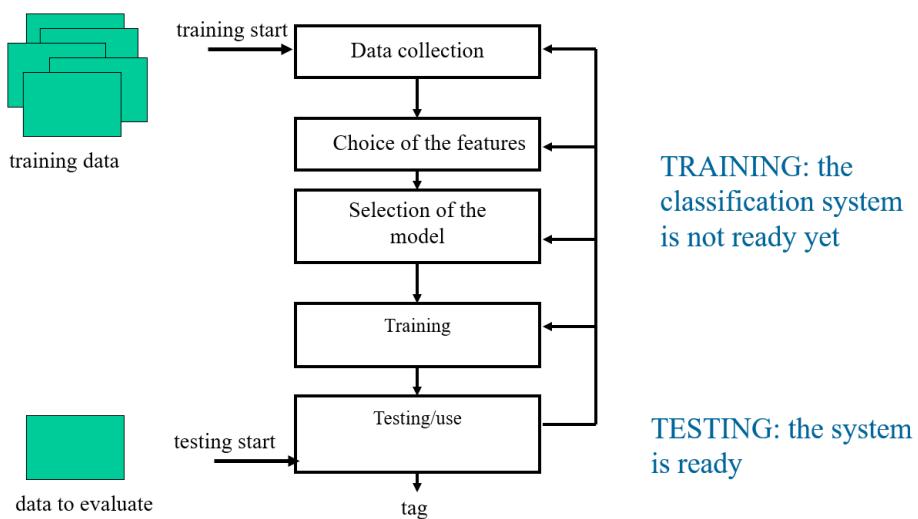
Capitolo 1

Introduzione

1.1 Sistemi di riconoscimento

Lo scopo principale della *pattern recognition* è quello di costruire degli oggetti detti *classificatori*, ovvero delle funzioni che permettano di determinare la natura di un particolare soggetto (detto *pattern*). La procedura per costruire un sistema di riconoscimento si articola in 5 fasi:

1. Raccolta dei dati.
2. Scelta delle features.
3. Selezione del modello.
4. Training (addestramento del modello).
5. Testing (valutazione del modello).



1.1.1 Raccolta dei dati

A seconda dello studio che si vuole fare, utilizzare sensori più o meno precisi può fare la differenza sia per quanto riguarda la chiarezza delle informazioni sia per le dimensioni dei dati stessi. I dati raccolti dovrebbero generare un insieme il più possibile *sufficiente* e *rappresentativo* delle classi che si vogliono studiare.

1.1.2 Scelta delle features

Per scegliere le features più significative si può procedere con la *feature extraction*, ovvero creare nuove features combinando quelle esistenti; oppure, in alternativa, si può procedere con la *feature selection*, ovvero derivare il miglior sottosinsieme delle features immediatamente reperibili dall'osservazione dei dati.

Le features dovrebbero essere semplici da calcolare, invarianti rispetto a trasformazioni irrilevanti, affidabili, indipendenti, discriminanti e, possibilmente, poche (per il problema dalla *curse of dimensionality*).

1.1.3 Selezione del modello

La scelta del modello corrisponde alla scelta della *struttura logica* e delle *basi matematiche* delle regole di classificazione. Lo scopo di un sistema di classificazione (e quindi del classificatore utilizzato) è tipicamente quello di definire il *grado di appartenenza* di un oggetto ad una particolare classe.

Allo stato attuale non esiste un classificatore in grado di spiegare bene tutti i casi di studio; ad esempio le reti neurali sono tra i classificatori più versatili, ma non sono comunque in grado di spiegare ottimamente tutti i possibili fenomeni.

1.1.4 Addestramento del modello

La fase di *training* (detta anche *learning*) permette di definire la tolleranza e la correttezza di un classificatore. Si parte dall'analisi di un *Training Set* (d'ora in poi TS) di cui si hanno conoscenze a priori, come la natura/classe dei suoi elementi. La fase di training consiste tipicamente nel trovare una variabile di soglia (detta *threshold*) che può essere ad esempio uno scalare, un vettore o una matrice. L'addestramento di un modello di classificazione si divide in tre categorie:

- *Addestramento supervisionato* (classificazione), dove l'idea è quella di creare uno strumento in grado di etichettare nuovi oggetti a partire da un TS di cui si conosce la natura degli elementi; i problemi legati a questo approccio sono capire se le soluzioni trovate sono quelle ottimali, capire se lo strumento converge e capire se lo strumento è sufficientemente scalabile (ovvero se la qualità del training è fortemente vincolata al numero di elementi studiati).
- *Addestramento semi-supervisionato*, dove si effettua un'etichettatura parziale ma significativa (sia in termini di dimensioni che in termini di natura degli oggetti selezionati) del TS e si lascia al classificatore il compito di completare tale operazione; conoscendo a priori il TS è possibile correggere il modello in modo opportuno.

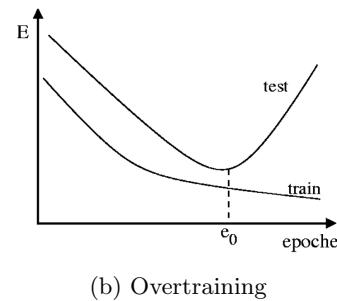
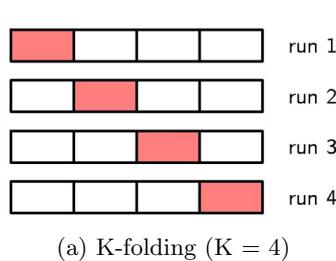
- *Addestramento non supervisionato* (clustering), dove, in opposizione alla classificazione, si dividono gli oggetti in funzione della somiglianza tra le loro caratteristiche e non in base alla loro natura; questo permette di costruire dei gruppi naturali, i clusters appunto, la cui divisione è fortemente collegata alle caratteristiche prese in considerazione.
- *Addestramento per rinforzo* (learning with a critic), ovvero una soluzione intermedia alla classificazione e al clustering in cui si procede con una metodologia test-and-try che porta ad un aggiornamento del classificatore mano a mano che si osservano nuovi pattern.

1.1.5 Valutazione del modello

In questa fase si valutano le prestazioni del classificatore in termini di *generalizzazione*. Una possibile soluzione al problema di ottenere i migliori risultati di addestramento consiste nel dividere il TS in due: una parte per l'addestramento e l'altra per la validazione (training + validation set). Questo approccio evita il problema dell'*overfitting* (o overtraining) del modello sul TS. Il rischio di un training non controllato porta infatti alla creazione di un classificatore che valuta perfettamente gli elementi del TS ma che alla minima variazione dei valori delle features in un oggetto sconosciuto, rispetto a quelle del TS, sbaglia a classificare. La divisione del TS può procedere per:

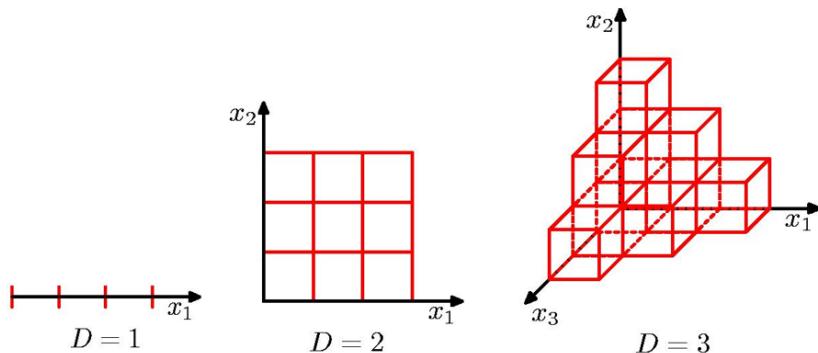
- *Holdout*, in cui il TS viene suddiviso casualmente in due parti uguali.
- *Averaged holdout*, in cui viene fatta una media sul risultato di più partitioni con holdout.
- *Leave P-Out*, in cui si effettua il training con tutti i pattern del TS tranne P elementi che vengono usati per il testing; questa procedura viene ripetuta per tutte le combinazioni possibili.
- *K-folding* (o Cross-Validation), in cui ogni osservazione viene usata almeno una volta per il testing; questo metodo prevede quindi di testare tutte le combinazioni legate alla suddivisione del TS per poi confrontare i risultati ottenuti.

La fase di training necessita di essere fermata prima di incorrere nell'overfitting. Infatti, l'errore di valutazione E di un certo pattern cala con l'avanzare dell'addestramento. Effettuare troppi passi di training porta al superamento di una soglia di generalizzazione e_0 , con conseguente errore durante la fase di test.



1.2 Course of dimensionality

Il problema della *course of dimensionality* è legato al numero di dati del TS che si devono utilizzare per ottenere una classificazione corretta. Infatti, a seconda del numero di features necessarie, il numero di campioni deve essere sufficiente (definito da una relazione esponenziale) al fine di rappresentare lo spazio delle features nella sua totalità. Si arriva quindi ad una tassellazione dello spazio delle features, in cui gli oggetti del TS devono occupare un sottospazio significativo.



1.3 Approcci per la pattern recognition

La pattern recognition può essere affrontata in due modi: tramite un *approccio sintattico*, in cui si considera la struttura dei pattern come la sintassi di un linguaggio; oppure tramite un *approccio statistico*, in cui i pattern sono associati a vettori di features che rappresentano i punti nello spazio mentre tutto il resto è espresso in termini di probabilità.

In quest'ultimo caso si parla di *classificazione statistica* in cui, come abbiamo detto, i pattern sono costituiti da vettori di features e tutte le possibili combinazioni di essi definiscono lo spazio delle features. Se in tale spazio esiste un'ipersuperficie che separa le classi, allora si parla di problema di classificazione a classi separabili.

1.4 Classificatore di Bayes

Alla base dei principi di classificazione c'è il *classificatore di Bayes*, il quale è in grado di dare una descrizione probabilistica delle conoscenze a priori, della bontà del TS e di tutto ciò che riguarda la classificazione.

La *legge di Bayes* calcola la probabilità che un certo oggetto x sia associato ad una determinata classe C_i (probabilità a posteriori, o *posterior*). Tale probabilità è calcolata moltiplicando la probabilità a priori della classe C_i (*prior*) con la *likelihood*, ovvero con la probabilità che la classe descriva correttamente l'oggetto in esame, e dividendo tale prodotto con la *evidence*, ovvero la probabilità con cui x si presenta. Quindi:

$$P(C_i|x) = \frac{P(C_i) \cdot P(x|C_i)}{P(x)}$$

La probabilità di evidence, invece, è definita come:

$$P(x) = \sum_{i=1}^N P(C_i) \cdot P(x|C_i)$$

1.5 Probability Density Functions

Il classificatore di Bayes pone le basi ad un grande insieme di approcci per la costruzione di classificatori e algoritmi per le procedure di pattern recognition. Infatti, esistono tre approcci per il calcolo delle *Probability Density Functions* (PDF):

- *Classificatori parametrici*, in cui, fissato il modello, tramite il TS si stimano i parametri che rendono ottimale la PDF (ad es. classificatore Gaussiano).
- *Classificatori non parametrici*, in cui si tenta di stimare direttamente la PDF basandosi esclusivamente sui dati (ad es. K-Nearest Neighbor).
- *Classificatori semi-parametrici*, in cui la classe di modelli della PDF è molto generale e permette di variare il numero di parametri che la definiscono (ad es. reti neurali).

1.5.1 K-Nearest Neighbor

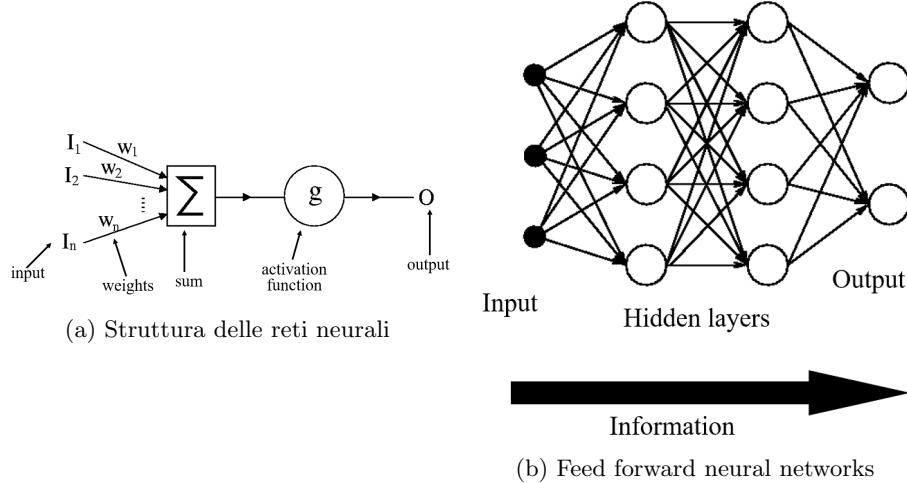
Il *KNN* (K-Nearest Neighbor) è un algoritmo che parte dall'idea che oggetti simili saranno vicini (secondo una certa metrica Φ) anche nello spazio delle features. Quindi, dato un oggetto x_0 e definito un valore intero positivo e non nullo K , si cercano i K elementi più vicini a x_0 nello spazio delle features; x_0 viene associato alla classe C che si presenta maggiormente tra i K oggetti selezionati. Il problema di questo approccio sta nel definire opportunamente K .

1.5.2 Reti neurali

Le *reti neurali* sono sistemi artificiali di elaborazione che emulano il sistema nervoso animale. Si presentano come un modello robusto, compatto, resistente ai guasti e flessibile (si adattano alle nuove situazioni). Inoltre, supportano il calcolo parallelo e permettono di lavorare anche con informazioni incomplete o affette da errore.

Una rete neurale è costituita da: unità elementari chiamate *neuroni*, definiti da una serie di ingressi pesati; un sommatore con la relativa funzione di attivazione; un output. I collegamenti tra neuroni sono detti *sinapsi*. Gli ingressi dei neuroni corrispondono quindi alle features decise dal modello e sono pesati in base all'importanza attribuita a queste ultime. Se i pesi sono associati in modo da costruire un output come combinazione lineare delle features, allora si può parlare di classificazione probabilistica.

Una rete neurale non permette di osservare direttamente i livelli nascosti, e quindi i processi interni al modello, rendendo difficile lo studio della likelihood (ovvero della descrizione della classe). Un esempio di reti neurali è costituito dalle *feed forward neural networks*.



1.6 Combinazioni di classificatori

Esiste la possibilità di effettuare una *combinazione di classificatori*. Infatti, anche se i modelli di classificazione sono costruiti per rispondere ad esigenze specifiche, essi permettono comunque un certo grado di adattamento ai problemi. L'idea è quindi quella di unire vari metodi risolutivi con lo scopo di aumentare le prestazioni di un modello generico. I classificatori possono essere combinati in *parallelo*, in modo *seriale* o in modo *gerarchico*.

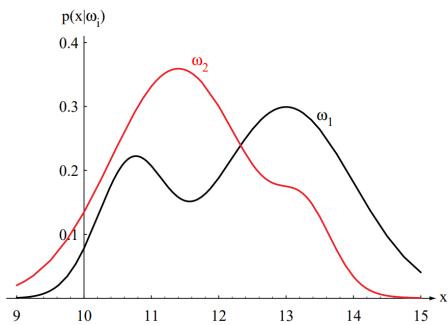
Capitolo 2

Teoria della decisione di Bayes

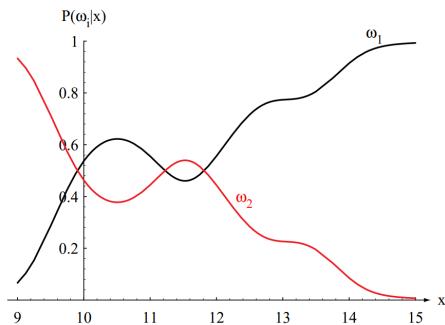
2.1 Introduzione

La teoria di Bayes è l'approccio statistico fondamentale per la classificazione di patterns. Una volta posto un problema in termini probabilistici e conosciute tutte le *probabilità* rilevanti ad esso associate, si ha l'obiettivo di definire una *regola di decisione* usando le probabilità e i costi associati. La *teoria di Bayes* si basa quindi su quattro concetti probabilistici:

- $P(\omega_j) = \text{Prior}$, ossia la probabilità di esistenza di una classe nello stato naturale (o spazio di esistenza).
- $p(x|\omega_j) = \text{Likelihood}$, ossia la probabilità che una certa classe descriva un determinato oggetto (probabilità di misurare x sapendo che lo stato naturale è ω_j).
- $p(x) = \text{Evidence}$, ossia la probabilità di esistenza di un oggetto (fattore di normalizzazione).
- $P(\omega_j|x) = \text{Posterior}$, ossia la probabilità che un oggetto sia descritto da una certa classe (formula di Bayes).



(a) Grafico di likelihood (classi ω_1 e ω_2)



(b) Grafico di posterior (classi ω_1 e ω_2)

2.1.1 Regola di decisione di Bayes

Dalla formula di Bayes

$$P(\omega_j|x) = \frac{p(x|\omega_j) \cdot P(\omega_j)}{p(x)} \equiv \text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

deriva la *regola di decisione di Bayes*, la quale non è altro che la definizione di una soglia che permetterà di selezionare una classe piuttosto che un'altra. Ovvero:

$$\text{Decide } \omega_1 \text{ if } P(\omega_1|x) > P(\omega_2|x), \text{ else } \omega_2$$

La regola di decisione di Bayes minimizza la *probabilità di errore*. Inoltre, rimuovendo la probabilità di evidence, è possibile ottenere una regola di decisione equivalente:

$$\text{Decide } \omega_1 \text{ if } p(x|\omega_1) \cdot P(\omega_1) > p(x|\omega_2) \cdot P(\omega_2), \text{ else } \omega_2$$

Infine, esistono due diverse strategie per definire una regola di decisione:

- *Approccio generativo*, in cui il problema di decisione si divide in due fasi, una di inferenza (addestramento) e una di decisione (basata sull'uso della probabilità a posteriori).
- *Approccio discriminativo*, in cui si genera una funzione (detta funzione discriminante) che permette di classificare gli input mano a mano che vengono letti.

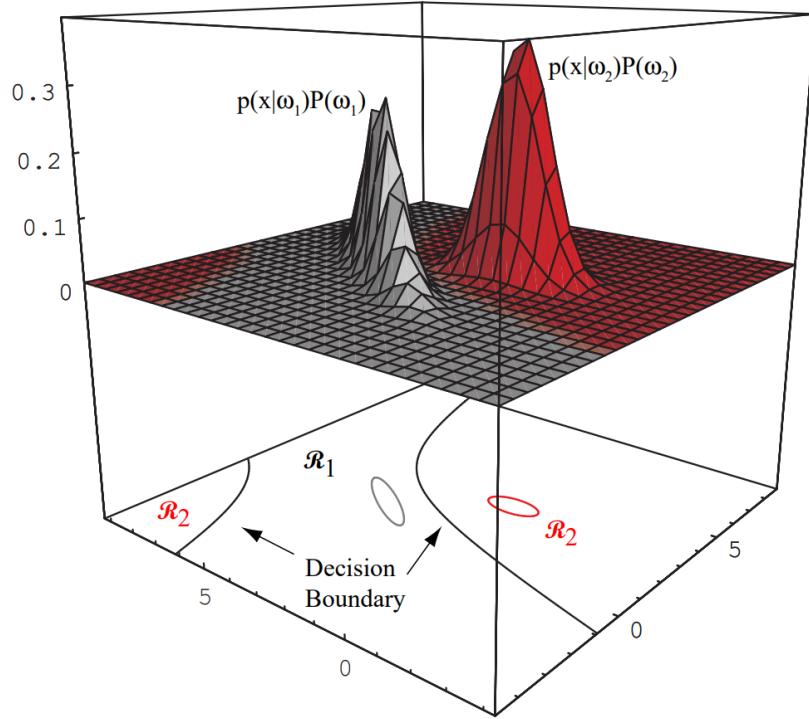
2.2 Funzione discriminante

Una *funzione discriminante* è una funzione che divide lo spazio delle features in due semispazi. Quindi, dato un oggetto in ingresso, a seconda del semispazio in cui viene localizzato si può dire se appartiene ad una certa classe o meno. Per ogni classe è infatti definita una particolare funzione discriminante; l'unione delle funzioni discriminanti permette di dividere lo spazio in maniera complessa.

A patto che ci sia *equivalenza*, una funzione discriminante può essere espressa mediante altre funzioni, come il logaritmo. Il motivo di questa operazione sta nella semplicità con cui è possibile trattare certe funzioni rispetto ad altre. Se g_i è la funzione discriminante della classe ω_i si ha infatti che:

$$g_i(x) = p(x|\omega_i) \cdot P(\omega_i) = \ln p(x|\omega_i) + \ln P(\omega_i)$$

L'effetto finale delle funzioni discriminanti è quello di partizionare lo spazio delle features in *superfici di separazione* (o di decisione). Tali regioni sono separate dai confini di decisione (*decision boundaries*).



2.3 Funzione Gaussiana

La più importante funzione discriminante, nella teoria di Bayes, è la *funzione Gaussiana* (definita come densità normale o Gaussiana multivariata). Essa infatti è analiticamente trattabile e fornisce la miglior modellazione per problemi sia teorici che pratici. Inoltre, la funzione Gaussiana ha le seguenti *proprietà*:

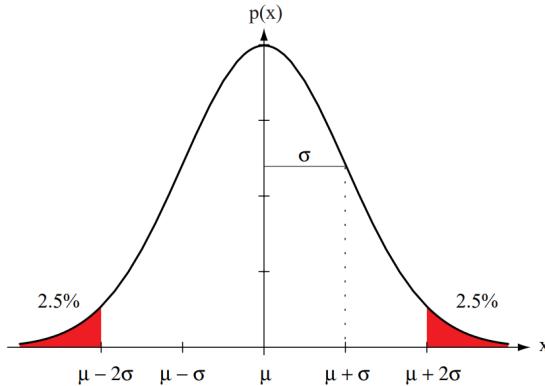
- La sua trasformata di Fourier è sempre una funzione Gaussiana.
- Il prodotto di due funzioni Gaussiane è ancora una funzione Gaussiana.
- È ottimale per la localizzazione in tempo o in frequenza.

2.3.1 Densità normale univariata

La densità normale *univariata* $N(\mu, \sigma^2)$ è specificata da due parametri: la media e la varianza.

$$\begin{aligned} p(x) &= \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \\ \mu &\equiv \mathcal{E}[x] = \int_{-\infty}^{\infty} x \cdot p(x) dx \\ \sigma^2 &\equiv \mathcal{E}[(x - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 \cdot p(x) dx \end{aligned}$$

Inoltre, la densità normale è quella dotata di massima entropia, ovvero è in grado di fornire una descrizione più precisa della distribuzione di un certo fenomeno.



2.3.2 Densità normale multivariata

La densità normale *multivariata* è un'estensione d -dimensionale della densità normale univariata, ed è scritta come:

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

dove μ è un vettore lungo d e Σ è la matrice (quadrata) di covarianza di dimensione d che descrive la distribuzione dei punti nello spazio, ovvero il layout della loro disposizione (ipersfere, iperellissi, eccetera). Analiticamente si ha che:

$$\begin{aligned} \mu &= \frac{1}{N} \cdot \sum_{k=1}^N x^{(k)} \\ \Sigma &= \frac{1}{N} \cdot \sum_{k=1}^N (x_i^{(k)} - \mu_i)(x_j^{(k)} - \mu_j) \end{aligned}$$

La *matrice di covarianza* Σ è simmetrica, semidefinita positiva (determinante maggiore di 0 e autovalori tutti maggiori o uguali a 0) e se due features hanno covarianza nulla questo significa che sono indipendenti. In generale, la matrice di covarianza permette di calcolare la dispersione dei dati in ogni superficie (o sottospazio) descrivendone la distribuzione nello spazio. I suoi autovalori forniscono un valore di dispersione lungo i rispettivi autovettori (assi dello spazio d -dimensionale) indicando di fatto l'importanza data ad ogni feature.

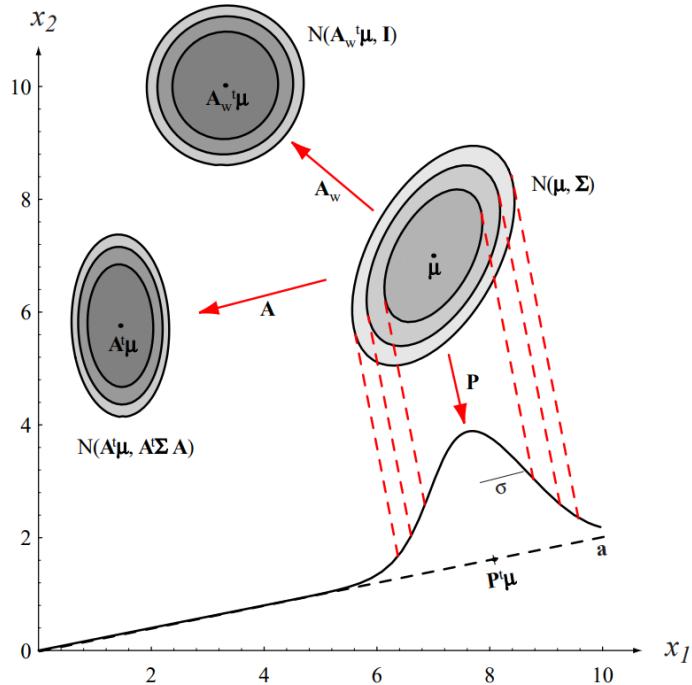
L'obiettivo della classificazione sarà quindi quello di trovare la proiezione migliore, ovvero la distanza minore tra un oggetto e una data classe. Quest'ultima viene chiamata *distanza di Mahalanobis* ed è definita come:

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

Nel caso della normale multivariata, infine, è possibile effettuare una proiezione dello spazio delle features d -dimensionale in uno k -dimensionale (con $k < d$). Se il prodotto della densità univariata per x , componente per componente, è descritto da una distribuzione $P(x) = N(\mu, \Sigma)$ e se data una matrice $A_{d,k}$ si ha che $y = A^T \cdot x$ allora

$$P(y) = N(A^T \cdot \mu, A^T \cdot \Sigma \cdot A)$$

dove $P(y)$ sarà la proiezione di $P(x)$ nello spazio k -dimensionale.



2.4 Funzioni discriminanti - densità normale

La generica formula per la funzione discriminante nel caso della densità normale è data da $g_i(x)$, la quale si traduce come:

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \Sigma_i^{-1} (x - \mu_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma_i| + \ln P(\omega_i)$$

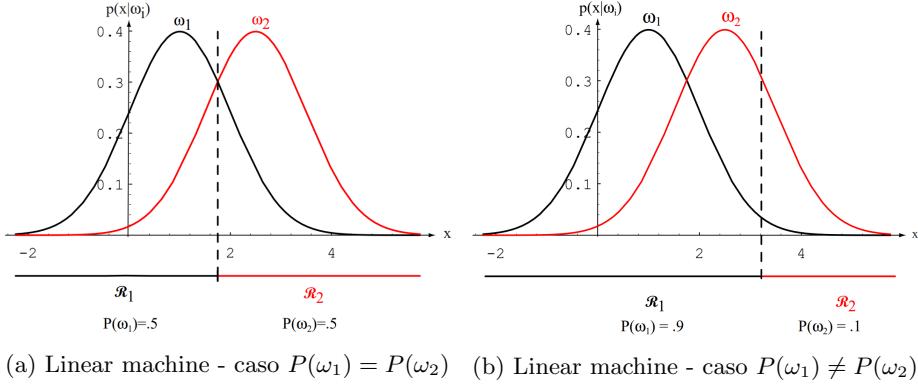
2.4.1 Primo caso: $\Sigma_i = \sigma^2 I$

Nel caso 1-D, in cui la matrice di covarianza è *uguale* per tutte le classi ed è data come $\sigma^2 I$ (ossia quando le features sono indipendenti), la funzione discriminante definisce una retta del tipo $g_i(x) = w_i^t \cdot x + w_{i0}$ dove:

$$w_i = \frac{1}{\sigma^2} \cdot \mu_i$$

$$w_{i0} = -\frac{1}{2\sigma^2} \cdot \mu_i^t \cdot \mu_i + \ln P(\omega_i)$$

Le funzioni di questo tipo sono dette *funzioni discriminanti lineari* o linear machines. Le funzioni discriminanti definiscono generalmente un iperpiano ortogonale alla (iper)linea che unisce le medie delle distribuzioni delle classi. Nel caso in cui le prior risultino uguali, la funzione discriminante si riduce ad un *classificatore di minima distanza*, ovvero si assegna l'elemento x alla classe la cui media μ è la più vicina all'elemento stesso.



2.4.2 Secondo caso: $\Sigma_i = \Sigma$

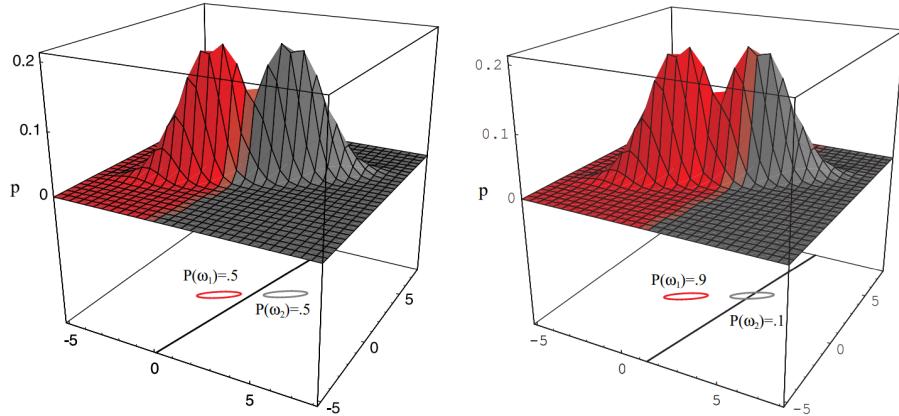
Sempre considerando la densità normale ma nel caso 2-D, in cui la matrice di covarianza è arbitraria ma *uguale* per tutte le classi, $g_i(x)$ può semplificarsi in:

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \Sigma^{-1} (x - \mu_i) + \ln P(\omega_i)$$

Anche in questo caso la funzione discriminante è nella forma di una retta, ma coefficiente angolare e termine noto diventano rispettivamente:

$$\begin{aligned} w_i &= \Sigma^{-1} \cdot \mu_i \\ w_{i0} &= -\frac{1}{2} \cdot \mu_i^t \cdot \Sigma^{-1} \cdot \mu_i + \ln P(\omega_i) \end{aligned}$$

Siccome le funzioni discriminanti sono lineari, i *confini di decisione* sono ancora degli iperpiani.



2.4.3 Terzo caso: Σ_i arbitraria

In quest'ultimo caso le matrici di covarianza sono *diverse* per ciascuna classe. Le funzioni discriminanti sono quindi intrinsecamente quadratiche:

$$g_i(x) = x^t \cdot W_i \cdot x + w_i^t \cdot x + w_{i0}$$

dove

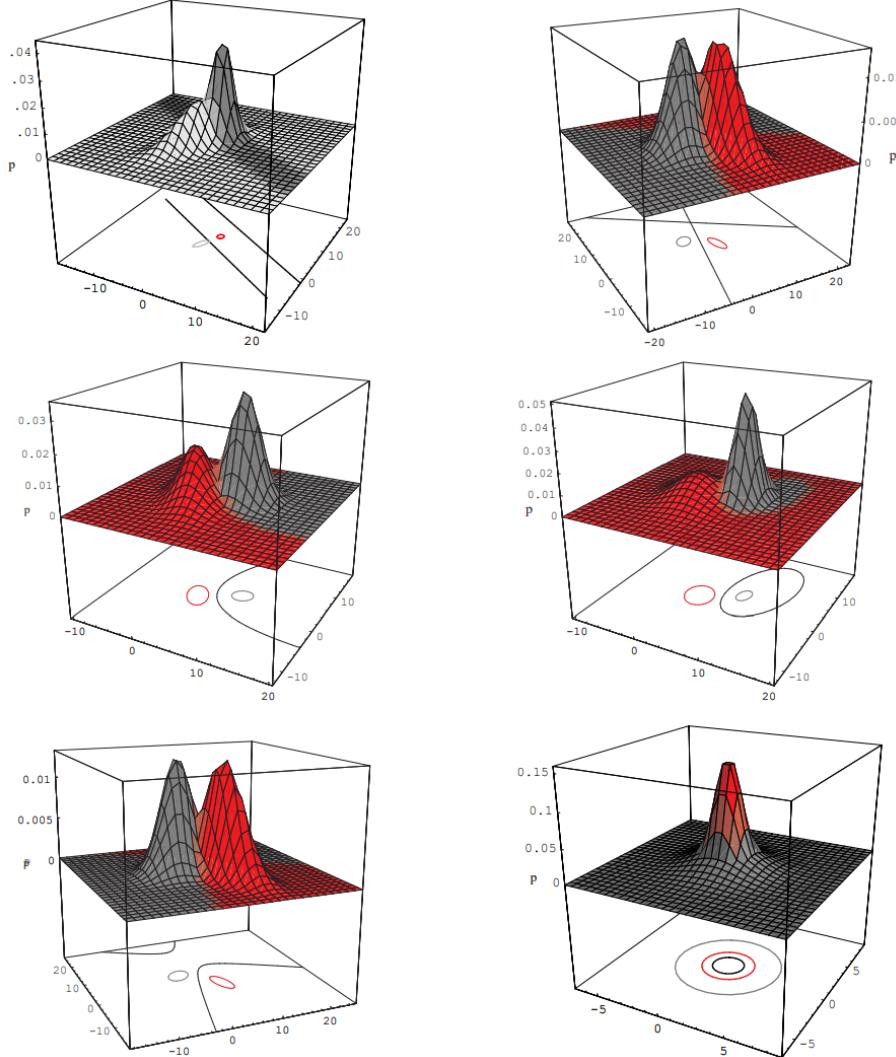
$$W_i = -\frac{1}{2} \cdot \Sigma_i^{-1}$$

$$w_i = \Sigma_i^{-1} \cdot \mu_i$$

e

$$w_{i0} = -\frac{1}{2} \cdot \mu_i^t \cdot \Sigma_i^{-1} \cdot \mu_i - \frac{1}{2} \cdot \ln |\Sigma_i| + \ln P(\omega_i)$$

Nel caso 2-D le *superfici di decisione* sono iperquadratiche e, solitamente, non semplicemente connesse.



Capitolo 3

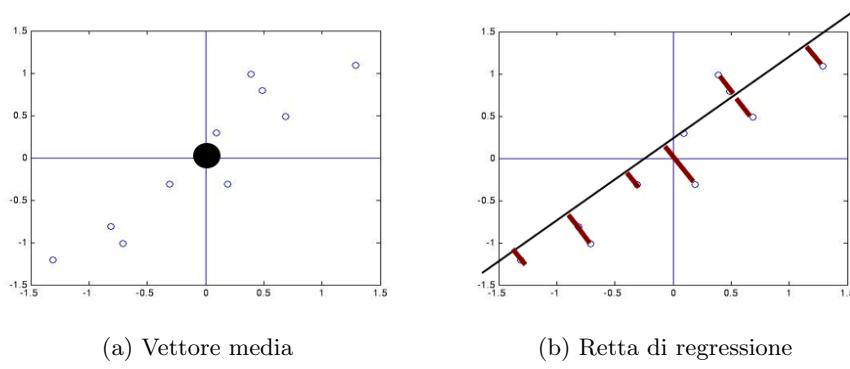
Estrazione delle features

3.1 Principal Component Analysis

La PCA (Principal Component Analysis), è una tecnica utilizzata per descrivere lo spazio delle features facendone un uso ridotto e semplificandone la struttura, ma garantendo un grado di informazione adeguato al problema studiato.

Dato infatti un vettore N-dimensionale di features, lo scopo della PCA è descrivere in maniera compatta i punti (N-dimensional) che popolano lo spazio delle features. A seconda del metodo adottato (ad es. *vettore media*) si può incorrere in una rappresentazione poco espressiva e non adeguata a risolvere il problema.

La PCA parte quindi dall'idea di voler proiettare lo spazio delle features in uno spazio dimensionalmente più compatto, costituito dal sottoinsieme di features considerate più significative. Per fare ciò, si vuole calcolare una *retta di regressione*, ovvero una retta su cui proiettare lo spazio, che passa per la media dei punti e che riduce al minimo lo scarto quadratico medio rispetto alla distanza punti-retta. Questo permette di proiettare le coordinate dei punti in uno spazio dimensionalmente più trattabile e, se possibile, con una distribuzione dei dati più facile da analizzare e quindi classificare.



3.1.1 Passi formali della PCA

1. Ri-denominazione dei punti per specificare la iper-retta:

$$x^{(k)} = m + a^{(k)} \cdot e$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} m_1 \\ \vdots \\ m_M \end{bmatrix} + a \cdot \begin{bmatrix} e_1 \\ \vdots \\ e_M \end{bmatrix}$$

dove x è il vettore di coordinate (features) di un punto, m è un termine noto che corrisponde alla media dei punti studiati, a è il vettore di coefficienti delle coordinate del punto, mentre e è un vettore unitario che indica la direzione (rispetto alla media) in cui si trova il punto x .

2. Trovare i coefficienti $\{a^{(k)}\}_{k=1 \dots M}$ isolando la variabile $a^{(k)}$:

$$a^{(k)} = e^T \cdot (x^{(k)} - m)$$

3. Otttenere il funzionale che descrive il layout degli M punti così da descrivere meglio i punti stessi:

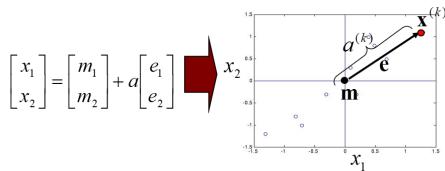
$$J_1(e) = -e^T S e + \sum_{k=1}^M \|x^{(k)} - m\|^2$$

4. Minimizzare $J_1(e)$ massimizzando $e^T S e$ in cui S è la *scatter matrix* (o matrice di covarianza), la quale indica la variazione tra coppie di variabili aleatorie ed è definita come:

$$S = \sum_{k=1}^M (x^{(k)} - m) \cdot (x^{(k)} - m)^T$$

5. Alla fine il termine incognito e sarà un autovettore della matrice di covarianza e , siccome dopo la massimizzazione $e^T S e = \lambda e^T e = \lambda$, λ sarà il massimo autovalore.

In sintesi, la formula del funzionale $J_1(e)$ può essere letta come: calcolata la somma degli scarti quadratici, si sottrae un valore calcolato sulla distribuzione spaziale dei punti; massimizzare questo fattore da sottrarre significa che le proiezioni delle features sui nuovi assi saranno molto diverse fra loro; conseguentemente l'errore quadratico sulla distanza dalla retta è minimo.



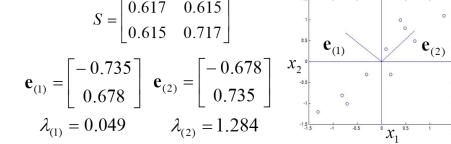
(a) Ri-denominazione dei punti

$$S = \begin{bmatrix} 0.617 & 0.615 \\ 0.615 & 0.717 \end{bmatrix}$$

$$\mathbf{e}_{(1)} = \begin{bmatrix} -0.735 \\ 0.678 \end{bmatrix} \quad \mathbf{e}_{(2)} = \begin{bmatrix} -0.678 \\ 0.735 \end{bmatrix}$$

$$\lambda_{(1)} = 0.049 \quad \lambda_{(2)} = 1.284$$

(b) Autovettori e autovalori



3.1.2 Procedura di feature extraction con PCA

La procedura alla base di PCA si interessa di sapere dove sono i punti (media) e quindi come sono localizzati (retta di proiezione). Le fasi di tale procedura di estrazione delle features sono:

1. Si calcola la media dei dati che vivono nello spazio delle features; solitamente si procede con lo spostamento delle classi di oggetti nell'origine per semplificare i calcoli della *matrice di covarianza*.
2. Presa la matrice di covarianza se ne calcolano *autovettori* e *autovalori*, corrispondenti rispettivamente alle dimensioni dello spazio delle features e ai relativi pesi.
3. Si decide quali features scartare; ad esempio, una volta normalizzati gli autovalori e disposti in ordine decrescente si seleziona il più grande (o i più grandi).

Riguardo all'ultima fase si fa notare che gli autovalori più grandi si avranno nelle direzioni di massima dispersione dei dati, ovvero in quelle corrispondenti alle features più significative. Quindi, i corrispondenti autovettori selezionati identificheranno gli *assi* di un sottospazio trasformato in cui sono proiettati i punti.

3.1.3 Considerazioni su PCA

La PCA parte dal presupposto che i dati siano descrivibili tramite una distribuzione gaussiana e con un numero di features *drammaticamente minore* rispetto a quello di partenza. Per soddisfare queste condizioni, i dati devono essere ragionevolmente *simili* nelle condizioni di acquisizione, ovvero non deve essere necessaria alcuna elaborazione particolare per riuscire a distinguere le classi.

3.1.4 Esempio in MATLAB

```

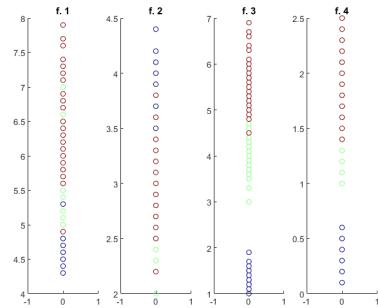
1 load 'irisSet.mat' % X = data, l = labels
2
3 % 0: scatters of the 4 features
4 figure; hold on
5 subplot(1,4,1);
6 scatter(0.*X(1,:), X(1,:), 20, 1);
7 title('f. 1');
8 subplot(1,4,2);
9 scatter(0.*X(2,:), X(2,:), 20, 1);
10 title('f. 2');
11 subplot(1,4,3);
12 scatter(0.*X(3,:), X(3,:), 20, 1);
13 title('f. 3');
14 subplot(1,4,4);
15 scatter(0.*X(4,:), X(4,:), 20, 1);
16 title('f. 4');
17 colormap jet
18

```

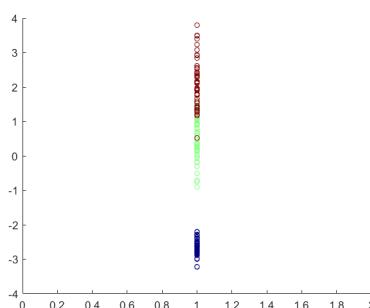
```

19 % Apply PCA on X, in order to have a set of new axis, where each one of them can
20 % be sorted depending on the variance it contains
21 [d,N] = size(X); % number of features, number of points
22
23 % 1a: means of the 4 features
24 u = mean(X,2);
25
26 % 1b: centering
27 h = ones(1,N);
28 B = X-u*h; % translate the points by the means
29
30 % 1c: covariance matrix
31 C = 1/(N-1) * B*B';
32
33 % 2: eigenvectors and eigenvalues
34 [V,D] = eig(C);
35
36 % 3b: sorting eigenvalues
37 D = diag(D);
38 [D,ind] = sort(D,'descend');
39 V = V(:,ind); % V contains the eigenvectors organized by columns (the first
40 % column has the largest eigenvector and so on)
41
42 % 3b: selecting a single dimension (the largest eigenvector)
43 V1 = V(:,1);
44
45 % 3c: projection
46 W = V1'*B;
47
48 % 4: plot on a single dimension
49 figure, scatter(ones(1,N),W, 20, 1)
set(gcf, 'name', '1D projection');
colormap jet

```



(a) Grafico delle 4 features iniziali



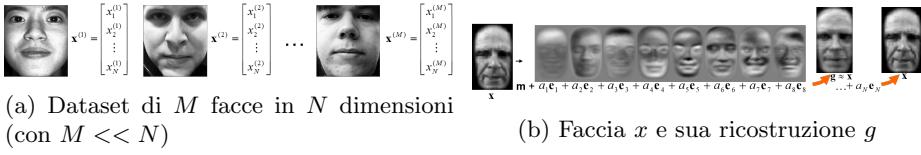
(b) Grafico dei punti proiettati dopo PCA

3.1.5 Da PCA alle eigenfaces

Il problema delle *eigenfaces* è un esempio di applicazione che fa uso di PCA. Una volta ottenuti gli autovettori e (o eigenfaces) dalla matrice di covarianza, posso calcolare le componenti a (o proiezioni) per la faccia x come:

$$x = m + a_1 e_1 + a_2 e_2 + \dots + a_N e_N$$

Gli *autovettori* rappresenteranno quindi varie caratteristiche della faccia (vedi figura (b) sottostante).



3.1.6 Problemi delle eigenfaces

Se A è una matrice di dimensione $N \times M$, allora la matrice di covarianza $S = AA'$ avrà dimensione $N \times N$. Questo porta a problemi di *overflow*: per un'immagine 256×256 avrei infatti una matrice S di dimensione 65536×65536 . Il trucco che si può utilizzare per aggirare questo problema consiste nel calcolare gli autovettori come $A'A$ (che ha una dimensione più gestibile pari a $M \times M$) in quanto nelle fasi successive, generalmente, non vengono usate più di 20-30 eigenfaces. Infatti, gli M autovalori di $A'A$ corrispondono agli M autovalori più grandi di S :

$$A'A\tilde{e} = \lambda\tilde{e} \rightarrow AA'A\tilde{e} = \lambda A\tilde{e} \rightarrow SA\tilde{e} = \lambda A\tilde{e}$$

Dal risultato ottenuto si può osservare che $e = A\tilde{e}$.

3.1.7 Algoritmo per le eigenfaces

1. Sottrarre da ogni immagine $x^{(k)}$ la media $m = \frac{1}{M} \cdot \sum_{k=1}^M x^{(k)}$.
2. Costruire la matrice A dai vettori $\tilde{x}^{(k)}$ ottenuti al passo precedente.
3. Calcolare gli M autovettori \tilde{e}_i di $A'A$ formando la matrice V .
4. Ottenere gli M autovettori più grandi di $S = AA'$ calcolando $e_i = A\tilde{e}_i$ (oppure sotto forma di matrice calcolando $U = AV$).
5. Ottenere i corrispondenti coefficienti di proiezione calcolando $a_i^{(k)} = e_i' \tilde{x}^{(k)}$ (oppure sotto forma di matrice calcolando $\omega^{(k)} = U' \tilde{x}^{(k)}$).

3.2 Linear Discriminant Analysis

La *Linear Discriminant Analysis* (LDA), o Fisher Discriminant Analysis, è una procedura di features extraction che, come PCA, esegue proiezioni lineari. Rispetto a quest'ultima, tuttavia, offre un grado informativo maggiore sulle classi di oggetti da classificare.

LDA effettua una riduzione delle dimensioni dello spazio delle features con lo scopo di mantenere le classi il più possibile compatte (varianza interna piccola) e distanti fra loro (medie distanti). Lo *spazio delle features* è abitato da M oggetti N dimensionali, ognuno dei quali appartiene ad una delle i classi in cui sono stati partizionati. In LDA ogni classe può quindi contenere un numero variabile di oggetti ma, a differenza di PCA, non si fanno assunzioni sulla loro distribuzione. È quindi possibile avere casi in cui due o più classi si sovrappongano parzialmente o totalmente.

Come detto, l'obiettivo di LDA è lo stesso di PCA, ovvero si vuole trovare una *proiezione* w tale per cui le coordinate di un punto x siano trasformate in quelle di un punto $y = w^T x$.

3.2.1 LDA nel caso di due classi

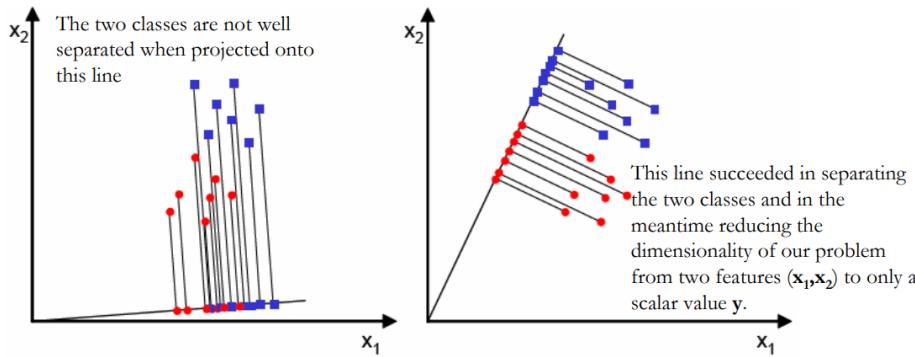
Nel caso in cui si trattano due sole classi, w è un vettore con la stessa dimensionalità del punto x . Per trovare un buon *vettore di proiezione* è quindi necessario definire una misura di separazione tra le proiezioni. Scegliere l'asse su cui le *medie proiettate* sono le più distanti possibili non risulta ragionevole in un'ottica generale in quanto non tiene conto del layout delle classi. Si potrebbe infatti avere una grande distanza tra *medie*, ma la dispersione dei dati potrebbe causare una sovrapposizione eccessiva. Tutti questi concetti sono rappresentabili analiticamente come:

$$\mu_i = \frac{1}{M_i} \cdot \sum_{x \in C_i} x$$

$$\tilde{\mu}_i = \frac{1}{M_i} \cdot \sum_{y \in C_i} y = \frac{1}{M_i} \cdot \sum_{x \in C_i} w^T x = w^T \cdot \frac{1}{M_i} \cdot \sum_{x \in C_i} w^T \mu_i$$

$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |w^T \mu_1 - w^T \mu_2| = |w^T \cdot (\mu_1 - \mu_2)|$$

dove M_i è il numero di oggetti per la classe C_i e $J(W)$ è il funzionale che misura la distanza tra le medie.



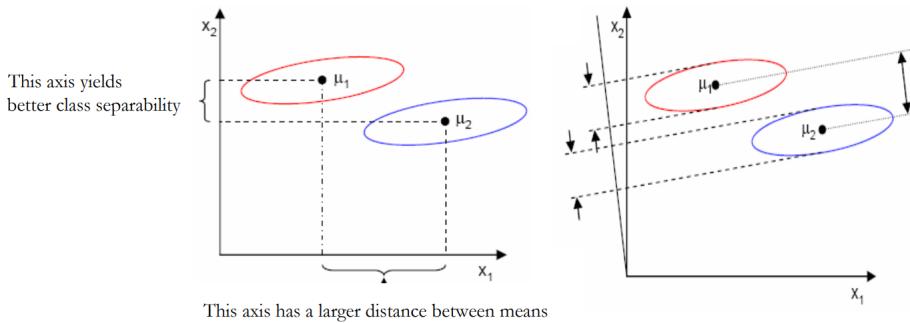
L'idea di Fisher è quella di studiare un funzionale che massimizzi la distanza tra le medie, ma che tenga anche conto della distribuzione dei dati. Per questo, per ogni classe, si calcola lo *scatter* nello spazio proiettato

$$\tilde{s}_i^2 = \sum_{y \in C_i} (y - \tilde{\mu}_i)^2$$

che rappresenta le informazioni sulla dispersione dei dati rispetto alla relativa media della classe di appartenenza. La somma degli scatter di più classi genera la *within-class* scatter, ovvero una misura della compattezza delle proiezioni delle classi che tiene conto di quanto esse siano compatte in termini spaziali (indipendentemente dalle medie). La LDA è quindi una funzione lineare $w^T x$ che massimizza la funzione criterio

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

la quale mostra chiaramente come la massimizzazione sia data nel momento in cui la within-class scatter è piccola (classi compatte) e le medie sono distanti (classi ben separate).



Per trovare la *proiezione ottimale* w^* , dobbiamo esprimere $J(w)$ come funzione esplicita di w . Calcoliamo quindi la within-class scatter matrix S_W definita come somma delle matrici scatter delle singole classi $S_W = S_1 + S_2$, dove:

$$S_i = \sum_{x \in C_i} (x - \mu_i) \cdot (x - \mu_i)^T$$

Ne consegue che

$$\tilde{s}_i^2 = \sum_{x \in C_i} (w^T x - w^T \mu_i)^2 = \sum_{x \in C_i} w^T (x - \mu_i)(x - \mu_i)^T w = w^T S_i w$$

da cui deriviamo la within-class scatter matrix dello spazio proiettato:

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T S_1 w + w^T S_2 w = w^T (S_1 + S_2) w = w^T S_W w = \tilde{S}_W$$

Allo stesso modo è possibile calcolare la *between-class* scatter matrix che mostra la distribuzione delle classi nello spazio proiettato:

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = (w^T \mu_1 - w^T \mu_2)^2 = w^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w = w^T S_B w = \tilde{S}_B$$

A partire da queste considerazioni, si può riscrivere il funzionale come:

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

Massimizzando il criterio di Fisher si riesce a comprendere che è possibile ridursi ad uno spazio con al più $i - 1$ dimensioni, dove i è il numero di classi. La *proiezione ottimale* sarà quindi data come:

$$w^* = \arg \max_w J(w) = \arg \max_w \left(\frac{w^T S_B w}{w^T S_W w} \right)$$

3.2.2 LDA nel caso di C classi

Nel caso in cui si stia considerando un numero di classi $C > 2$ è possibile generalizzare il calcolo delle matrici di scatter in questo modo:

$$S_W = \sum_{i=1}^C S_i$$

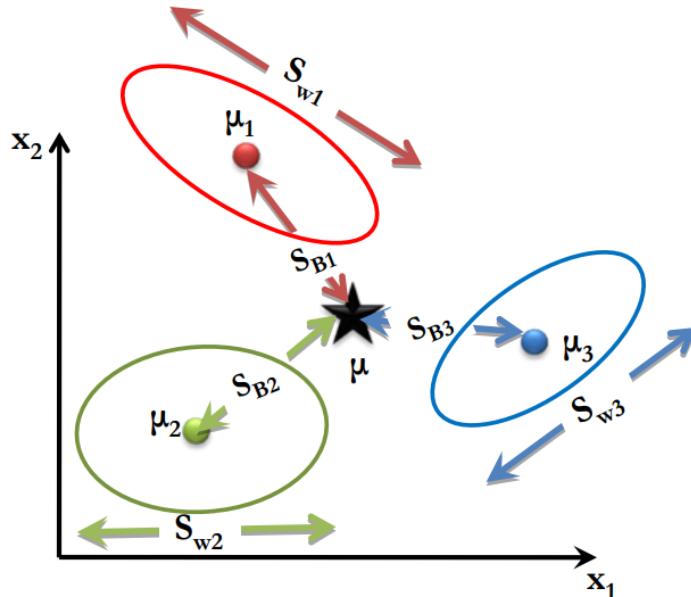
$$S_B = \sum_{i=1}^C M_i(\mu_i - \mu)(\mu_i - \mu)^T$$

dove

$$S_i = \sum_{x \in C_i} (x - \mu_i) \cdot (x - \mu_i)^T$$

$$\mu_i = \frac{1}{M_i} \cdot \sum_{x \in C_i} x$$

$$\mu = \frac{1}{M} \cdot \sum_{\forall x} x = \frac{1}{M} \cdot \sum_{\forall x} M_i \mu_i$$



3.2.3 Esempio in MATLAB

```

1 load 'irisSet.mat' % X = data, l = labels
2
3 % 0: 2D projection
4 figure; hold on
5 subplot(2,3,1);

```

```

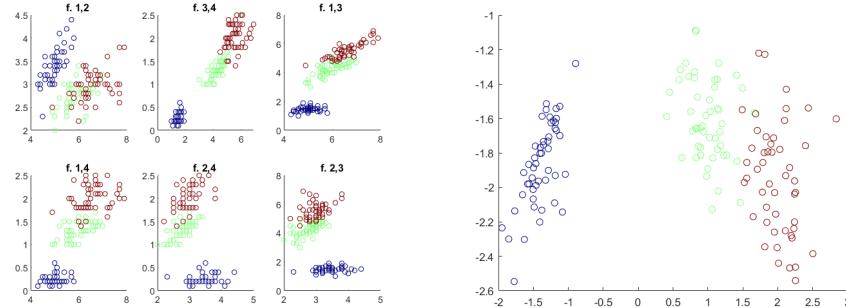
6 scatter(X(1,:),X(2,:), 20, 1);
7 title('f. 1,2 ');
8 subplot(2,3,2);
9 scatter(X(3,:),X(4,:), 20, 1);
10 title('f. 3,4 ');
11 subplot(2,3,3);
12 scatter(X(1,:),X(3,:), 20, 1);
13 title('f. 1,3 ');
14 subplot(2,3,4);
15 scatter(X(1,:),X(4,:), 20, 1);
16 title('f. 1,4 ');
17 subplot(2,3,5);
18 scatter(X(2,:),X(4,:), 20, 1);
19 title('f. 2,4 ');
20 subplot(2,3,6);
21 scatter(X(2,:),X(3,:), 20, 1);
22 title('f. 2,3');
23 colormap jet
24
25 % Apply LDA on X
26 [d,N] = size(X); % number of features, number of points
27 K = length(unique(l)); % number of classes
28
29 % 1: collect the classes Ck
30 for k = 1:K
31     a = find(l == k);
32     Ck{k} = X(:,a);
33 end
34
35 % 2: compute the means
36 for k = 1:K
37     mk{k} = mean(Ck{k},2);
38 end
39
40 % 3: get the cardinalities (for the scatter matrices)
41 for k = 1:K
42     [d, Nk(k)] = size(Ck{k});
43 end
44
45 % 4: within-class covariance for compactness estimation
46 for k = 1:K
47     S{k} = zeros(d,d);
48     for i = 1:Nk(k)
49         S{k} = S{k} + (Ck{k}(:,i)-mk{k})*(Ck{k}(:,i)-mk{k})';
50     end
51     S{k} = S{k}. / Nk(k);
52 end
53 Swx = zeros(d,d);
54 for k = 1:K
55     Swx = Swx + S{k};
56 end

```

```

57
58 % 5: between-class covariance
59 m = mean(X,2);
60 Sbx = zeros(d,d);
61 for k=1:K
62     Sbx = Sbx + Nk(k)*((mk{k} - m)*(mk{k} - m)');
63 end
64
65 % 6: compute the LDA projection
66 MA = inv(Swx)*Sbx;
67
68 % 7: projection vector
69 [V,D] = eig(MA);
70 W = V(:,1:2);
71 Y = W'*X;
72
73 % 8: plot
74 figure, scatter(Y(1,:),Y(2,:),[],1)
75 set(gcf,'name','LDA 2D projection');
76 colormap jet

```



(a) Grafico delle 4 features iniziali (b) Grafico dei punti proiettati dopo LDA

3.2.4 Fisherfaces: algoritmo ed esempio in MATLAB

1. Fare PCA per passare da una matrice AA di dimensione $N \times M$ (M oggetti N dimensional) ad una matrice X di dimensione $low_dim \times M$.
2. Fare LDA per passare dalla matrice X ad una matrice MA di dimensione $low_dim \times low_dim$.
3. Calcolare i low_dim autovettori di MA e tenerne i primi $C - 1$ (con C numero delle classi).
4. Fare la proiezione (vedere la figura alla fine del codice per un esempio di risultato).

```

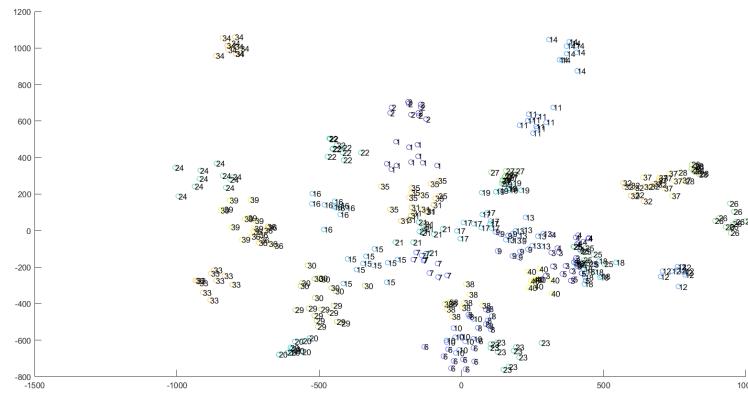
1 dire      =  '.\images'; % path of a directory containing 400 bitmap images
2 list      =  dir(strcat(dire,'*.bmp'));
3 M         =  size(list,1);
4 tmp       =  imread(strcat(dire,'\',list(1).name));
5 [r,c]    =  size(tmp);
6 TMP      =  zeros(r*c,M);
7 for i=1:M
8     tmp = imread(strcat(dire,'\',list(i).name));
9     TMP(:,i) = tmp(:,);
end
10 TMP      =  double(TMP);
11
12
13 % O: perform PCA (N x M -> low_dim x M)
14 media      =  mean(TMP,2);
15 AA(:, :)   =  TMP-repmat(media,1,M);
16 [U,lambda] =  eigen_training(AA); % A'A trick
17
18 T          =  200; % low_dim
19 X          =  U(:,1:T)'*AA; % projection
20 l          =  reshape(repmat([1:40],10,1),400,1);
21
22 % LDA code (low_dim x M -> low_dim x low_dim)
23 [d,N] = size(X);
24 K = max(l);
25 for k = 1:K % 1
26     a = find(l == k);
27     Ck{k} = X(:,a);
end
28 for k = 1:K % 2
29     mk{k} = mean(Ck{k},2);
end
30 for k = 1:K % 3
31     [d, Nk(k)] = size(Ck{k});
end
32 for k = 1:K % 4
33     S{k} = 0;
34     for i = 1:Nk(k)
35         S{k} = S{k} + (Ck{k}(:,i)-mk{k})*(Ck{k}(:,i)-mk{k})';
36     end
37     S{k} = S{k}/Nk(k);
end
38 for k = 1:K % 5
39     Swx = 0;
40     for k = 1:K
41         Swx = Swx + S{k};
end
42     m = mean(X,2);
43     Sbx = 0;
44     for k=1:K
45         Sbx = Sbx + Nk(k)*((mk{k} - m)*(mk{k} - m)');
46     end
47     Sbx = Sbx/K;
48
49
50
51

```

```

52 MA = inv(Swx)*Sbx; % 6
53
54 % C = 40 classes
55 % take the first 39 eigenvectors (C-1)
56 [V,D] = eig(MA);
57 W = V(:,1:39);
58 Y = W'*X; % 7
59
60 % 8: plot
61 figure, scatter(Y(1,:),Y(2,:),[],1)
62 for i=1:M
63     text(Y(1,i),Y(2,i),num2str(l(i)))
64 end
65
66
67 function [U,lambda] = eigen_training(A)
68
69 M = size(A,2);
70 N = size(A,1);
71 L = A'*A;
72
73 % eigenvectors and eigenvalues of A'A
74 [V,values] = eig(L);
75 values = diag(values);
76 [lambda, ind] = sort(values,'descend');
77 V = V(:,ind);
78
79 % eigenvectors of AA'
80 U = A*V;
81 for i=1:M
82     U(:,i)=U(:,i)/norm(U(:,i));
83 end
84
85 end

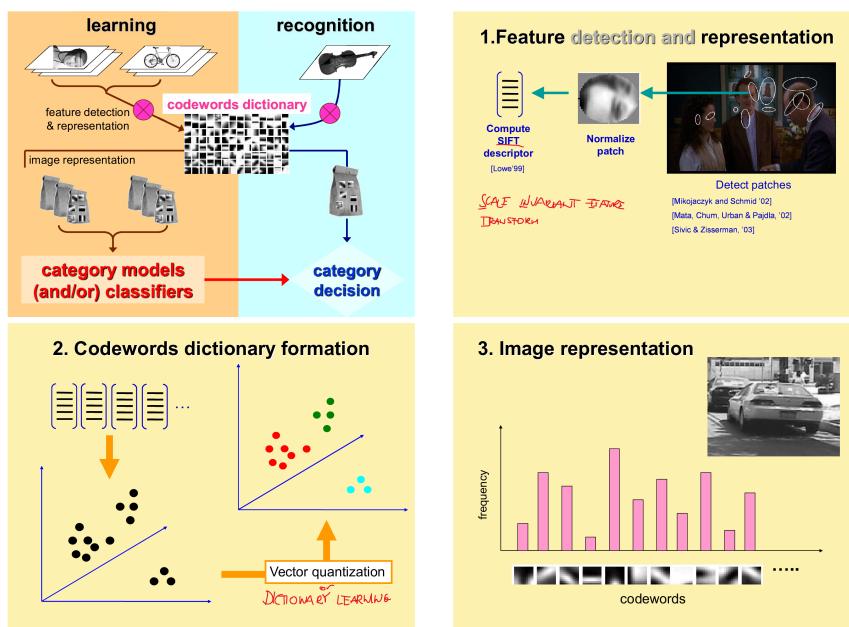
```



3.3 Bag of words

Questa metodologia di classificazione basa il suo funzionamento sul calcolo della *frequenza* con cui un pattern si presenta all'interno di un documento di testo (bag of words) o di un'immagine (bag of visual words). L'idea di fondo è quella di *individuare* degli elementi caratterizzanti da un dataset tramite procedure adatte al problema studiato (suddivisione tramite griglia, individuazione di punti di interesse, campionamento casuale, segmentazione basata su patches, algoritmo SIFT, ecc.).

Poi, tramite il *dictionary learning* si costruiscono i clusters, ovvero dei gruppi naturali di campioni accomunati da un determinato insieme di pattern caratterizzanti. L'estrazione di queste patch permette di generare un dizionario di elementi avente la forma di un *istogramma*. Nella fase di *riconoscimento* ogni elemento viene confrontato con gli elementi estratti dal campione da classificare utilizzando un opportuno modello. Solitamente, il riconoscimento di oggetti incogniti avviene tramite approccio generativo (modelli grafici) o discriminativo (Support Vector Machines, SVM).



Capitolo 4

Learning

4.1 Stima parametrica

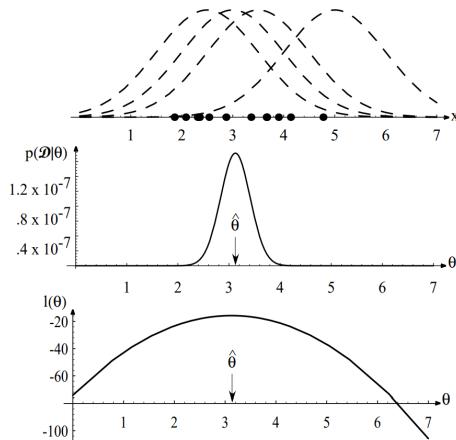
Lo scopo del *learning* è quello di definire delle superfici nello spazio delle features che dividano le diverse classi. In particolare, nel learning *parametrico* si vogliono stimare i parametri di una distribuzione i cui confini sono proprio le superfici. Un possibile approccio per la stima di questi parametri è chiamato maximum likelihood estimation.

4.1.1 Approccio maximum likelihood (caso mono-modale)

Per ipotesi, sappiamo che θ è un set di parametri fissato (ma sconosciuto) tale per cui dato un certo dataset D :

$$p(D|\theta) = \prod_{k=1}^n p(x_k|\theta)$$

Tale funzione di θ è chiamata likelihood, per cui la stima della sua massimizzazione è detta, per definizione, *maximum likelihood* di θ ed è rappresentata dal valore $\hat{\theta}$.



Per scopi analitici, è preferibile scrivere la funzione di likelihood in forma logaritmica (vedi figura precedente). Per cui prima di tutto definiamo il gradiente di $\theta = (\theta_1, \dots, \theta_p)^t$ (in funzione del parametro t) come:

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

La *log-likelihood* sarà quindi espressa come:

$$l(\theta) \equiv \ln p(D|\theta) = \sum_{k=1}^n \ln p(x_k|\theta)$$

A questo punto, per ottenere il set di parametri che massimizza $l(\theta)$, dovremo stimare:

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

Per calcolare questo massimo, facciamo la derivata della log-likelihood

$$\nabla_{\theta} l = \sum_{k=1}^n \nabla_{\theta} \ln p(x_k|\theta)$$

e la poniamo a zero per trovare i punti di minimo/massimo/flesso:

$$\nabla_{\theta} l = 0$$

Nel caso di *distribuzioni multivariate* dovremo quindi stimare μ e Σ . Per semplicità, consideriamo solo μ e concentriamoci su un singolo dato x_i . Avremo:

$$\ln p(x_i|\mu) = -\frac{1}{2} \ln[(2\pi)^d |\Sigma|] - \frac{1}{2} (x_i - \mu)^t \Sigma^{-1} (x_i - \mu)$$

Facendo la derivata passiamo a:

$$\nabla_{\mu} \ln p(x_i|\mu) = \Sigma^{-1} (x_i - \mu)$$

Per calcolare il massimo poniamo la derivata a zero

$$\Sigma^{-1} (x_i - \mu) = 0$$

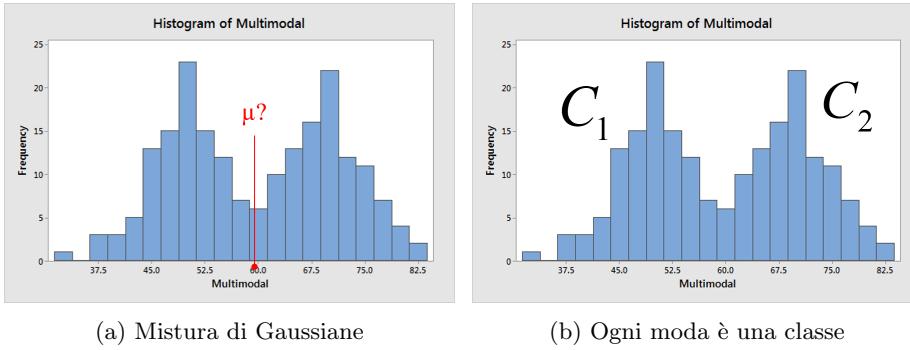
e troviamo che:

$$\hat{\mu} = \frac{1}{n} \cdot \sum_{i=1}^n x_i$$

4.1.2 Approccio maximum likelihood (caso multi-modale)

Nel caso in cui la funzione di likelihood risulta particolarmente complessa (ad esempio è una *mistura di Gaussiane*), non si è in grado di calcolarne la differenziabilità per trovare i punti di interesse.

Per una distribuzione multi-modale, i parametri da stimare saranno un certo insieme ripetuto per ciascuna moda. Si può quindi pensare che ciascuna moda potrebbe essere vista come una *classe* e, secondo lo stesso principio, i parametri delle distribuzioni potrebbero essere visti come un classificatore.



L’assegnazione di ogni campione ad una moda può essere trovata applicando il teorema di Bayes:

$$p(C_j|x_i) = \frac{p(x_i|C_j) \cdot p(C_j)}{p(x_i)} = \frac{p(x_i|C_j) \cdot p(C_j)}{\sum_j p(x_i|C_j) \cdot p(C_j)}$$

Per studiare la probabilità a posteriori dovremo stimare le probabilità a priori $p(C_j)$ e le funzioni di likelihood $p(x_i|C_j)$. Di solito, le probabilità a priori vengono fornite e perciò non è necessario stimarle:

$$p(C_j) = \frac{n_j}{n}$$

dove n_j è il numero di campioni con classe C_j .

Per stimare le funzioni di likelihood, invece, dobbiamo prima stimare il tipo di funzione $p(x_i|C_j)$ da utilizzare (Gaussian, esponenziale, ecc.) e poi stimarne i parametri. Nel caso della Gaussian, la distribuzione da stimare sarebbe:

$$\sum_{j=1}^K \pi_j \cdot \mathcal{N}(\mu_j, \Sigma_j)$$

La stima di questa distribuzione è decisamente più difficile in quanto, dopo aver derivato per ogni singolo parametro, non si riesce ad ottenere un sistema lineare chiuso (ovvero non si riescono a disaccoppiare i parametri).

In questi casi si adotta un algoritmo ausiliario chiamato *Expectation Maximization* (EM), il quale prende il nome dalle due fasi in cui si articola:

1. Assumere di focalizzarsi su ogni moda della densità valutando $p(C_j|x_i)$ per ogni j (fase di Expectation).
2. Calcolare i parametri per ogni moda C_j (fase di Maximization).
3. Ripetere i punti 1 e 2 fino alla convergenza dei parametri.

4.1.3 K-means

Prima di mostrare il funzionamento dell’algoritmo EM diamo uno sguardo a *K-means*. Questo algoritmo è una variante più semplice di EM, in quanto stima solo la media μ_j .

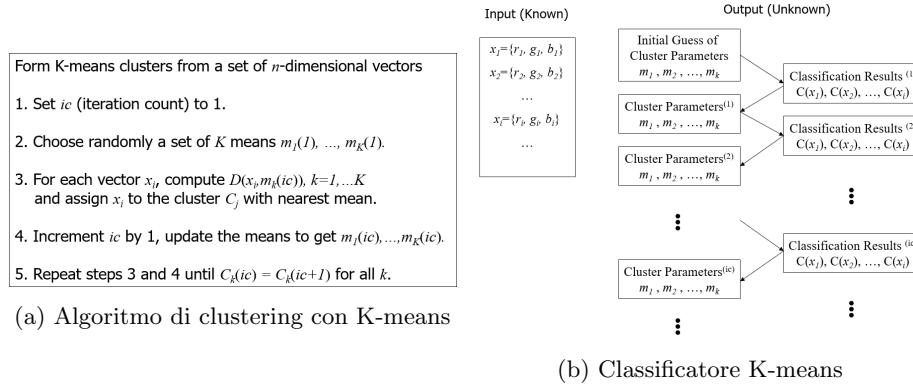
K-means può quindi essere visto come un algoritmo EM in cui le due fasi si mappano in:

1. Calcolare $p(C_j|x_i)$ come la distanza euclidea d di x_i rispetto a μ_j .
2. Associare ogni x_i alla classe \tilde{C}_j per cui vale

$$\tilde{C}_j = \arg \max_{C_j} p(C_j|x_i) = \arg \min_{\mu_j} d(x_i, \mu_j)$$

3. Ricalcolare μ_j per ogni j .

A partire da una clusterizzazione iniziale (il numero di clusters deve essere fissato a priori), l'*algoritmo K-means* assegna ad ogni iterazione tutti i pattern al cluster con la media più vicina e poi aggiorna le medie, terminando nel caso in cui sia giunto a convergenza. Questo algoritmo ha un'efficienza lineare nella dimensione del dataset, perciò risulta ottimo per grandi insiemi di dati.

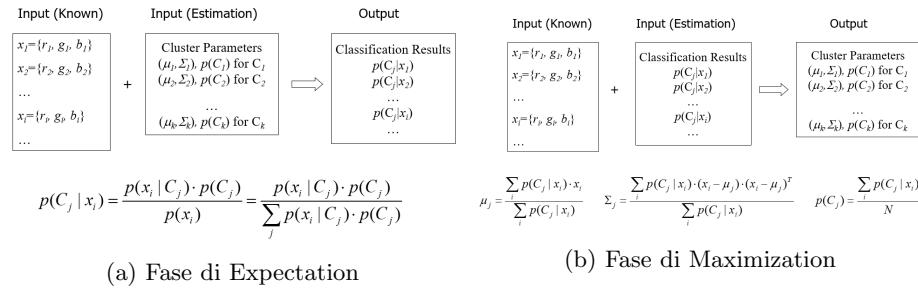


4.1.4 Algoritmo EM

L'*algoritmo EM* tende a convergere ad un massimo locale adottando una procedura iterativa, che ha lo scopo di calcolare la maximum-likelihood, in cui si alternano due fasi:

1. Quella di *Expectation*, che crea una funzione di stima per la media della log-likelihood a partire dagli ultimi parametri stimati.
2. Quella di *Maximization*, che calcola i parametri che massimizzano tale funzione; questi ultimi verranno poi riutilizzati per eseguire la successiva fase di Expectation.

Il caso tipico in cui si usa EM è quello in cui i dati sono distribuiti secondo una *mistura di Gaussiane*. Infatti, EM analizza i dati che si sovrappongono e, col procedere delle iterazioni, identifica i valori massimi che corrispondono alle medie delle Gaussiane presenti nella mistura.



4.2 Stima non parametrica

Nel *learning non parametrico* non si fanno assunzioni sulla forma delle densità di probabilità, in quanto potrebbero essere sovra-strutturate. L'idea è quindi quella di studiare le regioni in cui si suddivide lo spazio. Infatti, se mi interessa studiare la probabilità $P(x|C)$, andrò ad osservare la regione R dello spazio in cui l'oggetto è localizzato. Un problema di questo approccio consiste nella stima di $P(x|C) = P(x)$. I principali metodi utilizzabili per stimare $P(x)$ (per ogni possibile x) sono:

- *Parzen Windows*, in cui, fissata la regione R centrata in x , si calcola la kernel function K dai dati e si stima $P(x)$ (più punti sono presenti in R e migliore sarà la stima).
 - *Meanshift*, in cui si cerca il centro di massa dei punti contenuti nella regione di interesse R andando a calcolare il meanshift vector, il quale collega il centro della regione con il centro di massa appena calcolato (quest'ultimo diventerà poi il centro di R al prossimo step).

4.2.1 Parzen Windows

Per ottenere la *kernel function* K (in forma analitica) ci affidiamo ad una window function che rappresenta la regione R tale per cui, dato un punto x_i , se questo cade nella regione allora la funzione ritorna 1, altrimenti ritorna 0. Grazie a ciò, il metodo Parzen Windows riesce a garantire una serie di vantaggi:

- Permette più tipi di window functions.
 - Siccome la stima della PDF è data dalla media della window function su x e x_i , ogni campione contribuisce, a seconda della distanza dalla media x , alla forma della PDF.

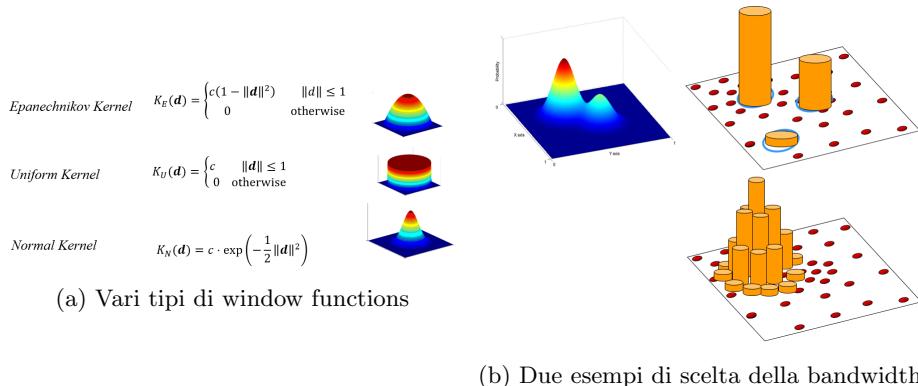
La formula finale della probabilità che un punto x si trovi in una determinata regione R è data quindi da:

$$P(x) = \frac{1}{n} \cdot \sum_{i=1}^n K(x - x_i)$$

Per riuscire a fornire una buona approssimazione, la funzione K deve avere le seguenti proprietà:

- $K(x - x_i)$ deve essere maggiore o uguale a 0.
- $\arg \max_x K(x - x_i) = x_i$ si deve avere per x_i uguale alla media x .
- Deve essere simmetrica, ovvero $K(x - x_i) = K(-x + x_i)$.
- Deve essere normalizzata, ovvero $\int K(x - x_i) dx = 1$.

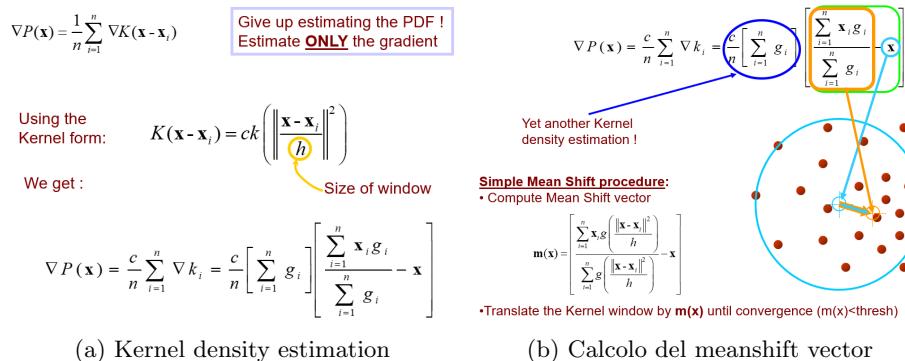
In Parzen Windows la scelta della *bandwidth* è cruciale: se è troppo grande, si avrà una PDF poco informativa; se è troppo piccola, si otterrà una distribuzione di picchi localizzati sui punti stessi.



4.2.2 Meanshift

Meanshift è una tecnica per la ricerca della regione (locale) più densa in un set di punti conosciuto. Questa tecnica evidenzia quindi la forma di una PDF in uno spazio d -dimensionale (dimensionalità di R). Le caratteristiche rappresentate possono essere di tipo posizionale, colorimetrico o altro.

La stima della densità dei punti nella regione studiata è detta *kernel density estimation*, in quanto la densità dei punti in R è calcolata sulla base di una funzione kernel K .



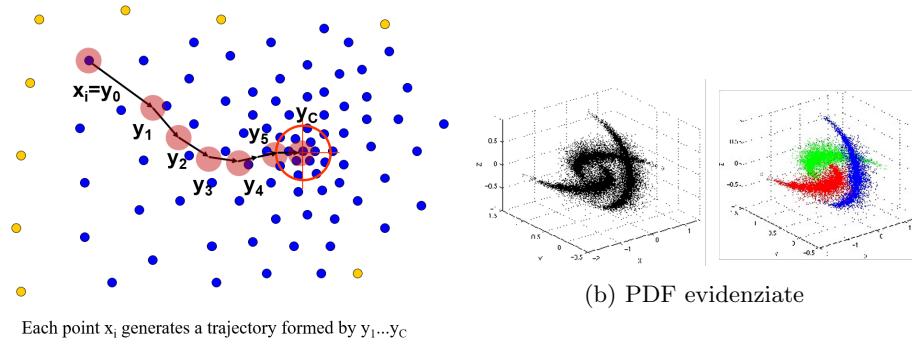
Siccome meanshift lavora con il gradiente del kernel, è possibile incappare in casi limite. È quindi più giusto parlare del meanshift come una tecnica non parametrica di stima del gradiente di densità. Le *proprietà* garantite da questo approccio sono:

- *Velocità di convergenza automatica*, ossia il meanshift vector varia (in modulo) automaticamente in quanto meanshift è una tecnica di ascesa del gradiente adattiva.
- *Infinitamente convergente*, ossia la convergenza è garantita solo per step infinitesimali (solitamente si pone una soglia alla lunghezza del meanshift vector).
- Un *kernel uniforme* garantisce la convergenza in un numero finito di passi.
- Un *kernel gaussiano* ritorna una traiettoria più smooth ma impiega più tempo rispetto a quello uniforme.

Inoltre, i *vantaggi* di meanshift sono che:

- È indipendente rispetto all'applicazione che si sta realizzando.
- È adatto per l'analisi di dati reali.
- Non necessita di assunzioni a priori sulla forma della distribuzione dei dati.
- Può gestire spazi delle features arbitrari.
- Richiede di scegliere un solo parametro, ossia la dimensione della finestra.

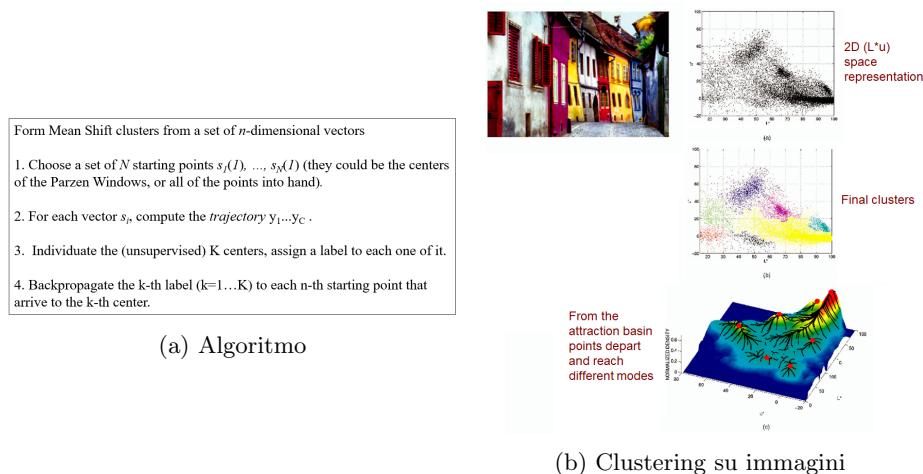
Infine il *lato negativo*, che è proprio scegliere la dimensione della finestra, in quanto una scelta non felice potrebbe provocare un'errata ricerca del centro di massa. Una metodologia che consente di risolvere questo problema consiste nell'adattare la dimensione della finestra in base allo stato in cui ci si trova.



4.2.3 Clustering con meanshift

Normalmente, meanshift si applica tassellando lo spazio con una serie di finestre e lanciando una procedura in parallelo per evidenziare tante traiettorie quante sono le finestre. Ogni finestra tende a seguire la direzione di convergenza più veloce, ovvero ogni finestra ricerca il centro di massa più vicino (massimo locale della PDF). Questa peculiarità garantisce, in base al numero di procedure lanciate in parallelo, un uso ottimale in uno studio legato al *clustering* dei dati. Infatti, la divisione in clusters avverrà considerando le traiettorie identificate dalle varie procedure.

Meanshift preserva le discontinuità. Nel caso delle immagini, quindi, viene lanciata per ogni pixel una procedura di meanshift (in cui la funzione kernel è pesata in funzione delle coordinate spaziali e colorimetriche) la quale arriverà ad una certa moda (centro di massa locale). Si assegna poi al pixel di partenza il valore (colore) della moda trovata. Ne consegue che meanshift permette la *segmentazione delle immagini*, mantenendo le discontinuità e applicando una procedura di smoothing (ogni pixel assume il valore della moda a cui è collegato).



4.3 Tracking (stima di densità dinamiche)

Lo scopo del *tracking* è quello di capire dove si spostano gli oggetti in una sequenza, stimandone così la traiettoria. Le operazioni di tracking si articolano in una prima fase di *inizializzazione*, dove si identificano i nuovi oggetti che entrano in scena (lo scopo è trovare un oggetto da seguire), e in una fase di *inseguimento*, dove si vuole mantenere l'attenzione sui soggetti (lo scopo è riconoscerne l'identità). Gli approcci alla fase di inseguimento sono di due tipi:

- Metodologia forza bruta, in cui si sfrutta la correlazione e la definizione di vincoli.
- Metodologia classica, in cui si fa predizione, verifica e associazione dei valori calcolati.

Tra le tecniche adottate nella fase di inizializzazione, invece, ci sono il template matching (con l'output di un classificatore come face detector o hand detector), la sottrazione del background e alcuni metodi ibridi.

4.3.1 Inseguimento

La fase di inseguimento vuole mantenere l'identità degli oggetti identificati, così da poterne tracciare e stimare la traiettoria di movimento. Il problema è trovare un metodo che permetta di associare gli oggetti identificati in un dato frame con quelli del frame successivo. Il metodo di forza bruta, basato sulla ricerca esaustiva dei singoli oggetti, risulta lento e impreciso. L'aggiunta di vincoli sulla scena e sulla natura degli oggetti, invece, permette una stima della posizione dei soggetti, pesando di conseguenza la correlazione in modo opportuno. Le tecniche di *particle filtering* come il condensation (o metodo di Montecarlo) sono attualmente i metodi di stima più adottati.

4.3.2 Particle filtering

L'approccio tramite particle filtering parte da diverse *assunzioni*:

- Lo stato del sistema studiato all'istante di tempo t è definito dalla posizione degli M punti $X_t = \{x_{1,t}, x_{2,t}, \dots, x_{M,t}\}$.
- La storia (discreta) del sistema fino al tempo t , che indica un'evoluzione del sistema nel tempo, è definita come $\mathcal{X}_t = \{X_1, X_2, \dots, X_t\}$.
- Lo stato del sistema fino al tempo t è osservabile da un set di osservazioni (immagini/frame) $\mathcal{Z}_t = \{Z_1, Z_2, \dots, Z_t\}$.
- Le osservazioni (vedi punto precedente) non modificano lo stato del sistema

$$p(\mathcal{Z}_{t-1}, X_t | \mathcal{X}_{t-1}) = p(X_t | \mathcal{X}_{t-1}) \cdot \prod_{i=1}^{t-1} p(Z_i | X_i)$$

e sono indipendenti fra loro

$$p(\mathcal{Z}_t | \mathcal{X}_t) = \prod_{i=1}^t p(Z_i | X_i)$$

- L'evoluzione del sistema è considerato un processo stocastico Markoviano tempo-indipendente di ordine 1 $P(X_t | \mathcal{X}_t) = P(X_t | X_{t-1})$.

Quindi, l'*obiettivo* del particle filtering è quello di stimare lo stato più probabile (atteso) del sistema X_t date tutte le osservazioni \mathcal{Z}_t . Per fare questo, è necessario studiare la seguente probabilità (in forma Bayesiana):

$$p(X_t | \mathcal{Z}_t) = k_t \cdot p(Z_t | X_t) \cdot p(X_t | \mathcal{Z}_{t-1})$$

dove

$$p(X_t | \mathcal{Z}_{t-1}) = \int_{X_{t-1}} p(X_t | X_{t-1}) \cdot p(X_{t-1} | \mathcal{Z}_{t-1}) dX_{t-1}$$

Il controllo ad ogni istante di tempo per questa probabilità corrisponde alla valutazione dell'evoluzione di una distribuzione nel tempo.

4.3.3 Algoritmo di Condensation

L'algoritmo di Condensation è una tecnica di particle filtering che si basa sul concetto di localizzazione spaziale dello stato del sistema. Assumendo di poter rappresentare in uno spazio (campo di esistenza R) finito e monodimensionale lo stato del sistema, si può affermare che lo stato è rappresentabile come un punto su R e quindi che il sistema è una particella puntiforme su tale campo di esistenza.

La fase di *inizializzazione* prevede che siano dati N campioni (particelle su R) che rappresentano i possibili stati del mio sistema ad un certo istante $t - 1$; tale set di particelle è definito come:

$$\{s_{t-1}^{(n)}\}$$

All'istante iniziale si settano i pesi per tutte le particelle (ogni campione rappresenta un intorno spaziale locale); tale set di pesi è definito come:

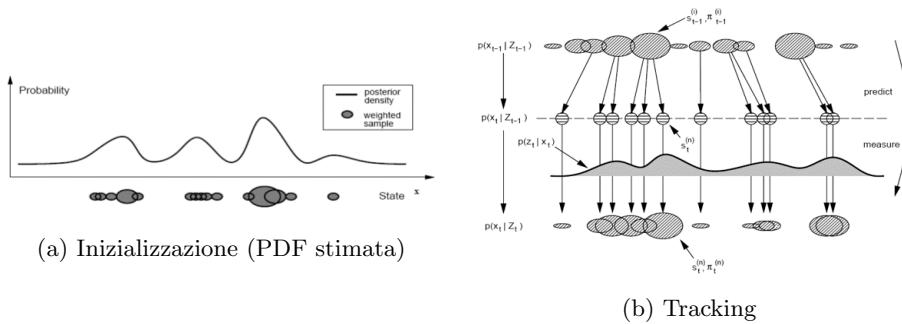
$$\{\pi_{t-1}^{(n)}\}$$

L'assegnazione dei pesi alle particelle permette di stimare una PDF su tutto R .

Finita l'inizializzazione inizia la fase di *tracking*, in cui il particle filtering fa evolvere il set di particelle in tre passi.

1. *Selezione o campionamento*: a partire dalla PDF al tempo $t - 1$, estraggo N nuove particelle (ogni particella ha probabilità di estrazione pari al suo peso).
2. *Applicazione della dinamica*: ad ognuna delle nuove particelle applico un moto con una componente deterministica (basata sul passo precedente) e una probabilistica aggiungendo del rumore gaussiano per modellare l'incertezza sul moto; questo permette di calcolare la parte predittiva (a priori) $p(X_t | \mathcal{Z}_{t-1})$ della formulazione Bayesiana del filtro.
3. *Valutazione/pesatura*: a partire dalle osservazioni, calcolo la likelihood di ogni campione.

Con la nuova PDF calcolata si associano i pesi (normalizzati) alle nuove particelle. A questo punto basta prendere una *decisione* sulla posizione del soggetto ad un certo istante t , ad esempio si può effettuare un calcolo basato sulla media pesata oppure tramite MAP (Maximum A Posteriori).



4.3.4 Campionamento delle particelle

Uno dei problemi da affrontare è il metodo di *campionamento* delle particelle. Presi tutti i pesi dei campioni, ne faccio la somma cumulativa, ottenendo così una serie di coefficienti

$$\{c_t^{(n)}\}$$

Questo permette di effettuare i calcoli in modo efficiente (complessità pari a $O(N \log N)$). Infatti, generato casualmente un valore intero $r \in [0, 1]$, si cerca per divisione binaria il più piccolo indice j tale per cui

$$c_{t-1}^{(j)} \geq r$$

e infine si setta

$$s_t'^{(n)} = s_{t-1}^{(j)}$$

4.3.5 Multi-object tracking

Per quel che riguarda il tracking, l'algoritmo di Condensation tende effettivamente a seguire un unico oggetto. Quindi la robustezza di questa fase è vincolata alla stima della *dinamica*, che tipicamente è fatta off-line (via correlazione esaustiva o manualmente) così da averla il più precisa possibile.

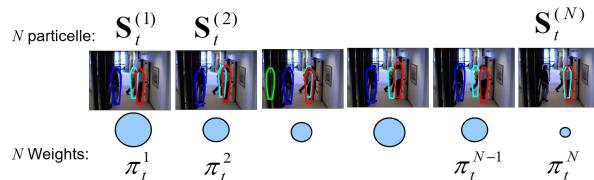
Il *metodo Bramble* (Bayesian Multiple-Blob) è un metodo di tracking che può essere considerato come l'evoluzione del Condensation (Condensation multi-oggetto). L'idea consiste, come per il Condensation, di esprimere la probabilità di stato in funzione delle osservazioni. Tuttavia, lo stato del sistema non è più costituito solo dalle posizioni degli oggetti ma da una serie di parametri, ovvero il numero di oggetti e lo stato degli oggetti: $X_t = (m, x_t^1, \dots, x_t^m)$. Ogni oggetto è poi una struttura di parametri $x = (i, l, v, s)$ dove:

- i è l'identità, ossia un indice identificativo.
- l è la posizione (locazione), ossia le coordinate spaziali.
- v è la velocità (espressa come vettore).
- s è l'aspetto, ossia un fattore legato all'occlusione.

Lo stato del sistema X_t è approssimato con un set di N particelle (configurazioni):

$$\{S_t^{(n)}, \pi_t^{(n)}\}$$

Le osservazioni prevedono un campionamento delle immagini (frame) sufficiente a poter valutare il filtraggio in tempo reale. Ogni cella (o filtro) corrisponde ad una determinata posizione spaziale che avrà di conseguenza una particolare probabilità di successo (peso).



Capitolo 5

Classificatori generativi

5.1 Processi di Markov

Un *processo di Markov* è costituito da N stati s_1, \dots, s_N ed è caratterizzato da un certo numero di passi discreti. La probabilità di partire da un determinato stato è dettata dalla distribuzione $\Pi = \{\pi_i\} : \pi_i = P(q_1 = s_i)$ dove $1 \leq i \leq N$, $\pi_i \geq 0$ e $\sum_{i=1}^N \pi_i = 1$.

Al t -esimo istante il processo si trova esattamente in uno degli stati a disposizione, indicato dalla variabile q_t . Ad ogni iterazione, lo stato successivo viene scelto con una certa probabilità. Tale probabilità è unicamente determinata dallo stato precedente; per questo si parla di Markovianità del primo ordine.

$$P(q_{t+1} = s_j | q_t = s_i, q_{t-1} = s_k, \dots, q_1 = s_l) = P(q_{t+1} = s_j | q_t = s_i)$$

Per descrivere un processo Markoviano si costruisce la matrice A detta di transizione tra stati (invariante nel tempo), la quale è una matrice $N \times N$ in cui ogni elemento è definito come $a_{i,j} = P(q_{t+1} = s_j | q_t = s_i)$. Quindi, i processi Markoviani (discreti) sono caratterizzati da:

- Markovianità del primo ordine.
- Stazionarietà.
- Avere una distribuzione iniziale.

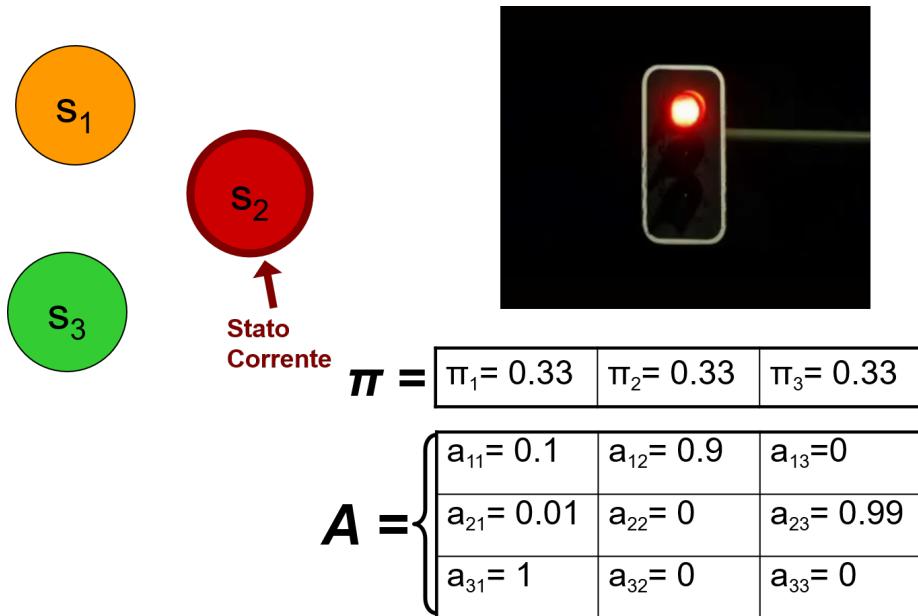
5.1.1 Modelli di Markov

Conoscendo le caratteristiche dei processi di Markov è possibile esibire un modello probabilistico di Markov (MM), definito come $\lambda = (A, \pi)$. Un *modello di Markov* è un modello stocastico che permette di descrivere tramite probabilità le cause che portano da uno stato all'altro del sistema, definendo così delle relazioni (probabilistiche) di causa-effetto tra gli stati. Su questo tipo di modelli si possono effettuare operazioni di *training* (costruendo gli elementi costituenti del modello) e operazioni di *interrogazione* (come per esempio richiedere la probabilità di avere una certa sequenza di stati).

I modelli di Markov modellano comportamenti stocastici Markoviani (di ordine N) di un sistema in cui gli stati sono:

- *Esplicativi*, ovvero riesco a dar loro un nome.
- *Osservabili*, ovvero ho delle osservazioni che identificano univocamente lo stato.

Per esempio, in un sistema *semaforo* gli stati sono tre e sono esplicativi (lampade accese/spente) e osservabili (colori delle lampade). Inoltre, ogni stato ha la stessa probabilità iniziale e la matrice di transizione A definisce le probabilità di passare da rosso a verde, da rosso a giallo, da verde a giallo, e così via.



5.1.2 Inferenze

Per definire le relazioni probabilistiche dell’evoluzione di un modello di Markov (ovvero di una sequenza di stati da q_1 a q_t) possiamo scrivere:

$$P(q_t, \dots, q_1) = P(q_t|q_{t-1})P(q_{t-1}|q_{t-2}) \dots P(q_2|q_1)P(q_1)$$

Inoltre, il calcolo della probabilità di trovarsi in un determinato stato in un certo istante di tempo è $P(q_t = s_j)$. La sua valutazione si articola in due step:

1. Valuto come calcolare $P(Q)$ per ogni cammino di stati $Q = \{q_1, q_2, \dots, q_t = s_j\}$, ossia $P(q_t, q_{t-1}, \dots, q_1)$.
2. Calcolo $P(q_t = s_j) = \sum_Q P(Q)$ dove con Q si indicano tutti i cammini di lunghezza t che finiscono in s_j .

Questa valutazione risulta tuttavia molto onerosa (complessità pari a $O(N^t)$).

L'idea è quindi quella di riscrivere la probabilità $P(q_t = s_j)$ come $p_t(j)$, da cui possiamo definire induttivamente

$$\forall i p_1(i) = \begin{cases} \pi_i & \text{se } s_i \text{ è lo stato in cui mi trovo} \\ 0 & \text{altrimenti} \end{cases}$$

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

e ottenere

$$\sum_{i=1}^N P(q_{t+1} = s_j | q_t = s_i) \cdot P(q_t = s_i) = \sum_{i=1}^N a_{ij} \cdot p_t(i)$$

Se si usa questo metodo partendo da $P(q_t = s_j)$ e procedendo a ritroso, il costo della computazione diventa $O(tN^2)$.

5.1.3 Limiti dei modelli Markoviani

Un limite dei modelli Markoviani è la necessità di avere stati *osservabili deterministicamente*, e questo non sempre è possibile. Se ho un sistema dove ci sono stati non esplicativi posso tuttavia studiarne la natura osservandone l'*evoluzione*. A questo punto non conosco ancora quali siano gli stati regolatori, ma posso determinarne il numero (e la categoria) *osservando* il sistema (introduco una conoscenza a priori). A partire dalle osservazioni stabilisco poi una relazione tra osservazione e stato nascosto.

5.2 Hidden Markov Models

Tra i principali modelli generativi troviamo i modelli Markoviani a stati nascosti (HMM). Gli *HMM* hanno lo scopo di descrivere probabilisticamente la dinamica di un sistema tralasciando la definizione delle cause che ne regolano l'evoluzione. Gli stati vengono quindi identificati solo tramite le osservazioni e in maniera probabilistica. In sostanza, gli HMM possono essere considerati come un'estensione dei modelli di Markov, in cui l'unica differenza sta nella non osservabilità degli stati. Un HMM (discreto e addestrato) è formato da:

- Un insieme $S = \{s_1, s_2, \dots, s_N\}$ di stati nascosti.
- Una matrice di transizione $A = \{a_{ij}\}$ tra stati nascosti, dove $1 \leq i, j \leq N$.
- Una distribuzione iniziale sugli stati nascosti $\pi = \{\pi_i\}$.
- Un insieme $V = \{v_1, v_2, \dots, v_M\}$ di simboli di osservazione (per ogni stato).
- Una distribuzione di probabilità sui simboli di osservazione $B = \{b_{jk}\} = \{b_j(v_k)\}$, ovvero la probabilità $P(v_k | s_j)$ di emissione del simbolo v_k quando lo stato del sistema è s_j .

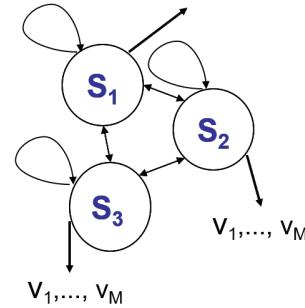
Possiamo quindi denotare un HMM con la tripla $\lambda = (A, B, \pi)$. Infine, analogamente ai processi di Markov, valgono le *indipendenze condizionali* per cui:

$$P(O_t = X | q_t = s_j, q_{t-1}, \dots, q_1, O_{t-1}, \dots, O_1) = P(O_t = X | q_t = s_j)$$

$$\boldsymbol{\pi} = \begin{array}{|c|c|c|}\hline \pi_1 & \pi_2 & \pi_3 \\ \hline 0.33 & 0.33 & 0.33 \\ \hline \end{array}$$

$$\boldsymbol{A} = \begin{array}{|c|c|c|}\hline a_{11} & a_{12} & a_{13} \\ \hline a_{21} & a_{22} & a_{23} \\ \hline a_{31} & a_{32} & a_{33} \\ \hline \end{array} \begin{array}{|c|c|c|}\hline a_{11} & a_{12} & a_{13} \\ \hline a_{21} & a_{22} & a_{23} \\ \hline a_{31} & a_{32} & a_{33} \\ \hline \end{array}$$

$$\boldsymbol{B} = \begin{array}{|c|c|c|}\hline b_{11} & b_{21} & b_{31} \\ \hline b_{12} & b_{22} & b_{32} \\ \hline b_{1M} & b_{2M} & b_{3M} \\ \hline \end{array} \begin{array}{|c|c|c|}\hline b_{11} & b_{21} & b_{31} \\ \hline b_{12} & b_{22} & b_{32} \\ \hline b_{1M} & b_{2M} & b_{3M} \\ \hline \end{array}$$



5.2.1 Problemi chiave degli HMM

I problemi chiave degli HMM sono:

1. *Evaluation*, per cui data una stringa di osservazioni $O = (O_1 \dots O_t \dots O_T)$ si vuole calcolare $P(O|\lambda)$; questo problema si risolve con la *procedura di forward*.
2. *Decoding*, per cui data una stringa di osservazioni O e un modello HMM λ si vuole calcolare la più probabile sequenza di stati $s_1 \dots s_t$ che ha generato O ; questo problema si risolve con l'*algoritmo di Viterbi*.
3. *Training*, per cui dato un insieme di osservazioni $\{O\}$ si vuole determinare il miglior modello HMM λ , ovvero quello per cui $P(O|\lambda)$ è massimizzata; questo problema si risolve con l'*algoritmo di Baum-Welch*.

Alla base di tutti questi problemi sta il fatto che gli stati non sono direttamente osservabili.

5.2.2 Procedura di forward

Data una serie di osservazioni, si potrebbe procedere per *forza bruta*:

$$P(O) = \sum_Q P(O, Q) = \sum_Q P(O|Q) \cdot P(Q)$$

dove Q indica i cammini di lunghezza pari al numero di osservazioni. La probabilità a priori $P(Q)$ della sequenza di stati Q sapremmo calcolarla: è la probabilità condizionata tra coppie di stati, ovvero $P(q_1)P(q_2|q_1)\dots P(q_t|q_{t-1})$. La probabilità condizionata $P(O, Q)$ dell'osservazione dati gli stati, invece, si dovrebbe calcolare facendo la moltiplicazione delle probabilità di aver osservato un simbolo dato uno specifico stato, ovvero $P(O_1|q_1)P(O_2|q_2)\dots P(O_t|q_t)$. Questo metodo, tuttavia, non è efficiente in quanto la sua complessità è pari a $O(tN^t)$.

Tramite la *procedura di forward* è possibile ridurre la complessità computazionale delle operazioni appena presentate. Se si considerano le prime t osservazioni O_1, O_2, \dots, O_t si ha che la probabilità

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, q_t = s_i | \lambda)$$

è definibile ricorsivamente come:

$$\alpha_t(i) = P(O_1, q_1 = s_i) = P(q_1 = s_i) \cdot P(O_1|q_1 = s_i) = \pi_i \cdot b_i(O_1)$$

$$\begin{aligned} \alpha_{t+1}(j) &= P(O_1, O_2, \dots, O_t, O_{t+1}, q_{t+1} = s_j | \lambda) = \\ &= \sum_{i=1}^N P(q_{t+1} = s_j | q_t = s_i) \cdot P(O_{t+1}|q_{t+1} = s_j) \cdot \alpha_t(i) = \\ &= \sum_{i=1}^N [a_{ij} \cdot \alpha_t(i)] \cdot b_j(O_{t+1}) \end{aligned}$$

Ne deriva che data la sequenza di osservazioni $O_1, O_2, \dots, O_t, \dots, O_T$ e conoscendo $\alpha_t(i)$ (variabile di forward) possiamo calcolare

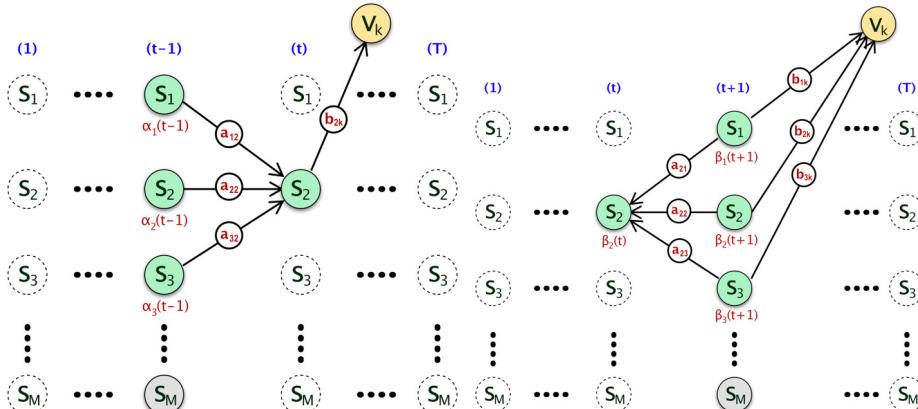
$$P(O_1, O_2, \dots, O_T) = \sum_{i=1}^N \alpha_T(i)$$

con una complessità computazionale pari a $O(TN^2)$. Oppure, in alternativa, si può calcolare ricorsivamente introducendo la variabile di backward

$$\beta_t(j) = P(O_{t+1}, \dots, O_T | q_t = s_j, \lambda) = \sum_{i=1}^N \beta_{t+1}(i) \cdot a_{ij} \cdot b_j(O_{t+1})$$

da cui si ottiene:

$$P(O|\lambda) = \sum_{j=1}^N \beta_0(j)$$



5.2.3 Algoritmo di Viterbi

Il secondo problema è quello del decoding, ovvero quello di stabilire il cammino di stati più probabile che ha generato O_1, O_2, \dots, O_T .

Questo può essere espresso come $\arg \max_Q P(Q|O_1, O_2, \dots, O_T)$. Quindi, una strategia di *forza bruta* si potrebbe calcolare come:

$$\arg \max_Q \frac{P(O_1, O_2, \dots, O_T|Q) \cdot P(Q)}{P(O_1, O_2, \dots, O_T)}$$

In alternativa, un calcolo efficiente può essere fatto tramite l'*algoritmo di Viterbi*. Tale algoritmo sfrutta infatti un approccio di programmazione dinamica che cerca il più probabile stato singolo alla posizione i -esima date le osservazioni e gli stati precedenti. Si vuole infatti trovare la singola migliore sequenza di stati singoli (path) massimizzando la probabilità $P(Q|O, \lambda)$. L'algoritmo si articola in quattro fasi:

1. *Inizializzazione*, in cui si definiscono il caso base e il passo induttivo

$$\begin{cases} \delta_1(i) = \pi_i \cdot b_i(O_1) & \text{con } 1 \leq i \leq N \\ \psi_1(i) = 0 \end{cases}$$

$$\delta_{t+1}(j) = [\max_i \delta_t(i)a_{ij}] \cdot b_j(O_{t+1})$$

2. *Ricorsione*, in cui per ogni $1 \leq j \leq N$ abbiamo che

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}] \cdot b_j(O_t)$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}]$$

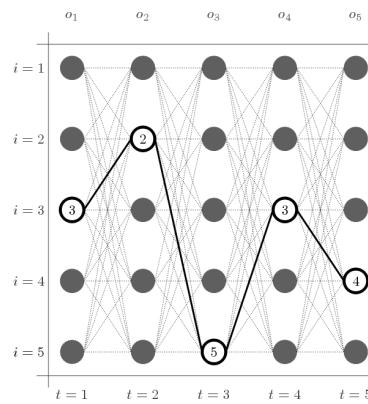
3. *Terminazione*, in cui arriviamo a

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$

$$q_t^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]$$

4. *Path backtracking*, in cui otteniamo per $t = T-1, T-2, \dots, 1$

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$



5.2.4 Stima dei parametri di un HMM

L'ultimo problema relativo agli HMM è il training. Tale problema è risolvibile tramite l'algoritmo di Baum-Welch oppure, in alternativa, si può usare anche l'approccio Maximum Likelihood. Nella *procedura di ri-stima* di Baum-Welch si definiscono inizialmente due misure:

$$\gamma_t(i) = P(q_t = s_i | O_1, O_2, \dots, O_T, \lambda)$$

$$\xi_t(i, j) = P(q_t = s_i, q_{t+1} = s_j | O_1, O_2, \dots, O_T, \lambda)$$

Tali quantità possono essere calcolate in modo efficiente come

$$\sum_{j=1}^N \xi_t(i, j) = \gamma_t(i)$$

dove:

- $\sum_{t=1}^{T-1} \xi_t(i, j)$ è il numero atteso di transizioni dallo stato i allo stato j durante il cammino.
- $\sum_{t=1}^{T-1} \gamma_t(i)$ è il numero atteso di transizioni passanti dallo stato i durante il cammino.

Queste quantità permettono di calcolare, per esempio, la stima di $P(s_j | s_i)$ come il loro rapporto a_{ij} .

Come per l'approccio Expectation-Maximization, utilizzando le variabili di forward e backward possiamo calcolare ξ e γ in due step:

- *Expectation*

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}$$

- *Maximization*

$$\pi_i = \gamma_1(i)$$

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_j(k) = \frac{\sum_{t=1}^T \gamma_t(j) \text{ t.c. } O_t = v_k}{\sum_{t=1}^T \gamma_t(j)}$$

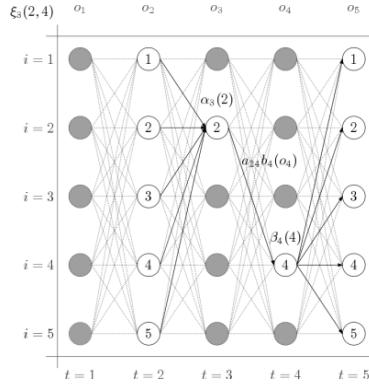
Le quantità appena descritte vengono utilizzate nel processo di stima dei parametri di un HMM in modo iterativo. Si utilizza quindi una variante dell'algoritmo EM che esegue un'ottimizzazione locale per *massimizzare la log-likelihood* del modello rispetto ai dati.

5.2.5 Algoritmo di Baum-Welch

Conoscendo le quantità che descrivono le frequenze attese degli stati e le transizioni da essi, si può costruire l'*algoritmo di Baum-Welch* il quale si articola in cinque fasi:

1. Inizializzo il modello $\bar{\lambda} = (A_0, B_0, \pi_0)$.
2. Il modello corrente è $\lambda = \bar{\lambda}$.
3. Uso il modello λ per calcolare la parte destra delle formule di ri-stima (Expectation).
4. Uso la statistica per la ri-stima dei parametri ottenendo il nuovo modello $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ (Maximization).
5. Vai al passo 2 finché non si verifica la terminazione (dopo un numero fissato di cicli oppure alla convergenza del valore di likelihood).

È dimostrato che ad ogni passo $P(O_1, O_2, \dots, O_T | \bar{\lambda}) > P(O_1, O_2, \dots, O_T | \lambda)$. Si può quindi descrivere l'algoritmo di Baum-Welch come una tecnica di ottimizzazione a discesa del gradiente (ottimizzatore locale) in cui la log-likelihood è fortemente multimodale. Queste caratteristiche fanno capire quanto la scelta dei parametri per il training sia cruciale per la convergenza.



5.3 Questioni aperte

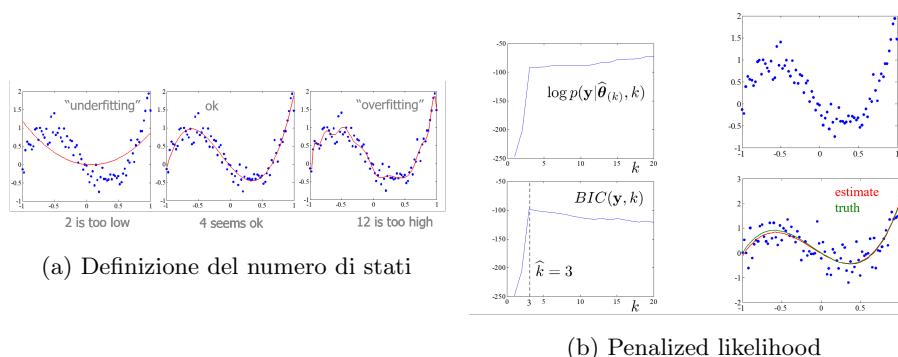
Nel campo del training degli HMM ci sono tre filoni di ricerca:

- *Selezione del modello*, quale topologia e quale numero di stati.
- *Estensione dei modelli standard*, tramite modifica delle componenti e delle dipendenze.
- *Inserimento di comportamenti discriminativi*.

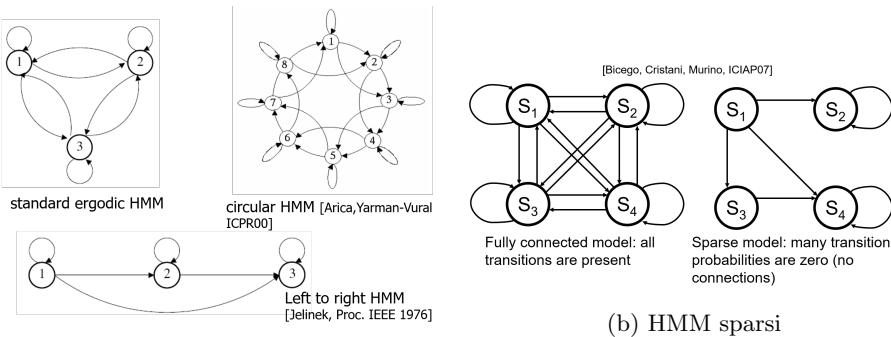
5.3.1 Selezione dell'ordine del modello

Nella selezione dell'ordine del modello, da adottare quando si usano gli HMM, bisogna definire il numero di stati (tipicamente non è un grosso problema) e la topologia, ovvero il set di collegamenti tra gli stati.

Il problema della definizione del *numero di stati* (tramite selezione di modelli standard) permette di evitare l'overfitting. La scelta del modello ottimale può avvenire per strategia forza bruta (si provano vari modelli e si sceglie quello che massimizza un certo criterio) o per maximum likelihood. Poiché il criterio della massimizzazione della log-likelihood può non funzionare (non decresce con l'aumentare dell'ordine del sistema), si inserisce un fattore penalizzante che cresce con l'aumentare dell'ordine del sistema, così da scoraggiare la scelta di un modello complesso. Sono esempi di *penalized likelihood* i criteri BIC e AIC. Un approccio alternativo è basato sul modello split and merge, in cui a partire da modelli grossolani si prosegue con la loro divisione o unione ad altri fino ad ottenerne uno ottimale.



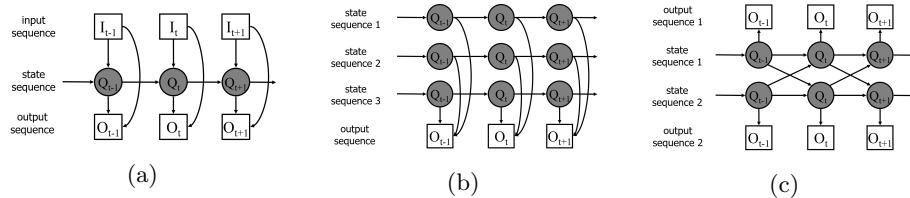
La definizione della *topologia* si basa invece sul tipo di problema da affrontare. Soluzioni ad-hoc per questo problema sono per esempio: topologie a grafo completo (ergodic), collegamenti left to right e HMM circolari. Un'altra possibile soluzione è rappresentata dagli *HMM sparsi*, ovvero grafi sparsi in cui componenti irrilevanti o ridondanti vengono pesate a zero.



5.3.2 Estensione dei modelli standard

Una prima *estensione dei modelli standard* può avvenire tramite aggiunta di dipendenze tra le componenti, così da modellare differenti comportamenti. Questa estensione si basa sul formalismo delle reti Bayesiane, in cui ogni nodo rappresenta una variabile e i link rappresentano le relazioni di causalità. Tre possibili estensioni sono:

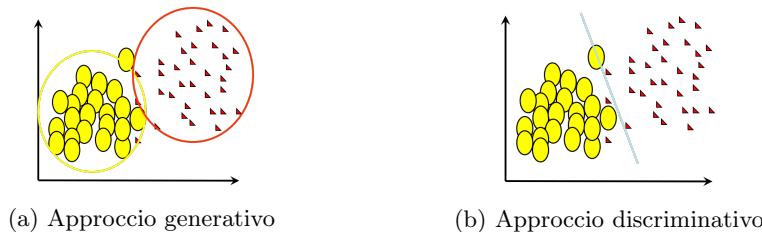
- Input-output HMM*, in cui la sequenza di osservazioni dipende dallo stato nascosto e dall'input (che determina a sua volta lo stato).
- Factorial HMM*, in cui la sequenza di osservazioni dipende da più di una sequenza di stati nascosti.
- Coupled HMM*, costituite da una coppia di HMM che interagiscono fra loro.



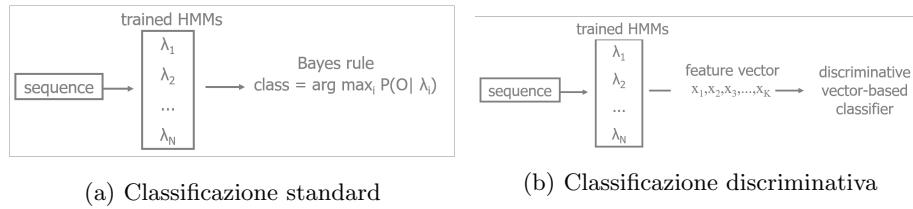
In alternativa, si possono costruire delle funzioni per modellare i simboli in output sulla base di un *sistema di emissione di probabilità*. Esempi di questa seconda possibile estensione sono: reti neurali, HMM gerarchici, kernel machines (SVM), factor analysis, distribuzioni Gaussiane generalizzate. Il problema di questa modalità di estensione è rappresentato dal fatto di dover fare il training per ogni tecnica adottata all'interno del framework. L'ottimalità si avrebbe con un training simultaneo dei parametri. Inoltre, se si utilizzano modelli a kernel (SVM) o reti neurali è necessario convertire il risultato del classificatore in un opportuno valore di probabilità.

5.3.3 Inserimento di comportamenti discriminativi

Il tipico problema nella modellazione dei classificatori è dato dalla divisione tra approccio generativo e discriminativo. L'approccio *generativo* (HMM) costruisce un modello per ogni classe/gruppo fornendo soluzioni migliori nella descrizione delle classi. L'approccio *discriminativo* (SVM), invece, costruisce un unico modello per separare tutte le classi/gruppi fornendo soluzioni migliori al problema in esame.



Dato che gli HMM sono un modello generativo, l'inserimento di informazioni discriminative (provenienti direttamente dalle classi) può essere fatto in fase di training o in fase di classificazione. Il *training discriminativo* degli HMM si basa sull'idea di addestrare il modello tramite un criterio discriminativo, considerando ad esempio le informazioni provenienti da altre classi. Infine, mentre nella classificazione standard prima si addestra un HMM per ogni classe e poi si applica la regola di Bayes, nella *classificazione discriminativa* si usa un HMM addestrato per definire uno spazio delle features dove addestrare poi un classificatore discriminativo. I tipi di features da trattare in questo caso possono essere la log-likelihood o il suo gradiente.



5.4 Sommario

Un *processo di Markov* si può vedere come un automa a stati finiti in cui la transizione da uno stato all'altro avviene con una certa probabilità. Tale probabilità dipende solamente dallo stato immediatamente precedente e non dall'intera storia degli stati passati (Markovianità del primo ordine).

Un *modello di Markov* a N stati può essere rappresentato tramite una matrice $A = \{a_{ij}\}$ di dimensione $N \times N$. Ogni elemento a_{ij} della matrice corrisponde alla probabilità di transizione dallo stato i allo stato j . Dato che questi valori rappresentano delle probabilità, la somma di ogni riga della matrice di transizione deve essere pari a 1. Per completare il modello, è necessario definire anche la distribuzione di probabilità π degli stati iniziali, ovvero le probabilità di trovarsi in un determinato stato all'inizializzazione della macchina a stati. Anche la somma di queste probabilità deve essere pari a 1.

Un *Hidden Markov Model* è un modello di Markov in cui non è possibile osservare direttamente lo stato corrente, per via del fatto che gli stati sono nascosti. Tuttavia, è possibile ottenere l'osservazione legata ad un certo stato. Infatti, ad ogni valutazione della macchina a stati, lo stato corrente s_j produrrà un simbolo v_k in uscita, il quale dipenderà da una distribuzione di probabilità $B = \{b_j(v_k)\}$. Ovvero $P(v_k|s_j)$ è la probabilità di emissione del simbolo v_k quando lo stato del sistema è s_j . In sostanza, un HMM (discreto e addestrato) è descritto dai seguenti parametri:

- Una matrice di transizione $A = \{a_{ij}\}$.
- Una distribuzione di probabilità sui simboli di osservazione $B = \{b_j(v_k)\}$.
- Una distribuzione di probabilità degli stati iniziali $\pi = \{\pi_i\}$.

Data una sequenza di osservazioni $O = (O_1, O_2, \dots, O_t)$ ed un HMM λ noto, è possibile ottenere, tramite la *procedura di forward*, un valore che descrive la probabilità che tale sequenza sia stata generata dal modello in questione. Questo valore, denominato *likelihood* (verosimiglianza), è scritto come $P(O|\lambda)$. Invece, se si desidera conoscere la sequenza di stati più probabile, è possibile utilizzare l'*algoritmo di Viterbi*, il quale restituisce la likelihood lungo la sequenza di stati più probabile. Infine, per determinare il miglior modello HMM (quello per cui $P(O|\lambda)$ è massimizzata), si deve utilizzare l'*algoritmo di Baum-Welch*. Questo algoritmo non ha bisogno di conoscere gli stati associati alle osservazioni. Infatti, partendo da un modello iniziale (una stima), l'algoritmo esegue iterativamente una procedura di Expectation-Maximization convergendo ad un massimo locale della likelihood. Dato che i parametri trovati corrispondono ad un massimo locale (e non globale) la stima iniziale dei parametri del modello può influenzare fortemente il risultato. Tuttavia, l'algoritmo trova una soluzione soddisfacente nella maggior parte dei casi.

La selezione del modello è una questione aperta nel campo del *training* degli HMM. Infatti, per costruire un HMM è necessario definire il numero dei suoi stati e la sua topologia. Per la scelta del *numero di stati* non esiste una regola precisa:

- Si può scegliere un valore che si crede ragionevole per l'applicazione.
- Si possono fare più tentativi per trovare il numero di stati che massimizza un certo criterio (brute force).
- Si può scegliere il valore che massimizza la log-likelihood (anche utilizzando fattori penalizzanti per evitare la scelta di un modello troppo complesso).
- Si può partire da un modello grossolano e dividerlo/unirlo ad altri fino a che non se ne ottiene uno ottimale.

Per *topologia*, invece, si intende l'insieme delle transizioni ammesse dal modello (collegamenti tra stati). Soluzioni ad-hoc per la scelta della topologia sono: topologie a grafo completo, collegamenti left to right, HMM circolari, HMM sparsi.

Capitolo 6

Classificatori discriminativi

6.1 Introduzione

I classificatori *generativi*:

- Modellano le probabilità condizionali (likelihood) e a priori delle classi per ogni classe (indipendentemente).
- Usano la regola di decisione di Bayes per classificare, prendendo la massima posteriore.

I classificatori *discriminativi*:

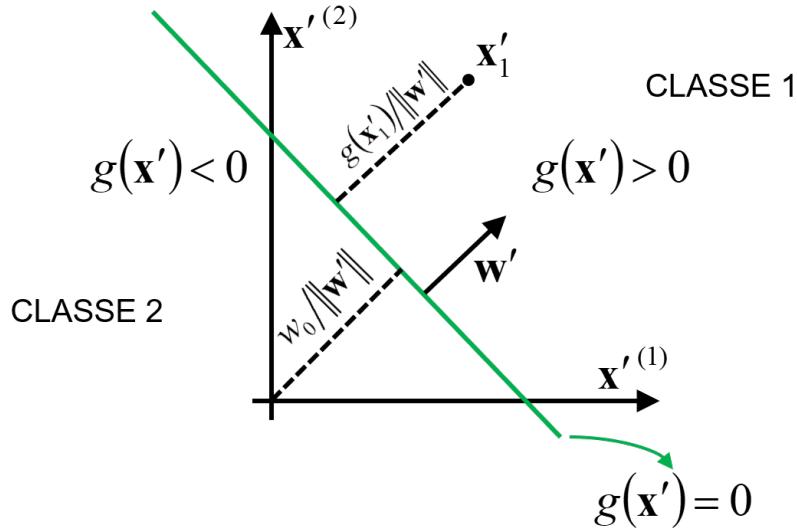
- Stimano direttamente il confine di decisione, producendo una stima di classificazione.
- Sono approcci geometrici (anche se non è semplice trovare un parallelismo con la regola di Bayes).

6.2 Funzioni discriminanti lineari

In un problema binario, l'obiettivo è trovare la retta che separa le due classi. Un approccio di questo tipo è stato visto finora solo per le distribuzioni gaussiane; il compito è ora quello di generalizzare. Si parla di *funzione discriminante lineare* quando la funzione discriminante è combinazione lineare delle varie features. Definiamo quindi la funzione discriminante lineare $g : \mathbb{R}^d \rightarrow \mathbb{R}$ come:

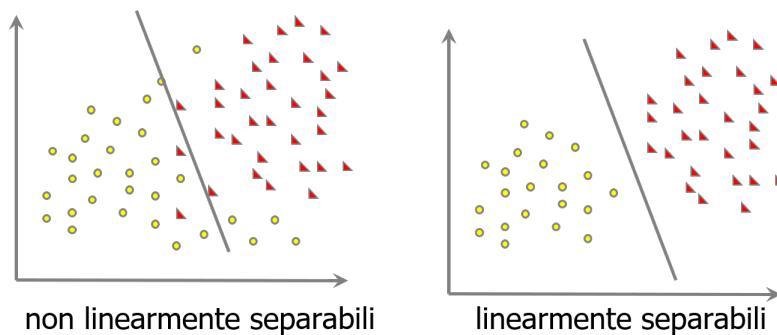
$$g(x') = w'^t x' + w_0$$

dove $x' = [x_1, \dots, x_d]$, $w' = [w_1, \dots, w_d]$ sono i pesi e w_0 è il termine noto (o bias). Un campione x' è classificato come appartenente alla classe ω_1 se $w'^t x' + w_0 > 0$, altrimenti se $w'^t x' + w_0 < 0$ verrà classificato come appartenente alla classe ω_2 . Geometricamente, settando $g(x') = 0$ si ottiene un *iperpiano* (o superficie di separazione). Infatti, w' determina l'orientazione del piano e, assieme a w_0 , la distanza del piano rispetto all'origine.



6.2.1 Training

Più genericamente, considerando $w = [w_0, w_1, \dots, w_d]$ e $x = [1, x_1, \dots, x_d]$, la regola di decisione diventa: un campione x è classificato come appartenente alla classe ω_1 se $w^t x > 0$, altrimenti se $w^t x < 0$ verrà classificato come appartenente alla classe ω_2 . Dato quindi come *training set* un insieme di N campioni x'_1, \dots, x'_N alcuni etichettati ω_1 ed altri etichettati ω_2 , si vogliono determinare il vettore dei pesi w' e il termine w_0 della funzione discriminante lineare. Un ragionevole approccio è la ricerca di un vettore di pesi tale che la probabilità di commettere errore sui campioni sia minima. Se esiste un vettore di pesi tale da rendere nulla la probabilità di errore, allora i campioni si dicono *linearmente separabili*.



L'obiettivo è quindi quello di calcolare pesi che soddisfino le seguenti condizioni:

$$\forall x \in \omega_1. w^t x > 0$$

$$\forall x \in \omega_2. w^t x < 0$$

Nella seconda condizione si può anche dire che x è classificato correttamente se $w^t(-x) > 0$. Ciò suggerisce una *normalizzazione* (cambiare il segno a tutti gli oggetti della classe ω_2) che semplifica il trattamento nel caso di due diverse classi, ossia il fatto che si possa solo trovare il vettore dei pesi tale per cui si abbia $w^t x > 0$ per tutti i campioni a prescindere dalle classi. Questo vettore w è chiamato *vettore soluzione* (o separatore).

Risulta chiaro che se il vettore soluzione esiste, esso non è unico. Ci saranno quindi diversi modi per imporre requisiti addizionali per vincolare il vettore soluzione. Uno potrebbe essere quello di cercare il vettore dei pesi che massimizzi la minima distanza dei campioni dal piano separatore. Un altro potrebbe essere quello di cercare il vettore dei pesi che soddisfi $w^t x \geq b$, dove b è una costante positiva chiamata *margine*.

6.2.2 Tecnica del gradiente discendente

Per determinare il vettore dei pesi w si ricorre a tecniche di minimizzazione, le quali minimizzano un funzionale in funzione di certe quantità. Una delle tecniche di base è quella del *gradiente discendente*, ossia uno degli approcci più semplici per il calcolo di una funzione discriminante. La tecnica del gradiente discendente è un metodo iterativo di assestamento progressivo dei pesi che si basa sulla seguente proprietà: il vettore gradiente nello spazio W (ossia i coefficienti di w) punta nella direzione di massimo scarto di una funzione da minimizzare.

La *procedura* consiste nell'aggiornare il valore del vettore dei pesi al passo $k+1$ con un contributo proporzionale al modulo del gradiente di un particolare funzionale al passo precedente. Tale procedura può essere formulata come:

$$w(k+1) = w(k) - \rho_k \nabla J(w)|_{w=w(k)}$$

dove $J(w)$ è la funzione di valutazione che deve essere minimizzata. Questa funzione deve essere scelta in modo tale che raggiunga il suo minimo quanto più w si avvicina alla soluzione ottima (nel caso in cui essa sia convessa). Quindi, il minimo di $J(w)$ si ottiene spostando w in direzione opposta al gradiente.

Chiaramente occorre scegliere un *criterio di ottimalità*. La scelta più ovvia è quella di definire un funzionale $J(w; x_1, \dots, x_N)$ rappresentato dal numero di campioni mal classificati da w su un totale di N campioni (error rate). Siccome tale funzionale è costante a tratti, il metodo della discesa del gradiente non è molto adatto a tale problema. Una scelta migliore per J può essere perciò:

$$J(w) = - \sum_{i \in X} w^t x_i$$

dove X è l'insieme di punti classificati non correttamente da w . Geometricamente, $J(w)$ è proporzionale alla somma delle distanze dei campioni mal classificati dal confine di decisione. Siccome $\nabla J = - \sum_{i \in X} x_i$, l'algoritmo di discesa del gradiente diventa:

$$w(k+1) = w(k) + \rho_k \cdot \sum_{i \in X} x_i$$

con $\rho_k = \frac{1}{k}$.

Si citano infine altri metodi per scegliere $J(w)$ e ottimizzarla: metodo del rilassamento, metodo del MSE (Minimum Square Error), metodo del Least-MSE (o Widrow-Hoff), metodo di Ho-Kashyap, stochastic gradient.

6.2.3 Funzioni discriminanti lineari generalizzate

Nel caso di problemi a C classi, possono essere costruiti $C - 1$ classificatori $g_j(x)$, uno per ogni problema C_j vs *non- C_j* (one-vs-rest). Oppure si possono costruire $C(C-1)/2$ classificatori binari che classificano un punto a maggioranza (one-vs-one). La funzione discriminante lineare può anche essere scritta come:

$$g(x) = w_0 + \sum_{i=1}^d w_i x_i$$

dove w_i è la componente i -esima del vettore dei pesi e w_0 è il solito termine noto. Possiamo quindi aggiungere altri termini in cui mettiamo i prodotti delle varie componenti di x , ad esempio un classificatore discriminante quadratico sarà nella forma:

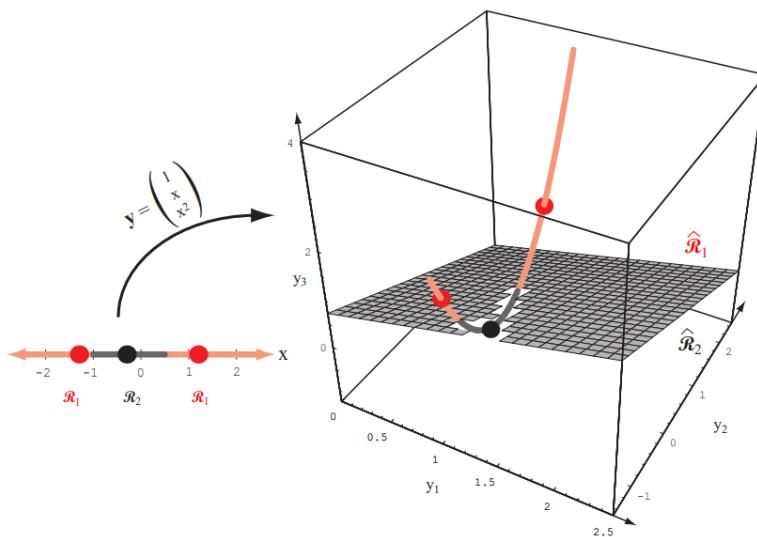
$$g(x) = w_0 + \sum_{i=1}^d w_i x_i + \sum_{i=1}^d \sum_{j=1}^d v_{ij} x_i x_j$$

Possiamo infine *generalizzare* il concetto scrivendo:

$$g(x) = \sum_{k=1}^K a_k \cdot y_k(x)$$

dove y_k sono funzioni arbitrarie su x che possono essere viste direttamente come nuove features. In altre parole, le funzioni y_k possono essere viste come *estrattori di features*. Di solito, la fase di estrazione delle features viene usata per diminuire la dimensionalità (PCA), mentre qui viene usata per aumentarla.

La funzione discriminante lineare generalizzata $g(x)$ è lineare nello spazio di y (è una funzione semplice) e non lineare nello spazio originale di x (è una funzione complessa). Per esempio, con la funzione $g(x) = a_1 + a_2 x + a_3 x^2$ si passa da uno spazio delle features monodimensionale ad uno spazio tridimensionale.

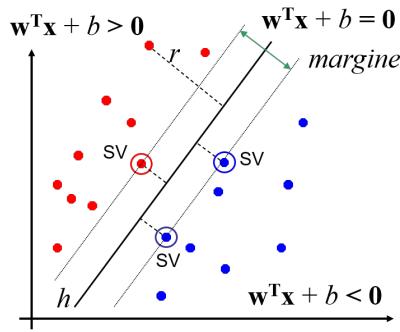


6.3 Support Vector Machines

Le *Support Vector Machines* (SVM) possono essere viste come separatori lineari (ipotizzando un problema linearmente separabile), ereditando la notazione vista finora, dove $b = w_0$ indica il margine. Per la classificazione basta applicare il segno alla funzione discriminante:

$$f(x) = \text{sign}(w^T x + b) = \begin{cases} +1 \\ -1 \end{cases}$$

Per trovare il separatore ottimo, chiamo r la distanza di un campione dall'iperpiano e chiamo *support vectors* i campioni più vicini all'iperpiano (tale definizione verrà poi generalizzata). Il margine dell'iperpiano h di separazione è la somma delle distanze minime dei campioni su di esso. Individuare l'iperpiano h ottimo corrisponde a massimizzare il margine. Questo implica che solo alcuni esempi sono importanti per l'apprendimento (i vettori di supporto SV), gli altri invece possono essere ignorati.



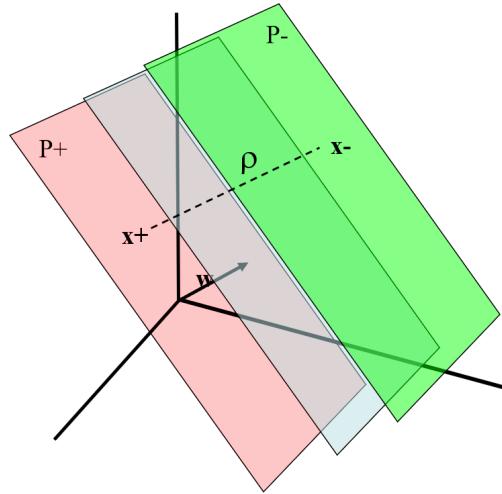
6.3.1 Addestramento di una SVM

L'addestramento della SVM consiste nel trovare il vettore w e il parametro b ottimali (che massimizzino il margine), a partire dal training set. Si hanno quindi i dati di addestramento $D\{(x_i, y_i)\}$ (dove $\{y\}$ indica label binarie di valore 1 o -1), a distanza almeno 1 dall'iperpiano, su cui valgono le seguenti condizioni:

$$\begin{aligned} w^T x_i + b &\geq 1 \text{ se } f(x_i) = +1 \\ w^T x_i + b &\leq -1 \text{ se } f(x_i) = -1 \end{aligned}$$

Per i vettori di supporto la diseguaglianza diventa una uguaglianza, perché essi stanno esattamente a distanza 1 dall'iperpiano separatore.

Le variabili incognite del problema sono il vettore w perpendicolare al piano di separazione (si suppone w versore per comodità) e la distanza $b = b/\|w\|$ del piano dall'origine. Indichiamo quindi con ρ la distanza fra i piani P^+ : $w^T x_i + b = 1$ e P^- : $w^T x_i + b = -1$. Se x^+ è un punto di P^+ e x^- è un punto di P^- a distanza minima da x^+ , allora posso scrivere $\rho = |x^+ - x^-|$, e posso anche scrivere $(x^+ - x^-) = \lambda w$. Essendo x^+ e x^- a distanza minima, muoversi da x^+ a x^- corrisponde a muoversi lungo un percorso $x^+ = x^- + \lambda w$ nella direzione di w e di lunghezza $\|\lambda w\|$.



6.3.2 SVM lineari

Mettendo assieme quanto visto nella sezione precedente, vogliamo capire chi sia ρ . Isolando λ e w da $x^+ = x^- + \lambda w$ e sfruttando le uguaglianze $w^T x_i + b = 1$ e $w^T x_i + b = -1$ otteniamo:

$$\rho = \left\| \frac{2}{w^T w} \cdot w \right\| = \left\| \frac{2}{w^T} \right\| = \frac{2}{\|w\|}$$

Per massimizzare il margine dovremo quindi minimizzare $\|w\|$. Il *problema di ottimizzazione* quadratica che ne risulta è illustrato (in due diverse formulazioni) nelle figure sottostanti.

Trova w e b tali che

$$\rho = \frac{2}{\|w\|} \text{ è massimo; e per ogni } \{(x_i, y_i)\} \in D \\ w^T x_i + b \geq 1 \text{ se } y_i = 1; \quad w^T x_i + b \leq -1 \text{ se } y_i = -1$$

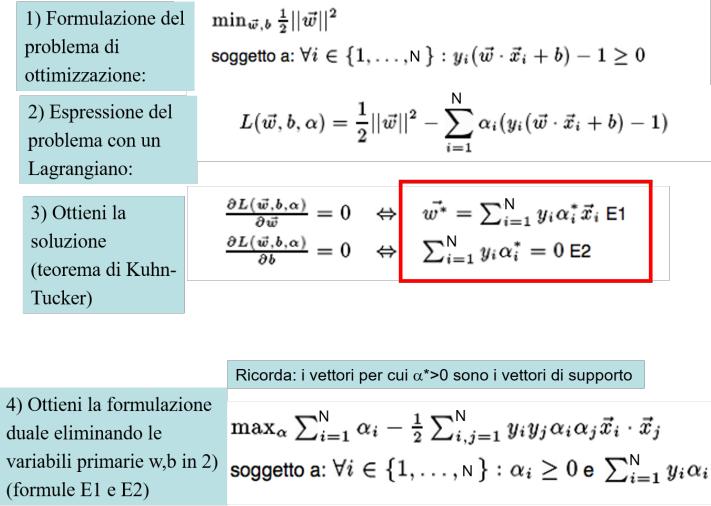
(a) Problema di ottimizzazione

Trova w e b t.c.

$$\Phi(w) = \frac{1}{2} w^T w \text{ è minimizzata;} \\ \text{E per ogni } \{(x_i, y_i)\} \in D : y_i (w^T x_i + b) \geq 1$$

(b) Formulazione duale

Quindi, per risolvere questo problema di ottimizzazione, si deve ottimizzare una funzione quadratica soggetta a vincoli lineari (uno per ogni x_i): $y_i (w^T x_i + b) \geq 1$. Siccome i problemi di ottimizzazione quadratica sono problemi di programmazione matematica ben noti, esistono diversi algoritmi; uno di questi comporta l'utilizzo dei moltiplicatori di Lagrange ed è illustrato nella prossima figura.



Usando le equazioni trovate al punto 3 nel punto 2, attraverso alcuni passaggi (figura (a)), otteniamo la formulazione finale (figura (b)).

$$\min_{\vec{w}, b} \frac{1}{2} \vec{w} \cdot \vec{w} - \sum_i (\alpha_i y_i \vec{x}_i) \cdot \vec{w} - b \sum_i \alpha_i y_i + \sum_i \alpha_i =$$

$$\min_{\vec{w}, b} \frac{1}{2} \sum_i (\alpha_i y_i \vec{x}_i) \cdot \sum_i (\alpha_i y_i \vec{x}_i) - \sum_i (\alpha_i y_i \vec{x}_i) \sum_i (\alpha_j y_j \vec{x}_j) - b \cdot 0 + \sum_i \alpha_i =$$

$$\max_{\vec{w}, b} \left(\sum_i \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \right)$$

Trova $\alpha_1, \dots, \alpha_N$ t.c.
 $\mathbf{Q}(\alpha) = \sum_i \alpha_i - 1/2 \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ è
 massimizzata e
 (1) $\sum_i \alpha_i y_i = 0$
 (2) $0 \leq \alpha_i$ per ogni α_i

(a) (b)

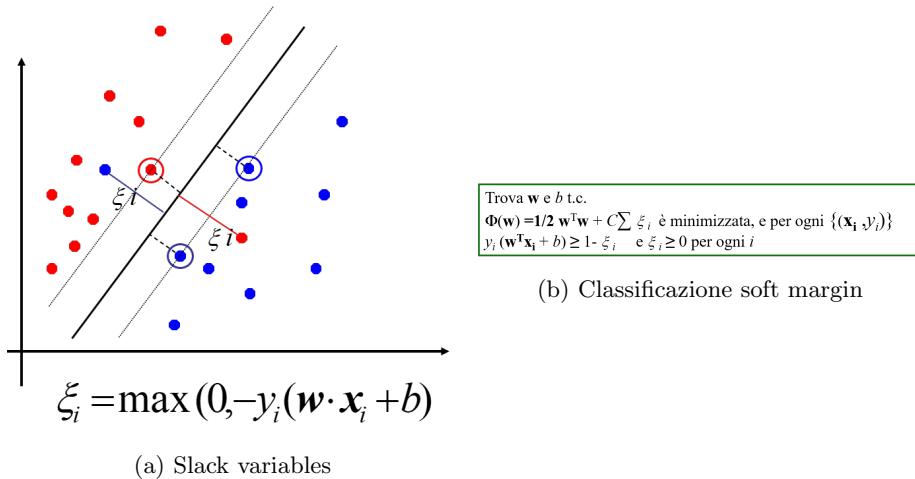
6.3.3 Margini soft

Se il set di addestramento non è linearmente separabile, si introducono le cosiddette *slack variables* ξ_i per consentire la classificazione errata di qualche punto. Nel problema di ottimizzazione con le slack variables viene introdotto il parametro C che pesa gli errori e controlla l'overfitting (più C è piccolo più ammetto errori). I valori di ξ_i si interpretano come:

- $\xi_i = 0$, classificazione corretta.
- $0 < \xi_i < 1$, classificazione corretta ma oltre il margine.
- $\xi_i \geq 1$, errore di classificazione.

Il sistema vincolato viene ancora risolto nella sua forma duale ed il risultato è ancora $w = \sum_{i=1}^N \alpha_i y_i x_i$, ma ora i support vectors saranno:

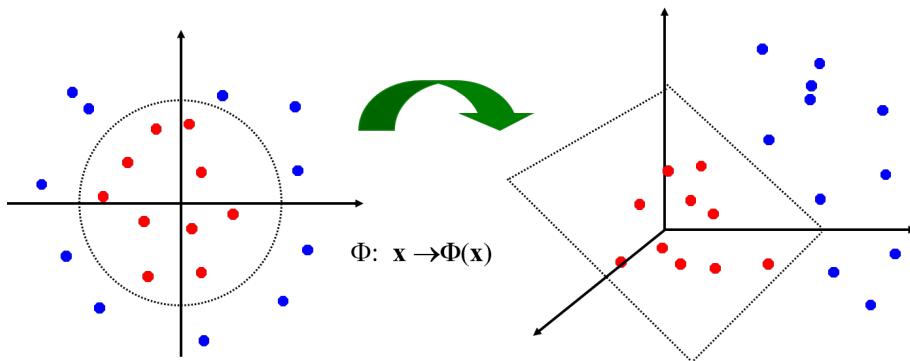
- I punti per cui α_i è diverso da zero.
- I punti sul margine.
- I punti classificati correttamente ma oltre il margine.
- I punti classificati non correttamente.



6.3.4 SVM non lineari

Se il problema non è linearmente separabile, l'idea è quella di proiettare le features in uno spazio nel quale i dati risultino separabili. I problemi di questa *proiezione ad alta dimensionalità* sono:

- Proiettare i punti in uno spazio a dimensionalità troppo elevata può portare alla curse of dimensionality.
- La complessità computazionale aumenta.
- Lo spazio risultante è quasi vuoto.
- Potrebbero in teoria esserci anche mapping in grado di proiettare i punti in uno spazio a dimensionalità infinita.
- Scegliere il mapping corretto potrebbe non essere facile.



6.3.5 Kernel trick

Il problema di ottimizzazione visto per le SVM lineari produce come risultato:

$$w = \sum_{i=1}^N \alpha_i y_i x_i$$

Siccome in fase di testing la *classificazione* avviene con $f(x) = \text{sign}(w^T x + b)$, possiamo scrivere:

$$f(x) = \text{sign} \left(\left[\sum_{i=1}^N \alpha_i y_i x_i \right]^T \cdot x + b \right) = \text{sign} \left(\sum_{i=1}^N \alpha_i y_i x_i^T x + b \right)$$

Quindi, i punti del training set (nel training e nel testing di una SVM) non appaiono mai da soli ma sempre in *prodotto scalare* con altri punti. Applicando il mapping $\Phi(x)$ per proiettare i punti in uno spazio a dimensionalità maggiore si ottiene:

$$w^T x = \sum_{i=1}^N \alpha_i y_i \Phi(x_i^T) \Phi(x)$$

Le *funzioni kernel* permettono di evitare di calcolare il prodotto in uno spazio ad elevata dimensionalità. Una funzione kernel corrisponde quindi ad un prodotto scalare in uno spazio esteso. Siccome il classificatore lineare si basa sul prodotto scalare fra vettori dello spazio delle istanze X (non esteso) $K(x_i, x_j) = x_i^T x_j$, se ogni punto di D è traslato in uno spazio di dimensioni maggiori attraverso una trasformazione $\Phi(x)$, il prodotto scalare diventa:

$$K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) = x_i'^T x_j'$$

dove x' indica una trasformazione non lineare. Si ottiene infine:

$$w^T x = \sum_{i=1}^N \alpha_i y_i K(x_i, x)$$

Per alcune funzioni, verificare che $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$ è complesso. Sono *esempi* di funzioni kernel:

- Funzione lineare, $K(x_i, x_j) = x_i^T x_j$.
- Funzione polinomiale di potenza p , $K(x_i, x_j) = (1 + x_i^T x_j)^p$.
- Funzione Gaussiana (Radial Basis Function), $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$.
- Percettrone a due stadi (sigmoide), $K(x_i, x_j) = \tanh(\beta_0 x_i^T x_j + \beta_1)$.

In generale, ogni funzione semi-definita positiva è un kernel (teorema di Mercer).

6.3.6 Selezione del kernel

Il kernel *lineare*:

- Viene utilizzato quando lo spazio delle features è già molto grande; ad esempio nella classificazione del testo, che utilizza i conteggi delle singole parole come features.
- È un caso speciale del kernel RBF.
- Non richiede parametri aggiuntivi.

Il kernel *polinomiale*:

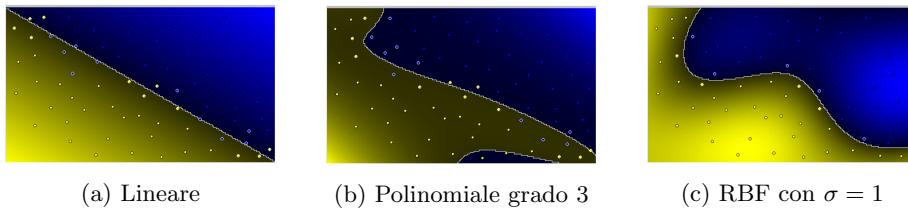
- Va in difficoltà in presenza di valori che si avvicinano allo zero o all'infinito.
- È una buona scelta in ambiti ben noti e ben condizionati.
- Ha un parametro aggiuntivo, ossia il grado p .

Il kernel *RBF* (Radial Basis Function):

- Viene indicato in generale come la scelta migliore.
- Ha bisogno del parametro aggiuntivo σ .

Il kernel *sigmoide*:

- Ha bisogno di due parametri aggiuntivi, ossia β_0 e β_1 .
- Per alcuni valori di β_0 e β_1 non soddisfa il teorema di Mercer.
- Deriva dalle reti neurali.
- Non viene consigliato dalla letteratura.



6.3.7 Sommario

Nel caso di una SVM lineare, il classificatore (funzione obiettivo) è un iperpiano di separazione. I punti più importanti sono i *vettori di supporto*, così chiamati per il fatto che sostengono l'iperpiano mantenendolo in equilibrio. Algoritmi di ottimizzazione quadratica identificano quali punti rappresentano il supporto. Nella formulazione di questo *problema di ottimizzazione* appaiono i prodotti scalari. Se i dataset non sono separabili si può proiettare il problema in uno spazio di dimensioni maggiori (SVM non lineari).

Le SVM sono attualmente fra i migliori classificatori in una varietà di problemi. I *vantaggi* delle SVM sono:

- Hanno un'interpretazione geometrica semplice.
- Il kernel trick permette di ottenere in modo efficiente un classificatore non lineare senza incorrere nella curse of dimensionality.
- La soluzione del training è ottimale.
- I support vectors danno una rappresentazione compatta del training set; il loro numero fornisce infatti un'idea della capacità di generalizzazione (meno sono e meglio è).
- In moltissimi contesti applicativi funzionano decisamente bene; la teoria alla base delle SVM (statistical learning theory) è infatti elegante e solida.

Gli *svantaggi*, invece, sono:

- La scelta di kernel e parametri è cruciale.
- A volte il training può essere oneroso in termini di tempo.
- Il training non è incrementale, ovvero se arriva un nuovo oggetto occorre rifare da capo tutto l'addestramento.
- La gestione del caso multiclass è non è ottimale (sia in termini di efficienza che in termini di soluzione proposta).

Il *tuning* dei parametri di una SVM è un'arte. Infatti, la selezione di uno specifico kernel e dei parametri viene spesso eseguita in modo empirico (tenta e verifica). Una possibile ricetta è:

1. Esegui un semplice ridimensionamento dei dati (ad es. z-normalizzazione).
2. Considera il kernel RBF.
3. Usa la cross-validation per trovare i migliori C e σ .
4. Usa i migliori C e σ per allenare l'intero training set.
5. Verifica.

Vediamo infine alcuni problemi di questa ricetta e le relative soluzioni.

- La ricerca dei parametri può richiedere molto tempo; soluzione: condurre la ricerca dei parametri in modo gerarchico.
- A volte i kernel RBF sono soggetti a overfitting; soluzione: usare kernel polinomiali di alto grado.
- La ricerca dei parametri deve essere ripetuta per ogni feature scelta (non c'è riuso dei calcoli); soluzione: confrontare le features su sottoinsiemi casuali dell'intero dataset per contenere i costi di computazione.
- Gli intervalli di ricerca per C e σ sono difficili da scegliere; soluzione: provare valori crescendo esponenzialmente, ad esempio $C = 2^{[-5...15]}$ e $\sigma = 2^{[-15...5]}$.

Capitolo 7

Clustering

7.1 Introduzione

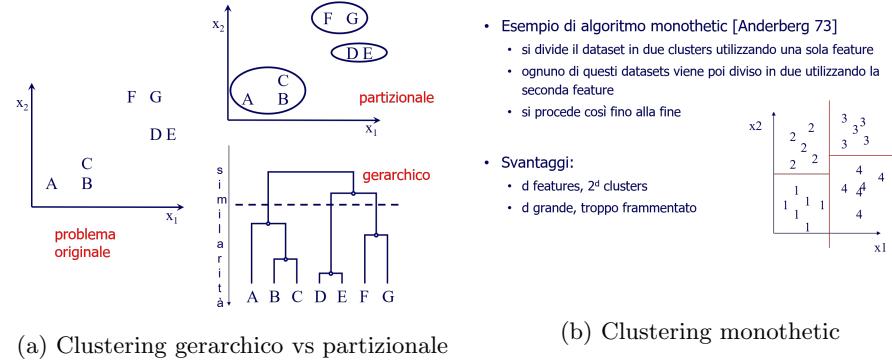
Il *clustering* è un metodo di classificazione che prevede il raggruppamento degli oggetti studiati secondo gruppi naturali, ovvero caratterizzati da features con valori simili. Ciò comporta che gli oggetti possono essere di classi diverse ma vicini nello spazio delle features. La differenza principale con i metodi di classificazione visti finora sta nel fatto che il fattore discriminante è la similarità tra gli oggetti.

7.2 Classi di approcci

A seconda del *punto di vista* adottato possiamo avere differenti tipologie (classi) di clustering.

- Tipo di risultato dell'operazione di clustering:
 - Clustering *gerarchico*, ritorna una serie di partizioni innestate (dendrogramma) e, tramite l'uso di matrici di prossimità, mira ad evidenziare le relazioni tra i vari pattern del dataset; non dovendo impostare il numero di clusters a priori permette un partizionamento più informativo, ma non è trattabile per dataset grandi; alcuni esempi sono complete link, single link e ward link.
 - Clustering *partizionale*, ritorna una singola partizione dei dati (il numero di clusters è dato a priori) mirando a identificare i gruppi naturali presenti nel dataset a partire da dati in forma vettoriale; nonostante sia ottimo per dataset grandi, il problema è scegliere il numero di clusters; alcuni esempi sono K-means e le sue varianti PAM e ISODATA.
- Natura dei clusters risultanti:
 - *Hard clustering*, associa ad ogni pattern un unico cluster (per questo è detto anche clustering esclusivo); un esempio di problema è il raggruppamento delle persone per età.

- *Soft clustering*, permette di associare un pattern a più clusters (clustering non esclusivo); un esempio è il raggruppamento delle persone per malattia.
- Come vengono formati i clusters:
 - Clustering *agglomerativo*, si costruiscono i clusters tramite operazioni di merge, inizializzando ogni pattern con un cluster, fino al raggiungimento di una determinata condizione.
 - Clustering *divisivo*, opera invece con operazioni di split, partendo da un unico cluster.
- Come vengono processati i pattern:
 - Clustering *sequenziale*, i pattern vengono processati uno alla volta.
 - Clustering *simultaneo*, i pattern vengono processati tutti assieme.
- Come vengono utilizzate le features:
 - Clustering *monothetic*, viene utilizzata una feature alla volta per fare clustering.
 - Clustering *polythetic*, vengono utilizzate tutte le features simultaneamente per fare clustering (la maggior parte delle tecniche di clustering sono polythetic).
- Come viene formulato matematicamente l'algoritmo:
 - *Graph theory*, gli algoritmi sono formulati in termini di teoria dei grafi e ne utilizzano definizioni e proprietà (ad esempio la connettività).
 - *Matrix algebra*, gli algoritmi sono espressi in termini di formule algebriche (ad esempio l'errore quadratico medio).
- Quando arrivano nuovi dati:
 - Clustering *incrementale*, è possibile aggiornare il clustering fino ad ora costruito (caratteristica cruciale in questi anni dove i dataset sono sempre più grandi e in espansione).
 - Clustering *non incrementale*, non permette l'aggiornamento e per questo necessita di riesaminare completamente l'intero dataset.
- Come viene ottimizzata la funzione di errore:
 - Clustering *deterministico*, effettua un'ottimizzazione classica (discesa lungo il gradiente).
 - Clustering *stocastico*, fa una ricerca stocastica nello spazio degli stati della soluzione (Monte Carlo).



(a) Clustering gerarchico vs partizionale

(b) Clustering monothetic

7.3 Clustering partizionale

Per quanto riguarda il clustering partizionale, ci sono ulteriori classi di approcci:

- Clustering *sequenziale*, ossia un approccio semplice ed intuitivo ottimo per cluster convessi in cui (tipicamente) i pattern sono processati poche volte; il risultato finale in genere dipende dall'ordine con cui sono presentati i pattern.
- *Center-based* clustering, in cui ogni cluster è rappresentato da un centro e l'obiettivo è minimizzare una funzione di costo; funziona bene per grandi dataset e cluster convessi.
- *Search-based* clustering, in cui si vuole minimizzare la funzione di costo in modo globale.
- *Model-based* clustering, in cui l'idea è quella di creare modelli (probabilistici) per i dati assumendo che ogni componente di queste distribuzioni di probabilità sia un cluster.

7.3.1 Clustering sequenziale

Un metodo per affrontare il problema del clustering partizionale di tipo sequenziale è l'*algoritmo BSAS* (Basic Sequential Algorithmic Scheme). In BSAS, ogni pattern viene processato una volta sola e nell'ordine in cui compare nel dataset. Tale pattern viene poi associato ad un cluster oppure ne crea uno nuovo, a seconda di una certa metrica adottata per misurare la distanza tra gruppi naturali (soglia di dissimilarità). Il numero di clusters non è conosciuto a priori ma viene stimato durante il processo.

L'update dei *rappresentanti dei clusters* è dato dalla condizione $d(x, C) = d(x, m_C)$, ossia se la distanza tra un punto x e un cluster C è uguale alla distanza dalla media del cluster, allora l'aggiornamento dei rappresentanti può essere fatto on-line come:

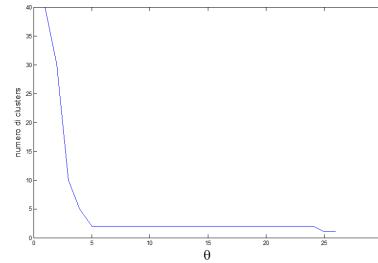
$$m_{C_k}^{new} = \frac{(n_{C_k}^{new} - 1) \cdot m_{C_k}^{old} + x}{n_{C_k}^{new}}$$

dove m_{C_k} è la media del cluster C_k , x è il punto aggiunto al cluster C_k e n_{C_k} è la cardinalità del cluster C_k .

Si può osservare che l'ordine con cui vengono processati i pattern è cruciale per definire il risultato finale, come anche la decisione della soglia (si possono infatti ottenere troppi clusters o troppo pochi). Inoltre, tramite l'uso dei rappresentanti, i clusters risultanti escono compatti. Per poter calcolare il *numero ottimale di clusters* basta eseguire BSAS più volte variando la soglia entro un certo range (secondo un passo di incremento fissato). Ciò che si ottiene è quindi la stima del numero di clusters in funzione della soglia (frequenza), in cui il numero che forma la regione piatta più lunga rappresenta il numero di clusters ottimale.

```
m=1 //numero di clusters trovati ad un determinato istante
C_m = x_1
for i = 2 to N
    trova C_k tale che d(x_i, C_k) = min_{1 ≤ j ≤ m} d(x_i, C_j)
    if d(x_i, C_k) > θ
        m = m+1
        C_m = {x_i}
    else
        C_k = C_k ∪ {x_i}
        (se necessario aggiornare i rappresentanti)
    end if
end for
```

(a) Algoritmo BSAS



(b) Stima del numero di clusters

7.3.2 Center-based clustering

Una tecnica per fare clustering partizionale center-based è l'*algoritmo K-means*. Tra le varianti del K-means c'è *ISODATA* (Iterative Self-Organizing Data Analysis Techniques), la quale permette lo split e il merge dei clusters risultanti e in cui, ad ogni iterazione, i clusters vengono aggiornati (divisi o fusi tra loro) a seconda della varianza e del numero di elementi di cui sono costituiti. La scelta della soglia è cruciale ed implica la scelta automatica del numero di clusters. Un'altra variante del K-means è *PAM* (Partitioning Around the Medoids), in cui invece delle medie dei clusters si prendono come riferimento i medoidi (punti più centrali).

7.3.3 Model-based clustering

Per affrontare il problema del clustering model-based si parte dall'idea di utilizzare un insieme di modelli per i clusters, con l'obiettivo di massimizzare il fit fra tali modelli e i dati. Assumendo che i dati siano generati da una mistura di funzioni di probabilità differenti (rappresentanti i clusters), si può affrontare il problema in due modi diversi:

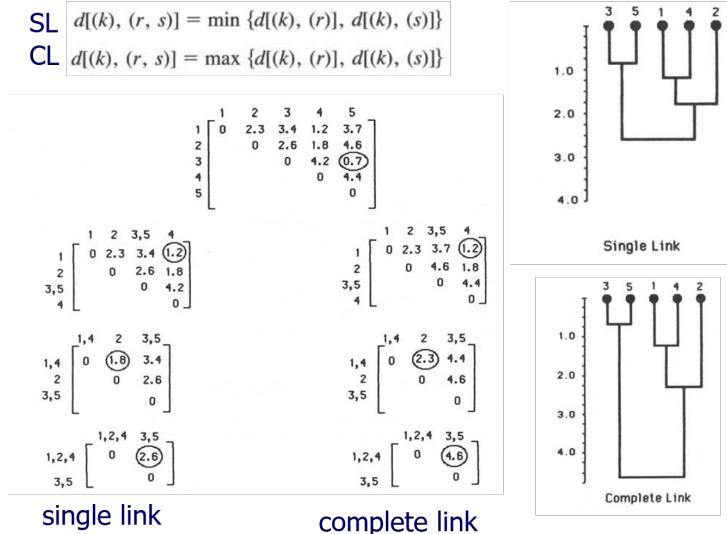
- *Classification likelihood approach*, il quale è equivalente al K-means se si assume che tutti i clusters abbiano la medesima matrice di covarianza e che questa sia proporzionale all'identità.
- *Mixture likelihood approach*, il quale sfrutta l'*algoritmo EM*; una tecnica di soft clustering in cui la mistura è composta da Gaussiane è il Gaussian Mixture Models (GMM).

Il problema di questi approcci è sempre lo stesso, ossia la scelta del numero di clusters in cui dividere i dati. Può essere paragonato al problema della scelta del numero di stati per gli HMM.

7.4 Clustering gerarchico

Un approccio per risolvere le problematiche del clustering gerarchico è l'utilizzo di algoritmi per la creazione di dendrogrammi sui dati, i quali generano una serie di partizioni innestate. Il numero di cluster non viene deciso a priori, ma dipende dalla soglia con cui si taglia l'albero dei link tra oggetti. Gli approcci più utilizzati sono:

- *Single link* (SL), il quale unisce due clusters se esiste un solo edge (tende a formare clusters allungati); si basa inoltre sull'assunzione che due clusters sono tanto simili quanto i loro elementi più simili (uno per cluster).
- *Complete link* (CL), il quale unisce due clusters se tutti gli elementi sono connessi, valutando quindi la similarità tra clusters prendendo gli elementi più dissimili fra loro; inoltre, è più conservativo, tende a formare cluster convessi ed è considerato l'approccio migliore.



7.5 Validazione

Per *validazione* del clustering si intende un insieme di procedure che valutano la qualità di un'analisi di clustering in termini quantitativi o oggettivi, ovvero misurando la capacità della struttura trovata di spiegare i dati indipendentemente dal contesto. A seconda del tipo di clustering (gerarchico o partizionale) si analizzano strutture diverse (rispettivamente gerarchie/dendrogrammi e categorie/partizioni).

La misura di validità si basa su una serie di strategie di validazione, ovvero gli *indici di validità*, i quali possono essere:

- Criteri esterni, che confrontano i risultati con la conoscenza a priori (etichette dei dati).
- Criteri interni, che misurano le performance di un clustering utilizzando solo i dati (completamente non supervisionato).
- Criteri relativi, che confrontano i risultati di due clustering differenti.

Quando si parla di indici di validità si fa riferimento ad una serie di misure statistiche. Perché un indice sia considerato attendibile deve essere intuitivo, basato su una teoria solida e facilmente calcolabile. Nel caso di criteri interni o esterni, si studia tipicamente quanto grande o piccolo sia un particolare indice di validità.

Nel caso di *clustering gerarchico* un possibile esempio di procedura di validazione è quella che misura la distanza nel dendrogramma tra due oggetti in base al livello in cui sono stati messi nello stesso cluster per la prima volta (similarità tra oggetti di un dendrogramma). Nel caso di *clustering partizionale*, invece, la validità è legata al numero di clusters identificati e tipicamente si va a confrontare il risultato del processo di clustering con un partizionamento conosciuto a priori (dovuto alle etichette assegnate agli oggetti). Quindi, la validità dei singoli cluster è fatta secondo misure di compattezza e isolamento.

Credits

Basato sulle slide fornite dal *prof. Marco Cristani*

Basato sui capitoli 2, 3, 4 e 5 del libro “*Pattern Classification*” (*Richard O. Duda, Peter E. Hart, David G. Stork*)

Vedere anche il gruppo Telegram **Magazzino Informatica Magistrale** per reperire ulteriore materiale

Repository github: <https://github.com/zampierida98/UniVR-informatica>
Indirizzo e-mail: zampieri.davide@outlook.com