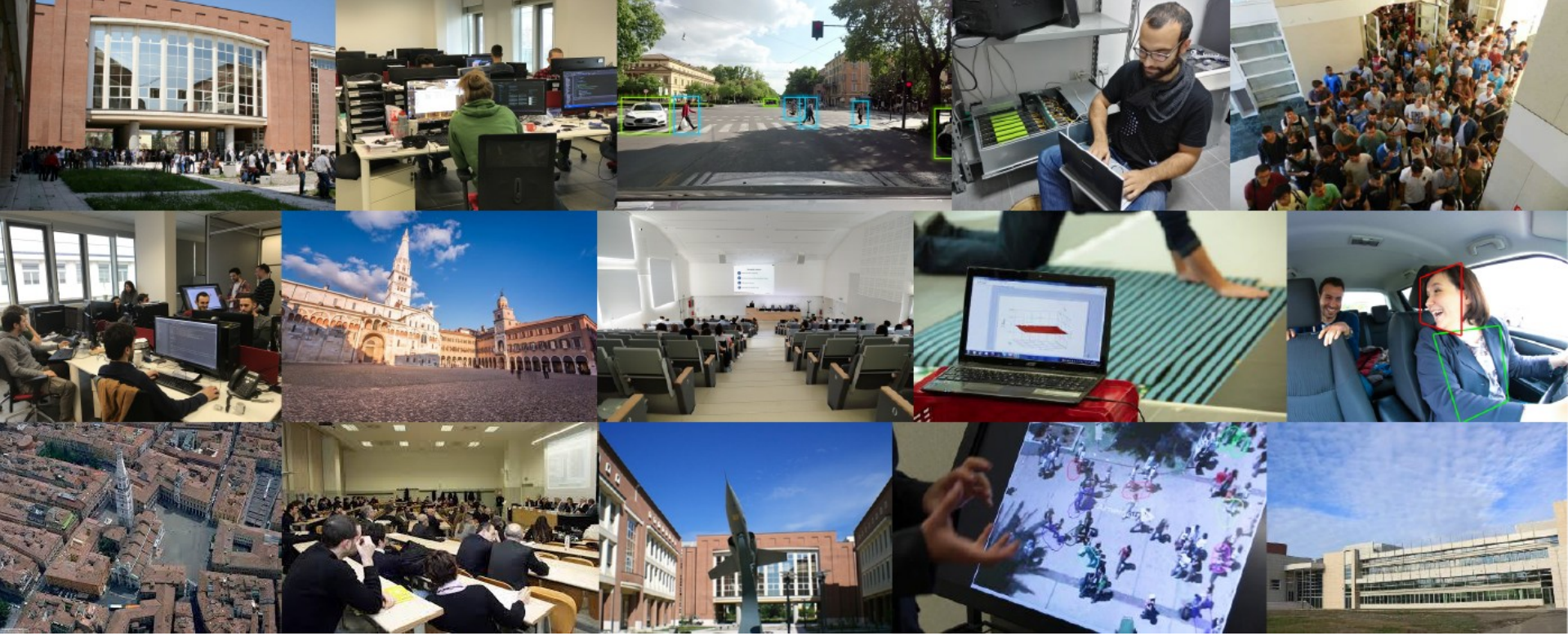


MACHINE LEARNING
- Metodi Non Supervisionati -

AImage Lab

UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Notazione

X	→	feature space
x_j, $j = 1, \ldots, d$	→	feature
$\mathbf{x}$, $\mathbf{x}^{(i)} = [x_1, \ldots, x_d]$	→	feature vector
$T = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$	→	dataset di training
$n = \lvert T \rvert$	→	numero di esempi di training
k	→	numero di cluster
$\mathbf{c}$, $\mathbf{m}$	→	baricentro o «centroide»
$\mathbf{c}_q$	→	baricentro o «centroide» del cluster q
n_q	→	numero di sample nel cluster q
C_q	→	insieme di sample del cluster q
$\boldsymbol{S}$	→	scatter matrix
$\lambda_1, \ldots, \lambda_i, \ldots, \lambda_d$ $\boldsymbol{e_1}, \ldots, \boldsymbol{e_i}, \ldots, \boldsymbol{e_d}$	→	autovalori e autovettori della scatter matrix

Metodi non supervisionati

Quando la variabie target (*y*) non esiste, oppure esiste ma il suo valore non è noto in fase di training (non ho la ground truth per *y*) allora si parla di metodi non supervisionati (**unsupervised**)

Il dataset di training quindi sarà del tipo $T = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$

Che tipo di task posso avere se non esiste la variabile target?

Ci sono diversi task e diverse applicazioni dei metodi non supervisionati. Due tra le principali (di cui vedremo esempi in questa lezione) sono:

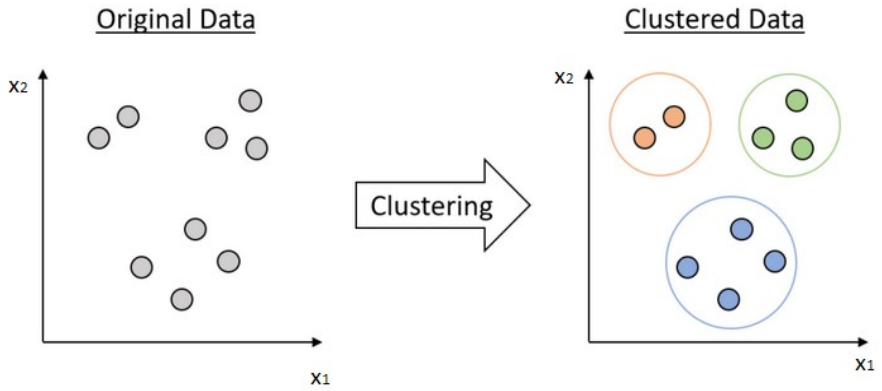
1. Clustering
2. Trasformazione delle feature

Clustering

Clustering

Un tipico task unsupervised è il clustering, ovvero il raggruppamento dei feature vector in sotto-gruppi spazialmente distinti

L'idea generale alla base del clustering è di raggruppare i dati rispetto alla loro «similitudine» (calcolata nel feature space)



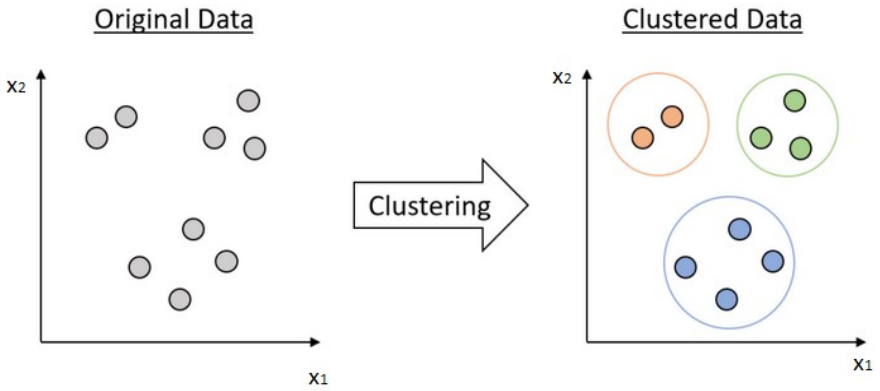
Clustering

Potrei, ad esempio, usare il clustering per suddividere i clienti di un servizio on-line in base ai loro profili o preferenze (rappresentati da corrispondenti feature vector)

Dopodichè, potrei suggerire ad un dato cliente di acquistare servizi già acquistati da altri clienti nel suo stesso gruppo, sfruttando l'ipotesi che, se i clienti sono «simili» allora probabilmente gradiscono prodotti simili

Oppure i feature vector potrebbero rappresentare, e.g., i meta-dati di contenuti mutimediali su un social media

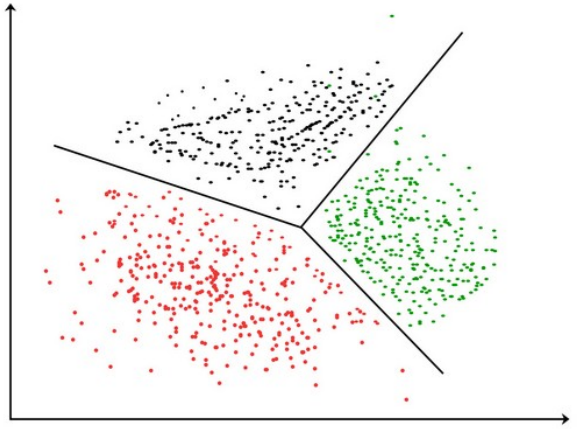
Dopo aver clusterizzato questi meta-dati, se un utente fruisce di un video in un dato cluster, la piattaforma può suggerirgli altri video dello stesso cluster



Clustering

In termini più precisi, i metodi di **clustering** (raggruppamento) suddividono i dati in gruppi (**cluster**) in modo tale che i punti "simili" finiscano nello stesso gruppo, mentre punti diversi finiscano in gruppi diversi

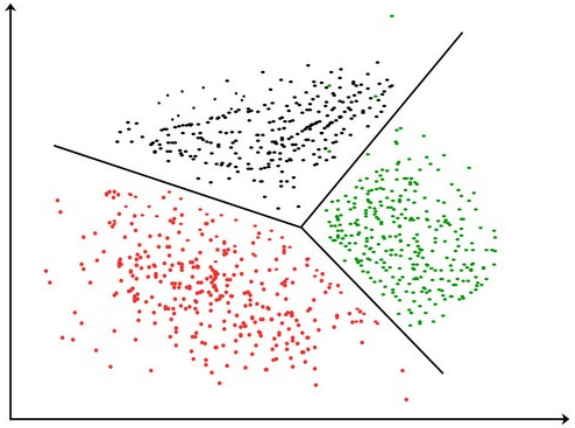
In qualche modo il clustering ricorda la classificazione, nella quale esempi simili appartengono alla stessa classe, con l'importante differenza che *non ho a disposizione il valore di y* (la variabile che mi definisce il raggruppamento) e, anzi, tale valore è proprio ciò che voglio trovare



Nell'esempio a fianco, il «colore» dei gruppi non è dato a training time, ed è il *risultato* del clustering. Ovvero, solo dopo che ho risolto il task del clustering posso capire, dato $\boldsymbol{x}^{(i)} \in T$, quali sono gli altri esempi appartenenti allo stesso gruppo

Clustering

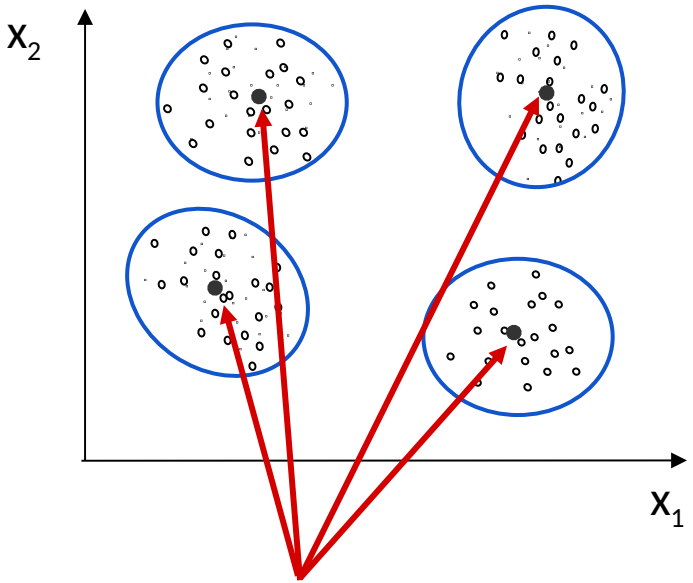
Inoltre, tipicamente, non so neanche quanti sono i cluster in un insieme
Il numero di cluster (*k*) da cercare devo trattarlo come un iper-parametro e fare prove diverse
Oppure *k* dipende dall'applicazione. Ad esempio: voglio *k* gruppi di clienti



k-means

Il k-means è l'algoritmo di clustering più usato, ed è anche uno dei più semplici

- Rappresenta la (dis-)similitudine tra punti con una distanza geometrica, (tipicamente la distanza **Euclidea**)
- Rappresenta ogni cluster col suo punto medio (*mean*, da cui il nome), detto anche baricentro o *centroide*
- Il centroide di un cluster è ottenuto sommando tutti i punti di quel cluster e dividendo per il numero di punti del cluster

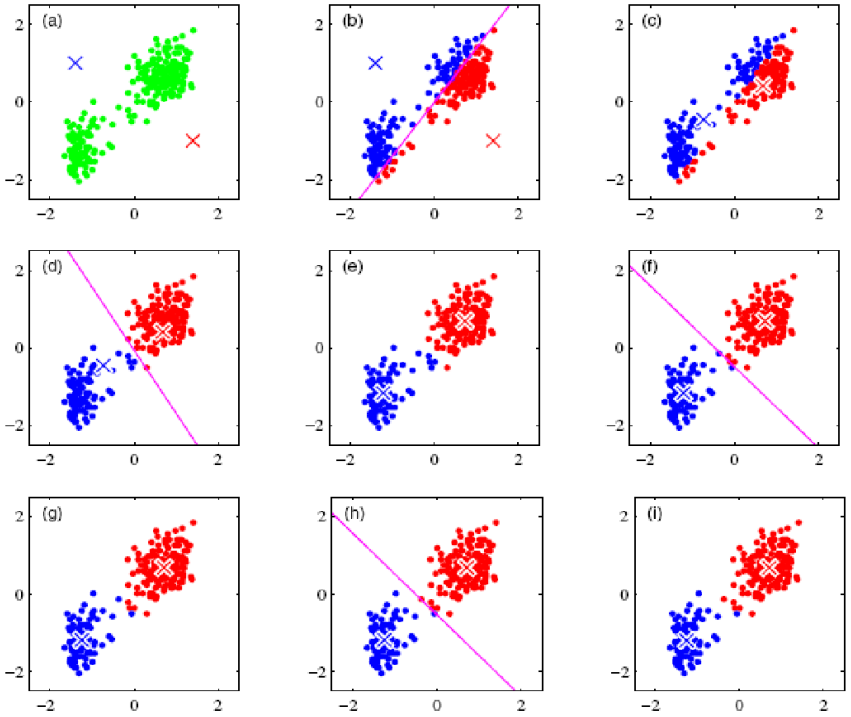


Centri dei cluster

k-means

Il k-means è un algoritmo greedy che può essere riassunto come segue:

1. Si scelgono **a caso** *k* centroidi $\{\mathbf{c}_q\}$ ($q = 1,\ldots, k$). Ad esempio, potrei inizializzare i centroidi estraendo a caso *q* sample da *T*, oppure scegliendoli all'interno del range di valori delle feature. Tutti i cluster sono inizializzati come vuoti
2. Si *assegna* ciascun punto $\mathbf{x} \in T$ al cluster C_q per cui è minima la distanza Euclidea $||\mathbf{x} - \mathbf{c}_q||$
3. Si *calcolano* i centri dei nuovi gruppi
4. Si itera finché gli spostamenti dei centroidi rispetto al passo precedente non diventano trascurabili



Hierarchical Methods

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Un'altra tipologia di metodi di clustering sono gli **algoritmi gerarchici**, che possono essere:

- **Agglomerativi**
- **Divisivi**

e che si basano unicamente sulla possibilità di misurare la distanza tra due cluster C_i e C_j

Sono meno comuni del k-means, ma hanno alcuni vantaggi, tra cui quello di poter essere usati per decidere il numero di cluster (k) in base ad una distanza massima consentita per gli elementi dello stesso cluster

Hierarchical Methods

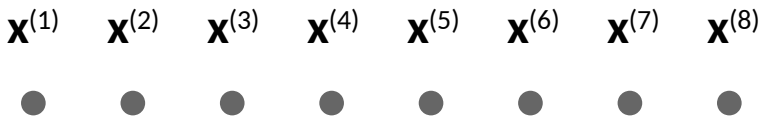
Nei metodi gerarchici *agglomerativi* si parte con n cluster C_1, …, C_n, ovvero si inizializzano n cluster, *ognuno composto da un singolo sample* di T

Dopodiché si procede iterativamente unendo sempre i due cluster più vicini

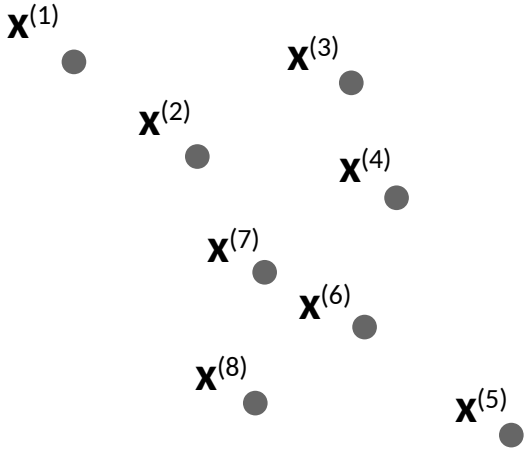
Nei metodi divisivi si procede al contrario, partendo da un unico cluster e dividendolo progressivamente

La natura gerarchica di questi metodi può essere illustrata con un albero detto **dendrogramma**

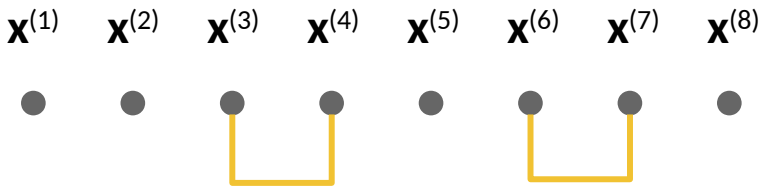
Hierarchical Methods



1° livello

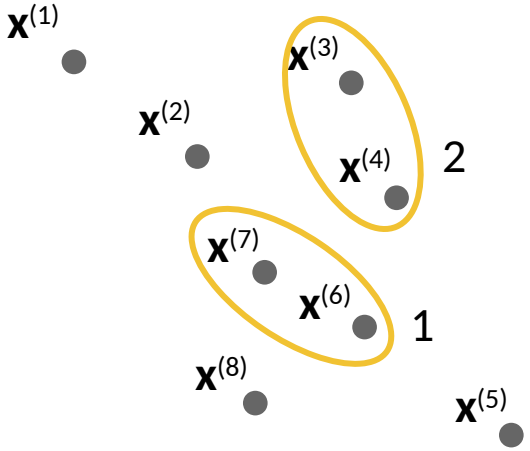


Hierarchical Methods

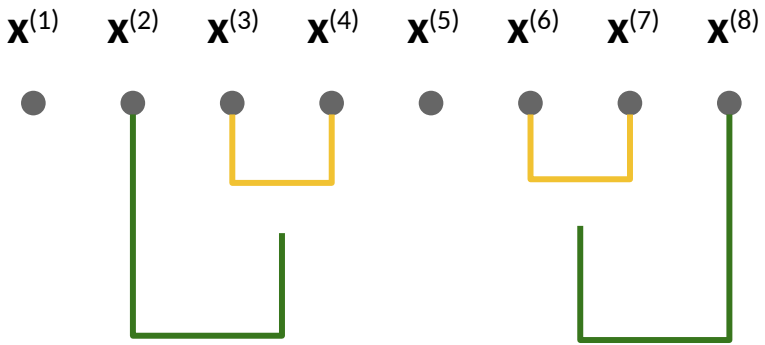


1° livello

2° livello



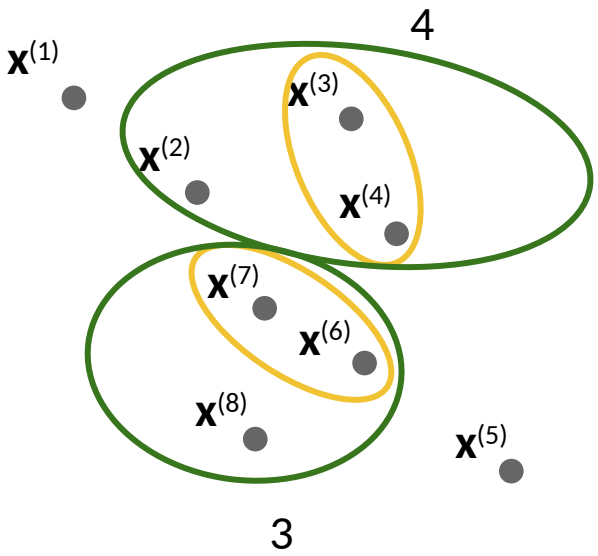
Hierarchical Methods



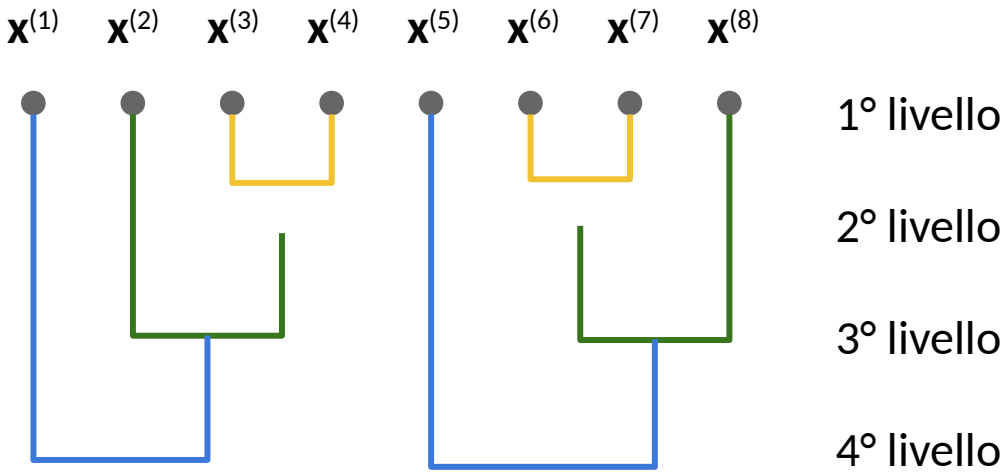
1° livello

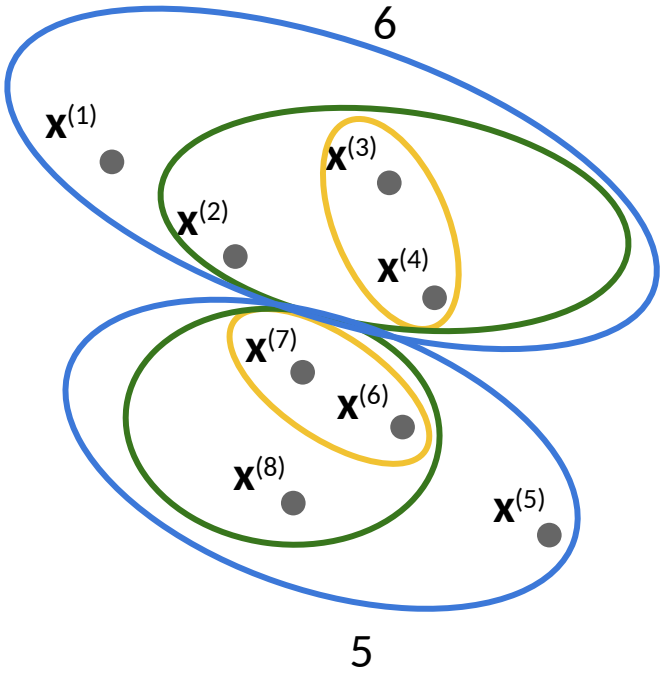
2° livello

3° livello

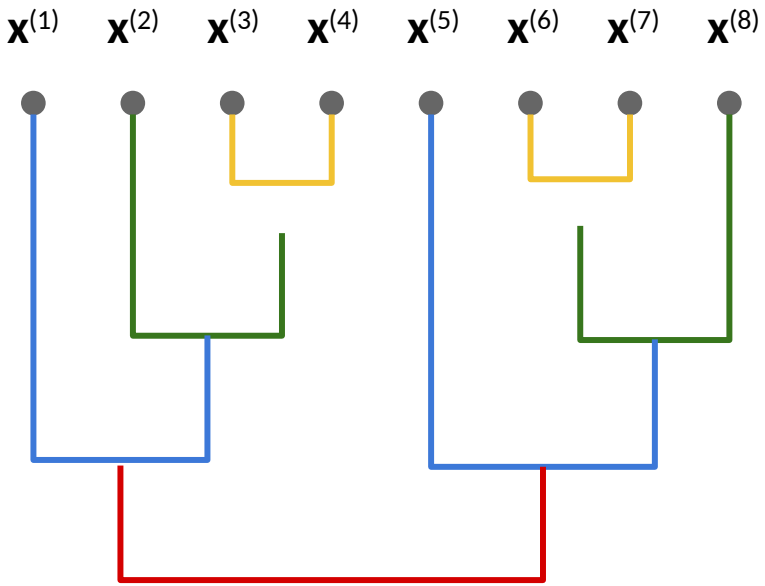


Hierarchical Methods





Hierarchical Methods



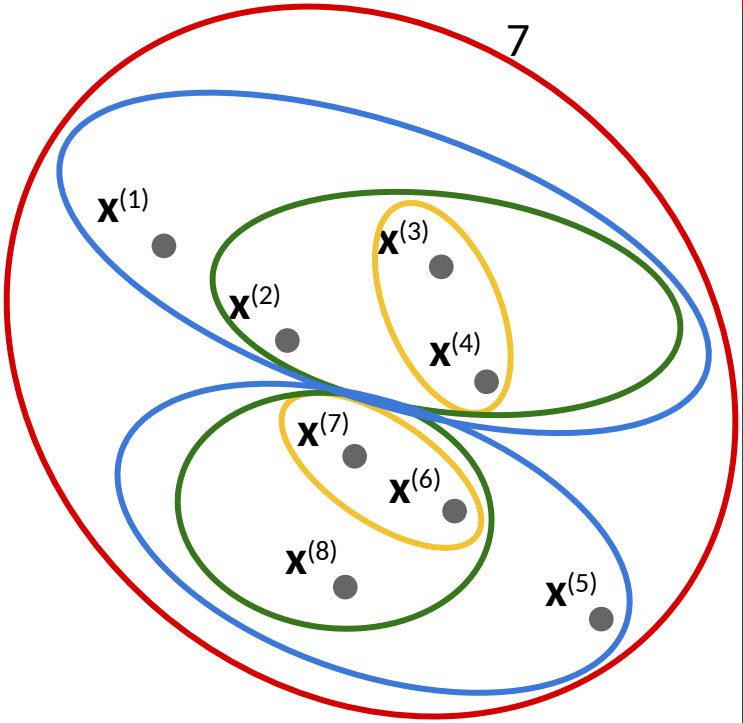
1° livello

2° livello

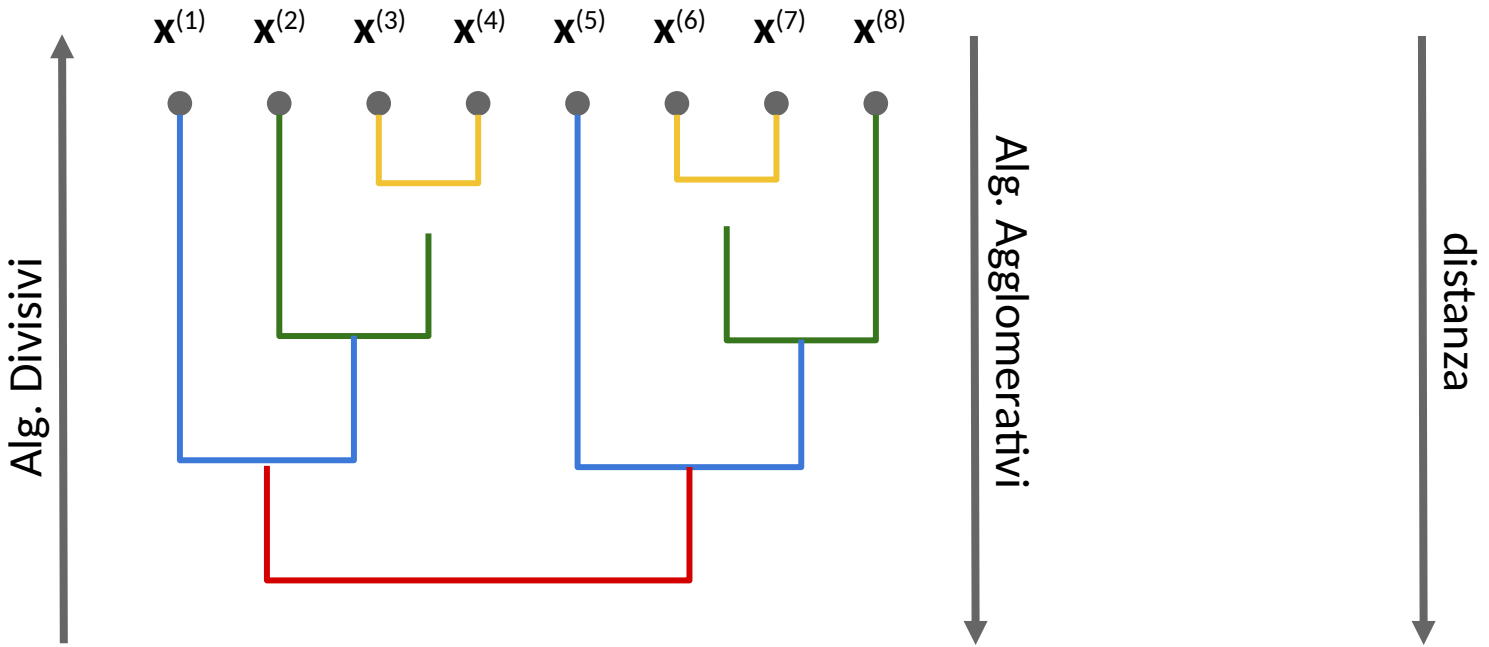
3° livello

4° livello

5° livello



Hierarchical Methods



Agglomerative Clustering

Se k è il numero di cluster desiderati, l'algoritmo gerarchico agglomerativo è il seguente:

1. Sia inizialmente $m = n$ e $C_1 = \{\boldsymbol{x}^{(1)}\}$, …, $C_n = \{\boldsymbol{x}^{(n)}\}$ (ovvero inizializzo n cluster di un solo elemento)
2. Si trovano i due cluster più vicini C_i e C_j
3. Si fondono insieme C_i e C_j
4. $m := m - 1$
5. Si ripetono i passi 2-4 finché $m = k$

Distanze possibili per i metodi gerarchici

Per misurare la vicinanza tra due cluster è possibile usare distanze diverse:

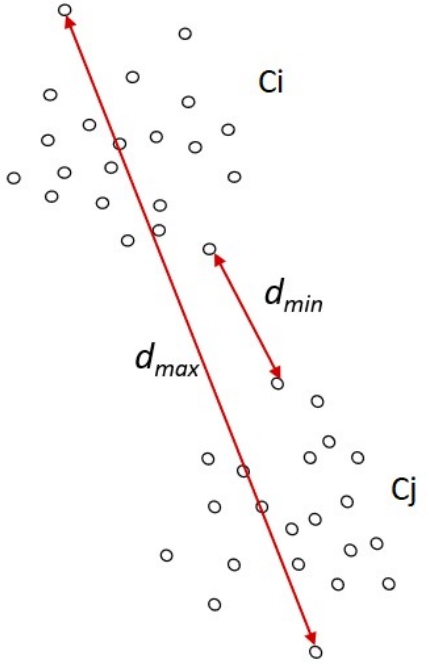
$$d_{min}(C_i, C_j) = \min_{\substack{x \in C_i \\ x' \in C_j}} \|x - x'\|$$

$$d_{max}(C_i, C_j) = \max_{\substack{x \in C_i \\ x' \in C_j}} \|x - x'\|$$

$$d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{x \in C_i} \sum_{x' \in C_j} \|x - x'\|$$

$$d_{mean}(C_i, C_j) = \|m_i - m_j\|$$

In $d_{mean}()$, $\boldsymbol{m}_i$ e $\boldsymbol{m}_j$ indicano i centroidi di C_i e C_j

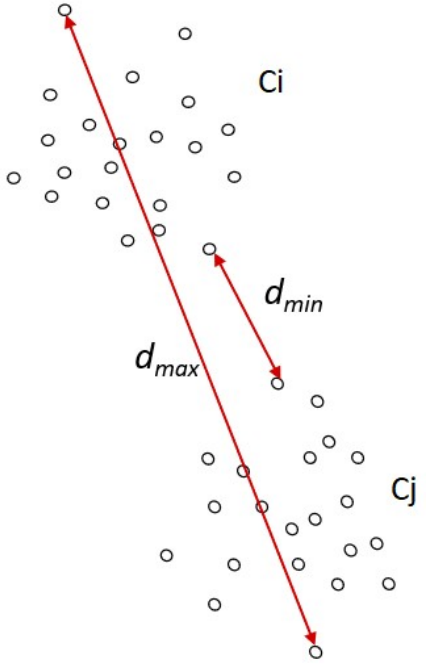


Determinare *k* in base ad una soglia di distanza

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Questa seconda versione dell'algoritmo agglomerativo permette di determinare *k* in base ad una distanza massima prefissata *th* (che diventa il nuovo iper-parametro):

1. Sia inizialmente $C_1 = \{\boldsymbol{x}^{(1)}\}$, …, $C_n = \{\boldsymbol{x}^{(n)}\}$ (ovvero inizializzo *n* cluster di un solo elemento) e *th* una soglia prefissata
2. Si trovano i due cluster più vicini C_i e C_j
3. Se la loro distanza è maggiore di *th*, allora mi fermo
4. Altrimenti fondo C_i e C_j e ripeto i passi 2-4



In particolare:

- Se uso come distanza tra cluster d_{min}, allora il metodo prende il nome di **single-linkage**
- Se uso d_{max}, allora si chiama **complete-linkage**

Trasformazione delle feature

Principal Component Analysis

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Abbiamo visto in passato che, per limitare l'overfitting, potrei usare la feature selection

La Principal Component Analysis (**PCA**) è un metodo alternativo (unsupervised) ai metodi di filter e di wrapper, con un funzionamento diverso da entrambi ma che ha in comune con i primi (filter) l'idea di selezionare le feature *prima e indipendentemente dal* metodo scelto per risolvere uno specifico task di classificazione/regressione

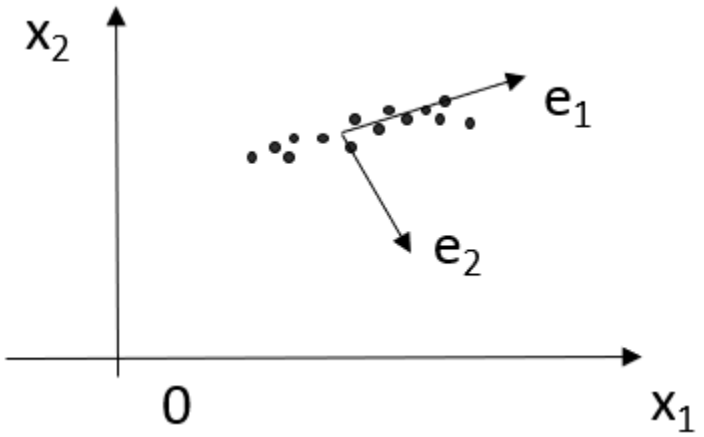
Principal Component Analysis

Nella PCA le feature vengono prima *trasformate* cercando una nuova *base* di rappresentazione del feature space e poi *selezionate* in base alla loro importanza

L'idea intuitiva è mostrata a fianco: la nuova base è costituita dai vettori $\boldsymbol{e}_1$ ed $\boldsymbol{e}_2$, che sono *ortogonali* tra di loro

Per ora tralasciamo il modo in cui $\boldsymbol{e}_1$ ed $\boldsymbol{e}_2$ vengono scelti (intuitivamente $\boldsymbol{e}_1$ ed $\boldsymbol{e}_2$ dovrebbero rappresentare le direzioni di dispersione principale dei dati in T)

Una volta scelti, $\boldsymbol{e}_1$ ed $\boldsymbol{e}_2$ sostituiranno le feature originarie x_1 ed x_2



Principal Component Analysis

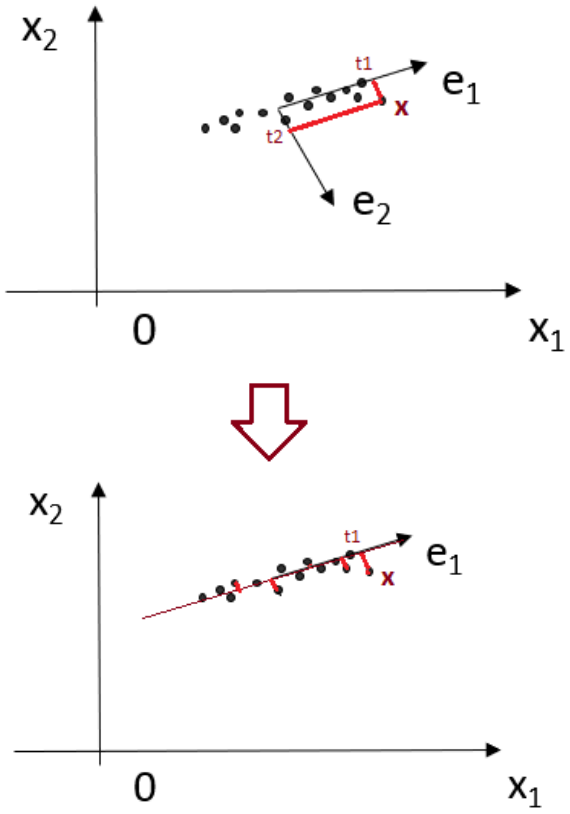
Ad esempio, dato un generico sample $\boldsymbol{x}$ (espresso usando le feature originarie), posso trasformarlo *proiettandolo* nella nuova base ed ottenere $\boldsymbol{t} = [t_1, t_2]$

A questo punto decido di "sacrificare" $\boldsymbol{e}_2$ perchè è la direzione che contribuisce in misura minore a descrivere la variabilità del dataset T e proietto T su $\boldsymbol{e}_1$

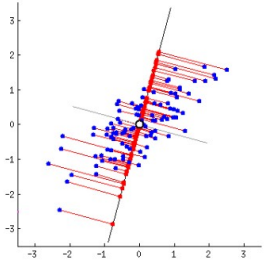
Se invece decidessi di eliminare $\boldsymbol{e}_1$, avrei una maggiore perdita di informazione una volta proiettati i dati su $\boldsymbol{e}_2$

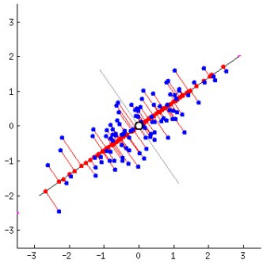
La rappresentazione finale di $\boldsymbol{x}$, in quest'esempio, sarà $\boldsymbol{t} = [t_1]$

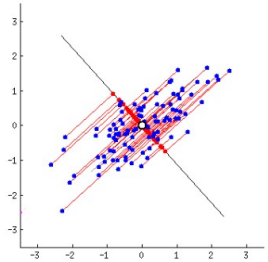
Lo scopo della PCA è trovare una base ortogonale di vettori che rappresentino le direzioni di maggiore variabilità dei dati in T, per poi eliminare quei vettori meno "significativi"



Principal Component Analysis







Il fatto che noi vogliamo preservare la maggiore variabilità possibile dei dati in *T* ci dà un criterio da seguire per la scelta delle "componenti principali"

Infatti io vorrei che $\boldsymbol{e}_1$ (la "prima componente") corrispondesse alla linea sulla cui proiezione ottengo la maggiore *varianza* dei dati

In generale avrò *d* feature nel feature space originario, per cui devo trovare una base ortonormale di *d* vettori $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d$

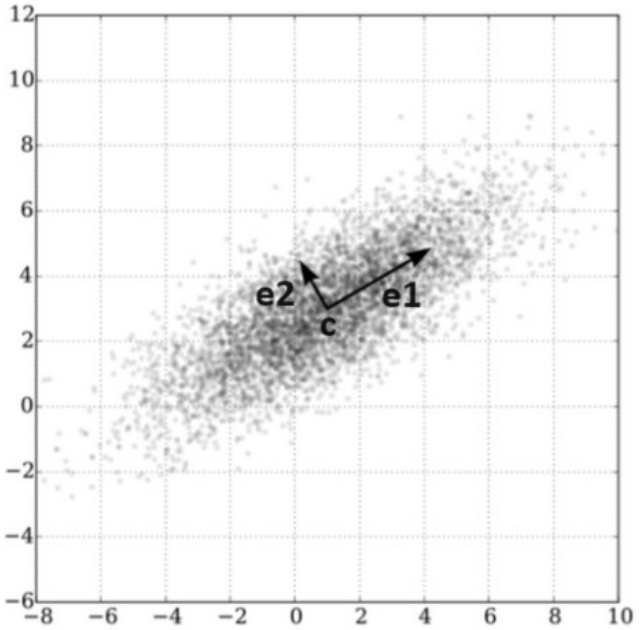
Per cui, una volta scelta la prima componente, la seconda componente deve essere ortogonale alla prima e corrispondere alla seconda maggiore direzione di variabilità dei dati

E così via per le altre componenti

Principal Component Analysis

Assumendo che i dati in *T* abbiano una distribuzione Gaussiana multidimensionale (cosa frequente ma non sempre vera…), la scelta delle componenti principali avviene in modo tale che (v. esempio mostrato nella figura a fianco):

- La nuova base sarà centrata sul centroide ***c*** di *T* (i.e., la media della Gaussiana)
- Le componenti principali corrispondono agli assi dell'ellissoide corrispondente alla Gaussiana, che a loro volta corrispondono agli autovettori della matrice di covarianza della Gaussiana



Una volta ottenuta la nuova base $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d$, scelgo di tenere le *p* componenti principali $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_p$ (corrispondenti ai *p* assi principali della Gaussiana) e di eliminare le *d-p* componenti minori

Questo procedimento può essere implementato usando l'algoritmo che segue

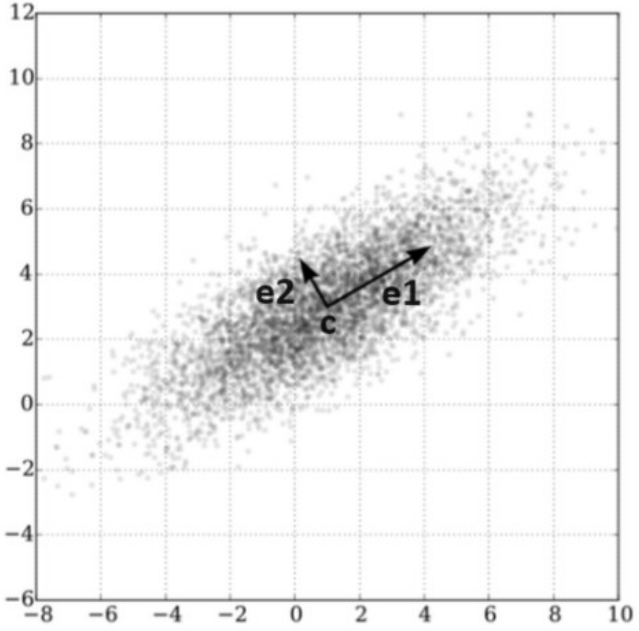
Principal Component Analysis: algoritmo

A partire da *T*, a matrice di covarianza ***S*** può essere calcolata usando:

$$\boldsymbol{S} = 1/n \sum_{i=1}^{n} (\boldsymbol{x}^{(i)} - \boldsymbol{c})(\boldsymbol{x}^{(i)} - \boldsymbol{c})^T$$

$$\boldsymbol{c} = 1/n \sum_{i=1}^{n} \boldsymbol{x}^{(i)}$$

dove ***c*** è il baricentro di *T*
S rappresenta la correlazione tra le feature originarie
Gli autovettori $\boldsymbol{e}_1$, …, $\boldsymbol{e}_d$ di ***S*** corrispondono alle direzioni di maggiore variabilità dell'informazione in *T*
S è semidefinita positiva, per cui ha tutti gli autovalori non negativi
L'autovalore λ_i corrispondente all'i-esimo autovettore $\boldsymbol{e}_i$ è proporzionale alla varianza degli esempi se proiettati sulla retta la cui direzione è rappresentata da $\boldsymbol{e}_i$



Principal Component Analysis: algoritmo

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

I passi fondamentali della PCA sono:

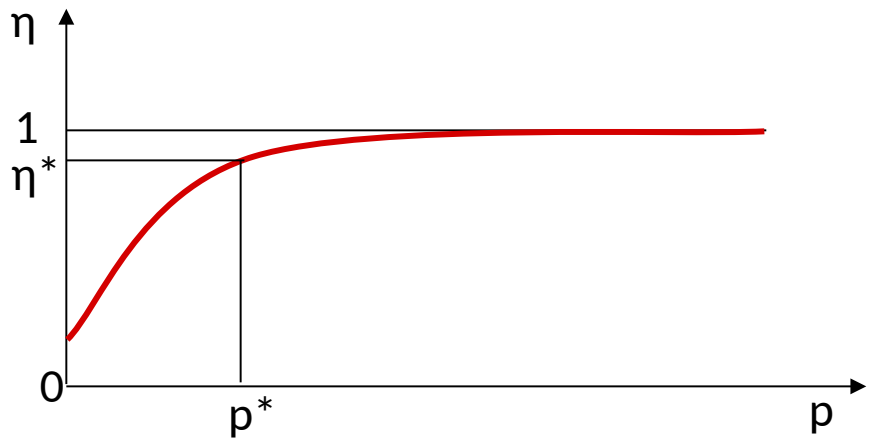
- Si calcola il baricentro $\boldsymbol{c}$ degli esempi in T
- Si calcola la matrice di covarianza $\boldsymbol{S}$
- Si ricavano gli autovalori $\lambda_1, \ldots, \lambda_i, \ldots, \lambda_d$ e gli autovettori $\boldsymbol{e_1}, \ldots, \boldsymbol{e_i}, \ldots, \boldsymbol{e_d}$ di $\boldsymbol{S}$
- Si ordinano gli autovettori $\{\boldsymbol{e_i}\}_{i=1,\ldots,n}$ in ordine decrescente rispetto alla grandezza dei corrispondenti autovalori $\{\lambda_i\}_{i=1,\ldots,n}$
- Si normalizzano gli autovettori per renderli una base: $\mathbf{e}_i := \mathbf{e}_i / ||\mathbf{e}_i||$
- Si selezionano i primi p autovettori $\boldsymbol{e_1}, \ldots, \boldsymbol{e_P}$ (corrispondenti ai p autovalori più grandi)
- È ora possibile trasformare un generico feature vector $\boldsymbol{x}$ (rappresentato con le feature originarie) nel nuovo feature vector $\boldsymbol{t} = [t_1, \ldots, t_p]$, dove: $t_i = \boldsymbol{e_i}^T (\boldsymbol{x} - \boldsymbol{c})$, $i = 1, \ldots, p$

Principal Component Analysis: iper-parametro

p è un iper-parametro che devo fissare

Posso farlo facilmente tenendo presente che la proporzione della varianza rappresentata da ciascun autovettore $\boldsymbol{e}_i$ può essere calcolata dividendo l'autovalore corrispondente (λ_i) per la somma di tutti gli autovalori:

- Decido la percentuale di varianza (informazione) che voglio mantenere. E.g., $\eta^* = 0.9$ (=90%)
- Scelgo il valore p^* corrispondente a η^* usando la formula qui sotto:

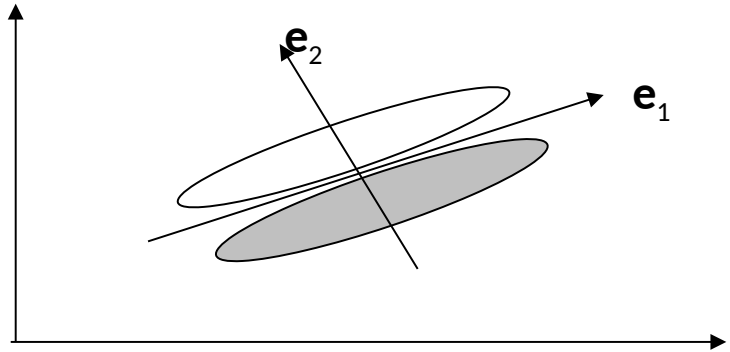


$$\eta = \frac{\sum_{l=1}^{p} \lambda_l}{\sum_{l=1}^{d} \lambda_l}$$

Principal Component Analysis: limitazioni

Le due maggiori limitazioni della PCA sono:

- Se i dati non hanno una distribuzione Gaussiana, le componenti principali calcolate usando l'algoritmo della PCA potrebbero non essere realmente rappresentative della distribuzione di *T*
- Similmente ai metodi di Filter, anche la PCA, eliminando le feature *prima e indipendentemente dal* training dello specifico classificatore/regressore, fa delle scelte che sono indipendenti dal task e dal metodo specifico usato per risolvere quel task, e che, quindi, potrebbero essere sub-ottimali (v. ese. nella figura qui sotto per un caso di classificazione binaria)



Riferimenti

- **A tu per tu col Machine Learning. L'incredibile viaggio di un developer nel favoloso mondo della Data Science**, Alessandro Cucci, The Dot Company, 2017 [cap.9]
- http://cs229.stanford.edu/notes2020spring/cs229-notes7a.pdf
- http://cs229.stanford.edu/notes2020spring/cs229-notes10.pdf
- https://builtin.com/data-science/step-step-explanation-principal-component-analysis
- Rousseeuw, P. J. (1987). **Silhouettes: a graphical aid to the interpretation and validation of cluster analysis.** Journal of computational and applied mathematics, 20, 53-65.
- Caliński, T., & Harabasz, J. (1974). **A dendrite method for cluster analysis.** Communications in Statistics-theory and Methods, 3(1), 1-27.
- Davies, D. L., & Bouldin, D. W. (1979). **A cluster separation measure.** IEEE transactions on pattern analysis and machine intelligence, (2), 224-227.
- **Pattern Classification, second edition,** R. O. Duda, P. E. Hart, D. G. Stork, Wiley-Interscience, 2000
- Link contenente la demo grafica vista a lezione: https://builtin.com/data-science/step-step-explanation-principal-component-analysis