



Fondamenti di Machine Learning

Laurea Triennale in Ingegneria delle Comunicazioni

8: Clustering e dimensionality reduction

Lecturer: S. Scardapane



SAPIENZA
UNIVERSITÀ DI ROMA

Unsupervised learning

Introduzione

Finora abbiamo considerato uno scenario **supervisionato**, nel quale conosciamo la predizione y ottimale per ogni input \mathbf{x} . Andiamo ora a considerare uno scenario **non supervisionato** (*unsupervised*) dove questa informazione non è disponibile.

In particolare, ci interessiamo a due problemi:

1. **Dimensionality reduction**: per ogni $\mathbf{x} \in \mathbb{R}^d$ vogliamo trovare un $\mathbf{x}' \in \mathbb{R}^q$, con $q \ll d$ e tale che (informalmente) $\mathbf{x}' \approx \mathbf{x}$.
2. **Clustering**: una forma estrema di dimensionality reduction in cui vogliamo raggruppare gli elementi in k *buckets* (**clusters**) omogenei fra loro.

Principal component analysis

Definizioni di base

Abbiamo una matrice $\mathbf{X} \in \mathbb{R}^{n \times d}$ con n esempi, ciascuno descritto da d feature. Vogliamo 'riassumere' la matrice in una seconda matrice $\mathbf{H} \in \mathbb{R}^{n \times q}$, con $q \leq d$:

- ▶ Con $q = d$ questo diventa uno step di preprocessing.
- ▶ Con $q < d$ abbiamo uno step di **feature reduction**.
- ▶ Se $q \leq 3$ possiamo usare questa trasformazione per graficare i dati.

Consideriamo una funzione $f(\mathbf{x}) = \mathbf{h}$ che vogliamo ottimizzare per ridurre la dimensionalità. Non possiamo confrontare immediatamente \mathbf{h} ed \mathbf{x} , in quanto hanno dimensionalità diverse.

Supponiamo però di saper (parzialmente) invertire questa trasformazione, $g(\mathbf{h}) \approx \mathbf{x}$. In questo caso possiamo formulare il problema di dimensionality reduction come un problema di ricostruzione:¹

$$f^*, g^* = \arg \min_{f, g} \left\{ \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - g(f(\mathbf{x}_i))\|^2 \right\} \quad (1)$$

¹Questo non è l'unico approccio possibile: ad esempio, per visualizzare i dati potremmo voler mantenere le distanze $d(\mathbf{x}_i, \mathbf{x}_j) \approx d(\mathbf{h}_i, \mathbf{h}_j)$, come fatto in t-SNE o UMAP.

Esistono infiniti algoritmi per la dimensionality reduction, in questa lezione ne vediamo due:

1. Se scegliamo una trasformazione lineare $\mathbf{h} = \mathbf{V}\mathbf{x}$ con $\mathbf{V} \in \mathbb{R}^{q \times d}$, sotto alcuni vincoli su \mathbf{V} otteniamo la **principal component analysis** (PCA). Nella PCA, in particolare, \mathbf{V} è **ortonormale**² tale che $g(\mathbf{h}) = \mathbf{V}^T \mathbf{h}$.
2. Se implementiamo f e g con due reti neurali otteniamo un **autoencoder**.

²Ogni colonna di \mathbf{V} ha norma unitaria, e le colonne sono ortogonali a coppie.

Possiamo descrivere una qualsiasi direzione nello spazio Euclideo con un vettore $\mathbf{v} \in \mathbb{R}^d$ a norma unitaria, $\|\mathbf{v}\| = 1$. Il vettore:

$$\mathbf{x}_v = \mathbf{v}\mathbf{v}^\top \mathbf{x} \quad (2)$$

è la **proiezione** di \mathbf{x} sull'asse definito da \mathbf{v} .³ $\mathbf{v}^\top \mathbf{x}$ è invece la distanza su quest'asse dall'origine a \mathbf{x}_v , ed infine:

$$\mathbf{x}_\perp = \mathbf{x} - \mathbf{x}_v \quad (3)$$

rappresenta il sottospazio **ortogonale** a \mathbf{x}_v .

³La matrice $d \times d$ $\mathbf{P} = \mathbf{v}\mathbf{v}^\top$ è una matrice a rango 1 detta di proiezione sul sottospazio definito da \mathbf{v} .

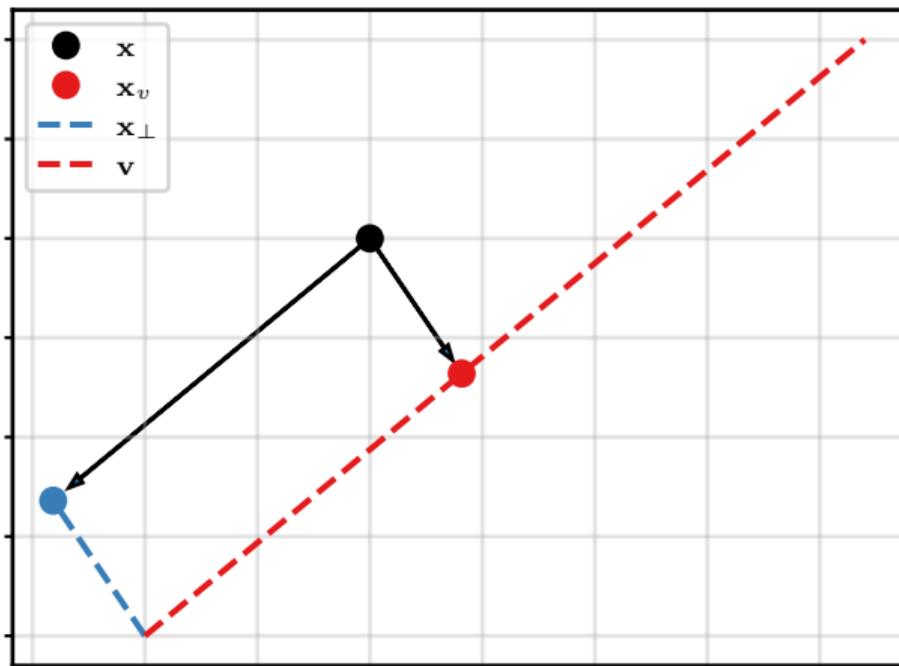


Figure 1: Un punto x in \mathbb{R}^2 proiettato su una linea v , insieme alla rispettiva proiezione ortogonale.

Se consideriamo d vettori $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d$ fra loro ortogonali, questi definiscono una **base** nello spazio Euclideo. Il vettore:

$$\mathbf{h} = \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_d^T \end{bmatrix} \mathbf{x} = \mathbf{V}\mathbf{x} \quad (4)$$

rappresenta le **coordinate** di \mathbf{x} nel nuovo orientamento definito da \mathbf{V} , ottenuto tramite una rotazione ed uno scaling degli assi di partenza. Quello che cerchiamo è una base nel quale gli assi siano ordinati per 'importanza'.

Per semplicità assumiamo che la matrice \mathbf{X} ha media nulla sulle colonne (in caso alternativo, è possibile eseguire un preprocessing per garantire questa proprietà).

Vedremo che nel seguito gioca un ruolo molto importante la **covarianza** della matrice \mathbf{X} , definita nel caso di media nulla come:

$$\mathbf{S} = \frac{1}{n} \mathbf{X}^T \mathbf{X} \quad (5)$$

Gli elementi sulla diagonale rappresentano la varianza delle colonne di \mathbf{X} , mentre gli altri elementi rappresentano le correlazioni tra coppie di feature.

La seguente relazione ci sarà molto utile. Data una proiezione 1D definita dal vettore \mathbf{v} , la varianza dei punti sul nuovo asse è uguale a:

$$\text{var}(\mathbf{Xv}) = \mathbf{v}^T \mathbf{Sv} \quad (6)$$

Vedremo che la PCA si può interpretare in maniera equivalente come una proiezione che massimizza la varianza dei nuovi dati, oppure come una proiezione che minimizza l'errore di ricostruzione.

Principal component analysis

PCA in 1 dimensione

Consideriamo per ora una proiezione 1D. Da quanto detto prima, il miglior asse \mathbf{v} è quello che garantisce il minor errore quadratico medio. Consideriamo un generico punto \mathbf{x} , abbiamo che:

$$(\mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x})^2 = \mathbf{x}^T\mathbf{x} - (\mathbf{v}^T\mathbf{x})^2 \quad (7)$$

Per minimizzare l'errore è quindi sufficiente massimizzare $(\mathbf{v}^T\mathbf{x})^2$.

Consideriamo ora gli n punti originali, abbiamo la seguente funzione costo:

$$\mathbf{v}^* = \arg \max_{\|\mathbf{v}\|=1} \left\{ \frac{1}{n} \sum_{i=1}^n (\mathbf{v}^\top \mathbf{x}_i)^2 \right\} = \arg \max_{\|\mathbf{v}\|=1} \{ \mathbf{v}^\top \mathbf{S} \mathbf{v} \} \quad (8)$$

dove $\mathbf{S} = \frac{1}{n} \mathbf{X}^\top \mathbf{X}$ è la matrice di covarianza dei dati. La soluzione ottimale è quindi scegliere una proiezione che massimizza la varianza dei punti proiettati su quell'asse (una direzione nella quale la varianza è nulla è, fondamentalmente, inutile).

A differenza di quanto visto finora, la funzione costo è **vincolata** e richiede l'introduzione dei cosiddetti **moltiplicatori di Lagrange**.⁴ In particolare, consideriamo la funzione costo alternativa e **non vincolata**:

$$L(\mathbf{v}) = \mathbf{v}^T \mathbf{S} \mathbf{v} - \lambda(\mathbf{v}^T \mathbf{v} - 1) \quad (9)$$

La soluzione ottimale rispetta sia $\nabla_{\mathbf{v}} L(\mathbf{v}) = 0$ (come nel caso non vincolato) che $\nabla_{\lambda} L(\mathbf{v}) = 0$. In questo caso la seconda equazione ritorna solo il vincolo $\mathbf{v}^T \mathbf{v} = 1$.

⁴Una derivazione alternativa considera un vettore non vincolato \mathbf{v} ed il cosiddetto **quoziente di Rayleigh** $\frac{\mathbf{v}^T \mathbf{S} \mathbf{v}}{\mathbf{v}^T \mathbf{v}}$.

Considerando il gradiente rispetto a \mathbf{v} otteniamo:

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v} \quad (10)$$

Questo ci dice che la soluzione ottimale è scegliere l'**autovettore** della matrice di covarianza corrispondente all'**autovalore** più alto: possiamo farlo con qualsiasi libreria di algebra lineare, ad esempio, **eig** in NumPy. Gli autovalori sono in genere numeri complessi, ma poiché \mathbf{S} è una matrice di covarianza (simmetrica e definita positiva) in questo caso sono tutti reali e positivi.

Principal component analysis

Algoritmo generale della PCA

Denotiamo il vettore di prima con \mathbf{v}_1 . Possiamo seguire iterativamente lo stesso ragionamento per trovare un secondo vettore (e quindi una seconda coordinata) dove proiettare i nostri dati.

In particolare, possiamo costruire un nuovo dataset $\mathbf{X}' = \mathbf{X} - \mathbf{X}\mathbf{v}\mathbf{v}^\top$, che rappresenta il sottospazio non 'considerato' dalla proiezione su \mathbf{v}_1 . Il ragionamento di prima dimostra che la nuova soluzione è l'autovettore di \mathbf{S} corrispondente al **secondo autovalore** in valore assoluto.

L'algoritmo risultante viene detto **Principal Component Analysis** (PCA):

1. Partiamo da una matrice \mathbf{X} con media (per colonne) uguale a 0.
2. Costruiamo la matrice empirica di covarianza $\mathbf{S} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$.
3. Costruiamo una matrice \mathbf{V}_q data dai q autovettori di \mathbf{S} corrispondenti ai q autovalori più grandi.
4. La proiezione è data da $\mathbf{H} = \mathbf{X}\mathbf{V}_q$. Se $q = d$ possiamo invertire la trasformazione (in quanto \mathbf{V}_q è ortonormale) come $\mathbf{X} = \mathbf{H}\mathbf{V}_q^T$.

I valori di \mathbf{H} vengono detti i **principal components** dei dati (a volte con questo termine si indicano invece gli autovettori stessi).

Consideriamo la matrice di covarianza dei dati trasformati:

$$S' = \frac{1}{n} (\mathbf{XV})^\top (\mathbf{XV}) = \quad (11)$$

$$\mathbf{V}^\top \mathbf{S} \mathbf{V} = \Lambda \quad (12)$$

dove Λ è la matrice diagonale con gli autovalori sulla diagonale. La PCA è quindi una trasformazione che **diagonalizza** la matrice di correlazione dei dati. Se imponiamo anche che le varianze siano unitarie, otteniamo la cosiddetta **whitening transform**.

L'autovalore λ_i viene detto la **explained variance** su quell'asse. Il grafico ordinato degli autovalori nel contesto della PCA viene detto uno **scree plot**.

Per scegliere q si considera di solito il rapporto della varianza spiegata sulla varianza complessiva:

$$\frac{\sum_{i=1}^q \lambda_i}{\sum_{i=1}^d \lambda_i} \quad (13)$$

scegliendo q di modo da spiegare almeno il 90% o il 95% della varianza complessiva.

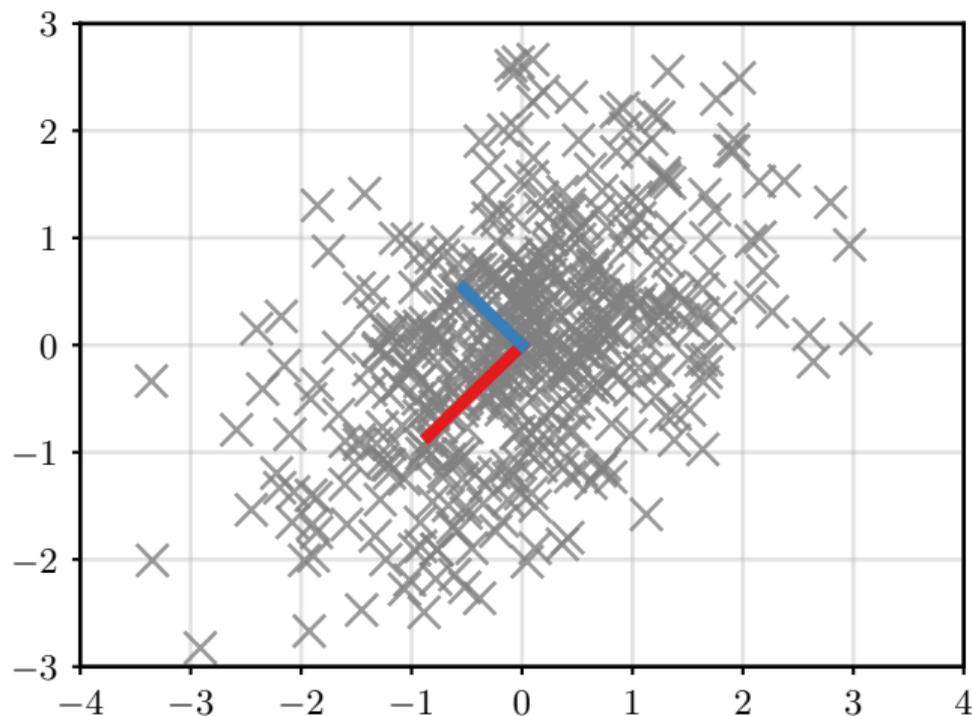


Figure 2: Gli assi della PCA in 2D. Stiamo graficando gli assi come $\mathbf{v}_i\sqrt{\lambda_i}$, a volte detti i **loadings** della PCA (gli assi scalati in funzione della varianza spiegata).

Principal component analysis

Autoencoding

Ritorniamo ora alla funzione generale:

$$f^*, g^* = \arg \min_{f, g} \frac{1}{n} \sum_{i=1}^n \| \mathbf{x}_i - g(f(\mathbf{x}_i)) \|^2 \quad (14)$$

Se f è g sono due reti neurali, abbiamo un cosiddetto **autoencoder**, dove f viene detto **encoder** e g **decoder**.

Si noti che in generale la soluzione banale è $f = g = \text{identità}$, a meno di non imporre ulteriori vincoli, ad esempio che la dimensionalità di output dell'encoder sia più piccola di quella di input.

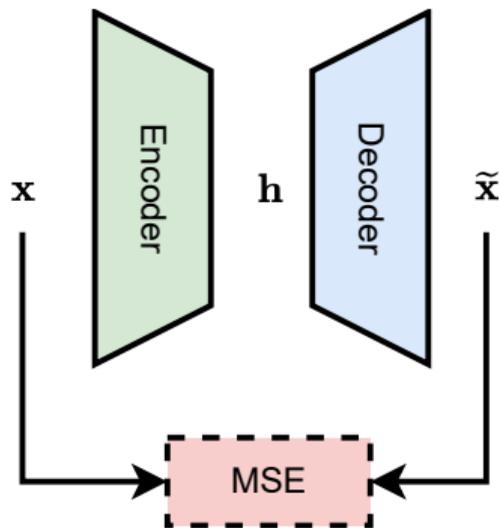


Figure 3: Un autoencoder implementato con due reti neurali dette **encoder** e **decoder**, allenate a ricostruire gli input originali a partire dalla rappresentazione intermedia.

Si può dimostrare che se f e g sono implementati come due reti neurali con un solo strato nascosto (anche se con funzioni di attivazione), la soluzione ottima converge sullo stesso sottospazio della PCA. Con due o più strati nascosti, la soluzione è invece diversa.

Gli autoencoder sono il punto di partenza per diverse architetture di interesse. Ad esempio, imponendo una trattazione probabilistica sulle variabili latenti si ottengono i **variational autoencoder**, un modello allo stato dell'arte per la generazione di immagini.

Clustering

k-means

Il problema del **clustering** è raggruppare le righe di X in k gruppi (**clusters**) tra loro omogenei (esistono anche varianti nei quali il numero di cluster non è scelto a priori dall'utente).

Vedremo di seguito l'algoritmo di clustering più usato, il cosiddetto **k -means**. Il k -means ha anche una semplice estensione probabilistica nota come **Gaussian mixture models** (GMMs). Entrambi possono essere ottimizzati con una tecnica nota come **expectation-maximization** (EM).

Non toccheremo il problema di **valutare** i risultati del clustering (metriche di clustering).

Nel k -means, rappresentiamo ogni cluster come un **centroide** (punto) nello spazio \mathbb{R}^d . Ogni elemento del nostro dataset è assegnato al centroide a lui più vicino.

Possiamo ottimizzare questo modello con una tecnica iterativa che ottimizza, in maniera alternata, il posizionamento dei centroidi nello spazio e l'assegnazione dei punti ai clusters.

Denotiamo con \mathbf{c}_j l' i -esimo centroide (possiamo inizializzarli in maniera casuale). Ogni punto del dataset è assegnato al centroide più vicino:

$$\pi_{ij} = \begin{cases} 1 & \text{se } j = \arg \min_{z=1, \dots, k} \{d(\mathbf{x}_i, \mathbf{c}_z)\} \\ 0 & \text{altrimenti} \end{cases} \quad (15)$$

dove d è la distanza Euclidea o un'altra distanza scelta dall'utente come iper-parametro. π_{ij} è 1 se il punto i è stato assegnato al cluster j , 0 altrimenti.

Supponiamo ora che i centroidi (ovvero le variabili π_{ij}) siano stati fissati. Quali sono i centroidi ottimali? Una scelta ragionevole è spostarli al centro dei punti che appartengono a quel cluster:

$$\mathbf{c}_j = \frac{1}{\sum_i \pi_{ij}} \sum_{i=1}^n \pi_{ij} \mathbf{x}_i \quad (16)$$

dove nella nostra notazione, $\sum_i \pi_{ij}$ è il numero di punti assegnati al cluster j -esimo.

Per formalizzare meglio il problema, notiamo che data una scelta dei centroidi e delle assegnazioni dei punti, possiamo scrivere una funzione costo come lo scarto medio tra ogni punto ed il centroide ad esso assegnato:

$$L(\mathbf{c}_j, \pi_{ij}) = \frac{1}{n} \sum_{i=1}^n \left\| \mathbf{x}_i - \sum_{j=1}^k \pi_{ij} \mathbf{c}_j \right\|^2 \quad (17)$$

Si può dimostrare che ogni iterazione dei due passi prima non può che diminuire o mantenere stabile $L(\mathbf{c}_j, \pi_{ij})$, e quindi l'algoritmo converge ad un minimo locale.

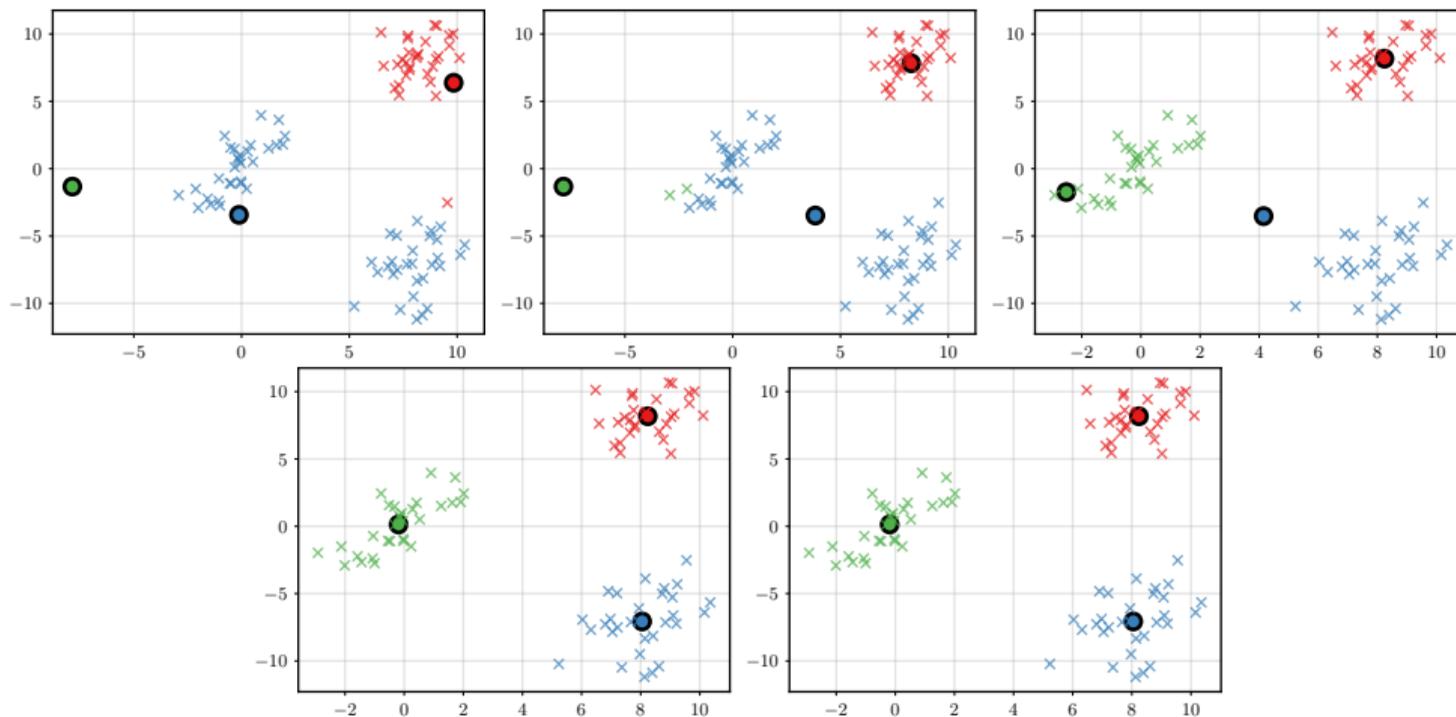


Figure 4: Esempio di iterazioni del k -means con $k = 3$.