SAPIENZA
UNIVERSITÀ DI ROMA

METODI DI INTELLIGENZA ARTIFICIALE E MACHINE LEARNING IN FISICA

S. Giagu - AA 2019/2020

Lezione 11: 1.4.2020

CENNI: CLASSIFICATORI UNSUPERVISED E CLUSTERING

- Fino ad ora abbiamo considerato classificatori basati su apprendimento supervisionato:
  - abbiamo a disposizione campioni di training $\{\mathbf{x}_i, y_i\}$ per tutte le classi
  - utilizziamo tali campioni per minimizzare la distanza dalla funzione discriminante rispetto agli eventi del campione ed ottenere i “pesi” che definiscono il modello stesso

- vogliamo analizzare in modo intuitivo e senza rigore formale come sia possibile costruire un algoritmo di classificazione avendo ancora un campione di eventi $\{\mathbf{x}_i\}$, ma in questo caso di eventi completamente indistinti per i quali non conosciamo a priori la classe di appartenenza

CENNI: CLASSIFICATORI UNSUPERVISED E CLUSTERING

- Tre approcci principali:

  - approccio parametrico:
    - in analogia al caso supervised si assume una distribuzione parametrica per ciascuna classe
    - si stimano i parametri della distribuzione (fittandoli su i dati stessi), incluse le frazioni relative
    - model dependent e con prestazioni peggiori rispetto al caso supervised → utilizzato solo in casi particolari

  - approccio non-parametrico (clustering):
    - si raggruppano i dati del campione in “cluster”, basandosi su opportuni criteri di similitudine, sperando che ogni cluster ci possa dire qualche cosa circa la categoria di appartenenza
    - oggi tratteremo questo approccio …

  - auto-encoders:
    - si addestra l’algoritmo ad apprendere direttamente informazioni sulla rappresentazione dei dati in input
    - utilizzati come algoritmi di dimensionality reduction, feature engineering, anomaly detection …
    - tipicamente implementati come reti neurali artificiali (li tratteremo successivamente insieme a queste ultime)

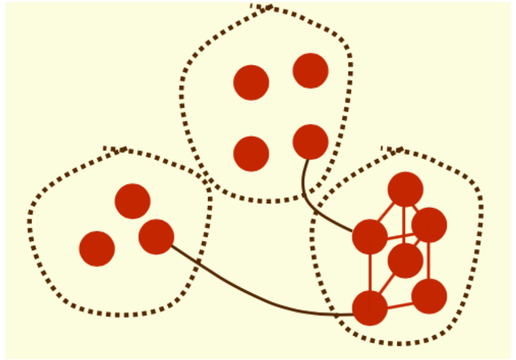
CENNI: CLASSIFICATORI UNSUPERVISED E CLUSTERING

- NOTA:
  - l'apprendimento non supervisionato è molto più complesso di quello supervised
  - non è nemmeno immediato decidere in modo "teorico" in base a principi primi se i risultati ottenuti siano sensati o meno
  - i risultati andranno sempre testati e valutati sul campo …

  - tuttavia vi sono molti casi in cui l'approccio unsupervised risulta l'unica via percorribile:
    - dataset estremamente grandi (in cui il costo legato al labelling in classi dei campioni è troppo grande): ad esempio: riconoscimento vocale o di immagini
    - dati in cui non si conosce il numero di classi di appartenenza a priori: esempio foto di osservazioni astronomiche
    - uso del clustering come criterio di descrizione dei dati stessi (cioè un modo per scoprire nuove categorie nei dati stessi)
      - può in principio essere usato come input per un classificatore supervisionato (apprendimento auto-supervisionato)
      - rappresenta una delle possibili direzioni oggi perseguite per superare le limitazioni del DL supervisionato …

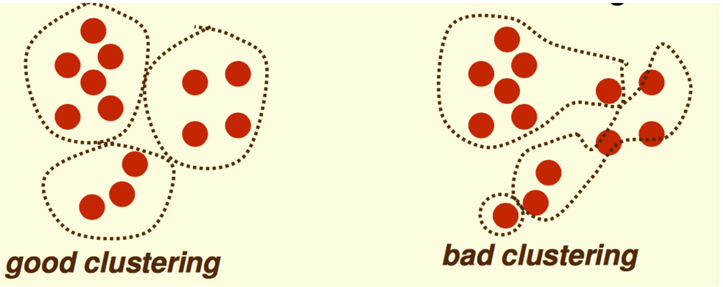
CLUSTERING

- come si cercano cluster in un dato campione?



- un buon algoritmo di clustering deve aggregare gli eventi in modo da garantire:
  - piccola distanza tra gli elementi appartenenti allo stesso cluster
  - grande distanza tra i vari cluster

- ingredienti:
  - misura di vicinanza (prossimità):
    - similarità o distanza: ex. $d(x_i,x_k)$ piccola se x_i e x_k sono simili



  - funzione obiettivo: per valutare la bontà del un clustering

  - algoritmo per calcolare il clustering (tipicamente basato sulla ottimizzazione della funzione obiettivo)

  - criterio per la scelta del numero di cluster:
    - si fissa il numero di cluster a priori
    - si sceglie il numero migliore di cluster in accordo alla una funzione obiettivo stessa

MISURA DI VICINANZA

- in generale problem dependent
  - è utile sfruttare il fatto che i cluster debbano essere invarianti sotto le trasformazioni “naturali” del problema

esempio: il riconoscimento di oggetti deve essere inv. sotto roto-traslazioni, il riconoscimento di caratteri no (9 VS 6)

- tipiche metriche utilizzate:
  - euclidea

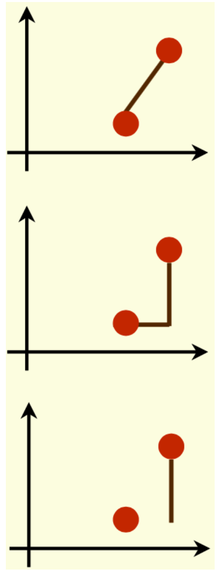
$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{d} \left(x_i^{(k)} - x_j^{(k)}\right)^2}$$

  - manhattan (city block)

$$d(x_i, x_j) = \sum_{k=1}^{d} \left| x_i^{(k)} - x_j^{(k)} \right|$$

  - chebyshev

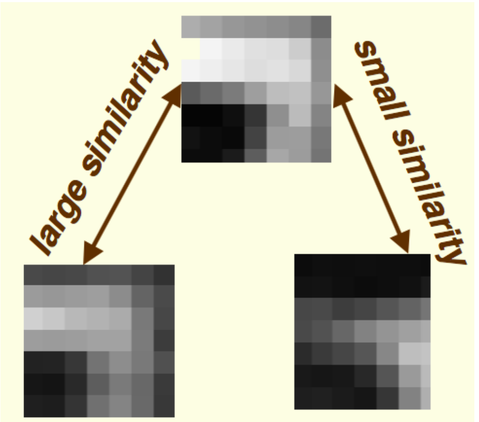
$$d(x_i, x_j) = \max_{1 \le k \le d} \left| x_i^{(k)} - x_j^{(k)} \right|$$



approssimano la distanza euclidea
ma sono più semplici da calcolare

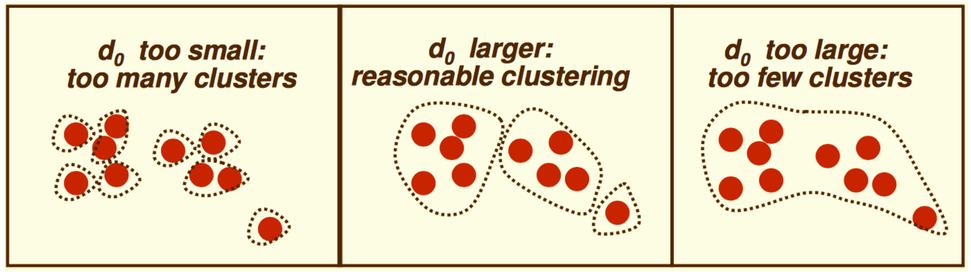
  - coefficiente di correlazione (molto usato nel image processing):

$$s(x_i, x_j) = \frac{\sum_{k=1}^{d} \left(x_i^{(k)} - \bar{x}_i\right)\left(x_i^{(k)} - \bar{x}_i\right)}{\left[\sum_{k=1}^{d} \left(x_i^{(k)} - \bar{x}_i\right)^2 \sum_{k=1}^{d} \left(x_j^{(k)} - \bar{x}_j\right)^2\right]^{1/2}}$$

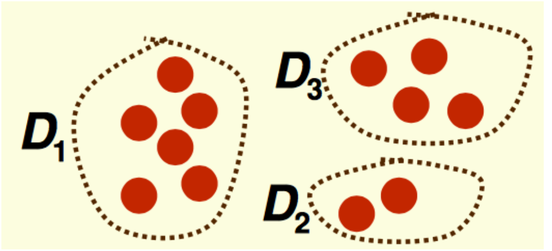


ALGORITMO DI CLUSTERING ELEMENTARE

- il più semplice di tutti:
  - loop su tutte le coppie di eventi del campione
  - se la distanza tra la coppia è < soglia d_0 vengono inserite nello stesso cluster
  - dipende fortemente dalla scelta del valore di soglia



- aggiungiamo una funzione obiettivo:
  - supponiamo di partizionare il campione di N eventi nei cluster D_1, .. , D_c (ognuno dei D_i è un cluster)
  - possiamo definire una funzione obiettivo che misuri la qualità di una data partizione D_1, .. , D_c
  - il problema del clustering è trasformato nel trovare la partizione che ottimizza la funzione obiettivo

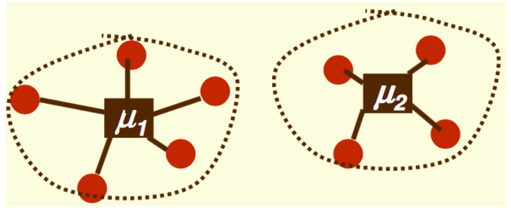


- funzione obiettivo SSE:
  - sia n_i il numero di eventi nel cluster D_i e sia μ_i la media degli eventi in D_i :

$$\mu_i = \frac{1}{n_i} \sum_{x \in D_i} x$$

  - la funzione obiettivo SSE è definita come la somma degli scarti quadratici:

$$J_{SSE} = \sum_{i=1}^{c} \sum_{x \in D_i} \| x - \mu_i \|^2$$

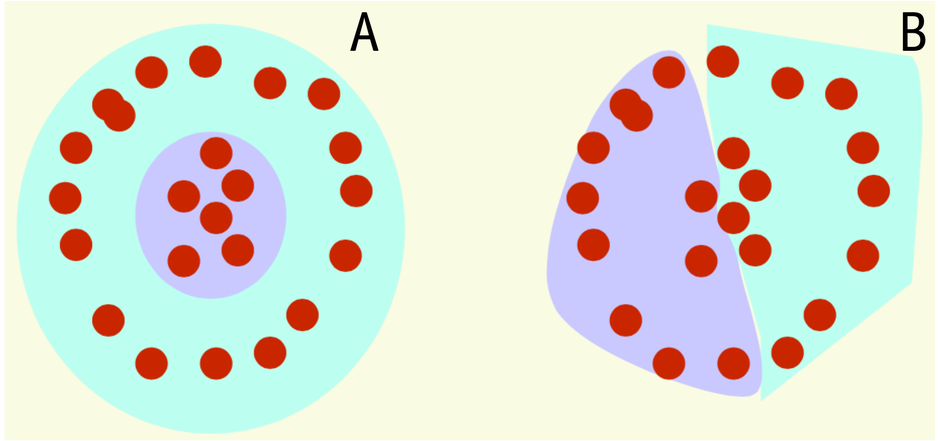


il criterio consiste nel minimizzare J_{SSE}
che corrisponde a minimizzare la varianza all'interno di ciascun cluster

SSE

- il criterio SSE funziona molto bene quando i dati formano strutture compatte all'interno di ogni classe che risultano relativamente ben separate

- in pratica il criterio SSE favorisce cluster con dimensione simile e risulta meno appropriato quando il raggruppamento naturale dei dati ha strutture con dimensioni molto disomogenee



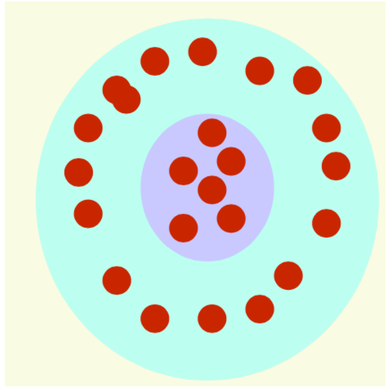
l'SSE sceglie il clustering sbagliato … la ragione è che uno dei cluster "naturali" non è compatto (l'anello esterno)

- Per ovviare al problema si possono utilizzare criteri differenti (model dependence) …

- Maximum distance:

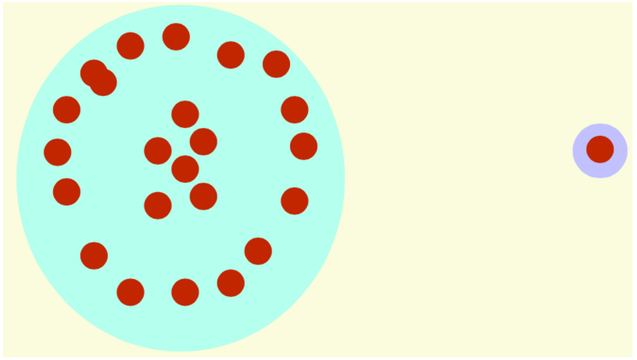
$$J_{max} = \sum_{i=1}^{c} n_i \left[\max_{y \in D_i, x \in D_i} \| x - y \|^2 \right]$$

  - risolve il problema precedente:



minimo J_{max}

… ma fallisce quando ci sono outlier



minimo J_{max}

- NOTA:
  - JSSE può essere riscritto come:

$$J_{SSE} = \sum_{i=1}^{c} \sum_{x \in D_i} \| x - \mu_i \|^2 = \frac{1}{2} \sum_{i=1}^{c} n_i \left[\frac{1}{n_i^2} \sum_{y \in D_i} \sum_{x \in D_i} \| x - y \|^2 \right]$$

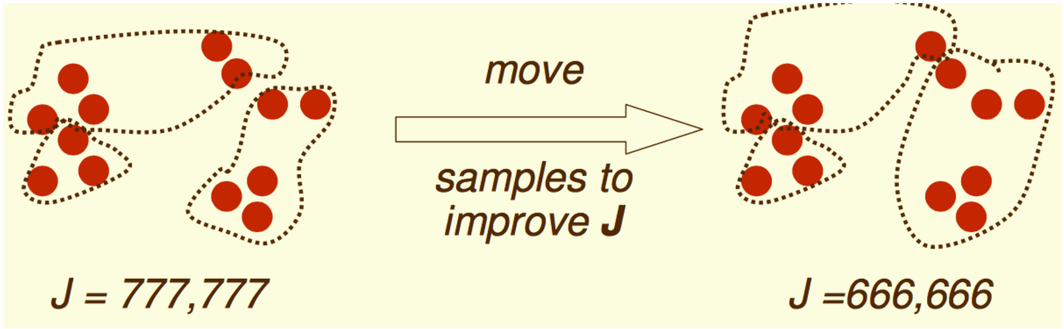
  - si possono ottenere diversi funzioni obiettivo semplicemente cambiando $||x-y||^2$ con una qualsiasi altra misura della distanza tra i punti in D_i

OTTIMIZZAZIONE ITERATIVA

- Definita la distanza e la funzione obiettivo ci rimane solo da definire un algoritmo per trovare il clustering ottimale

  - la ricerca esaustiva non è consigliabile visto il numero di partizioni possibili che cresce esponenzialmente con N

  - si utilizzano algoritmi iterativi:
    - si trova una partizione iniziale ragionevole
    - si ripete muovendo eventi da un cluster all'altro in modo da migliorare il valore della funzione obiettivo

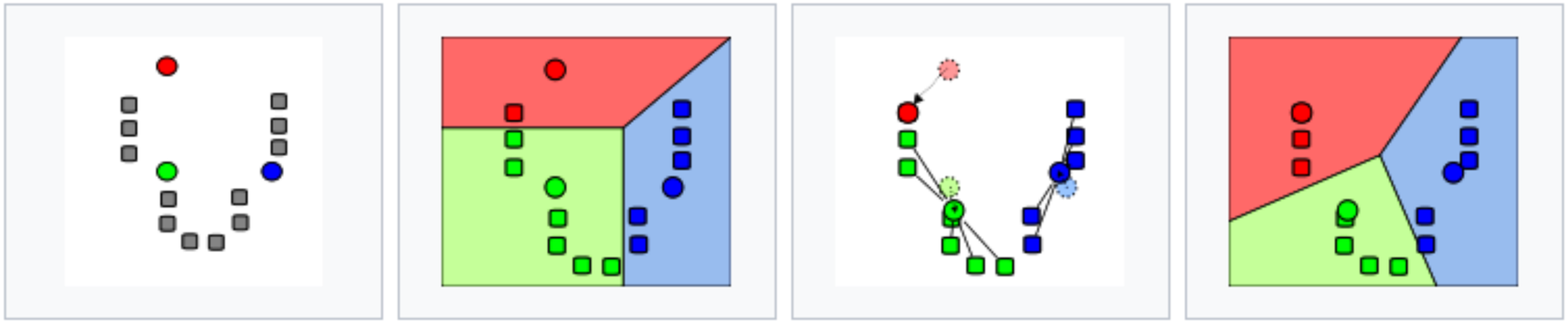


  - in pratica sono algoritmi simili alla discesa lungo il gradiente
  - ci si muove nella direzione in cui la funzione obiettivo diminuisce

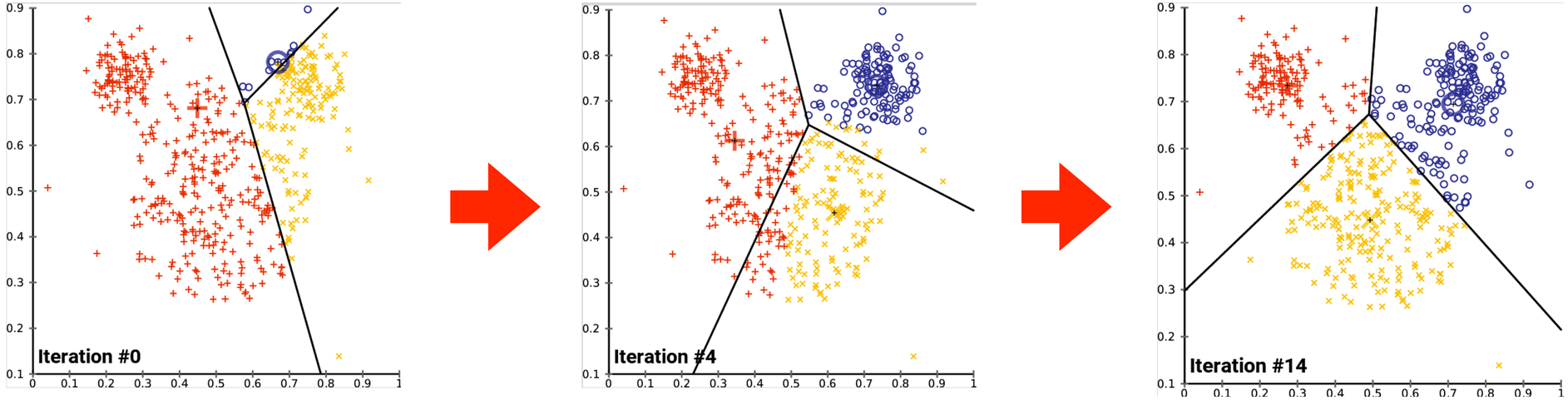
NOTA: poiché la funzione obiettivo tipicamente non è derivabile ⇒ la soluzione dipende dalla condizioni iniziali e l'algoritmo non sempre converge a un minimo globale

  - si utilizzano tecniche ad hoc, dipendenti dal tipo di problema i.e. non esiste un algoritmo di clustering general-purpose!

K-MEANS CLUSTERING



1. si decide k (ex. 3)
2. vengono scelte random 3-medie iniziali tra i dati
3. si costruiscono 3-cluster associando ad ogni media gli eventi vicini secondo la metrica SSE e si costruiscono i domini di decisione corrispondenti
4. i centroidi di ogni cluster vengono calcolati e le 3-medie divengono tali centroidi
5. si ripete iterativamente fino ad arrivare a convergenza o ad un numero di step fissato



K-MEANS CLUSTERING

- in dettaglio: supponiamo si avere un training set T{$\mathbf{x}_i$} i=1,…,N con $\mathbf{x}_i$ feature vector d-dimensionale dell'evento i-esimo del set

definiamo cluster means: $\{\underline{\mu}_k\}\ k = 1,\ldots,K$ le coordinate dei centri dei di K cluster ($\underline{\mu}_k$ vettore d-dimensionale)

$\boldsymbol{\mu}_k$ possono essere interpretate come rappresentative di ogni cluster a cui vengono associati i dati di T

L'algoritmo k-means viene formulato come problema di minimizzazione: in cui fissato K si deve trovare il vettore {$\underline{\mu}$} e le assegnazioni dei dati in T in modo da minimizzare la loss function:

$$L(\mathbf{x}, \underline{\mu}) = \sum_{k=1}^{K} \sum_{i=1}^{N} r_{ik} (\mathbf{x}_i - \underline{\mu}_k)^2$$

con r_{ik} "assignement":

$$r_{ik} = \begin{cases} 1 & \text{se } \mathbf{x}_i \text{ assegnata al cluster } k \\ 0 & \text{altrimenti} \end{cases}$$

$$\sum_k r_{ik} = 1 \ \forall i \qquad \sum_i r_{ik} = N_k$$

la minimizzazione di L corrisponde a trovare il migliore {$\underline{\mu}$} e {r_{ik}} tale che la varianza all'interno di ciascun cluster è minimizzata (interpretazione fisica: L è la somma dei momenti di inerzia di ciascun cluster e {$\underline{\mu}$} corrisponde al centro di massa di ciascun cluster)

K-MEANS CLUSTERING

- la procedura iterativa alterna du step sequenziali:

update rule

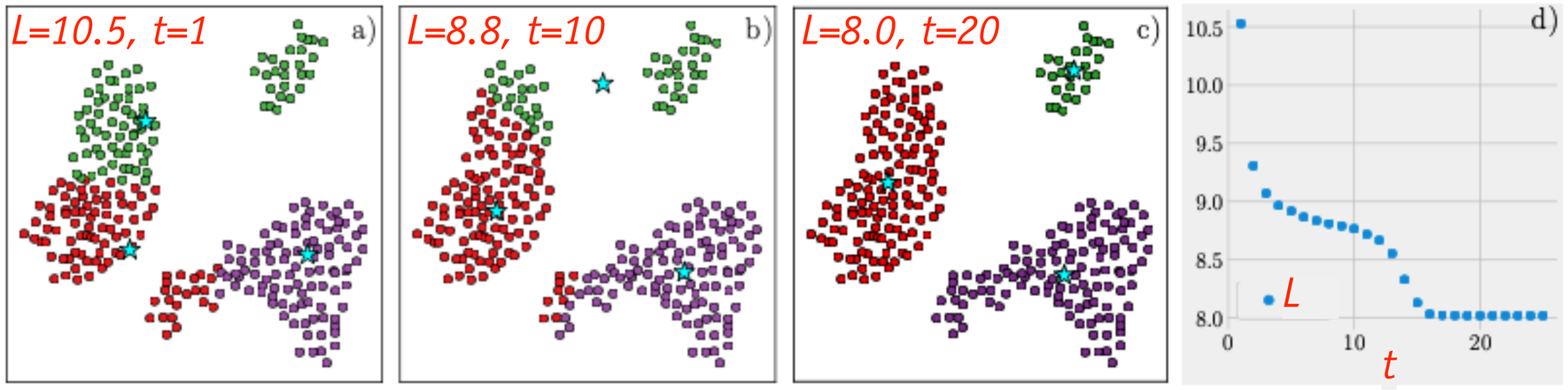
EXPECTATION: dato un set di assignement $\{r_{ik}\}$ si minimizza L rispetto a μ_k

$$\nabla_{\mu_k} L(\mathbf{x}, \mu) = 0 \Rightarrow \mu_k = \frac{1}{N_k} \sum_{i=1}^{N} r_{ik} \mathbf{x}_i$$

MAXIMIZATION: dato un set di cluster $\{\mu\}$ si trova l'assignment $\{r_{ik}\}$ che minimizza L → i.e. si associa ogni $\mathbf{x}_i$ a cluster means più vicino

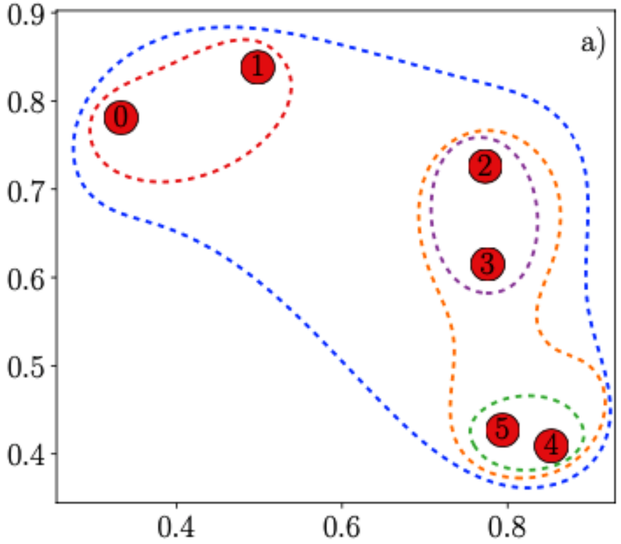
$$r_{ik} = \begin{cases} 1 & \text{se } k = \underset{k'}{argmin}[(\mathbf{x}_i - \mu_{k'})^2] \\ 0 & \text{altrimenti} \end{cases}$$

L'algoritmo alterna tra questi due step per il numero di iterazioni prescelto o quando $L<L_{soglia}$ …

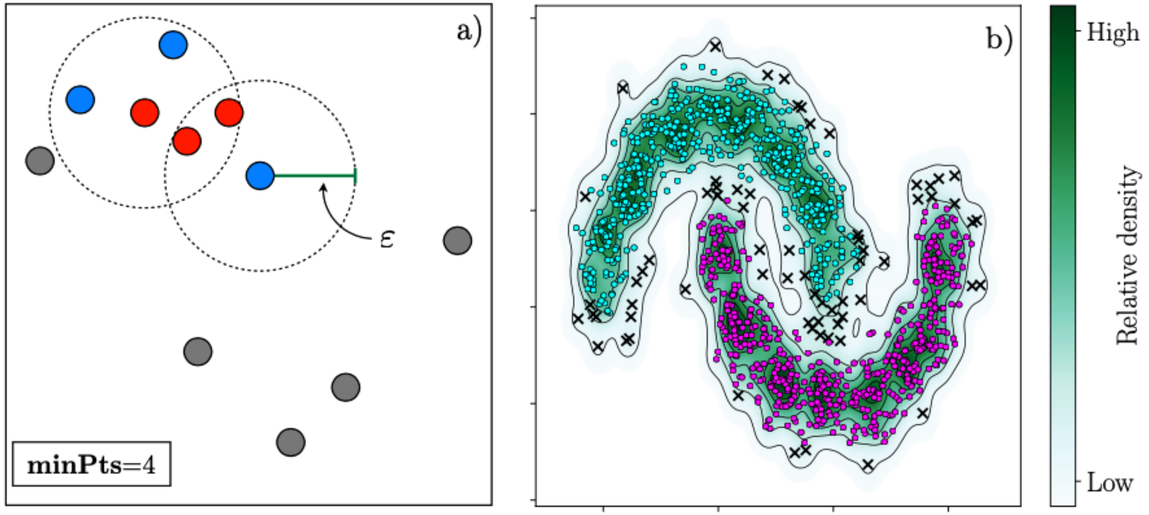


ALTRI ALGORITMI

- metodi agglormerativi (clustering gerarchico): si parte da un cluster per ogni evento e poi si agglomerano basandosi su una misura di distanza



- metodi density-based (DBSCAN): is basano sull'assunzione intuitiva che i cluster sono definiti da regioni dello spazio con più alta densità di punti



- Gaussian Mixture Models → …

VARIABILI LATENTI

- vogliamo guardare al clustering da un punto di vista più astratto e formale, meno intuitivo a prima vista ma che permette di capire più facilmente le connessioni con le idee concettuali del apprendimento non-supervisionato

- iniziamo con introdurre il concetto di variabile latente (nascosta) e di spazio latente, uno dei concetti cardine dell'unsupervised learning
  - sono variabili non direttamente osservabili ma che vengono dedotte da altre variabili che sono osservabili (attraverso un modello matematico)
  - anche se non sono osservabili influenzano la struttura visibile dei dati
    - esempio: nel clustering la cluster identity (a quale cluster un certo dato appartiene) è una variabile latente
    - anche se non conosciamo esplicitamente le label di ogni dato sappiamo che eventi dello stesso cluster tendono a stare vicini tra loro
  - le variabili latenti forniscono una metodo elegante per codificare le complicate correlazioni tra le feature osservabili

VARIABILI LATENTI

- la ragione sottostante per cui le variabili latenti sono in grado di rappresentare tali correlazione è legata al fatto che marginalizzare su un sottoinsieme di variabili (cioè integrare out un set di gradi di libertà nel linguaggio dei fisici) induce interazioni complesse tra le restanti variabili (le osservabili)
- esempi in fisica:
  - elettroni liberi di muoversi in un reticolo cristallino → integrare out i fononi da luogo ad interazioni elettrone-elettrone (correlazioni nei superconduttori o di tipo magnetico)
  - le teorie effettive di campo (nel paradigma del gruppo di rinormalizzazione) sono prodotte dalla operazione di integrazione out dei gradi di libertà di alta energia

VARIABILI LATENTI E CLUSTERING

- nel contesto del ML possiamo pensare al clustering come: un algoritmo che apprende il valore più probabile per una variabile latente (cluster identity) associata ad ogni campione del dataset

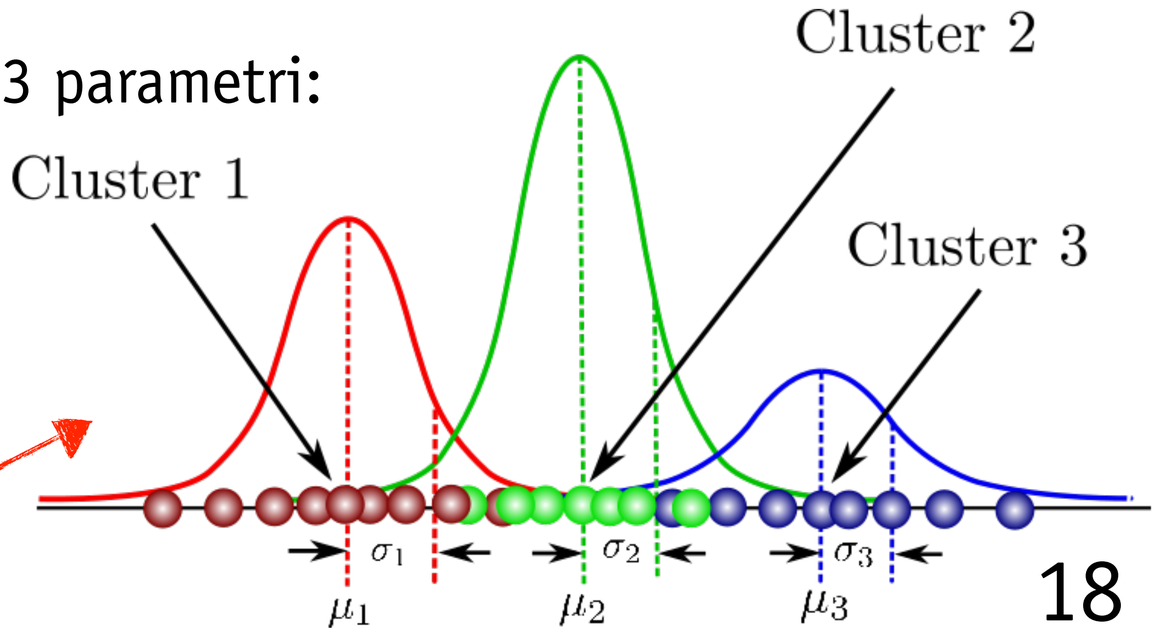
- calcolare le variabili latenti richiede:

  1. un modello matematico (modello generativo) della struttura del dataset
     - si assume una distribuzione di probabilità sottostante dalla quale sono stati generati i dati del dataset
       - es. clustering: i dati sono associati ai cluster, ogni cluster è caratterizzato da una distribuzione di probabilità (per esempio gaussiana)
  2. una procedura per trovare il valore di tali variabili latenti
     - tipicamente si scelgono i valori delle variabili latenti che minimizzano una data loss function

GAUSSIAN MIXTURES MODELS (GMM)

- i modelli basati su gaussian mixture (GMM) sono un esempio di modello generativo spesso usato nel contesto del clustering che offrono la possibilità di avere informazioni sulla probabilità di appartenenza ad un cluster (soft clustering)
- Definizione:
  - una mistura gussiana è una funzione composta da K gaussiane ognuna identificata da un indice $k \in \{1,\ldots,K\}$ in cui K è il numero di cluster del dataset
  - ogni gaussiana $N(\mathbf{x}; \boldsymbol{\theta}=\{\mu, \Sigma, \pi\})$ è definita da 3 parametri:
    - μ, Σ
    - π: mixture weight

esempio 1D: K=3



- Proprietà:
  - ogni gaussiana spiega i dati contenuti nel cluster associato
  - i mixture weight sono probabilità (la probabilità che un dato sia estratto dalla k-esima gaussiana)

$$\Rightarrow \sum_{k=1}^{K} \pi_k = 1$$

  - in generale ogni gaussiana ha forma (**x** feature vector multidimensionale di dimensione D):

$$N(\mathbf{x} \mid \{\mu_k, \Sigma_k\}) = \frac{1}{(2\pi)^{D/2} \mid \Sigma_k \mid^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_k)\Sigma_k^{-1}(\mathbf{x}-\mu_k)^t\right)}$$

x, μ_k: vettori a D-dimensioni

Σ_k: matrice (covarianza) DxD

- la probabilità di generare il punto **x** nella GMM è data da:

$$p(\mathbf{x} \mid \{\mu_k, \Sigma_k, \pi_k\}) = \sum_{k=1}^{K} \pi_k N(\mathbf{x} \mid \{\mu_k, \Sigma_k\})$$

- dato un dataset T={$\mathbf{x}_1$,…, $\mathbf{x}_N$} possiamo scrivere la likelihood del dataset come probabilità congiunta di tutti gli eventi del dataset stesso:

$$p(T \mid \{\mu_k, \Sigma_k, \pi_k\}) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \{\mu_k, \Sigma_k, \pi_k\}) = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\})$$

- con log-likelihood:

$$\ln p(T) = \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\})$$

Problema: servono i parametri {μ_k,Σ_k,π_k} ma derivare ln$p(T)$ e uguagliare a zero risulta complicato da lnΣ … si deve procedere in modo alternativo …

- introduciamo per ogni dato **x** la variabile latente z che puo assumere solo due valori:
  - 1 se **x** proviene dalla gaussiana k
  - 0 altrimenti
- z non è osservabile (il campione non ha label) ma ci permette di esprimere:

$$p(z_k = 1 \mid \mathbf{x})$$

probabilità che dato **x** questo venga dalla gaussiana k

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

probabilità totale di osservare un evento che proviene dalla gaussiana k (definizione di π)

- indichiamo con **z**={z_1,…,z_K} l'insieme di tutte le variabili latenti delle K-gaussiane

esempio: K=3
z = (1,0,0) o (0,1,0) o (0,0,1)

NOTA:
variabili di questo tipo sono dette "one-hot" vector

- poiché ogni z_k occorre indipendentemente dalle altre e può assumere solo valore 1 quando k è uguale al cluster a cui il punto appartiene:

$$p(\mathbf{z}) = p(z_1 = 1)^{z_1} p(z_2 = 1)^{z_2} \dots p(z_K = 1)^{z_K} = \prod_{k=1}^{K} \pi_k^{z_k}$$

probabilità di osservare un dato z

- seguendo la stessa logica possiamo scrivere:

$$p(\mathbf{x} \mid \mathbf{z}) = \prod_{k=1}^{K} N(\mathbf{x} \mid \{\mu_k, \Sigma_k\})^{z_k}$$

probabilità di osservare **x** condizionato ad avere un dato vettore latente **z**

- il goal è quello di trovare la probabilità di avere **z** data l'osservazione **x**, possiamo farlo sfruttando le regole della probabilità condizionata e il teorema di Bayes:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})$$

probabilità congiunta di osservare **x** e **z**

- applicando Bayes:

$$p(\mathbf{x}\,|\,\mathbf{z}) = \prod_{k=1}^{K} N(\mathbf{x}\,|\,\{\mu_k, \Sigma_k\})^{z_k} \Rightarrow$$

$$\Rightarrow p(\mathbf{x}\,|\,z_k = 1) = N(\mathbf{x}\,|\,\{\mu_k, \Sigma_k\})$$

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

$$p(z_k = 1\,|\,\mathbf{x}) = \frac{p(\mathbf{x}\,|\,z_k = 1)p(z_k = 1)}{\sum_{j=1}^{K} p(\mathbf{x}\,|\,z_j = 1)p(z_j = 1)}$$

- per cui:

$$p(z_k = 1\,|\,\mathbf{x}) = \frac{\pi_k N(\mathbf{x}\,|\,\{\mu_k, \Sigma_k\})}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}\,|\,\{\mu_j, \Sigma_j\})} = \gamma(z_k)$$

responsibility della mistura **k** nello spiegare **x**

ALGORTIMO EXPECTATION-MAXIMIZATION (EM)

- l'algoritmo EM fornisce un metodo potente per trovare iterativamente i valori di minimo locale in modelli con variabili latenti
- dati i parametri del modello: $\theta=\{\mu, \Sigma, \pi\}$ l'algoritmo procede con i seguenti passi:

1. sceglie un valore iniziale $\hat{\theta}$ per θ
2. EXPECTATION step:
   valuta:

$$Q(\hat{\theta}, \theta) = E[\ln p(T, \mathbf{Z} \mid \hat{\theta})] = \sum_{\mathbf{Z}} p(\mathbf{Z} \mid T, \theta) \ln p(T, \mathbf{Z} \mid \hat{\theta})$$

valore atteso

con:
dataset $T=\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
$\mathbf{Z}$ set delle variabili z associate al dataset (sono N z-variabili ognuna di dimensione K)

- osserviamo che abbiamo già calcolato $p(\mathbf{Z}|T,\theta)$, infatti non sono altro che le responsability $\gamma(z_k)$:

$$p(z_k = 1 \mid \mathbf{x}_i) = \frac{\pi_k N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\})}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}_i \mid \{\mu_j, \Sigma_j\})} = \gamma(z_{k_i})$$

variabile latente z_k dell'i-esimo evento di T

- per cui:

$$Q(\hat{\theta}, \theta) = \sum_{\mathbf{Z}} \gamma(z_{n_k}) \ln p(T, \mathbf{Z} \mid \hat{\theta})$$

- rimane da calcolare $p(T,\mathbf{Z}|\hat{\theta})$, ma questo è facile visto che non è altro che la likelihood completa del modello includendo sia **Z** che T data dalla prob. congiunta:

$$p(T, \mathbf{Z} \mid \hat{\theta}) = \prod_{i=1}^{N} \prod_{k=1}^{K} \pi^{z_{k_n}} N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\})^{z_{k_n}}$$

- passando al logaritmo:

$$\ln p(T, \mathbf{Z} \mid \hat{\theta}) = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{k_n} \left[\ln \pi_k + \ln N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\}) \right]$$

che è un'espressione molto più semplice da trattare rispetto al ln*p(T)* trovata all'inizio …

- possiamo semplificare la notazione eliminando z_{kn} osservando che sarà 1 un'unica volta ogni volta che la sommatoria viene valutata. Otteniamo quindi per Q:

$$Q(\hat{\theta}, \theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{n_k}) \left[\ln \pi_k + \ln N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\}) \right]$$

3. MAXIMIZATION step:

ottiene una nuova stima dei parametri $\hat{\theta}$:

$$\hat{\theta} = \underset{\theta}{argmax}[Q(\hat{\theta}, \theta)]$$

- con il vincolo: $\sum_{k=1}^{K} \pi_k = 1$

- questo viene implementato usando un moltiplicatore di lagrange opportuno:

$$Q(\hat{\theta}, \theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{n_k}) \left[\ln \pi_k + \ln N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\}) \right] - \lambda \left[\sum_{k=1}^{K} \pi_k - 1 \right]$$

così i parametri sono facili da ottenere usando MLE:

- derivando Q rispetto a π ed eguagliando a zero:

$$\frac{dQ(\hat{\theta},\theta)}{d\pi_k} = \sum_{i=1}^{N} \frac{\gamma(z_{k_i})}{\pi_k} - \lambda = 0 \Rightarrow$$

$$\sum_{i=1}^{N} \gamma(z_{k_i}) = \pi_k \lambda \Rightarrow \sum_{k=1}^{K} \sum_{i=1}^{N} \gamma(z_{k_i}) = \sum_{k=1}^{K} \pi_k \lambda = \lambda$$

applicando una somma su k ad entrambi i membri ed osservando che:

$$\sum_{k=1}^{K} \pi_k = 1$$

$$\sum_{k=1}^{K} \sum_{i=1}^{N} \gamma(z_{k_i}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{k_i}) = \sum_{i=1}^{N} 1 = N \Rightarrow \lambda = N$$

$$\gamma(z_{k_i}) = \frac{\pi_k N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\})}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}_i \mid \{\mu_j, \Sigma_j\})}$$

- per cui:

$$\hat{\pi}_k = \frac{\sum_{i=1}^{N} \gamma(z_{k_i})}{N}$$

stima di π_k alla data iterazione

- procedendo in modo identico derivando Q rispetto a μ e a Σ si ottiene:

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \gamma(z_{k_i})\mathbf{x}_i}{\sum_{i=1}^{N} \gamma(z_{k_i})}$$

NOTA: $\hat{\mu}$, $\hat{\Sigma}$, $\hat{\pi}$ non sono altro che le stime usuali di media e varianza in cui ogni punto è pesato in accordo alla migliore stima disponibile della probabilità che appartenga al cluster k → questo è essenzialmente algoritmo k-means visto precedentemente …

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma(z_{k_i})(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^t}{\sum_{i=1}^{N} \gamma(z_{k_i})}$$

4. si usa la nuova stima di $\hat{\theta}=\{\hat{\mu}, \hat{\Sigma}, \hat{\pi}\}$ per ricalcolare γ nella successiva iterazione E-M fino a quando non si osserva una qualche convergenza nei valori della likelihood totale dell'evento che vogliamo massimizzare:

$$L = \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \hat{\pi}_k N(\mathbf{x}_i \mid \{\hat{\mu}_k, \hat{\Sigma}_k\})$$

ESEMPIO: IMPLEMENTAZIONE GMM IN PYTHON

https://www.dropbox.com/s/fxsb3vn4x905rqw/GMM_esempio.ipynb?dl=0

```
import imageio
import matplotlib.animation as ani
import matplotlib.cm as cmx
import matplotlib.colors as colors
import matplotlib.pyplot as plt
import numpy as np

from matplotlib.patches import Ellipse
from PIL import Image
from sklearn import datasets
from sklearn.cluster import KMeans
```

```
iris = datasets.load_iris()
X = iris.data
X[:20]
```

$$N(\mathbf{x}\,|\,\{\mu_k,\Sigma_k\}) = \frac{1}{(2\pi)^{D/2}\,|\,\Sigma_k\,|^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_k)\Sigma_k^{-1}(\mathbf{x}-\mu_k)^t\right)}$$

```
def gaussian(X, mu, cov):
    n = X.shape[1]
    diff = (X - mu).T
    return np.diagonal(1 / ((2 * np.pi) ** (n / 2) * np.linalg.det(cov) ** 0.5) * np.exp(-0.5 * np.dot(np.dot(diff.T
```

```
x0 = np.array([[0.05, 1.413, 0.212], [0.85, -0.3, 1.11], [11.1, 0.4, 1.5], [0.27, 0.12, 1.44], [88, 12.33, 1.44]])
mu = np.mean(x0, axis=0)
cov = np.dot((x0 - mu).T, x0 - mu) / (x0.shape[0] - 1)

y = gaussian(x0, mu=mu, cov=cov)
y
```

```
array([[0.00159853],
       [0.00481869],
       [0.00276259],
       [0.0014309 ],
       [0.00143998]])
```

step 1

Inizializzione dei parametri π_k, μ_k, e Σ_k. Come guess iniziale usaimo il risultato di un k-means clustering

```python
def initialize_clusters(X, n_clusters):
    clusters = []
    idx = np.arange(X.shape[0])

    # We use the KMeans centroids to initialise the GMM

    kmeans = KMeans().fit(X)
    mu_k = kmeans.cluster_centers_

    for i in range(n_clusters):
        clusters.append({
            'pi_k': 1.0 / n_clusters,
            'mu_k': mu_k[i],
            'cov_k': np.identity(X.shape[1], dtype=np.float64)
        })

    return clusters
```

step 2 (EXPECTATION)

$$\gamma(z_k) = \frac{\pi_k N(\mathbf{x} \mid \{\mu_k, \Sigma_k\})}{\sum_{j=1}^{K} \pi_j N(\mathbf{x} \mid \{\mu_j, \Sigma_j\})}$$

```
def expectation_step(X, clusters):
    totals = np.zeros((X.shape[0], 1), dtype=np.float64)

    for cluster in clusters:
        pi_k = cluster['pi_k']
        mu_k = cluster['mu_k']
        cov_k = cluster['cov_k']

        gamma_nk = (pi_k * gaussian(X, mu_k, cov_k)).astype(np.float64)

        for i in range(X.shape[0]):
            totals[i] += gamma_nk[i]

        cluster['gamma_nk'] = gamma_nk
        cluster['totals'] = totals


    for cluster in clusters:
        cluster['gamma_nk'] /= cluster['totals']
```

step 3 (MAXIMIZATION)

```python
def maximization_step(X, clusters):
    N = float(X.shape[0])

    for cluster in clusters:
        gamma_nk = cluster['gamma_nk']
        cov_k = np.zeros((X.shape[1], X.shape[1]))

        N_k = np.sum(gamma_nk, axis=0)

        pi_k = N_k / N
        mu_k = np.sum(gamma_nk * X, axis=0) / N_k

        for j in range(X.shape[0]):
            diff = (X[j] - mu_k).reshape(-1, 1)
            cov_k += gamma_nk[j] * np.dot(diff, diff.T)

        cov_k /= N_k

        cluster['pi_k'] = pi_k
        cluster['mu_k'] = mu_k
        cluster['cov_k'] = cov_k
```

$$\hat{\pi}_k = \frac{\sum_{i=1}^{N} \gamma(z_{k_i})}{N}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \gamma(z_{k_i}) \mathbf{x}_i}{\sum_{i=1}^{N} \gamma(z_{k_i})}$$

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma(z_{k_i})(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^t}{\sum_{i=1}^{N} \gamma(z_{k_i})}$$

Calcolo Likelihood

$$\ln p(T) = \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k N(\mathbf{x}_i \mid \{\mu_k, \Sigma_k\})$$

```python
def get_likelihood(X, clusters):
    likelihood = []
    sample_likelihoods = np.log(np.array([cluster['totals'] for cluster in clusters]))
    return np.sum(sample_likelihoods), sample_likelihoods
```

NOTA: la seconda sommatoria è stata già calcolata nel expectation step (variabile totals)

Algortimo completo

```python
def train_gmm(X, n_clusters, n_epochs):
    clusters = initialize_clusters(X, n_clusters)
    likelihoods = np.zeros((n_epochs, ))
    scores = np.zeros((X.shape[0], n_clusters))
    history = []

    for i in range(n_epochs):
        clusters_snapshot = []

        # This is just for our later use in the graphs
        for cluster in clusters:
            clusters_snapshot.append({
                'mu_k': cluster['mu_k'].copy(),
                'cov_k': cluster['cov_k'].copy()
            })

        history.append(clusters_snapshot)

        expectation_step(X, clusters)
        maximization_step(X, clusters)

        likelihood, sample_likelihoods = get_likelihood(X, clusters)
        likelihoods[i] = likelihood

        print('Epoch: ', i + 1, 'Likelihood: ', likelihood)

    for i, cluster in enumerate(clusters):
        scores[:, i] = np.log(cluster['gamma_nk']).reshape(-1)

    return clusters, likelihoods, scores, sample_likelihoods, history
```

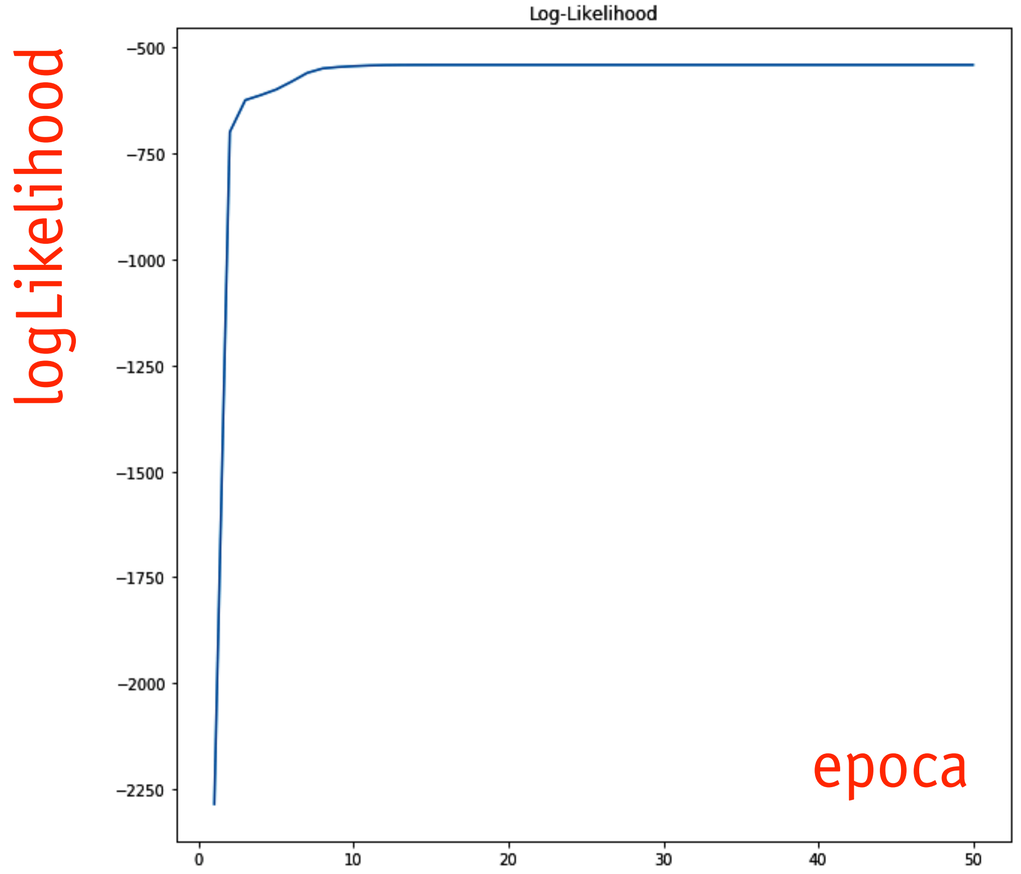
Run:

```
n_clusters = 3
n_epochs = 50

clusters, likelihoods, scores, sample_likelihoods, history = train_gmm(X, n_clusters, n_epochs)
```

```
Epoch:  1 Likelihood:  -2285.227823700732
Epoch:  2 Likelihood:  -698.8321661339035
Epoch:  3 Likelihood:  -623.6936980854894
Epoch:  4 Likelihood:  -611.8861190852117
Epoch:  5 Likelihood:  -598.6000407722117
Epoch:  6 Likelihood:  -579.8469927423686
Epoch:  7 Likelihood:  -559.4577300114757
Epoch:  8 Likelihood:  -548.9220532433164
Epoch:  9 Likelihood:  -545.6580893688225
Epoch:  10 Likelihood:  -543.5233690220482
Epoch:  11 Likelihood:  -541.987043457231
Epoch:  12 Likelihood:  -541.1233211005083
Epoch:  13 Likelihood:  -540.7510928912716
Epoch:  14 Likelihood:  -540.6188538766411
Epoch:  15 Likelihood:  -540.5761230989426
Epoch:  16 Likelihood:  -540.5626617732636
```

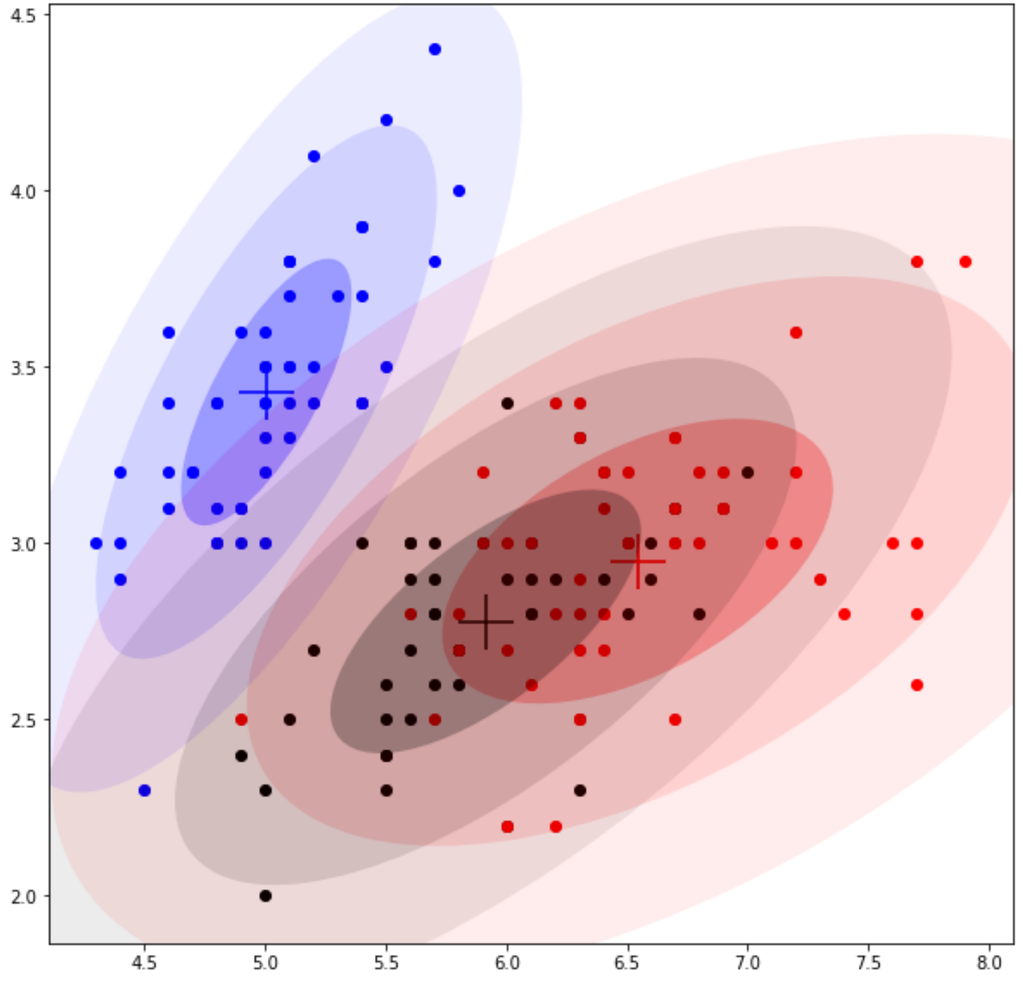
```
Epoch:  48 Likelihood:  -540.5564313939103
Epoch:  49 Likelihood:  -540.5564313939103
Epoch:  50 Likelihood:  -540.5564313939103
```

```
plt.figure(figsize=(10, 10))
plt.title('Log-Likelihood')
plt.plot(np.arange(1, n_epochs + 1), likelihoods)
plt.show()
```



Plot:

```
def create_cluster_animation(X, history, scores):
    fig, ax = plt.subplots(1, 1, figsize=(10, 10))
    colorset = ['blue', 'red', 'black']
    images = []

    for j, clusters in enumerate(history):

        idx = 0

        if j % 3 != 0:
            continue

        plt.cla()

        for cluster in clusters:
            mu = cluster['mu_k']
            cov = cluster['cov_k']

            eigenvalues, eigenvectors = np.linalg.eigh(cov)
            order = eigenvalues.argsort()[::-1]
            eigenvalues, eigenvectors = eigenvalues[order], eigenvectors[:, order]
            vx, vy = eigenvectors[:,0][0], eigenvectors[:,0][1]
            theta = np.arctan2(vy, vx)

            color = colors.to_rgba(colorset[idx])

            for cov_factor in range(1, 4):
                ell = Ellipse(xy=mu, width=np.sqrt(eigenvalues[0]) * cov_factor * 2, height=np.sqr
t(eigenvalues[1]) * cov_factor * 2, angle=np.degrees(theta), linewidth=2)
                ell.set_facecolor((color[0], color[1], color[2], 1.0 / (cov_factor * 4.5)))
                ax.add_artist(ell)

            ax.scatter(cluster['mu_k'][0], cluster['mu_k'][1], c=colorset[idx], s=1000, marker='+'
)
            idx += 1

        for i in range(X.shape[0]):
            ax.scatter(X[i, 0], X[i, 1], c=colorset[np.argmax(scores[i])], marker='o')

        fig.canvas.draw()

        image = np.frombuffer(fig.canvas.tostring_rgb(), dtype='uint8')
        image = image.reshape(fig.canvas.get_width_height()[::-1] + (3,))

        images.append(image)

    kwargs_write = {'fps':1.0, 'quantizer':'nq'}
    imageio.mimsave('./gmm.gif', images, fps=1)
    plt.show(Image.open('gmm.gif').convert('RGB'))


create_cluster_animation(X, history, scores)
```



Si poteva fare più velocemente ...

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=n_clusters, max_iter=50).fit(X)
gmm_scores = gmm.score_samples(X)

print('Means by sklearn:\n', gmm.means_)
print('Means by our implementation:\n', np.array([cluster['mu_k'].tolist() for cluster in clusters
]))
print('Scores by sklearn:\n', gmm_scores[0:20])
print('Scores by our implementation:\n', sample_likelihoods.reshape(-1)[0:20])
```

```
Means by sklearn:
 [[5.91697517 2.77803998 4.20523542 1.29841561]
 [5.006      3.428      1.462      0.246     ]
 [6.54632887 2.94943079 5.4834877  1.98716063]]
Means by our implementation:
 [[5.006      3.428      1.462      0.246     ]
 [6.54454865 2.94866115 5.47955343 1.98460495]
 [5.91496959 2.77784365 4.20155323 1.29696685]]
Scores by sklearn:
 [ 1.57050082  0.73787138  1.14436656  0.92913238  1.411028   -0.09451903
  0.05266884  1.62442195  0.27082378  0.16706624  0.83489877  0.77168582
  0.29597841 -1.79224582 -3.41557928 -2.10529279 -1.12995447  1.47503579
 -0.84612536  0.97699215]
Scores by our implementation:
 [ 1.57057947  0.73793642  1.14444614  0.92920539  1.41110417 -0.09448868
  0.05268031  1.62449505  0.27090462  0.16702226  0.83494742  0.77171947
  0.29597776 -1.79222469 -3.41562626 -2.1052825  -1.1300608   1.47509939
 -0.84608424  0.9770596 ]
```