

# Algoritmi di clustering

Algoritmi per l'intelligenza artificiale

Vincenzo Bonnici

Corso di Laurea Magistrale in Scienze Informatiche

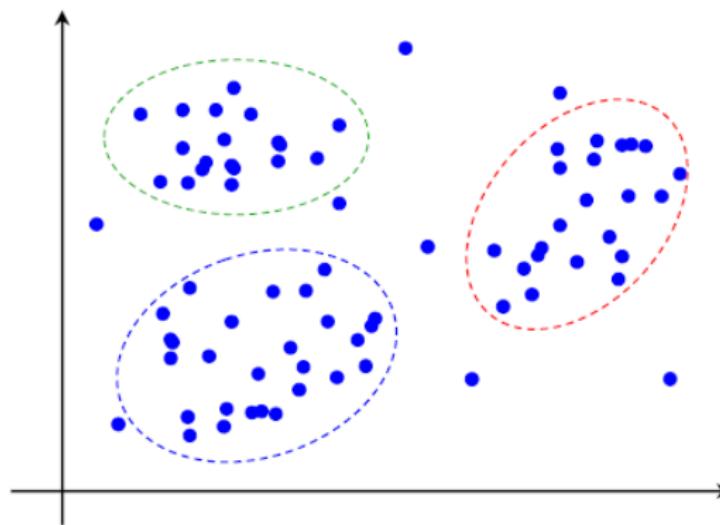
Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Università degli Studi di Parma

2025-2026

# Clustering

Con il termine **Clustering** (in italiano raggruppamento) si denota un famiglia di metodi **non supervisionati** in grado di individuare **raggruppamenti intrinseci** (cluster) di punti nello spazio multidimensionale e di definire in corrispondenza di tali raggruppamenti le classi (**incognite**).



un **Cluster** è una raccolta di oggetti (punti) dati:

- Simili tra loro all'interno dello stesso cluster
- Dissimili dagli oggetti in altri cluster

Clusterizzare (**raggruppare**) significa:

- Trovare somiglianze tra i dati in base ai caratteristiche riscontrate nei dati e raggruppamento di oggetti simili tra loro in cluster (gruppi)

Viene chiamato **apprendimento non supervisionato** perché non ci sono classi predefinite.

**Applicazioni** tipiche:

- Come strumento autonomo per ottenere informazioni sulla distribuzione dei dati
- Come fase di pre-elaborazione per altri algoritmi

Un buon metodo di clustering produrrà cluster di alta **qualità** con

- elevata somiglianza intra-classe
- bassa somiglianza inter-classe

La qualità di un risultato di clustering dipende sia dalla **misura di similarità** tra punti utilizzata dal metodo che dalla sua **attuazione**.

La qualità di un metodo di clustering si misura anche dalla sua capacità di **scoprire** alcuni o tutti gli **schemi (pattern, regolarità) nascosti**.

- Scalabilità
- Capacità di gestire diversi tipi di attributi
- Capacità di gestire dati dinamici
- Scoperta di cluster con forma arbitraria (usando la misura euclidea troviamo clustering sferico)
- Requisiti minimi per la conoscenza del dominio per determinare i parametri di input
- In grado di gestire rumore e valori anomali (dati mancanti, sconosciuti, errati)
- Insensibile all'ordine dei record di input
- Alta dimensionalità (gli occhi umani sono bravi a giudicare la qualità di raggruppamento fino a tre dimensioni) (i dati sono scarsi)
- Incorporazione di vincoli specificati dall'utente
- Interpretabilità e usabilità

## Clustering: complessità

Il problema è molto complesso (**NP-hard**): determinare la soluzione ottima con ricerca esaustiva è possibile solo nei casi semplici (pochi dati).

Dati  $n$  oggetti, assumendo di conoscere il numero di classi  $s$ , il numero di soluzioni distinte è dell'ordine di  $\frac{s^n}{s!}$  (approssimazione numero di Stirling di seconda specie). Esempio:  $n = 100$ ,  $s = 5$ , il numero di soluzione è circa  $10^{67}$ .

Se  $s$  non è noto, il numero di soluzioni è dato dai numeri di Bell:

$B_0 = 1$ ;  $B_{n+1} = \sum_{s=0}^n \binom{n}{s} B_s$ . Una approssimazione è data dalla formula di Dobinski:  $B_n = \frac{1}{e} \sum_{s=0}^{\infty} \frac{k^n}{k!}$ . Esempio: per l'insieme di oggetti  $\{A, B, C\}$ , quindi  $n = 3$  e  $s = 2$ , i cluster ottenibili sono:

$(A)(B, C); (A, B)(C); (B)(A, C)$ .

Le principali famiglie di algoritmi sono:

- **Clustering basato su partizionamento**: attraverso metodi euristici (iterativi) si individuano i cluster (partizioni) cercando di minimizzare la distanza dei punti dai centroidi dei cluster cui appartengono:
  - K-means
  - Fuzzy K-means
  - Expectation-Maximization (Gaussian Mixture)
- **Clustering gerarchico**: attraverso operazioni tipicamente **bottom up**, che aggregano punti in base a una misura di distanza, si organizzano i dati in **struttura ad albero** (dendogramma)
- **Clustering basato sulla densità**: i cluster individuati sono regioni connesse in aree ad elevata densità. L'approccio più noto è DBSCAN.

Distinguiamo inoltre:

- **Clustering hard**: ogni punto è assegnato in modo **esclusivo** a un solo cluster.
- **Clustering soft** (fuzzy, in italiano sfocato o di peluria) i punti appartengono ai diversi cluster con un certo **grado di appartenenza** (es tra 0 e 1).  
È più efficace nel gestire pattern vicino al bordo di due o più clusters e outliers. L'assegnazione può diventare esclusiva scegliendo, per ogni punto, il cluster verso cui il grado di appartenenza è massimo.

Criteri e algoritmi di clustering sono due cose ben distinte

- **criteri**: descrivono cosa si vuol ottenere specificando il grado di ottimalità di ogni soluzione ammissibile
- **algoritmi**: dato un criterio di clustering, forniscono una procedura algoritmica per determinare soluzioni che lo ottimizzano

La maggior parte dei **criteri** di clustering sono definiti sulla base delle due **osservazioni** seguenti:

- i punti all'**interno** (**intra**) dello stesso cluster devono essere tra loro **più simili** rispetto a punti appartenenti a cluster diversi (**inter**)
- i cluster sono costituiti da **nuvole di punti** a **densità** relativamente elevata separate da zone dove la densità è più bassa

Costruire una **partizione** di un data set  $D$  di  $n$  oggetti in un insieme di  $k$  cluster.

Dato un valore per  $k$ , trovare una partizione di  $k$  cluster che ottimizza il criterio scelto.

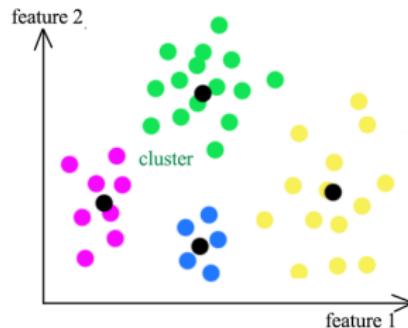
- metodi **ottimali**: enumerano esaustivamente tutte le possibili partizioni
- metodi **euristici**:
  - k-means: ogni cluster è rappresentato dal suo (bari)centro
  - k-medoids: ogni cluster è rappresentato da uno (il più centrale) degli oggetti del cluster

# Clustering: minimizzazione delle distanze tra centroidi (inter-classe)

Tra i criteri più usati c'è la **minimizzazione delle distanze dai centroidi** che minimizza la somma dei quadrati delle distanze dei punti  $x$  dai **centroidi** (i.e. baricentri) delle classi:

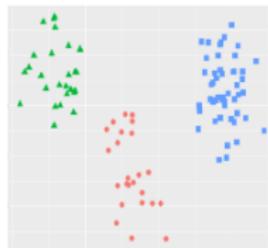
$$J_e = \sum_{i=1, \dots, k} \sum_{x \in C_i} \|x - \hat{x}_i\|^2, \text{ con } \hat{x}_i = \frac{1}{n} \sum_{x \in C_i} x$$

dove  $C_i$  è l' $i$ -esimo cluster,  $n_i$  il numero di punti che esso contiene e  $\hat{x}_i$  il suo centroide (media/baricentro).

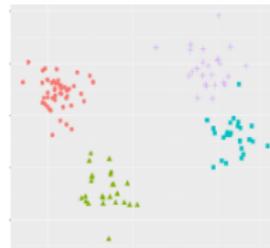


# Clustering: minimizzazione distanze tra centriodi (inter-classe)

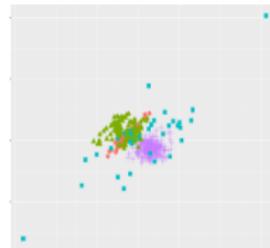
È un buon criterio per cluster a **simmetria radiale** (i.e. circolari), ma penalizza forme allungate o cluster innestati (i.e. un cluster a forma di anello con all'interno un altro cluster)



PAM,  $K = 3$  ( $ARI = 0.990$ )



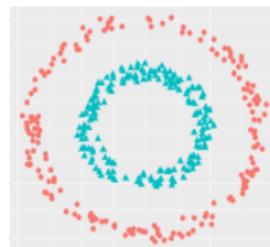
mclust,  $K = 4$  ( $ARI = 0.955$ )



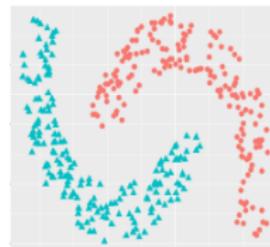
mclust,  $K = 4$  ( $ARI = 0.834$ )



Complete linkage,  $K = 2$  ( $ARI = 1.000$ )



Single linkage,  $K = 2$  ( $ARI = 1.000$ )



Spectral clustering,  $K = 2$  ( $ARI = 1.000$ )

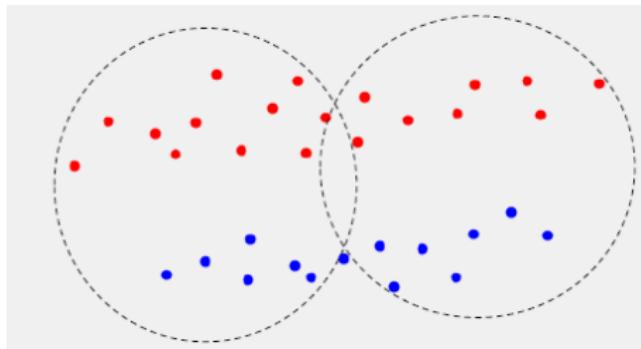
Un altro possibile criterio è **minimizzare le distanze intra-classe**:

$$J_e = \sum_{i=1, \dots, k} n_i \cdot \hat{s}_i, \text{ con } \hat{s}_i = \frac{1}{n_i} \sum_{x \in C_i} f_s(x, C_i)$$

dove  $f_s(x, C_i)$  è una misura di distanza tra  $x$  e il cluster cui appartiene. Ad esempio:

- ①  $f_s(x, C_i) = \frac{1}{n_i} \sum_{x' \in C_i} \|x - x'\|^2$ : criterio simile alla minimizzazione delle **distanze dal centroide**
- ②  $f_s(x, C_i) = \min_{x' \in C_i, x' \neq x} \|x - x'\|^2$ : non penalizza cluster allungati affinché  $f_s(X, C_i)$  assuma valore ridotto, non è necessario che tutti i (o gran parte dei) punti di  $C_i$  siano vicini a  $x$ , ma è sufficiente **un vicino**.

Nell'esempio sotto, per  $k = 2$ , (inter-classe) favorirebbe l'aggregazione come da cerchi tratteggiati, mentre (intra-classe) come da colori (rosso e blu).



# K-means (basato su centroidi)

**K-means** (o C-means) è un metodo **computazionalmente molto semplice** e altrettanto semplice da implementare per questo motivo è spesso la prima scelta per risolvere problemi di clustering.

- **Minimizza** implicitamente le distanze dai **centroidi**
- Richiede in **input** il **numero di cluster**  $k$  e una **soluzione iniziale**. Produce buoni risultati a patto di fornire una ragionevole soluzione iniziale e un numero adeguato di classi.
- Il tipo di **ottimizzazione** è **iterativa** e **locale**; pertanto il metodo può convergere a massimi locali della soluzione. La convergenza si ottiene solitamente in pochi passi ( $\leq 10$ )
- Identifica cluster **iper-sferici** nel caso in cui venga utilizzata la distanza euclidea come misura di distanza tra i punti o cluster **iper-ellissoidali** nel caso di distanza di Mahalanobis

$$(d(\vec{x}, \vec{y}) = \sqrt{\sum \frac{(x_i - y_i)^2}{\sigma_{x_i}^2}})$$

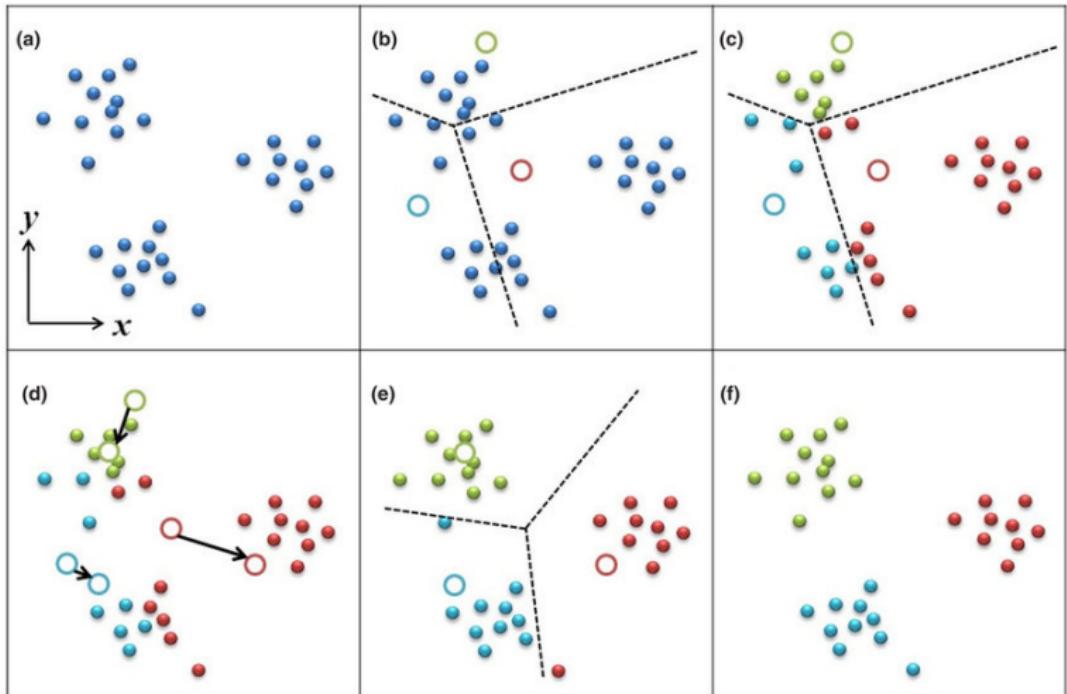
Nella sua versione base l'algoritmo può essere così descritto

- ① **Inizializza**  $n, k$ .
- ② Scegli **casualmente**  $k$  punti da utilizzare come **centroidi iniziali**
- ③ do
  - ① **assegna** ogni punti al cluster per cui è minima la distanza dal centroide
  - ② **calcola** i centroidi dei cluster come media dei rispettivi punti

while (i cluster sono stati modificati AND  $iteration < max$ )

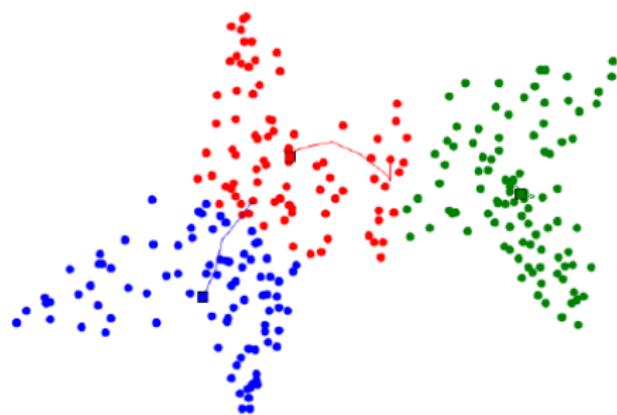
I cluster sono modificati iterativamente a seguito del ricalcolo del loro centroide. L'algoritmo **termina** (converge) quando i centroidi sono stabili e quindi le partizioni non cambiano.

# K-means: esempio



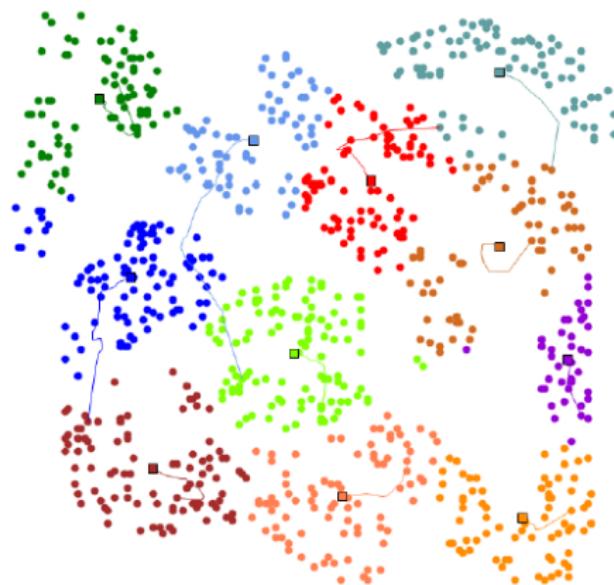
## 3 classi, 8 iterazioni

Minimizzando le distanze dai centroidi , K-means non è in grado di identificare cluster dalla forma non sferica.



## 11 classi, 31 iterazioni

anche in questo caso non tutti i cluster hanno forma sferica.



## Vantaggi:

E' relativamente efficiente, infatti prende tempo  $O(t \cdot k \cdot n)$ , dove  $t$  è il numero di iterazioni,  $k$  il numero di cluster richiesto e  $n$  il numero di oggetti da clusterizzare.

## Svantaggi:

- Spesso termina in ottimi locali (dipende dalla soluzione iniziale)
- necessita di specificare un valore per  $k$
- non è molto robusto al rumore e agli outliers
- non è in grado di trovare cluster di forma non convessa (sfera o ellise)
- non direttamente applicabile a dati categoriali

Diverse varianti sono state proposte per **generare buone soluzioni** iniziali e per determinare il numero di classi.

Per minimizzare il rischio di **convergenza verso minimi locali** l'algoritmo può essere **eseguito più volte** a partire da **soluzioni iniziali diverse**:

- **random**
- prodotte da diverse **euristiche**
- prodotte da un metodo **evoluzionistico** (algoritmo genetico)

# K-means: quanti cluster?

Le tecniche di validazione tendono a valutare a posteriori la bontà delle soluzioni prodotte per diversi valori di  $s$ , e a sceglierne una sulla base di un **criterio di validazione** che tenga conto sia della **bontà** della soluzione sia della sua **complessità**.

## Problema

Considerando il criterio di **minimizzazione distanze dai centroidi** e lanciando K-means con diversi valori di  $k$ , è molto più facile ottenere valori ridotti di  $J_e$  per valori elevati di  $k$ .

Quanto vale  $J_e$  per  $k = n$  (ogni cluster un solo pattern)?

## Possibile soluzione

Penalizzare le soluzioni con molti cluster, ad esempio:

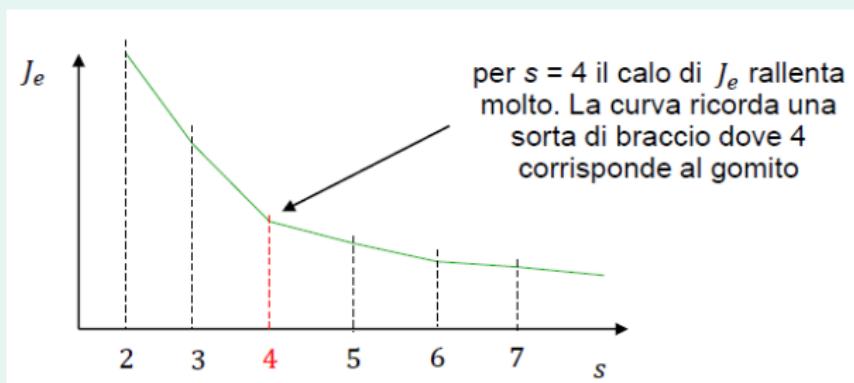
$$J_e^* = J_e + \text{penalty} \cdot k$$

Di fatto però il problema si ribalta nella scelta di *penalty*.

# K-means: quanti cluster?

## Metodo della discontinuità del grafico di $J_e$

Un modo più efficace per determinare il numero ottimale di cluster è quello di cercare **punti di inflessione** (anche detti elbow) nel valore di  $J_e$  al variare di  $k$ . Infatti, se i pattern formano raggruppamenti evidenti e ben identificabili, scegliendo il corretto valore di  $k$ , dovremmo riscontrare una discontinuità nel grafico di  $J_e$  variare di  $k$ .



## Metodo del silhouette score

Un criterio più oggettivo (che funziona per cluster sferici come quelli prodotti da k-means) è quello di calcolare il **silhouette score** come media dei **coefficienti di silhouette** di ciascun punto e scegliere  $k$  che lo **massimizza**.

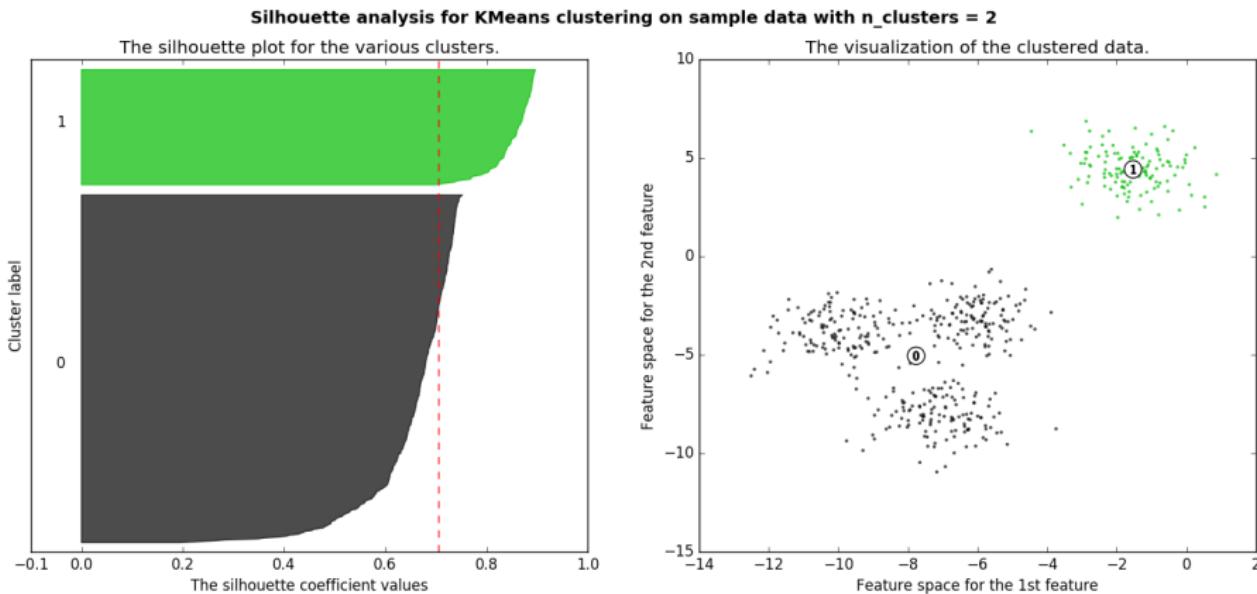
$$\text{silhouette coefficient} = \frac{|b - a|}{\max(a, b)}$$

dove  $a$  è la **distanza media intra-cluster** di un punto e  $b$  la distanza tra tale punto e i punti del cluster ad esso più vicino (escluso quello di appartenenza).

Risulta però computazionalmente pesante per grandi dataset.

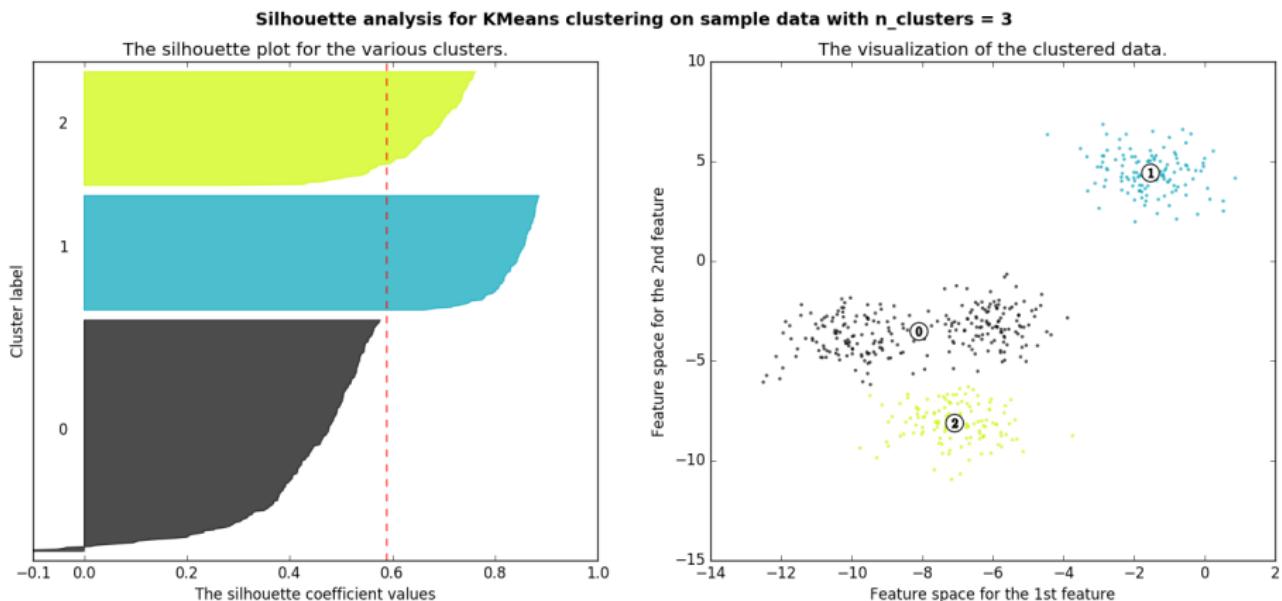
# K-means: quanti cluster?

## Metodo del silhouette score



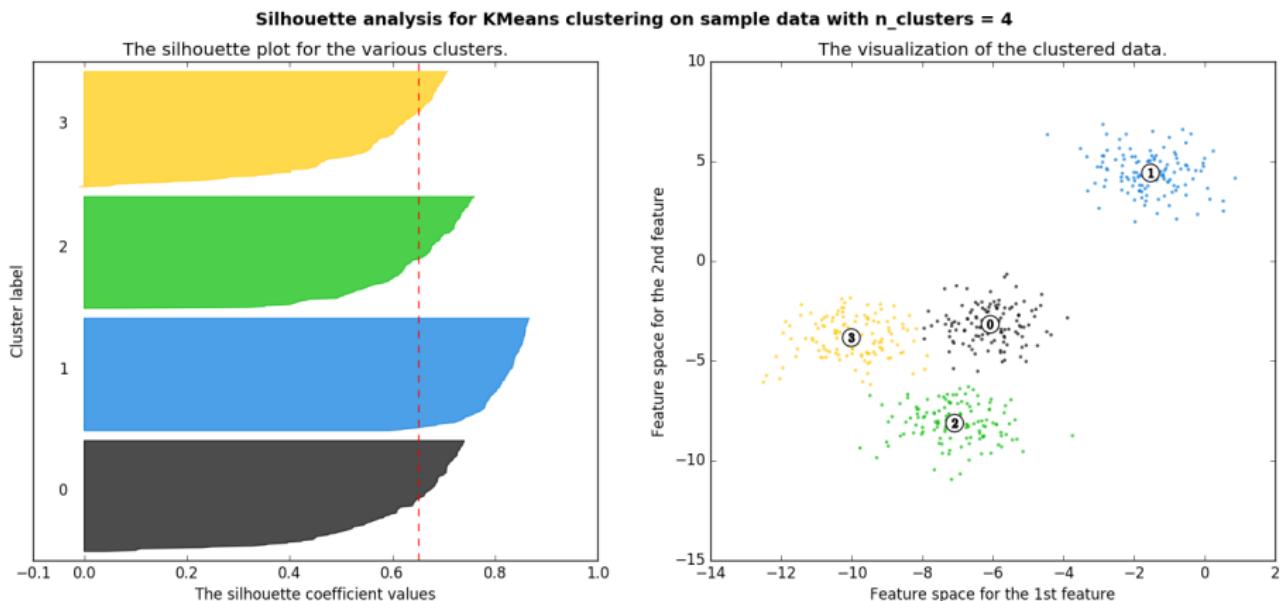
# K-means: quanti cluster?

## Metodo del silhouette score



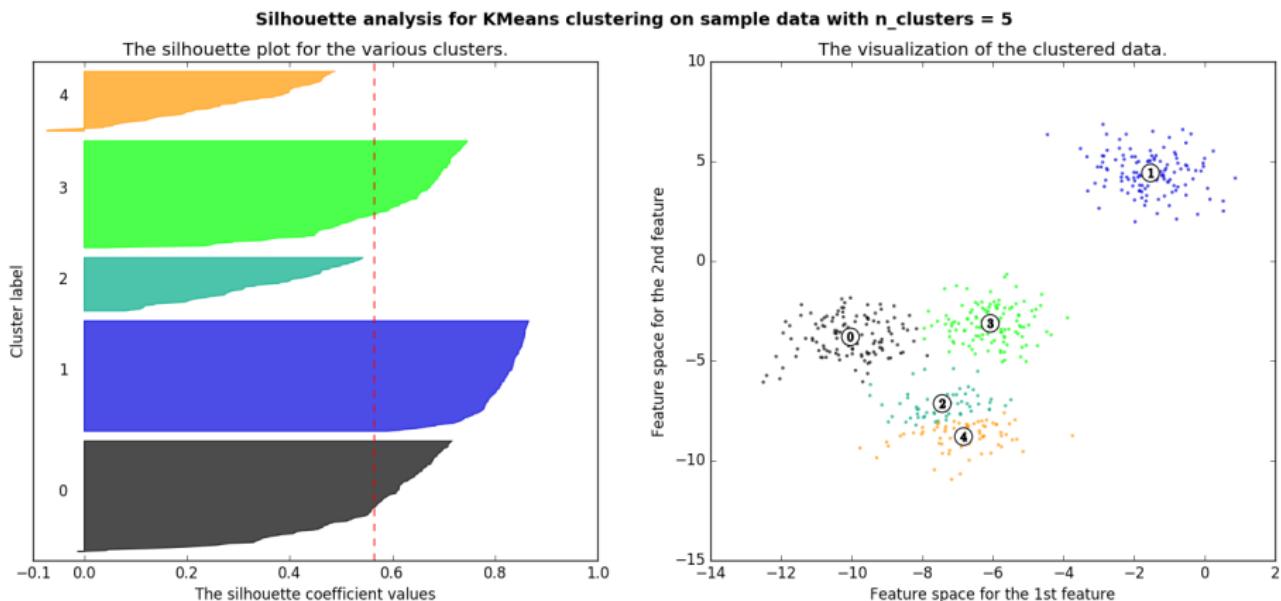
# K-means: quanti cluster?

## Metodo del silhouette score



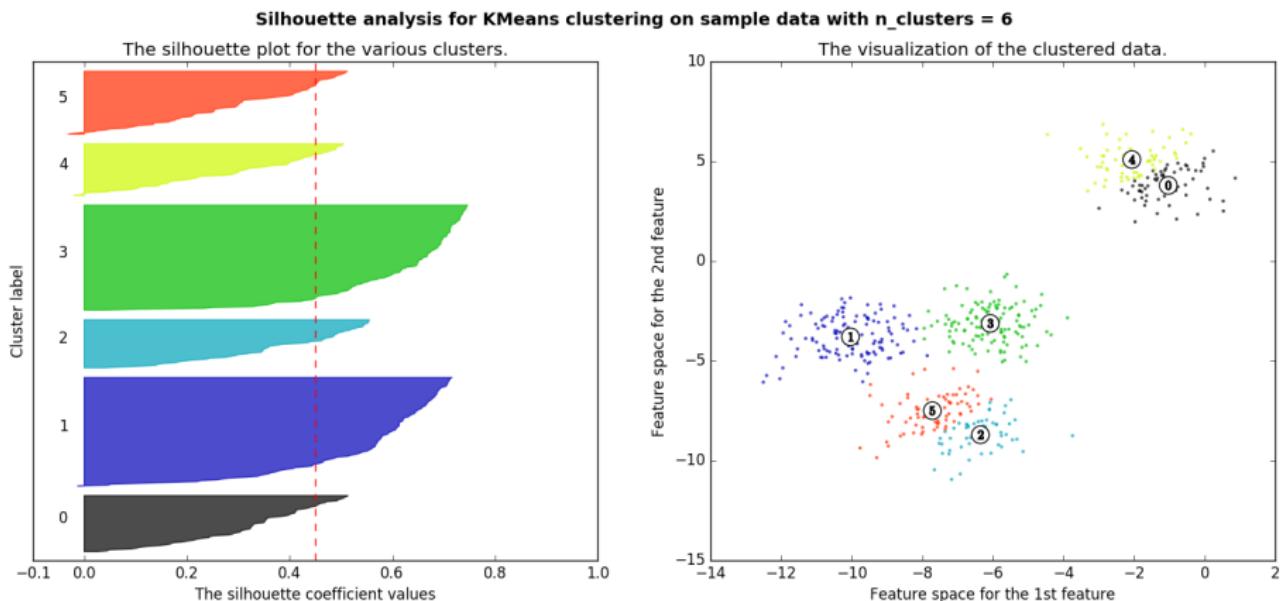
# K-means: quanti cluster?

## Metodo del silhouette score



# K-means: quanti cluster?

## Metodo del silhouette score

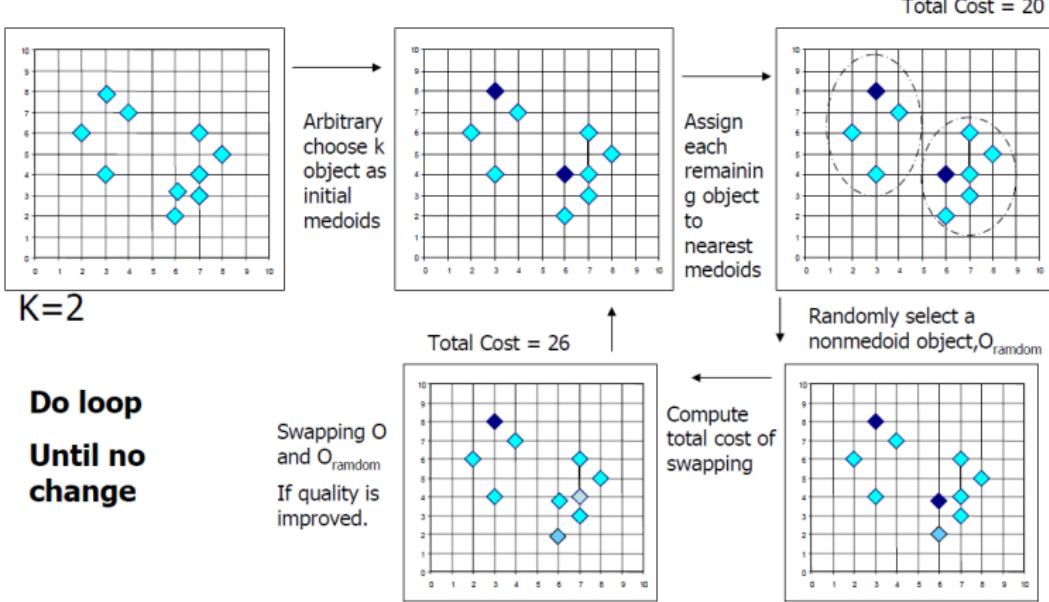


Cerca di risolvere il problema degli **outlier** di K-means. Infatti, k-means è sensibile agli outlier perchè un oggetto con dei valori sensibilmente più alti della media dei valori in un cluster può **distorcere** la distribuzione dei dati del cluster e quindi la media.

**K-medoids**: invece di prendere il centroide, si prende in considerazione l'oggetto più centrale al cluster, il medoide.

**PAM** (Partitioning Around Medoids, 1987) parte da un insieme iniziale di medoidi e sostituisce iterativamente uno dei medoidi da uno dei non medoidi se migliora la distanza totale del clustering risultante.

# K-medodis: PAM



PAM è più robusto di k-means in presenza di rumore e valori anomali perché un medoide è meno influenzato da valori anomali o valori estremi di una media.

PAM funziona in modo efficiente per **piccoli set di dati** ma non è scalabile bene per grandi set di dati:

$O(k(n - k)^2)$  per ogni iterazione, dove  $n$  è il numero di dati,  $k$  è il numero di cluster.

Alternativa:

Metodo basato sul **campionamento**: CLARA (raggruppamento di applicazioni "grandi")

**CLARA** disegna **più campioni** del set di dati, applica PAM su ciascuno campione e fornisce il miglior clustering come output.

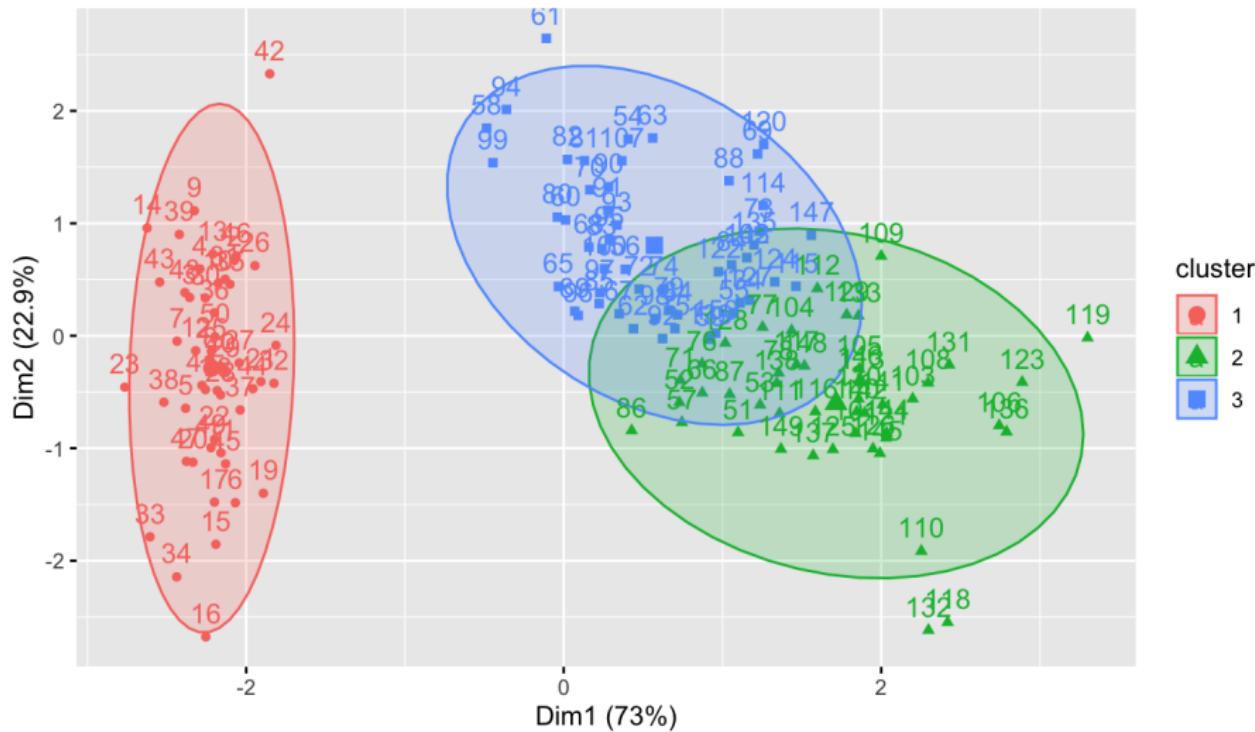
**Vantaggi:** si occupa di set di dati più grandi di PAM.

**Svantaggi:**

- l'efficienza dipende dalla dimensione del campione
- Un buon raggruppamento basato su campioni non necessariamente rappresenta un buon raggruppamento dell'intero del set di dati se il campione è distorto

# K-medoids: CLARA (Kaufmann & Rousseeuw, 1990)

Cluster plot



L'algoritmo **BFR** (Bradley-Fayyad-Reina) è un algoritmo di clustering **basato sul k-means**.

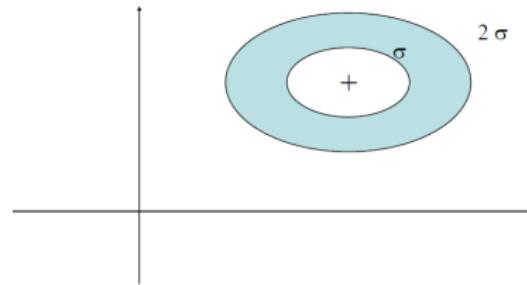
Lo scopo e' rendere l'algoritmo k-means piu' **scalabile**.

Esso individua tre **tipi di regioni** di dati:

- Regioni **scartabili**: oggetti la cui appartenenza ad un cluster e' certa.
- Regioni **comprimibili**: oggetti che non sono scartabili ma che appartengono a subcluster molto ben definiti e stretti/densi;
- Regioni che devono essere mantenute in **memoria principale** perché non ancora associate a nessun cluster o sotto-cluster specifico.

L'algoritmo funziona meglio se i punti sono distribuiti normalmente (**Gaussiana**) attorno a dei punti centrali magari con delle **deviazioni standard diverse su ogni dimensione**.

Se ad esempio nella dimensione orizzontale si ha una deviazione standard  $\sigma$  doppia di quella verticale si ha che il 70% circa di punti giace nell'ellisse interna di semiasse  $,\sigma$ , il 95% in  $2\sigma$ , 99,9% in  $3\sigma$ , il 99,9999 in  $4\sigma$ .



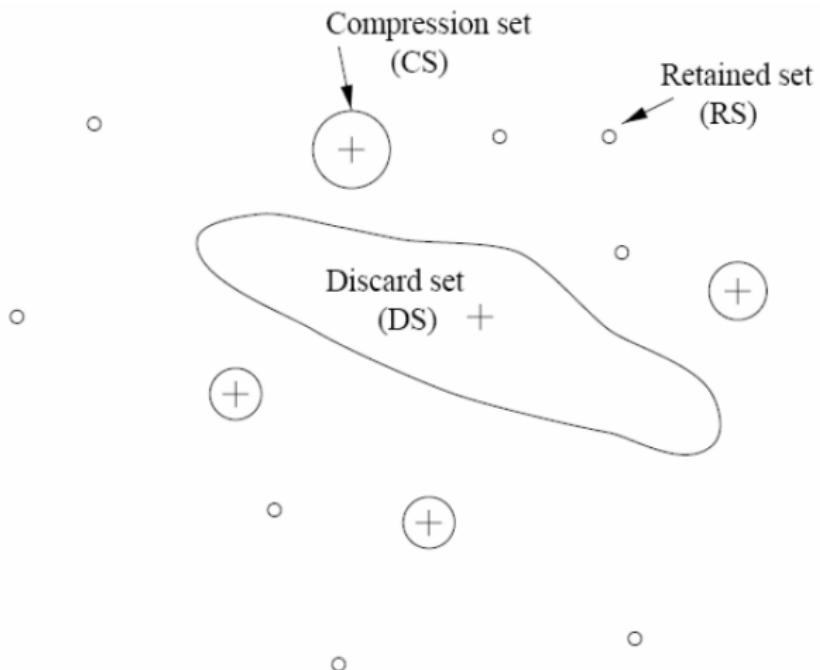
## BFR: rappresentazione di un cluster

Fayyad lavorò a Skycat e pensò ai cluster come galassie.

Ogni cluster consiste in :

- Un **nucleo centrale** di cluster collettivamente chiamati **Discard Set (DS)**. Ciascuno dei punti in un cluster del DS è considerato certo di appartenere al cluster. Ogni cluster viene rappresentato da semplici statistiche che dicono dov'è il centroide e quale è la deviazione standard su ogni componente.
- **Sottocluster**, collettivamente chiamati **Compression Set (CS)**. Ciascun sottocluster nel CS consiste di un gruppo di punti sufficientemente vicini da poter essere rappresentati dalle loro statistiche come i DS. Tuttavia essi sono abbastanza lontani dai centroidi dei DS da non sapere a quale associarli.
- **Singoli punti** che non sono parte né di DS né di CS sono chiamati collettivamente **Retained Set (RS)**. Essi non possono né essere assegnati a DS né raggruppati in CS. Sono immagazzinati nella memoria centrale, come singoli punti, assieme alle statistiche dei DS e CS.

# BFR: rappresentazione di un cluster



Le statistiche per rappresentare ogni cluster dei DS ed ogni sottocluster dei CS sono:

- ① La cardinalità  $N$  (numero di punti)
- ② Il vettore  $SUM$  delle somme  $SUM_i$  delle coordinate in ciascuna dimensione  $i$ .
  - $x = (x_1, x_2, \dots, x_k); y = (y_1, y_2, \dots, y_k)$
  - $SUM = (x_1 + y_1, \dots, x_k + y_k)$
- ③ Il vettore  $SUMSQ$  delle somme  $SUMSQ_i$  dei quadrati delle coordinate in ciascuna dimensione  $i$ .
  - $SUMSQ = (x_1^2 + y_1^2, \dots, x_k^2 + y_k^2)$

Se vi sono  $k$  dimensioni, osserviamo che queste tre statistiche, formano  $2k + 1$  numeri e sono sufficienti a calcolare le **caratteristiche** più importanti dei DS e CS.

Ad esempio:

- la coordinata  $\mu_i$  del centroide è  $SUM_i/N$
- la varianza della dimensione  $i$  ( $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$ ) è  $(SUMQ_i/N) - (SUM_i/N)^2$
- e la sua radice quadrata è la deviazione standard  $\sigma_i$

## BFR: RAM piena di punti

Con il primo **carico di punti** nella RAM, BFR seleziona i  $k$  centroidi con qualche algoritmo su RAM (esempio fissa un campione di punti, calcola i  $k$  centroidi esatti e sceglili come centroidi iniziali)

L'intera RAM piena di punti è trattata, così come i successivi caricamenti di punti, nel seguente modo:

- ① Determinare quali punti sono sufficientemente vicini ad un centroide da essere inseriti nel DS e le loro statistiche ( $N, SUM, SUMSQ$ ) combinati con quelle del cluster.
- ② Aggiorna le statistiche  $N, SUM, SUMSQ$  del cluster in DS.
- ③ Prendi i punti il cui **Raggio di Mahalanobis** è al di sotto di un certa soglia, ad esempio 4 volte la deviazione standard del DS.

### Raggio di Mahalanobis

E' essenzialmente la distanza dal centroide scalata mediante  $\sigma_i$ . Più precisamente se  $\mu_i$  è la media nella  $i$ -esima dimensione allora il Raggio di Mahalanobis del punto  $= [y_1, y_2, \dots, y_k]$  è  $\sqrt{\sum_{i=1}^k \left(\frac{y_i - \mu_i}{\sigma_i}\right)^2}$

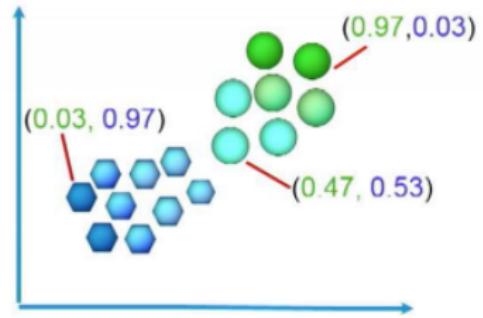
- ④ Nella RAM prova a fare il cluster di punti non in DS incluso quelli di RS di round precedenti. Se troviamo un cluster di punti la cui varianza è al di sotto di una soglia, allora assegniamo questi punti al CS e sostituiamo i punti con le statistiche.
- ⑤ Fondi un nuovo sottocluster con uno vecchio di CS purchè la varianza del sottocluster ottenuto(calcolabile dalle statistiche dei due CS) sia al di sotto della soglia
- ⑥ Se siamo all'ultimo round assegniamo elementi di CS e RS al più vicino cluster di DS anche se abbastanza lontano.

# Fuzzy K-means

**Hard Clustering**



**Soft Clustering**



La variante **Fuzzy** del K-means consente a un pattern di appartenere con un **certo grado di probabilità** a diverse classi. Il criterio di ottimizzazione in questo caso è:

$$J_{fux} = \sum_{i=1, \dots, s} \sum_{j=1, \dots, n} [P(C_i|x_j, \Theta)]^m \cdot ||x_j - \theta_i||^2$$

dove  $P(C_j|x_j, \Theta)$  è la **probabilità** che dato il punto  $x_j$  e l'insieme  $\Theta = \{\theta_1, \dots, \theta_s\}$  di centroidi che definiscono la **soluzione attuale**, il cluster di **appartenenza** sia  $C_j$ .

# Fuzzy K-means

I **centrodi** invece che come semplice media dei punti si calcolano come **media pesata** rispetto alle probabilità di appartenenza:

$$\theta_i = \frac{\sum_{j=1,\dots,n} P(C_i|x_j, \Theta) \cdot x_j}{\sum_{j=1,\dots,n} P(C_i|x_j, \Theta)}$$

Le **probabilità di appartenenza** si calcolano come:

$$P(C_i|x_j, \Theta) = \frac{1}{\sum_{k=1,\dots,s} \left( \frac{\|x_j - \theta_i\|}{\|x_j - \theta_k\|} \right)^{\frac{2}{m-1}}}$$

$m > 1$  è un parametro che definisce quanto l'appartenenza ai diversi cluster debba essere **sfumata**. Valore tipico  $m = 2$ .

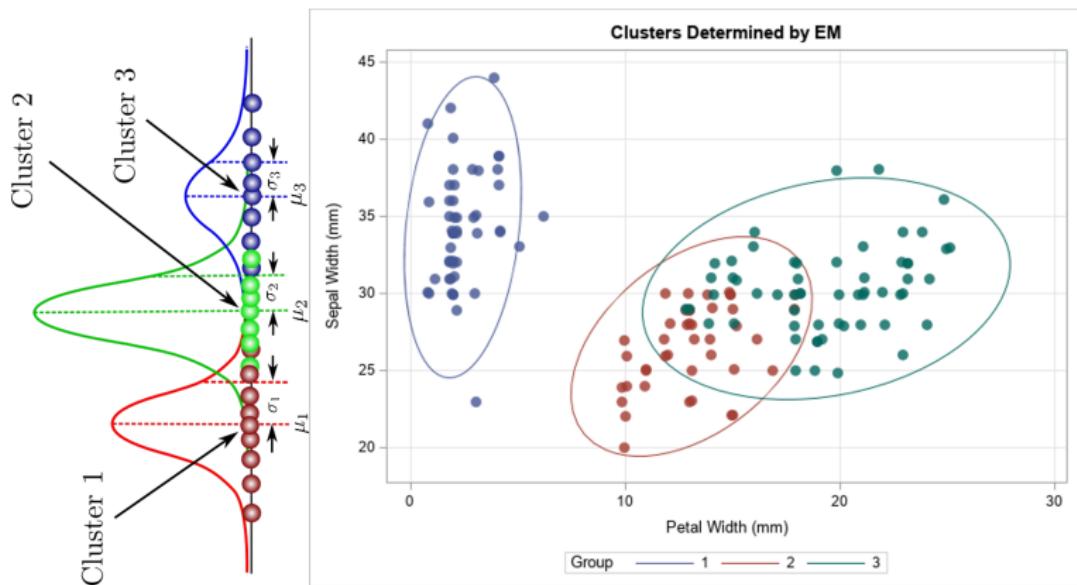
Lo pseudocodice dell'algoritmo è molto simile al K-means.

- ① **inizializza**  $n, s$
- ② **inizializza casualmente**  $P(C_i|x_j, \Theta), i = 1, \dots, x, j = 1, \dots, n$
- ③ do
  - ① calcola i centroidi  $\Theta = \{\theta_1, \dots, \theta_s\}$
  - ② calcola i gradi di appartenenza  $P(C_i|x_j, \Theta), i = 1, \dots, x, j = 1, \dots, n$
- while(riduzione  $J_{fuz}$  **significativa** AND *iteration* < *max*)

Rispetto al K-means questa variante fornisce a volte una **convergenza più robusta** verso la soluzione finale. D'altro canto uno **svantaggio** è legato al fatto che l'appartenenza di un punto a un cluster dipende implicitamente dal numero di cluster, e se questo è specificato in modo incorretto si possono ottenere cattive soluzioni.

# Gaussian mixture (EM)

Si ipotizza che i punti siano stati generati da un **mix di distribuzioni** (mixture in inglese): ogni classe ha generato dati in accordo con una **specifico distribuzione** ma al termine della generazione i punti appaiono come prodotti da un'**unica distribuzione multi-modale**.



**Obiettivo** del clustering è **risalire ai parametri delle singole distribuzioni** che li hanno generati. A tal fine si ipotizza nota la forma delle distribuzioni e si assume, per semplicità, che esse siano tutte dello stesso tipo. Il caso più frequente è quello di mix di  $s$  distribuzioni multinormali (gaussiane) di cui si vogliono stimare i parametri di definizione ( $s$  vettori medi +  $s$  matrici di covarianza +  $s$  coefficienti  $\alpha$ ):

$$p(x|\Theta) = \sum_{i=1,\dots,k} \alpha_i \cdot p_i(x|\theta_i)$$

dove  $\Theta = \{\alpha_1, \alpha_2, \dots, \alpha_k, \theta_1, \theta_2, \dots, \theta_k\}$  è il vettore di parametri che definisce il mix di distribuzioni,  $p_i(\cdot|\theta_i)$  è una multinormale con parametri  $\theta_i = \{\mu_i, \sigma_i\}$ .

La stima dei parametri avviene secondo il criterio generale del **maximum likelihood** (in italiano massima verosimiglianza).

In generale il **likelihood**  $\mathcal{L}$  dei **parametri**  $\Theta$  dati i **punti**  $\mathcal{X}$  corrisponde alla probabilità (densità di probabilità nel continuo) di aver ottenuto i punti dati i parametri (inversione):

$$\mathcal{L}(\Theta|\mathcal{X}) = p(\mathcal{X}|\Theta)$$

Per semplicità, al posto del likelihood si massimizza il suo logaritmo. Considerando i punti indipendenti, la probabilità di avere ottenuto i punti del training set è il prodotto delle probabilità delle singole generazioni:

$$\log(p(\mathcal{X}|\Theta)) = \log \prod_{j=1,\dots,n} p(x_j|\Theta) = \sum_{j=1,\dots,n} \log \left( \sum_{i=1,\dots,k} \alpha_i \cdot p_i(x_j|\theta_i) \right)$$

Purtroppo questa massimizzazione è molto difficile a causa della sommatoria dentro al logaritmo. Per eliminare la sommatoria, ci servirebbe sapere ogni punto  $x_j$  del training set da quale componente  $p_i(\cdot)$  della mixture è stato generato.

A tal fine si utilizza **EM** che è un approccio **iterativo** nato per il calcolo del maximum likelihood nel caso in cui i dati a disposizione  $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$  siano **incompleti** per la mancanza di alcuni valori  $\mathcal{Y} = \{y_1, y_2, \dots, y_n\}$ . Ogni punto completo  $z_j = [x_j, y_j]$ ,  $j = 1, \dots, n$ , è costruito da due parti di cui solo la prima è nota.

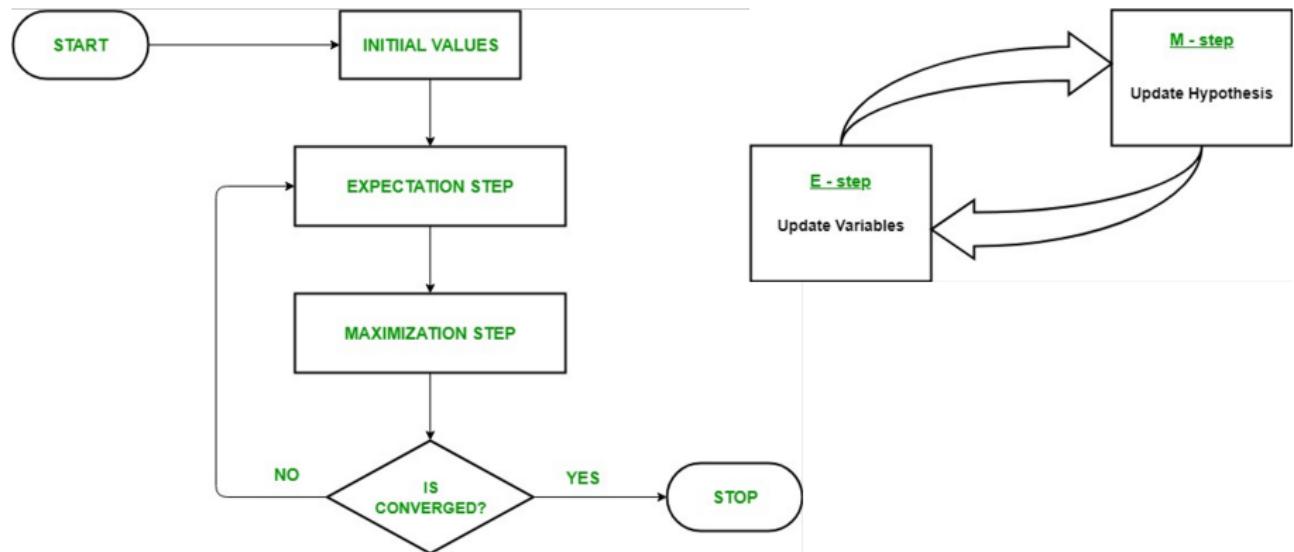
Nel caso di interesse (derivare i parametri di una gaussian mixture) i dati sono completi ma le potenzialità di EM sono utilizzate per rendere trattabile la massimizzazione sfruttiamo: i valori  $\mathcal{Y}$  come indicatori delle componenti ignote che hanno generato i singoli punti.

**Obiettivo** di EM è dunque la **massimizzazione** del log-likelihood dei dati completi o log-likelihood-completo:

$$\log(\mathcal{L}(\Theta|\mathcal{Z})) = \log(\mathcal{L}(\Theta|\mathcal{X}, \mathcal{Y})) = p(\mathcal{X}, \mathcal{Y}|\Theta)$$

A tal fine vengono eseguiti **iterativamente** (fino a convergenza) due passi, **Expectation** e **Maximization**.

# Expectation-Maximization (EM) con Gaussian Mixture



## Expectation

Viene calcolato il **valore atteso**  $E(\cdot)$  del log-likelihood-completo, dato il **training set**  $\mathcal{X}$  e una **stima dei parameteri**  $\Theta^g$  ottenuti alla iterazione precedente:

$$Q(\Theta|\Theta^g) = E(\log(p(\mathcal{X}, \mathcal{Y}|\Theta)|\mathcal{X}, \Theta^g))$$

Il valore atteso (media) è calcolato rispetto alla variabile aleatoria  $\mathcal{Y}$ , governata dalla distribuzione  $f(y|\mathcal{X}, \Theta^g)$

$$E(\log(p(\mathcal{X}, \mathcal{Y}|\Theta)|\mathcal{X}, \Theta^g)) = \int_{y \in \Psi} \log(p(\mathcal{X}, \mathcal{Y}|\Theta)) \cdot f(y|\mathcal{X}, \Theta^g) dy$$

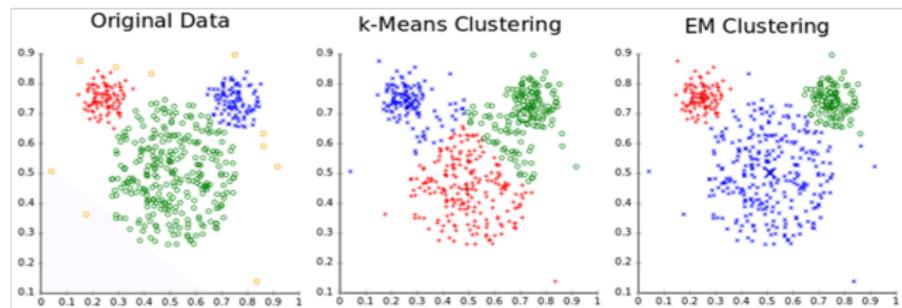
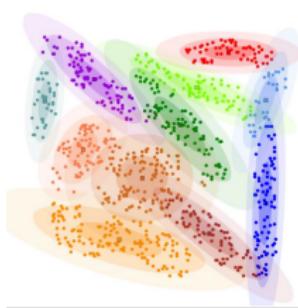
## Massimization

Determina il valore dei parametri che massimizzano il valore atteso calcolato al passo di Expectation:

$$\Theta^{g+1} = \operatorname{argmax}_{\Theta} Q(\Theta|\Theta^g)$$

# Expectation-Maximization (EM) con Gaussian Mixture

L'algoritmo è simile a fuzzy K-means ma in questo caso, oltre ai centroidi abbiamo maggiori gradi di libertà nella forma dei cluster che possono essere **ellisoidali**.

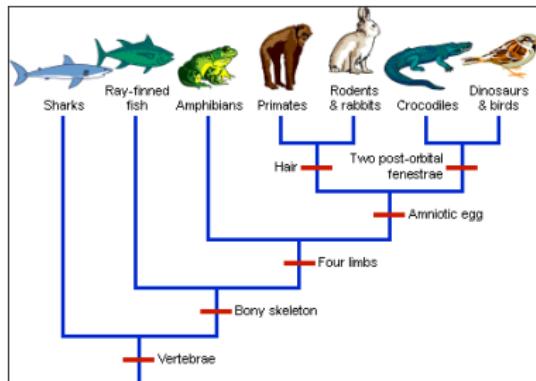


**Attenzione:** Le probabilità di appartenenza possono assumere valori molto piccoli e causare problemi numerici. Consiglio usare costanti moltiplicative se necessario e normalizzare alla fine.

# Clustering gerarchico

Simile al modo di eseguire classificazione in **tassonomia biologica**:

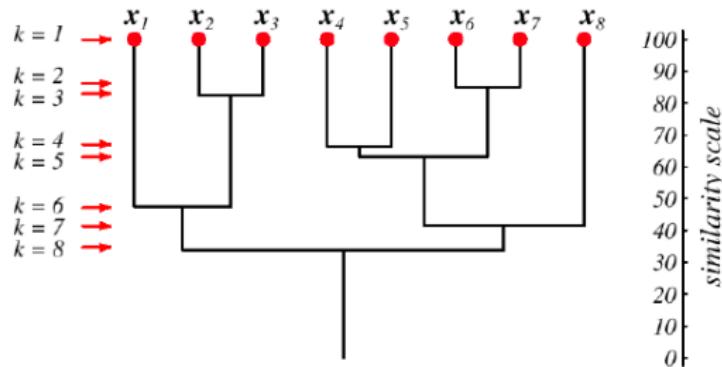
gli insetti sono **gerarchicamente** classificati **specializzandone** le specie a partire da famiglie molto **ampie** fino a famiglie più **ridotte**.



# Clustering gerarchico

Gli algoritmi sono generalmente **bottom-up (agglomerativi)**:

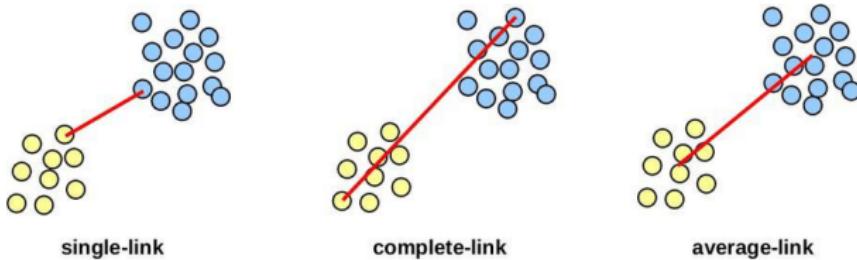
si parte cercando di **aggregare singoli** elementi e ad ogni passo (livello) si aggregano (punti a punti, punti a cluster o cluster a cluster) gli elementi tra loro più simili (i.e. **meno distanti** rispetto a una **soglia** (che dipende dal livello)).



# Clustering gerarchico

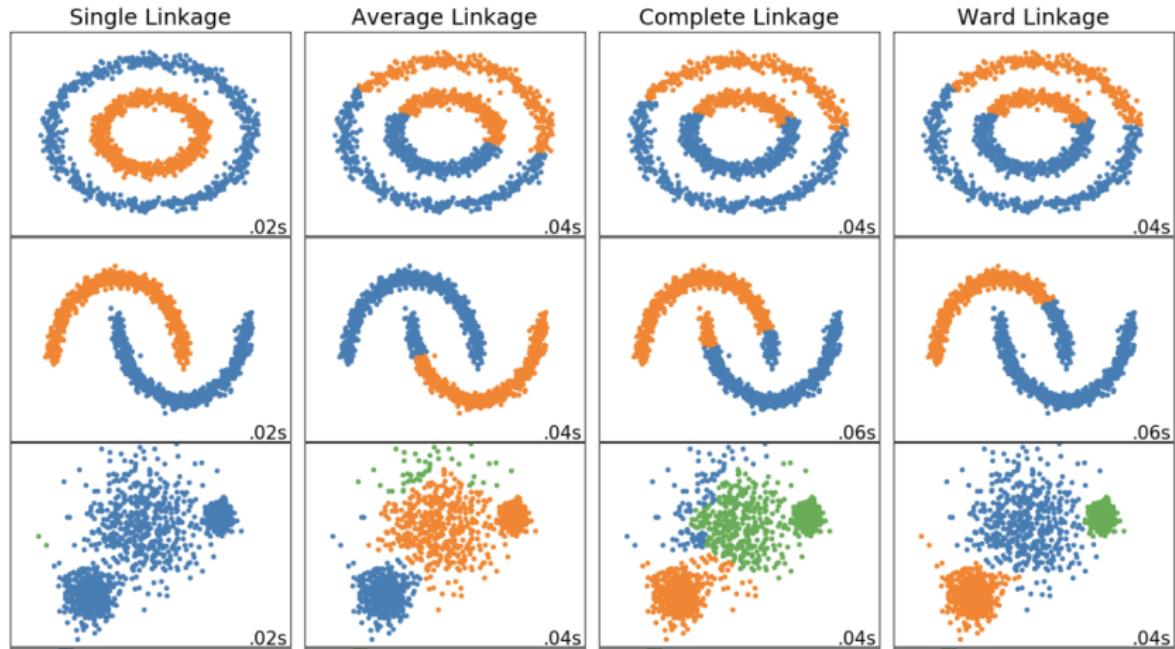
Le principali differenze implementative dipendono dalla definizione utilizzata per calcolare le distanze tra cluster :

- **Single linkage**: distanza **minima tra due punti** dei due cluster
- **Complete linkage**: distanza **massima tra due** punti dei cluster
- **Average linkage**: distanza **media tra tutti** i punti dei due cluster

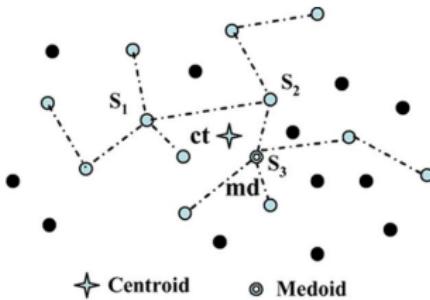


# Clustering gerarchico

- **Ward linkage**: minimizza (in modo ricorsivo) la **varianza intra-cluster**



- **distanza tra centroidi**: si calcola la distanza tra i **centroidi** (media dei punti di un cluster  $\mapsto$  punto possibilmente non nei dati di input) di due cluster
- **distanza tra medoidi**: si calcola la distanza tra i **medoidi** (punto collocato centralmente al cluster  $\mapsto$  obbligatoriamente tra i dati di input) di due cluster

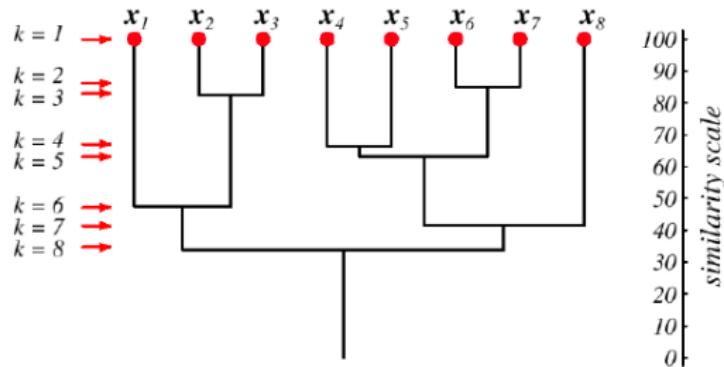


Concludiamo il processo di raggruppamento fino a quando abbiamo sufficientemente pochi cluster. Ovvero, quando si sono generati **k cluster**.

In congiunzione, si termina il merge di cluster quando ogni possibile merge causerebbe la **violazione** di qualche vincolo di compattezza: ad esempio la distanza media di ogni nodo dal suo centroide è troppo alta.

# Clustering gerarchico

A conclusione del processo di merging, decomponiamo gli oggetti in livelli diversi di partizione (tree of clusters), chiamato **dendrogramma**.



Il clustering e' ottenuto **tagliando** il **dendrogramma** in un desiderato livello.

## Problemi

- **complessità**  $O(n^2)$ , dove  $n$  è il numero di oggetti da clusterizzare, ogni decisione sul merge o sullo split richiede di conoscere un numero alto di altri dati.
- Non si puo' ovviare a cio' che e' stato fatto precedentemente

## Metodologie note:

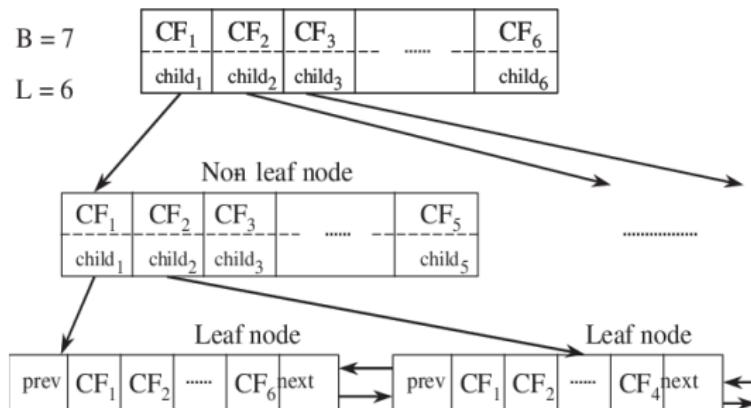
- **BIRCH** : aggiusta in modo incrementale la qualita' dei sottocluster.
  - BIRCH\*: Estende BIRCH per very large dataset
- **ROCK**: clustering di dati categoriali utilizzando la distanza minima tra cluster
- **CURE**: minimizzare la distanza media dei punti dal proprio centroide
- **Chameleon**: modellazione dinamica per determinare la similarità tra coppie di cluster.

# Clustering gerarchico: BIRCH

BIRCH: **Balanced Iterative Reducing and Clustering using Hierarchies** (Zhang, Ramakrishnan & Livny, SIGMOD'96).

E' basato su due concetti:

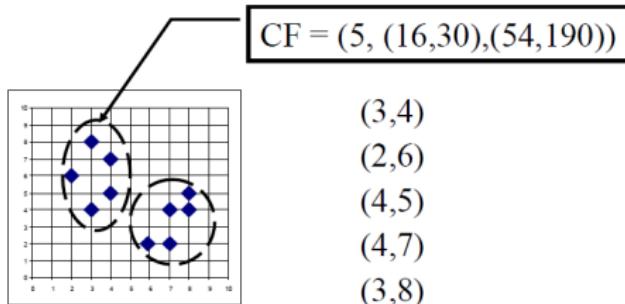
- **Cluster Feature (CF)**
- **Cluster Feature Tree (CF-tree)**



# BIRCH: cluster feature

Una **cluster feature** è una tupla  $CF = (N, \vec{LS}, \vec{SS})$ , dove

- $N$  è il numero di punti del cluster
- $LS = \sum_{i=1}^N \vec{X}_i$
- $SS = \sum_{i=1}^N \vec{X}_i^2$



Il CF serve a rappresentare il cluster (e/o i sottocluster) con delle statistiche e permette di utilizzare un sistema di immagazzinamento efficiente.

Inoltre definiamo anche  $\mu_c = LS/N$  ed il **raggio** =  $\sqrt{(SS/N - (LS/N)^2)}$ .

Dato un cluster  $C$  composto da  $N$  elementi  $[\vec{X}_1, \dots, \vec{X}_n]$  definiamo:

- **centroide** (elemento medio):  $\vec{X}_0 = \frac{\sum_{i=1}^N \vec{X}_i}{N}$
- **raggio** (distanza media dei punti entro un cluster rispetto al centroide che lo rappresenta):  $R = \sqrt{\frac{\sum_{i=1}^N (\vec{X}_i - \vec{X}_0)^2}{N}} = \sqrt{\left(\frac{SS}{N} - \frac{LS}{N}\right)^2}$
- **diametro** (distanza media tra due punti di un cluster):

$$D = \sqrt{\frac{\sum_{i=1}^N \sum_{j=1, j \neq i}^N (\vec{X}_i - \vec{X}_j)^2}{N(N-1)}}$$

## Teorema di additività

Consideriamo due cluster  $C_1$  e  $C_2$  e le clustering features a loro associate  $CF_{C_1}$ ,  $CF_{C_2}$ , è possibile rappresentare la clustering feature del cluster  $C_3$  nato dalla fusione di  $C_1$  e  $C_2$  come :

$$CF_{C_3} = \langle n_{C_1} + n_{C_2}, LS_{C_1} + LS_{C_2}, SS_{C_1} + SS_{C_2} \rangle$$

Un **CF-tree** è un **albero bilanciato** costruito tenendo in considerazione due parametri

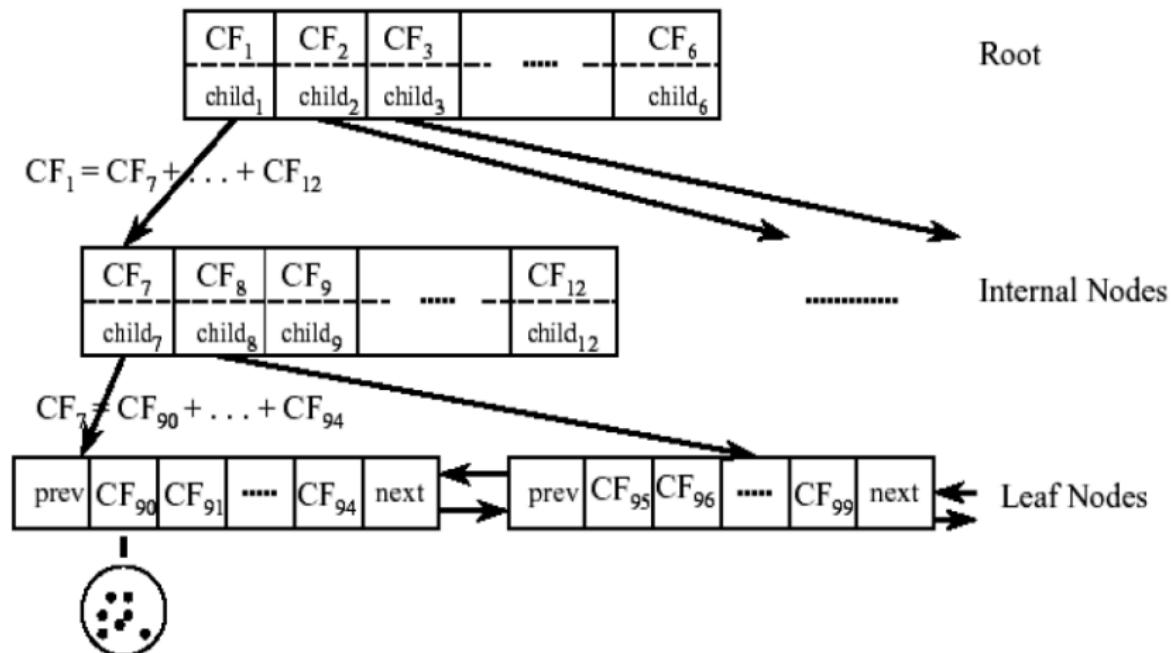
- **branching factor B**: numero massimo di elementi in un nodo non foglia
- **threshold T**: il diametro massimo dei sottocluster immagazzinati nei nodi foglia

tale che

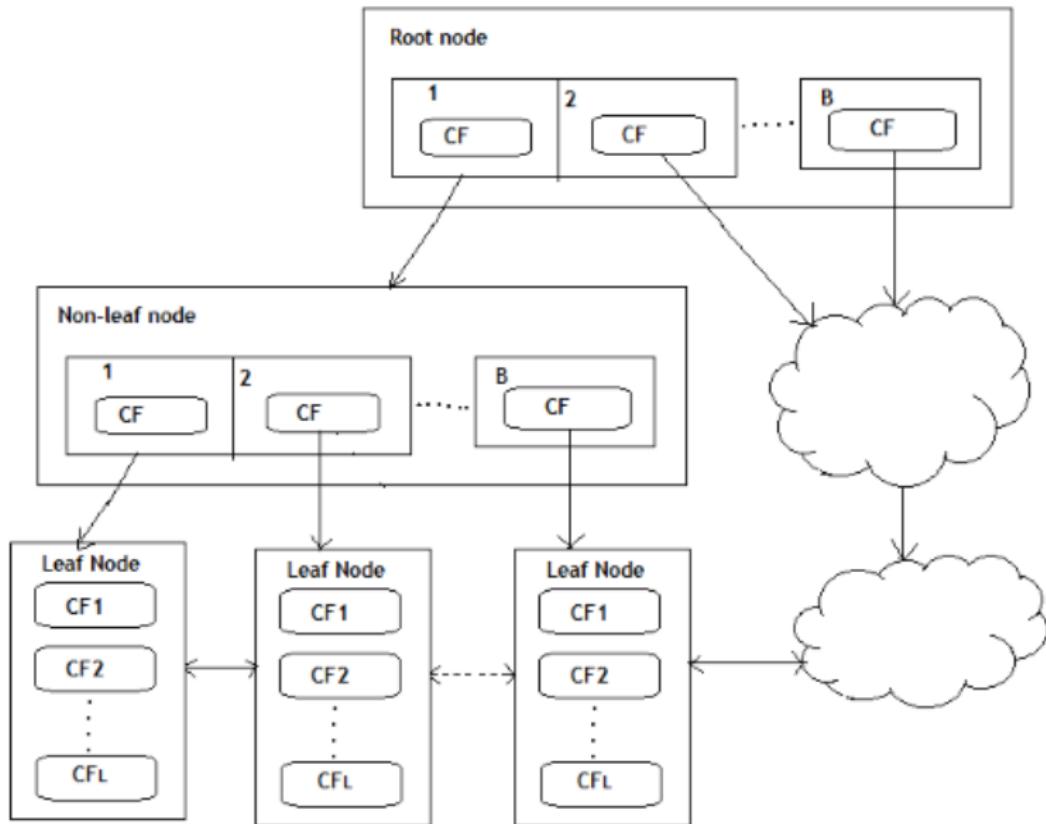
- il contenuto di un **nodo non foglia** è del tipo  $[CF_i, child_i]$ , dove
  - $child_i$  è un puntatore al  $i$ -esimo figlio del nodo in considerazione
  - $CF_i$  è la cluster feature relativa al sottocluster rappresentata da tale figlio
- il contenuto di un **nodo foglia** è nella forma  $[CF_j, \dots, CF_{j+L}]$ . Inoltre per ogni nodo foglia manteniamo i puntatori al nodo foglia che precede ( $prev$ ) e che succede ( $next$ ) il nodo

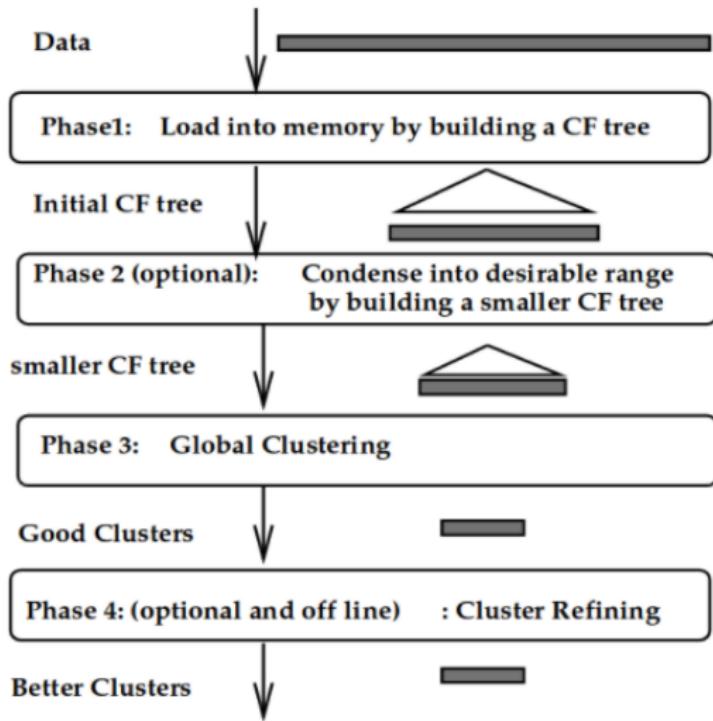
# BIRCH: CF-tree

$B = 7, L = 5$



# BIRCH: CF-tree

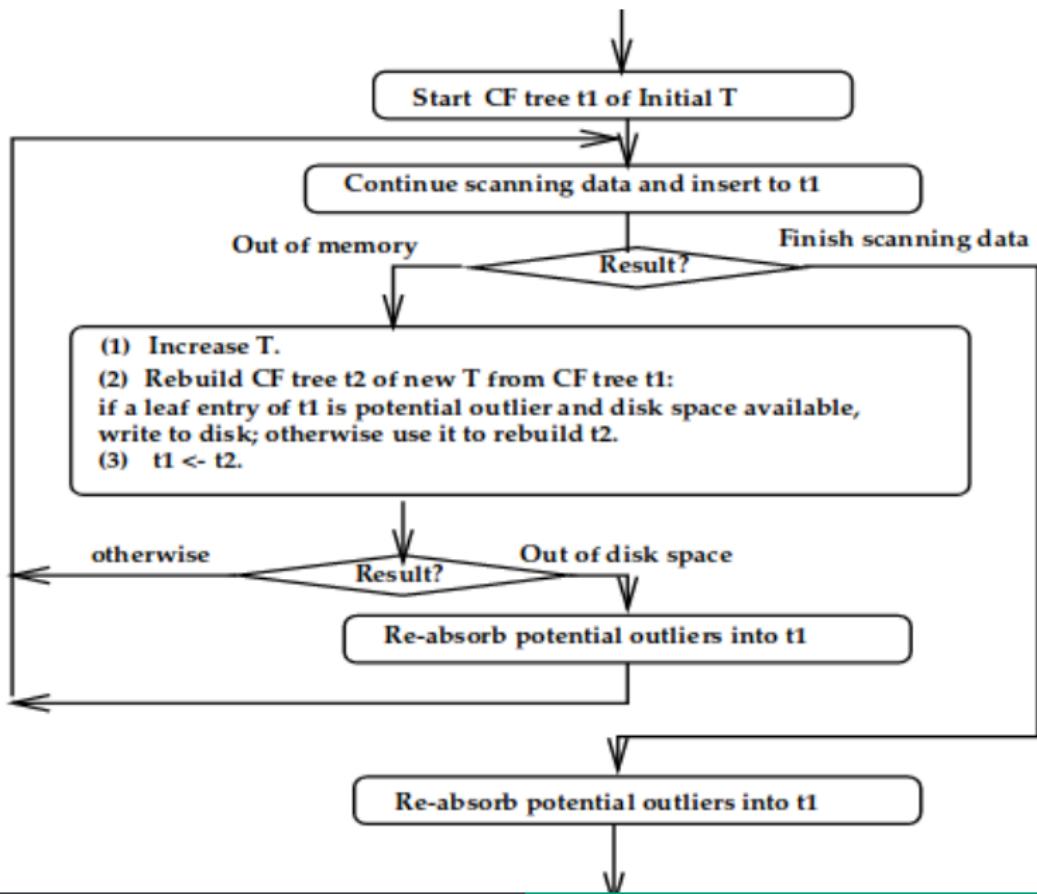




In modo incrementale costruisce un CF-tree, una struttura gerarchica per fare un clustering a piu' fasi (multiphase clustering):

**Fase 1:** scan del DS per costruire un initial in-memory CF-tree (una compressione dei dati a più livelli che prova a preservare la struttura a cluster dei dati).

**Fase 2:** utilizza un algoritmo di clustering per clusterizzare le foglie del CF-tree.



Inserimento di un Elemento nel CF Tree::

- ① Identifichiamo la **foglia appropriata** tramite una misura di **closeness**.
- ② Modifichiamo la foglia
  - se  $D < T$  allora la foglia può assorbire il punto, quindi inseriamo l'elemento e aggiorniamo le CF
  - altrimenti: creazione nuova entry e gestione splitting nodo (si scelgono i due elementi nell'entry più lontani e gli altri elementi vengono assegnati ad uno o all'altro in base alla loro vicinanza)
- ③ Modifichiamo il path per raggiungere la foglia
  - aggiornamento dei CF
  - gestione dei nodi parent fino alla root per possibili divisioni degli stessi

## Rifinitura:

- Entry nelle foglie del CF tree con pochi elementi possono essere considerati degli **outliers** ed eliminati dal tree.
- la threshold e il branching factor possono portare a lla **divisione di cluster reali**, cioe' ad un risultato che poco riflette come i dati realmente dovrebbero essere clusterizzati
  - (**fase 2**) Un algoritmo agglomerativo di cluster viene applicato ai subclusters delle foglie, considerando ognuno di essi come un singolo punto con valore il suo CF.

## Vantaggi:

scala linearmente: trova un buon clustering con una sola lettura dei dati e migliora la qualita' con altre letture (scan) dei dati.

## Svantaggi:

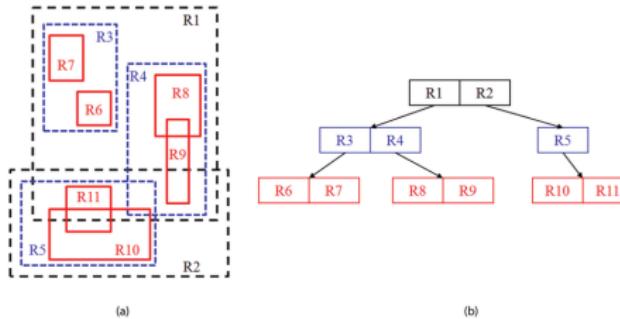
manipola solo dati numerici, è sensibile all'ordine con cui vengono letti i dati, trova SOLO cluster di forma sferica in quanto si basa sul raggio.

O anche GRGPF (Ganti, Ramakrishnan, Gehrke, Powell, French).

E' definita una **metrica**  $D$ , non necessariamente Euclidea.

I dati non entrano nella **main memory**.

I cluster sono immagazzinati in una struttura dati tipo R-tree detto **CF\*-tree**.



I **nodi** puntano a blocchi del disco e si conservano dati differenti nelle foglie e nei nodi interni.

Nei **nodi foglia** mettiamo le **caratteristiche** dei cluster come in CF-tree ma non essendo la metrica euclidea le caratteristiche sono differenti:

- ① **Cardinalità  $N$**  del cluster
- ② Il **clusteroido**  $\hat{C}$  (il punto del cluster che minimizza la **rowsum** cioè la somma dei quadrati delle distanze dagli altri punti del cluster.) e la sua **rowsum**
  - la **rowsum** di  $\hat{C}$  è  $\sum_{x \in C} (D(\hat{C}, x))^2$
  - La **rowsum** è l'analogo di **SUMSQ** di BFR. Può essere usato per calcolare il raggio del cluster analogo alla deviazione standard di BFR. La formula è  $\text{raggio} = \sqrt{\text{rowsum}/N}$ .
- ③ I  $p$  punti ( $p$  scelto a piacere) del cluster più vicini a  $\hat{C}$  e le loro  $p$  **rowsum** (somma dei quadrati delle distanze di ognuno di questi  $p$  punti da tutti i punti del cluster)
- ④ I  $p$  punti del cluster più lontani da  $\hat{C}$  e le loro  $p$  **rowsum**.

Nei **nodi interni** mettiamo

- i clusteroidi dei cluster nei nodi ad essi discendenti
- la sua rowsum e
- il numero di nodi (in modo da calcolare il raggio).

---

Quando si deve **inserire** un punto in qualche cluster, si inizia dalla radice e si discende l'albero solo in quei cammini lungo i quali si trovano cluster ragionevolmente vicini basandosi sui clusteroidi associati ai nodi.

Vi è una fase di **inizializzazione** sulla main memory piena di dati, dove si opera la prima stima dei cluster ed essi sono sistemati in una gerarchia corrispondenti al CF\*-tree.

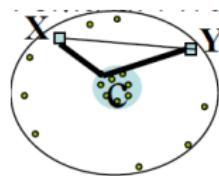
Successivamente esaminiamo i **rimanenti punti** e cerchiamo di inserirli nei cluster esistenti.

Questi possono essere riconsiderati se sono troppi per la RAM oppure se un cluster diventa troppo grosso (raggio troppo grande).

Se si vuole aggiungere un nuovo punto  $X$  al cluster  $C$  tale che  $D(\hat{C}, X)$  è minimo, allora si usano i campioni di clusteroidi nei punti interni per evitare di traversare tutto l'albero (cioè per guidare la ricerca e l'inserimento).

Se  $X$  è aggiunto a  $C$  allora aggiungiamo a ciascuna delle  $2p + 1$  rowsum (rowsum di  $\hat{C}$ , dei  $p$  punti più vicini e di quelli più lontani) il quadrato della distanza di  $X$  da ciascuno di quei  $2p+1$  punti.

Stimiamo anche la rowsum di  $X$  come  $N(r^2 + D(\hat{C}, X)^2)$ , dove  $r$  è il raggio. Questo deriva dall'analogia situazione Euclidea in cui il segmento  $XY$ ,  $Y$  generico punto del cluster, è decomposto in  $X\hat{C}e Y\hat{C}$  questi sono "perpendicolari" per uno dei fenomeni dell'alta dimensionalità. Basta quindi applicare il Teorema di Pitagora ( $r$  stima  $D(Y, \hat{C})$  ).



Dobbiamo considerare la possibilità che dopo aver aggiunto  $X$ , il cluster  $C$  ha un diverso clustroide che potrebbe essere  $X$  o uno dei  $p$  punti più vicini a  $\hat{C}$ . Se uno di questi punti ha rowsum più piccola di  $\hat{C}$  allora diventa clusteroide.

Ovviamente ciò non può essere fatto indefinitamente poiché alla lunga il clusteroide migrerà fuori dai  $p$  punti più vicini.

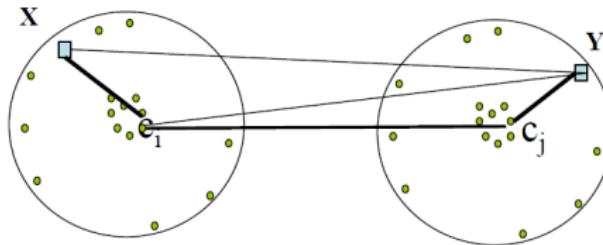
Pertanto potrebbe richiedersi una ricomputazione periodica delle caratteristiche di un cluster ,con relative I/O su disco dove ci sono tutti i punti.

Se il raggio del clusteroido eccede una certa soglia allora l'algoritmo decide di spezzare il cluster in due.

Ciò richiede di ricaricare tutto il cluster in main memory ed eseguire un algoritmo di spezzamento . Inoltre il numero di foglie nel CF\*-tree aumenta di un'unità e se il nodo (blocco di disco) va in overflow esso deve spezzarsi come in un B-tree. Questo può causare altri spezzamenti e nel caso peggiore l'albero può non stare in più in main memory. In questo caso conviene aumentare la soglia e provare a fondere cluster attraverso l'albero.

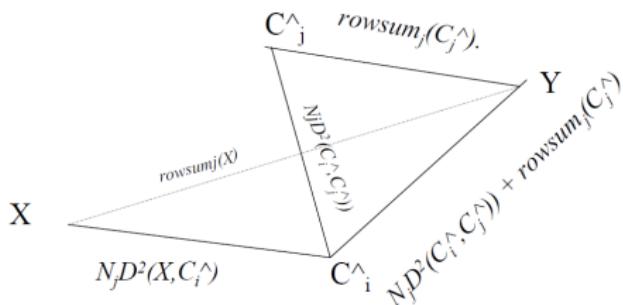
Supponiamo di voler fondere  $C_i$ ,  $C_j$ . Volendo evitare di caricare tutto sulla main memory possiamo sfruttare clustroidi e rowsum come segue:

- ➊ Assumiamo che il clustroide del cluster risultante sia uno dei punti più lontani da  $\hat{C}_i$  oppure  $\hat{C}_j$
- ➋ Per decidere chi è il nuovo clustroide dobbiamo stimare la rowsum per ogni punto  $X$  in  $C_i \cup C_j$ . Per esempio se  $X \in C_i$  allora stimiamo  $\text{rowsum}(X) = \text{rowsum}_i(X) + N_j(D^2(X, \hat{C}_i) + D^2(\hat{C}_i, \hat{C}_j)) + \text{rowsum}_j(\hat{C}_j)$ , dove  $N_j$  è la cardinalità di  $C_j$



## BIRCH\*: fondere cluster

$$\text{rowsum}(X) = \text{rowsum}_i(X) + N_j(D^2(X, \hat{C}_i) + D^2(\hat{C}_i, \hat{C}_j)) + \text{rowsum}_j(\hat{C}_j)$$



Il primo addendo tiene in conto le distanze di  $X$  dagli elementi del suo cluster originale  $C_i$ .

Il secondo riguarda le distanze di  $X$  dagli elementi  $Y$  di  $C_j$ . Nel senso che si considera il triangolo  $XC_iY$  ed applica il Teorema di Pitagora al triangolo  $XC_jY$  perché per l'alta dimensionalità esso è rettangolo. Tuttavia per stimare  $D^2(C_i, Y)$  si considera il triangolo  $C_iC_jY$  e si applica il Teorema di Pitagora dove i segmenti  $C_jY$  vengono stimati da  $\text{rowsum}(\hat{C}_j)$ .

- ③ Una volta deciso il clusteroido del nuovo cluster fusione di  $C_i$  e  $C_j$ , calcola la rowsum di tutti i punti che manteniamo nelle caratteristiche del cluster con la stessa stima espressa in (2).

Un metodo comune ma poco flessibile utilizza un valore di interruzione dell'altezza costante; questo metodo mostra prestazioni non ottimali su dendrogrammi complicati.

DynamicTreeCut (DTC) implementa nuovi metodi dinamici di taglio dei rami per rilevare i cluster in un dendrogramma a seconda della loro forma. Rispetto al metodo del taglio ad altezza costante, le tecniche DTC offrono i seguenti vantaggi:

- ① sono in grado di identificare cluster nidificati;
- ② sono flessibili: i parametri della forma del cluster possono essere regolati per adattarsi all'applicazione in questione;
- ③ sono adatti all'automazione
- ④ possono facoltativamente combinare i vantaggi del clustering gerarchico e del partizionamento attorno ai medoidi, fornendo un migliore rilevamento degli outlier.

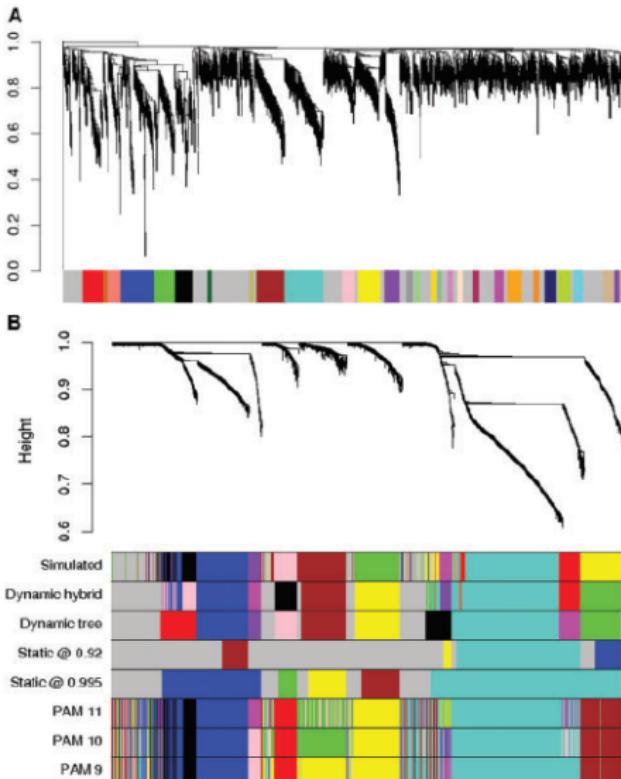
# Dynamic tree cut: taglio adattativo del dendogramma

(A) Cluster gerarchico di collegamento medio utilizzando la matrice di sovrapposizione topologica (Yip e Horvath, 2007) e il taglio dell'albero dinamico applicato alla rete di interazione proteina-proteina della Drosophila (dati PPI da BioGRID).

L'assegnazione dei moduli è rappresentata dalla riga di colore immediatamente sotto il dendrogramma, con il grigio che rappresenta le proteine non assegnate. Un'analisi di arricchimento funzionale ha dimostrato che i cluster sono significativamente arricchiti con ontologie genetiche note (Dong e Horvath, 2007).

Si noti che un limite di altezza fisso non sarebbe in grado di identificare molti dei cluster mostrati.

(B) Albero gerarchico dei cluster e vari metodi di rilevamento dei cluster applicati a un set di dati di espressione genica simulato.



## Dynamic tree cut: taglio adattativo del dendogramma

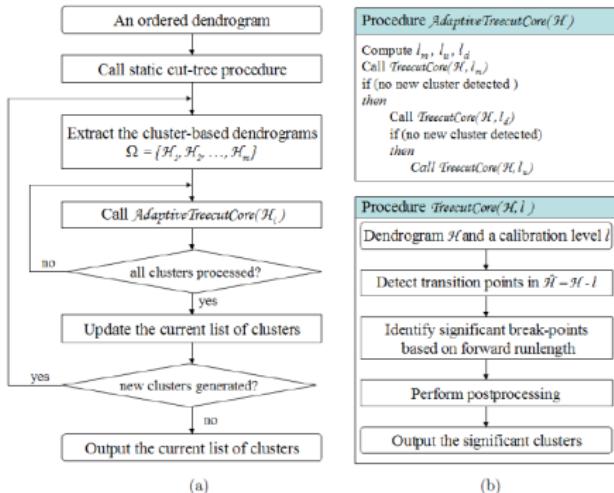
L'algoritmo implementa un processo adattivo e iterativo di scomposizione e combinazione dei cluster e si arresta quando il numero di cluster diventa stabile.

Si inizia ottenendo pochi grandi grappoli dal taglio statico dell'albero.

Le altezze di unione di ciascun cluster vengono analizzate attraverso le fluttuazioni delle altezze dei rami del dendogramma per identificare una struttura di sottocluster.

I cluster che presentano questo modello vengono suddivisi ricorsivamente.

Per evitare una suddivisione eccessiva, i cluster molto piccoli vengono uniti ai cluster più vicini.



Una variante, chiamata taglio "Dynamic Hybrid", è un algoritmo dal basso verso l'alto che migliora il rilevamento dei membri periferici di ciascun cluster.

La rilevazione procede in due fasi.

- ① Innanzitutto, il metodo identifica i cluster preliminari come rami che soddisfano i seguenti criteri:
  - contengono un certo numero minimo di oggetti;
  - gli oggetti troppo lontani da un cluster ne vengono esclusi anche se appartengono allo stesso ramo del dendogramma;
  - ogni cluster deve essere distinto da ciò che lo circonda
  - il nucleo di ciascun cluster, definito come la punta del ramo, dovrebbe essere strettamente connesso.
- ② tutti gli oggetti precedentemente non assegnati vengono testati per verificarne la sufficiente vicinanza ai cluster preliminari; se il cluster più vicino è abbastanza vicino, l'oggetto viene assegnato a quel cluster

**ROCK**: RObust Clustering using linKs (S. Guha, R. Rastogi & K. Shim, ICDE'99)

**Problema**: cluster di dati categoriali.

Metodi tradizionali per misurare dati categoriali possono non lavorare bene, e.g., Jaccard coefficient.



# ROCK: tre fasi

## Campione casuale

il campionamento viene utilizzato per garantire la scalabilità a set di dati molto grandi.

Il campione iniziale viene utilizzato per formare i cluster, quindi i dati rimanenti su disco vengono assegnati a questi cluster.

## Clustering

Esegue un algoritmo di clustering agglomerativo gerarchico con una particolare definizione di similarità (tramite link).

## Etichetta i dati su disco

Infine, i punti dati rimanenti nel disco vengono assegnati ai cluster generati.

Questo viene fatto selezionando un campione casuale  $L_i$  da ciascun cluster  $C_i$ , quindi assegniamo ogni punto  $p$  al cluster per il quale ha il legame (link) più forte con  $L_i$ .

**Vicinanza:** Se la somiglianza (tipicamente coefficiente di Jaccard) tra due punti supera una certa soglia di somiglianza  $\theta$ , sono considerati vicini.

Il **Link** per una coppia di punti è: il numero dei loro vicini comuni.

Ovviamente, Link incorpora informazioni globali sugli altri punti nelle vicinanze dei due punti. Più grande è il Link, maggiore è la probabilità che questa coppia di punti si trovi negli stessi cluster.

Si vuole massimizzare il seguente **criterio**:

$$E_I = \sum_{i=1}^k n_i \times \sum_{p_q, p_r \in C_i} \frac{\text{link}(p_q, p_r)}{n_i^{1+2f(\theta)}}$$

dove  $C_i$  è il cluster in questione,  $n_i$  è la cardinalità del cluster,  $k$  è il numero finale di cluster che si vuole ottenere. Tipicamente  $f(\theta)$  viene posta uguale a  $(1 - \theta)/(1 + \theta)$ .

Lo scopo è quello di massimizzare la somma dei link tra coppie di punti appartenenti allo stesso cluster, e allo stesso tempo minimizzare la somma dei link tra coppie di punti appartenenti a cluster diversi.

Utilizziamo la seguente **misura di goodness** (bontà) per massimizzare la funzione criterio.

$$g(C_i, C_j) = \frac{\text{link}[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

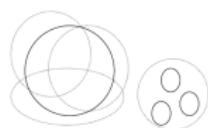
Tale misura viene utilizzata per trovare la migliore coppia di cluster da fondere durante la fase di clustering agglomerativo.

**CURE** Clustering Using REpresentatives (Guha, Rastogi & Shim, '98).

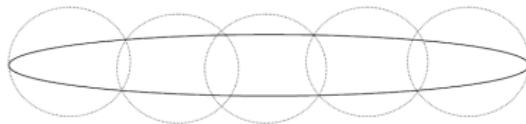


risolve il clustering in spazi Euclidei quando i cluster non hanno una distribuzione Gaussiana attorno ad un centroide (come in BFR). In questi casi il k-means può andare male.

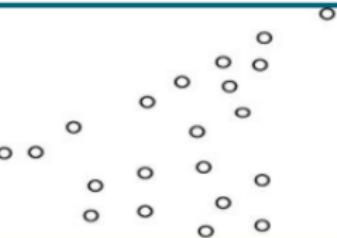
Esempio 1: Supponiamo di utilizzare un algoritmo di cluster con la distanza tra centroidi. Si vede che i cluster sono 4 (linee continue) ma non sono quelli giusti (output dell'algoritmo sono i cerchi tratteggiati) perché si è badato solo alla distanza tra centroidi.



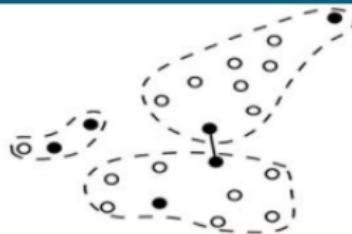
Esempio 2: Accade perché si vuole minimizzare la distanza media dei punti dal proprio centroide. Tuttavia ciò non accade se si usa il raggio di Mahalanobis come in BFR, perché viene visto come “circolare”.



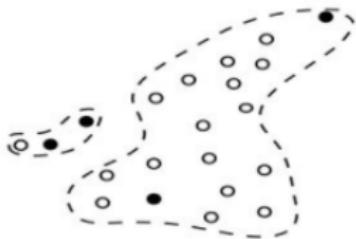
- ① Inizia con la main memory piena di punti random. Clusterizzali utilizzando un approccio gerarchico (che evita i problemi sopra detti).
- ② Per ogni cluster scegli un campione di  $c$  punti per qualche costante  $c$ . Questi punti sono scelti più distante possibile l'uno dall'altro, dopo di che essi si spostano leggermente nella direzione del centroide come segue:
  - Scegli il primo come il più lontano dal centroide.
  - Ripetutamente scegli il punto del cluster che massimizzi la minima distanza da quelli già scelti.
  - Quando si sono completati i  $c$  punti allora spostali tutti leggermente (20% ) verso il centroide.(questi in generale non saranno punti dell'insieme ma ciò non è un problema)
- ③ Partiziona l'insieme secondo tutti questi  $c$  punti.



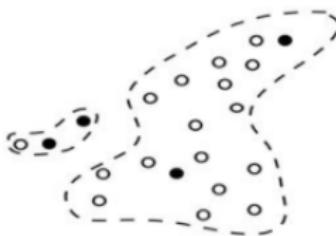
a) Sample of Data



b) Three Clusters with Representative Points



c) Merge Clusters with Closest Points



d) Shrink Representative Points

# Chameleon

**Chameleon** è un algoritmo di clustering **gerarchico** che utilizza la **modellazione dinamica** per determinare la **similarità tra coppie di cluster**.

E' stato sviluppato per gestire alcune debolezze osservate in ROCK e CURE.

- ROCK enfatizza l'**interconnettività** dei cluster ignorando le informazioni relative alla vicinanza dei cluster.
- CURE considera la **prossimità** dei cluster ma ignorano l'interconnettività dei cluster.



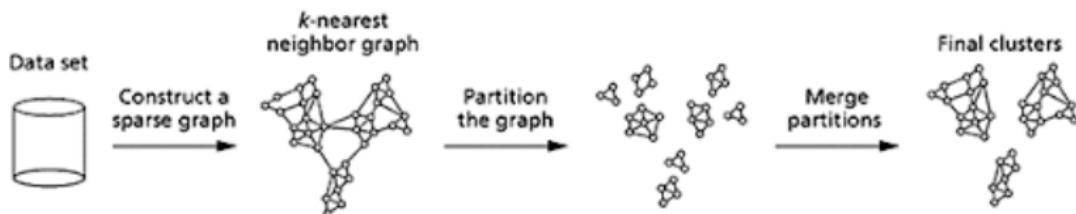
In Chameleon, la **somiglianza dei cluster** viene valutata in base a come gli **oggetti** sono ben collegati all'interno di un cluster e alla vicinanza dei **cluster**.

Due cluster vengono **uniti** se la loro **interconnettività** è elevata e sono **vicini** tra loro.

Pertanto, Chameleon non dipende da un modello statico fornito dall'utente e può **adattarsi automaticamente alle caratteristiche interne dei cluster che vengono fusi**.

Il processo di fusione facilita la **scoperta di cluster naturali e omogenei** e si applica a **tutti i tipi di dati** purché sia possibile specificare una funzione di somiglianza.

# Chameleon: kNN graph



Utilizza un approccio basato su un **k-nearest-neighbor graph**.

Un grafo sparso, in cui ogni **vertice** del grafico rappresenta un oggetto, ed esiste un **arco** tra due vertici se un oggetto è tra gli oggetti  $k$  più simili dell'altro. Gli archi sono ponderati per riflettere la somiglianza tra gli oggetti.

Chameleon utilizza un algoritmo di **partizionamento** del grafo per partizionare il grafo in un gran numero di sottogruppi relativamente piccoli. Quindi utilizza un algoritmo di **clustering gerarchico agglomerativo** che unisce ripetutamente i sottogruppi in base alla loro somiglianza. Per determinare le coppie di sottogruppi più simili, si tiene conto sia dell'**interconnettività** che della **vicinanza** dei cluster.

Si noti che il grafo cattura il concetto di vicinato in **modo dinamico**: il **raggio di vicinato** di un oggetto è determinato dalla **densità** della regione in cui risiede l'oggetto.

Ciò tende a produrre **cluster più naturali**, rispetto ai metodi basati sulla densità come DBSCAN che utilizzano invece un vicinato globale.

Inoltre, la **densità** della regione viene registrata come **peso dei archi** appartenenti a tale regione.

Un cluster  $C$  è **partizionato** in due sottogruppi  $C_i$  e  $C_j$  in modo da **minimizzare** il peso degli archi che verrebbero tagliati dividendo  $C$  in due nei due sottogruppi  $C_i$  e  $C_j$ .

Il peso totale del taglio del partizionamento è denotato come  $EC_{\{C_i, C_j\}}$  (**Edge Cut**) e rappresenta l'**interconnettività assoluta** tra i due sottocluster  $C_i$  e  $C_j$ .

Chameleon determina la somiglianza tra ciascuna coppia di cluster  $C_i$  e  $C_j$  in base alla loro **interconnettività relativa**,  $RI(C_i, C_j)$ , e alla loro **vicinanza relativa**,  $RC(C_i, C_j)$ .

L'**interconnettività relativa**,  $RI(C_i, C_j)$ , tra due cluster,  $C_i$  e  $C_j$ , è definita come l'interconnettività assoluta tra  $C_i$  e  $C_j$ , normalizzata rispetto all'interconnettività interna dei due cluster,  $C_i$  e  $C_j$ .

$$RI(C_i, C_j) = \frac{|EC_{\{C_i, C_j\}}|}{\frac{1}{2}(|EC_{C_i} + EC_{C_j}|)}$$

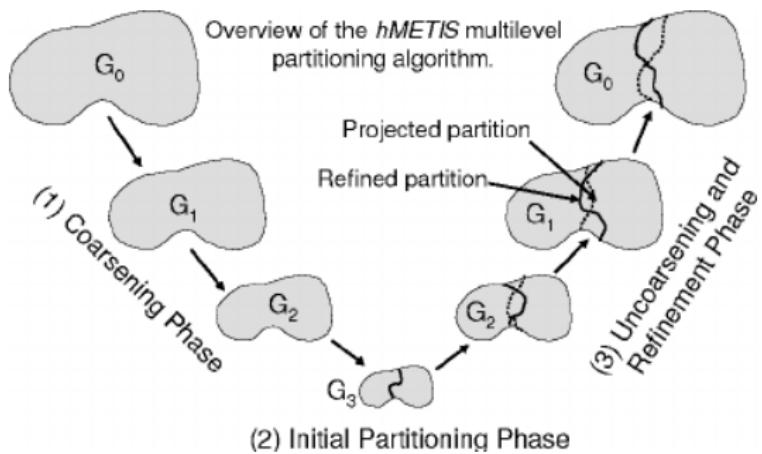
dove  $EC_{C_i, C_j}$  è l'interconnettività assoluta per il cluster formato dall'unione di  $C_i$  e  $C_j$ .  $EC_{C_i}$  (e similmente  $EC_{C_j}$ ) è la somma minima degli archi tagliati dal partizionamento che dividono  $C_i$  (o  $C_j$ ) in due parti approssimativamente uguali.

La **vicinanza relativa**,  $RC(C_i, C_j)$ , tra una coppia di cluster,  $C_i$  e  $C_j$ , è la vicinanza assoluta tra  $C_i$  e  $C_j$ , normalizzata rispetto alla vicinanza interna dei due cluster,  $C_i$  e  $C_j$ .

$$RC(C_i, C_j) = \frac{S_{EC_{\{C_i, C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|} S_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|} S_{EC_{C_j}}}$$

dove  $S_{EC_{\{C_i, C_j\}}}$  è il peso medio degli archi che connettono i vertici di  $C_i$  ai vertici di  $C_j$ ;  $S_{EC_{C_i}}$  è il peso medio degli archi che appartengono alla bisettrice di taglio minimo del cluster  $C_i$ .

Chameleon utilizza l'algoritmo **hMETIS** per definire il partizionamento iniziale dei cluster da cui far partire l'lagoritmo agglomerativo.



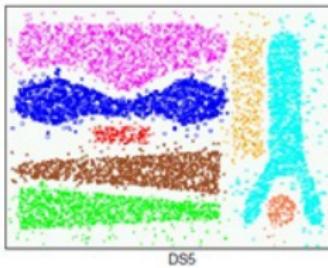
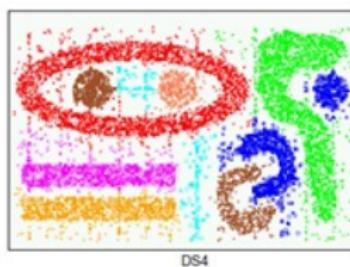
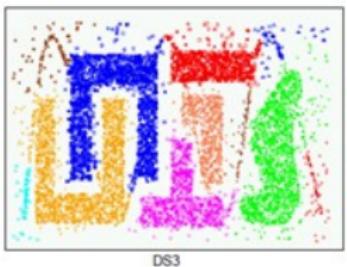
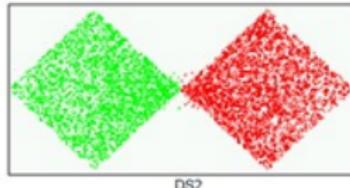
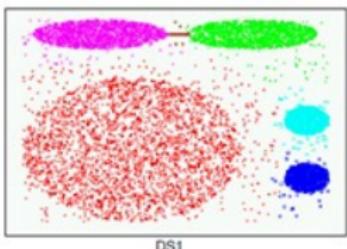
Il taglio (edge-cut) tra due cluster viene minimizzato ed ognuno dei sottocluster ottenuti contiene il 25% dei nodi del cluster iniziale.

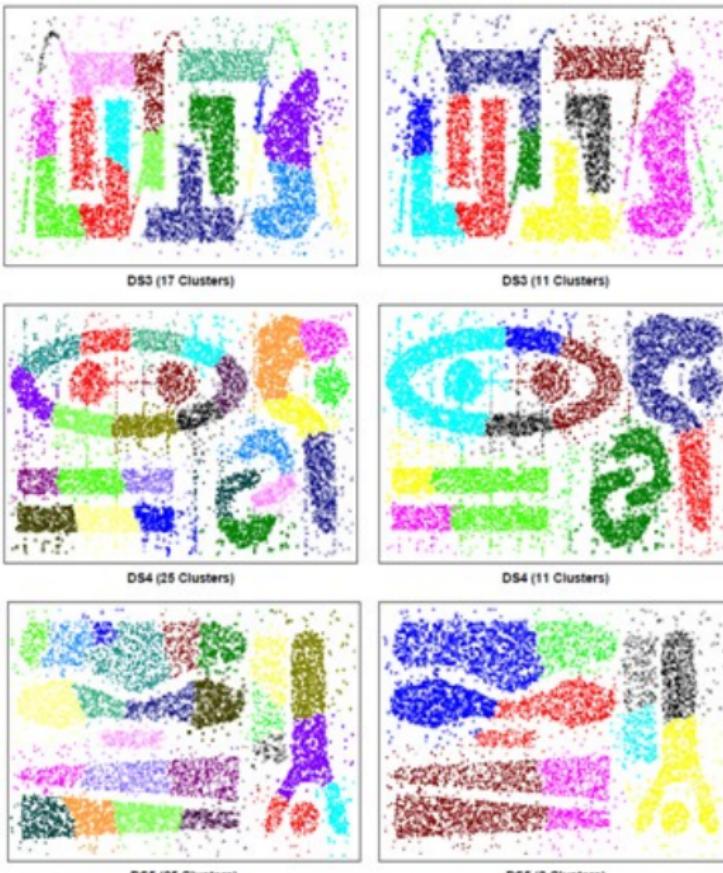
Si inizia dai singoli punti.

Si seleziona il più grande sottocluster tra quelli correnti e si usa hMETIS per bisezionarlo.

Si termina quando il più grande sottocluster contiene un numero di nodi inferiore ad una soglia data (sufficientemente grande, tra l'1% e il 5% dei nodi totali).

Chameleon ha dimostrato di avere un potere maggiore nello scoprire **cluster di forma arbitraria di alta qualità** rispetto a diversi algoritmi ben noti come BIRCH e DBSCAN basato sulla densità. Tuttavia, il costo di elaborazione per dati ad alta dimensionalità può richiedere  $O(n^2)$  nel caso peggiore.





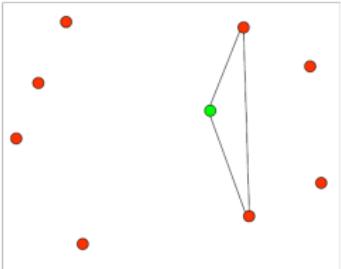
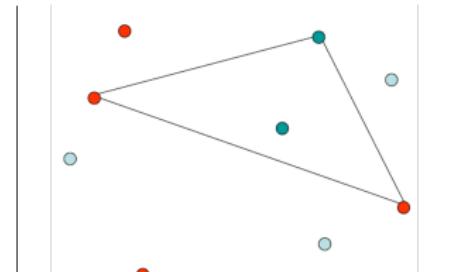
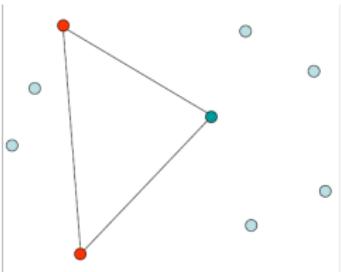
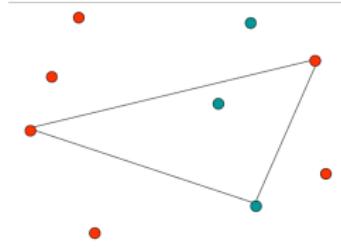
La 1-mediana  $c$  (centroide) di un sottoinsieme  $S$  di uno spazio metrico è quell'elemento che minimizza la somma delle distanze dagli altri elementi di  $S$ . Cioè  $c$  in  $S$  minimizza la funzione  $w(c) = \sum_{s \in S} d(s, c)$ .

Un modo efficiente per calcolarla in modo approssimato è:

- ① Dividi  $S$  in triple random
- ② Per ogni tripla  $T$  seleziona la 1-mediana di  $T$  e butta via gli altri due punti
- ③ Continua ricorsivamente suddividendo in triple e selezionando le 1-mediane
- ④ La 1-mediana finale è l'output

# Algoritmi randomizzati: 1-mediana (centroide)

Rosso gli eliminati, verde il finale.



- ① Ad ogni round, gli elementi che passano il turno precedente sono partizionati in modo casuale in sottoinsiemi,  $X_1, \dots, X_n$
- ② ogni sottoinsieme  $X_i$  è processato localmente tramite una procedura che calcola la 1-mediane  $x_i$
- ③ gli elementi  $x_1, \dots, x_k$  vanno alla fase successiva
- ④ l'algoritmo viene applicato ricorsivamente
- ⑤ l'algoritmo termina quando rimangono solo due elementi. Tali elementi sono scelti per approssimare il diametro di  $S$

## Teorema

Dato un insieme  $S$  di cardinalità  $n = 3^r$ , l'algoritmo esegue esattamente  $\frac{3}{2}(n - 1)$  distanze.

Per induzione su  $r$ . Per  $r = 1$  è banale infatti si eseguono 3 sole distanze.

Supponiamolo vero per  $n$  e sia  $n' = 3n$ . Allora si hanno  $n$  tornei di cardinalità 3 al primo round seguiti da un torneo di cardinalità  $n$ . Per l'ipotesi induttiva si hanno

$$3n + [(3/2)(n - 1)] = (6n + 3n - 3)/2 = 3(3n - 1)/2 = (3/2)(n' - 1)$$

□

# Antilope clustering

Sia  $S$  un insieme di oggetti di uno spazio metrico  $(X, d)$  e sia  $\sigma$  un numero reale positivo.

Un  **$\sigma$ -antipole** per  $S$  è una coppia  $(a, b)$  di elementi di  $S$  tale che  $d(a, b) > \sigma$ .



Due punti  $a, b$  di  $S$  tale che  $d(a, b)$  è **massimale** si dicono **poli** e la loro distanza si chiama **diametro**.

## Algoritmo fase 1:

- ① Dividi  $S$  in triple random.
- ② Per ogni tripla  $T$  seleziona i poli  $a$  e  $b$ .
  - Se il diametro  $d(a, b) > \sigma$  allora STOP e ritorna  $(a, b)$ .
  - Altrimenti  $a$  e  $b$  vanno al round successivo e butta via il terzo (che è la 1-medianà di  $T$ ).
- ③ Continua ricorsivamente suddividendo in triple, selezionando i poli ed eventualmente producendo in output il diametro  $> \sigma$ . (L'ultimo potrebbe avere 3,4 o 5 elementi)
- ④ ritorna  $FAIL$ .

### Teorema (complessità lineare)

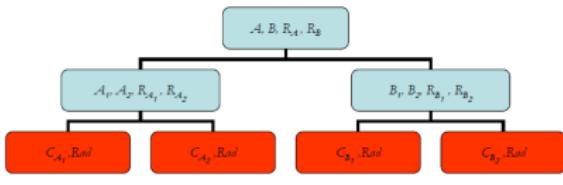
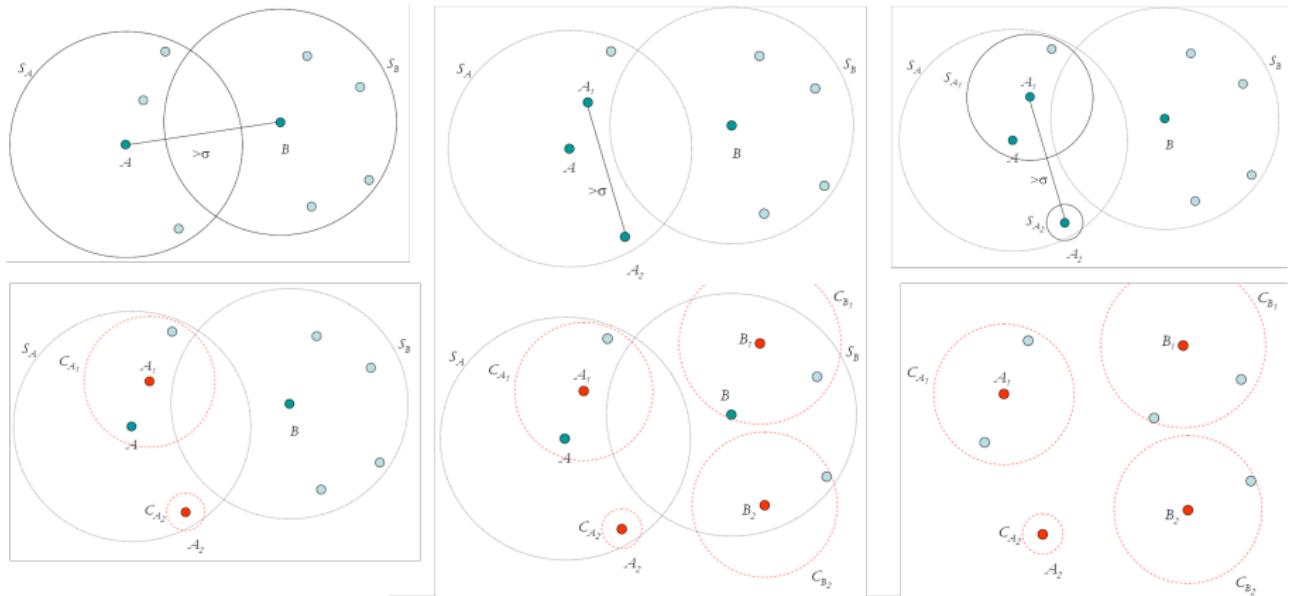
Dato un insieme  $S$  di cardinalità  $n$ , l'algoritmo esegue al più  $9n$  distanze.

Un clustering con diametro limitato di soglia  $\sigma$  è un clustering tale che il diametro di ogni cluster è  $< \sigma$ .  $\square$

Algoritmo fase 2:

- ① Se l'antipole è *FAIL* allora non spezzare e ritorna il cluster
- ② Altrimenti sia  $(a, b)$  l'antipole tale che  $d(a, b)$  è maggiore della soglia.  
Spezza in due sottoinsiemi secondo la maggiore vicinanza ad  $a$  o  $b$ .
- ③ Continua ricorsivamente sui due sottoinsiemi finchè si ottiene l'insieme finale di cluster.

# Antilope clustering



Dati un oggetto  $q$ , un database  $D$ , ed una soglia  $t$ , la **Range Query** ( $q, t$ ) chiede di trovare tutti gli oggetti di  $D$  che distano meno di  $t$  da  $q$ .

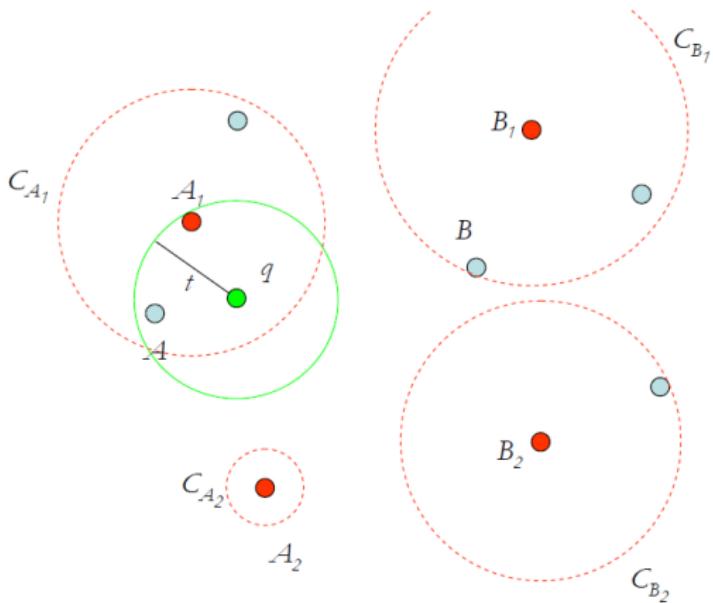
Osserviamo che se  $\hat{C}$  è un centroide per il cluster  $C$  con raggio  $r$  ( $\max d(x, \hat{C})$  in  $C$ ) allora se  $d(q, \hat{C}) > t + r$  tutto  $C$  si può scartare.  
Similmente se  $d(q, \hat{C}) < t - r$  allora tutto  $C$  sta nella range query.

Dati un oggetto  $q$  e un database  $D$ , la  **$k$ -Nearest Neighbor Query**  $q$  chiede di trovare i  $k$  oggetti di  $D$  più vicini a  $q$ .

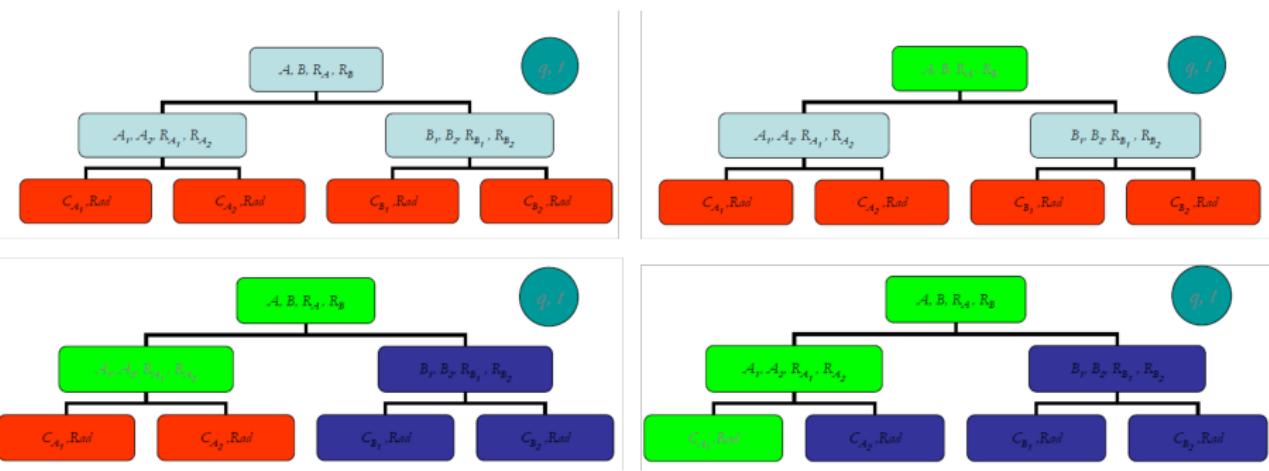
## Proprietà triangolare

- $d(q, C) \leq d(q, o') + d(o', C)$
- $d(q, o') \geq d(q, C) - d(o', C)$
- $d(o', C) \leq r$  quindi  $-d(o', C) \geq -r$
- $d(q, o') \geq d(q, C) - d(o', C) \geq d(q, C) - r > t$   
Possiamo scartare tutti gli elementi del cluster
- $d(o', q) \leq d(o', C) + d(C, q)$
- $d(o', C) \leq r$
- $d(o', q) \leq r + d(q, C) \leq t$   
Tutti gli elementi del cluster sono ad una distanza minore o uguale di  $t$  da  $q$
- $d(q, C) - r > t$  ne segue  $d(q, C) > t + r$
- $r + d(q, C) \leq t$  ne segue  $d(q, C) \leq t - r$
- $d(q, C) \leq t + r$  e  $d(q, C) > t - r$   
Ogni elemento del cluster deve essere esaminato per verificare se si trova ad una distanza minore di  $t$  da  $q$ .

# Proprietà triangolare



# Proprietà triangolare

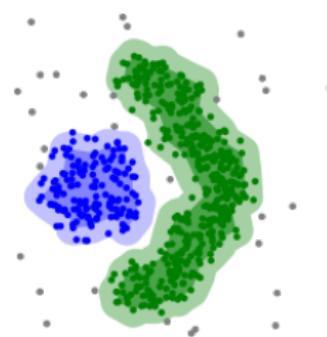


# Clustering basato sulla densità

Clustering basato sulla densità o density-based clustering (criterio di cluster locale), la densità di come sono connessi i punti.

Caratteristiche principali:

- Scoprire cluster di forma arbitraria
- Manipolare rumore
- 1 scan dei dati
- Ha bisogno di parametri di densità per definire la condizione di stop



DBSCAN: Ester, et al. (KDD'96)

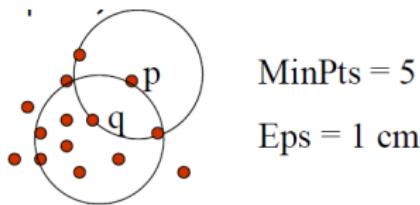
# Clustering basato sulla densità

Parametri:

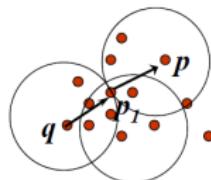
- $Eps$ : Raggio massimo del vicinato (intorno)
- $MinPts$ : Minimo numero di punti nel Eps-intorno  $N_{Eps}(q)$  di quel punto, con  $N_{Eps}(q) = \{p \in D : dist(p, q) \leq Eps\}$ .

Un punto  $p$  è direttamente raggiungibile per densità da un punto  $q$  rispetto a  $Eps, MinPts$  se

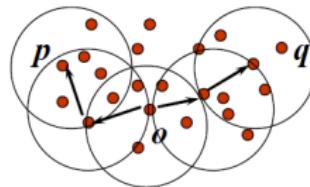
- $p$  appartiene a  $N_{Eps}(q)$
- Condizione di punto centrale (core point):  $|N_{Eps}(q)| \geq MinPts$



Un punto  $p$  è **raggiungibile per densità** da un punto  $q$  rispetto a  $Eps, MinPts$  se e solo se vi è una catena di punti  $p_1, \dots, p_n$ ,  $p_1 = q$ ,  $p_n = p$  tali che  $p_i + 1$  è direttamente raggiungibile per densità da  $p_i$ .



Un punto  $p$  è **connesso per densità** a un punto  $q$  rispetto a  $Eps, MinPts$  se vi è un punto  $o$  tale che sia  $p$  che  $q$  sono raggiungibili per densità da  $o$  rispetto a  $Eps$  e  $MinPts$ .

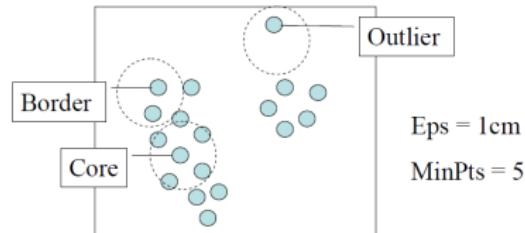


DBSCAN: Density Based Spatial Clustering (può manipolare rumore).

Un **cluster** è definito come un insieme massimale di punti connessi per densità:

Un cluster  $C$  in  $D$  rispetto a  $\epsilon$  ( $Eps$ ) e  $MinPts$  è un insieme non vuoto di punti di  $D$  tale che:

- (i)  $\forall p, q \in D$  se  $p \in C$  e  $q$  è raggiungibile per densità da  $p$  rispetto a  $\epsilon$  e  $MinPts$  allora  $q \in C$ . (**Massimalità**)
- (ii)  $\forall p, q \in C$ ,  $p$  è connesso per densità a  $q$ . (**Connettività**)



- ① Seleziona un punto  $p$  in modo arbitrario
- ② Se  $p$  è core point: ossia per cui sia  $|N_{Eps}(p)| \geq MinPts$ 
  - Ritrova tutti i punti raggiungibili per densità da  $p$  rispetto a  $Eps$  e  $MinPts$ , si e' formato un cluster.
  - Il cluster così individuato e' indipendente da quale dei suoi core point si è partiti poiché fra i core point la relazione di raggiungibile per densità è simmetrica
- ③ Altrimenti  $p$  e' marcato come noise. Se dovesse essere un punto nel border esso sarà recuperato durante successive iterazioni dell'algoritmo.
- ④ Continua il processo fino a quando tutti i punti sono stati processati.

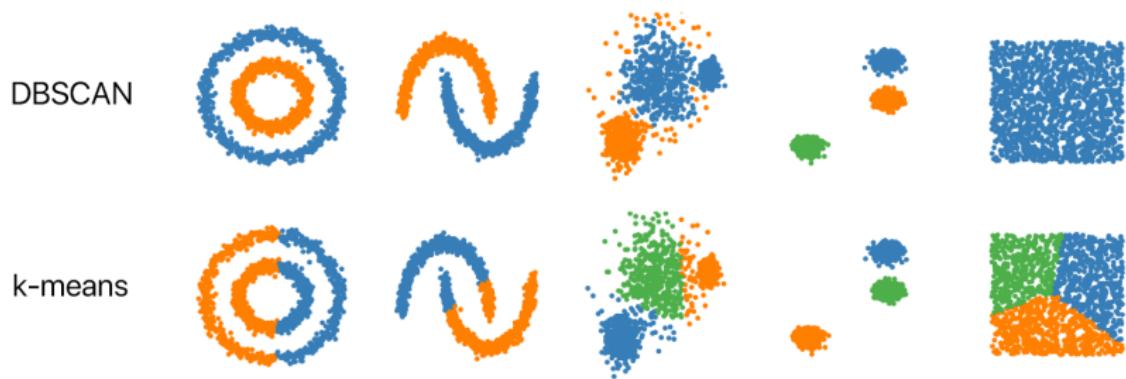
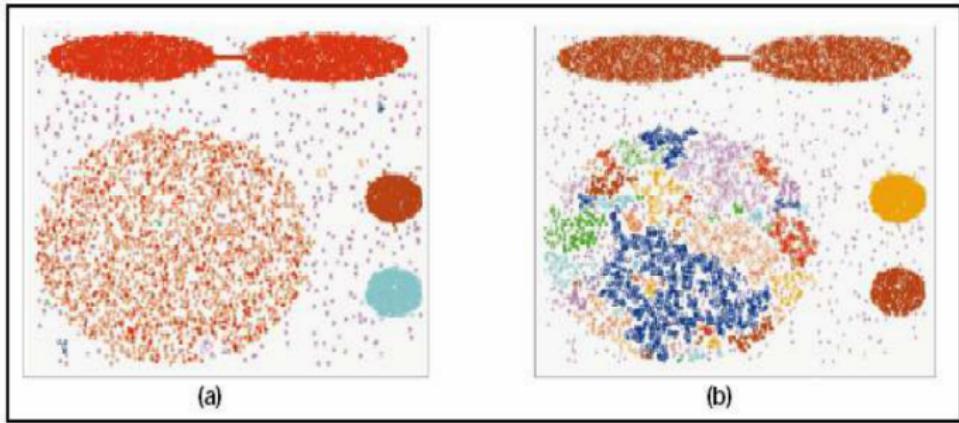
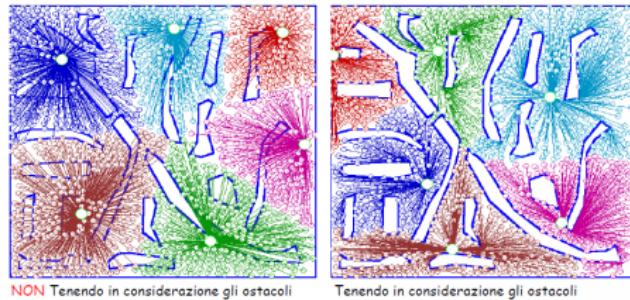


Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.



# Constraint-based clustering

Clusterizzare oggetti in presenza di ostacoli.

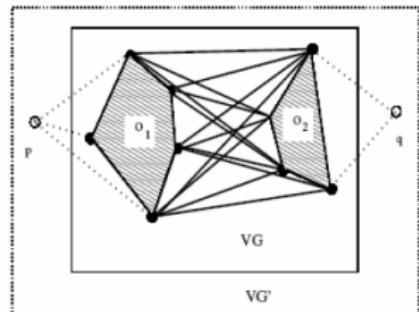


Supponiamo di volere decidere dove piazzare macchine ATM (bancomat). Le ATM dovrebbero essere accessibili in modo efficiente ai clienti della banca. Ma bisogna considerare gli ostacoli presenti nel piano (fiumi, montagne, condomini, etc..). La misura di distanza deve tener conto di questi **VINCOLI**.

Naturalmente un k-medoids e' preferibile rispetto a un k-means (il centro di un cluster potrebbe risultare all'interno di un ostacolo).

# Constraint-based clustering

Un punto  $p$  è **visibile** da un punto  $q$ , nella regione  $R$ , se una linea retta può essere tracciata tra  $p$  e  $q$  senza intersecare ostacoli.



Un **grafo di visibilità** è un grafo  $VG = (V, E)$ , tale che

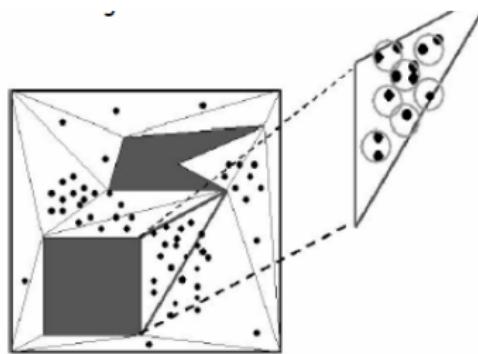
- ogni vertice degli ostacoli sono nodi del grafo
- Esiste un arco tra  $v_1$  e  $v_2$  in  $E$  se  $v_1$  è visibile da  $v_2$ .

Il grafo di visibilità è essere costruito considerando i punti da clusterizzare e i vertici degli ostacoli.

Lo shortest path in  $VG$  tra due punti  $p$  e  $q$  è la **distanza da usare per clusterizzare**.

Una volta ottenute le distanze, usiamo **k-means** per la clusterizzazione.

Si può ottimizzare effettuando una triangolazione dello spazio e creando dei micro cluster all'interno di ogni triangolo.



Ogni cluster verrà rappresentato usando un solo punto. Il numero di punti da clusterizzare viene scalato dal numero di cluster che si formano.