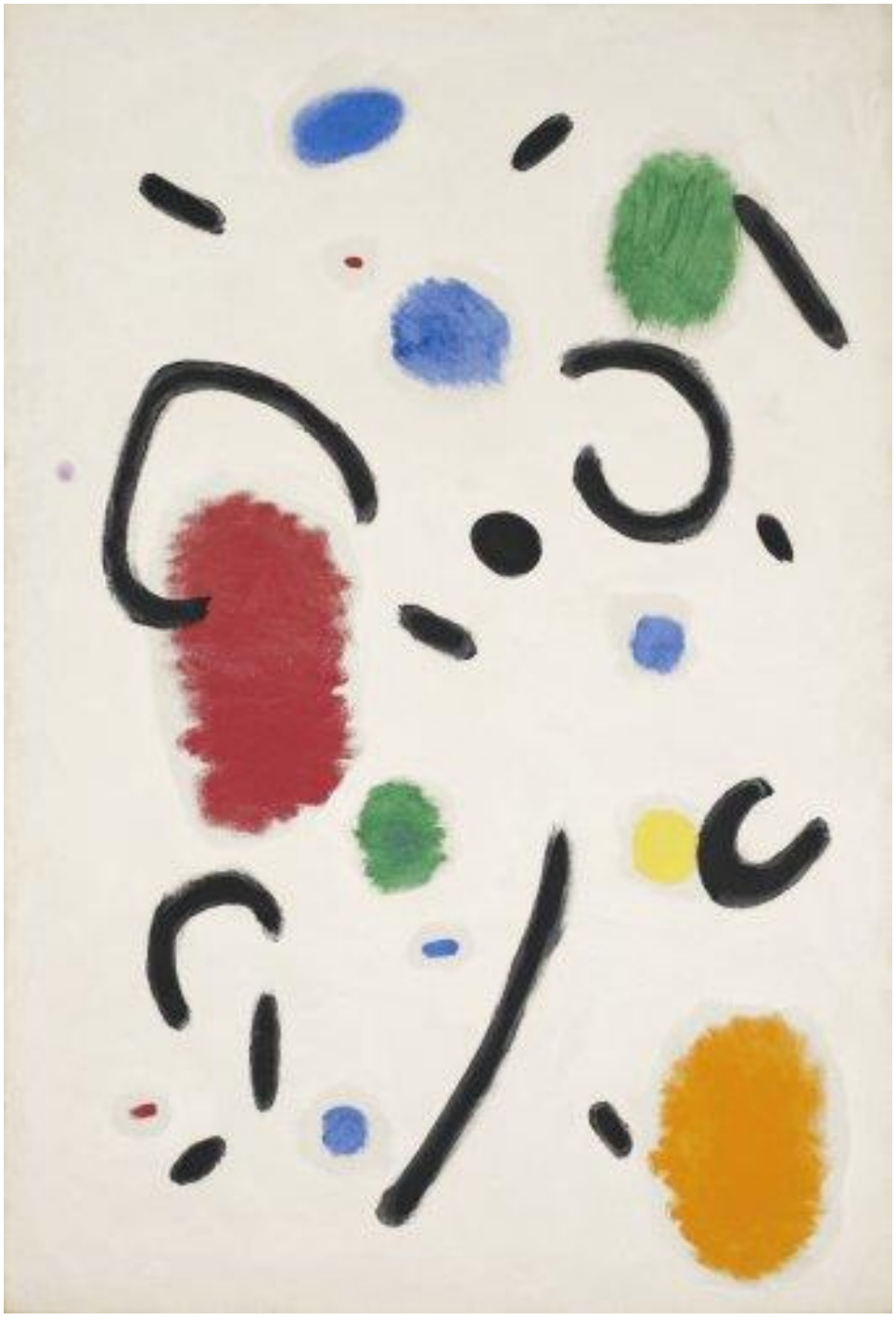


Chanson sur fond blanc, Joan Mirò, 1966, olio su tela.

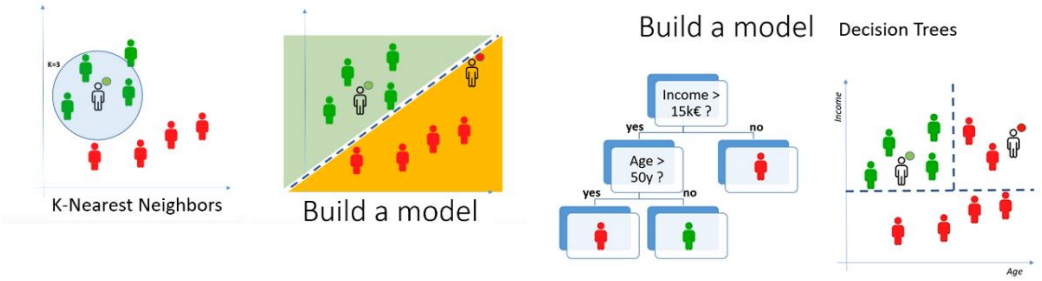
Classification and clustering.

Iniziamo la lezione chiedendoci: cos'è la classificazione? E cos'è il clustering?

Dati due gruppi di oggetti, ciascuno con determinate caratteristiche (per le persone possono essere: età, sesso, altezza; per le transizioni possono essere: tipo, quantità di denaro, tempo), vogliamo assegnare a ciascun oggetto di una lista una etichetta ("label").

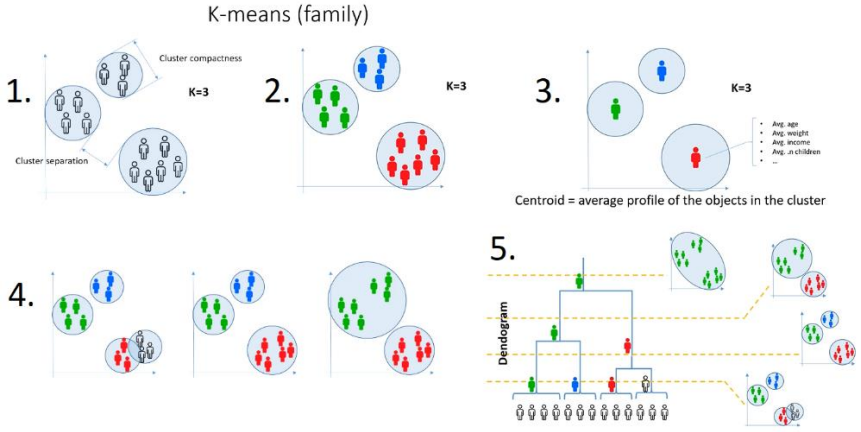
La **classificazione** non assegna a priori tale etichetta, ma cerca di assegnarla in base a degli esempi prestabiliti, inferendo una regola generale. Vari metodi di classificazione sono (v. immagine; in tali esempi il verde simboleggia "good customers", il rosso "bad customers"):

- **K-Nearest Neighbors**: si assegna al ns. oggetto lo stesso label dei tre oggetti più simili;
- **Build a model**: metodo più astratto: si realizza un *modello* piuttosto che effettuare una comparazione diretta con un altro oggetto. Immaginiamo uno spazio diviso in due sotto-spazi (green e red); a seconda di dove si trova il ns. oggetto gli assegniamo il label;
- **Decision Trees**: esempio particolare del Build a model. Lo spazio è diviso in più sub-spazi, con più linee;



Come visto nella classificazione è implicita la pre-esistenza di almeno due label (in questi casi: green e red). Se gli oggetti interessati non posseggono un label. Potremmo avere dei clienti e sapere le loro caratteristiche, ma non avere alcun esempi di "bad customers" and "good customers". Si parla in questo caso di **clustering.**

- **K-means (family)**: con questo metodo sono formati k sottogruppi, insiemi (cluster) compatti e ben separati. **A tali gruppi possiamo assegnare un colore, che non ha però alcun valore semantico.** Nell'esempio riportato sotto nessuno dei tre gruppi colorati indica "good customer". Da questi gruppi si possono estrasse i **valori medi** introducendo il concetto di ***centroid.*** Nondimeno, è importante ricordare **l'artificialità di fondo di tali divisioni** – gli stessi *samples* possono essere divisi in altri gruppi, come evidente nel quarto esempio riportato nell'immagine seguente. È anche possibile un **approccio gerarchico a scatole cinesi,** creando man mano diversi sottogruppi. **In ogni caso, è importante ricordare che non si parte da alcuna idea pre-definita di gruppo.**



-

Clustering. The basis.

La definizione di **clustering** è: *definire gruppi (sub-sets) di oggetti in modo tale che gli oggetti in ciascun gruppo siano simili l'un l'altro*. In altre parole, comparando fra loro gli oggetti interni di ciascun sub-set (ad esempio: l'oggetto rosso A con l'oggetto rosso B) essi risulteranno simili (tale nozione è nota come **Minimizzazione delle distanze intra-cluster**), mentre comparando fra loro oggetti appartenenti a cluster diversi essi saranno piuttosto differenti (**Massimizzazione delle distanze inter-cluster**). I sub-set hanno quindi il nome di *cluster* e pertanto **il clustering è un set di clusters.**

In che modo si usano i cluster?

- **Understanding.** I cluster sono utilizzati per meglio comprendere dei dati, raggruppando i dati simili e pensando a loro come un singolo *pseudo-oggetto* (cfr. il concetto di *centroid*). Un esempio pratico è la **document-repositary:** è creato un indice di documenti (oggetti) che parlano dello stesso argomento. Partiti da 4 milioni di oggetti, ne abbiamo in ultima istanza un quarto: sono stati astrattizzati nello stesso oggetto.
- **Summarization.** Si riduce la grandezza di un grande data set. Ad esempio nelle previsioni del meteo più livello di pioggia sono astrattizzatti in un unico concetto di "tempo piovoso". In questo modo l'informazione, troppo grande e caotica per la mente umana, risulta accessibile anche gli uomini e non solo alle macchine.

Un'altra definizione di clustering è quella di ***unsupervised classification,*** laddove la classificazione *strictu sensu* è nota come ***supervised classification***. In un caso, infatti, si hanno dei label pre-esistenti (si è quindi guidati), nell'altro no. Questa artificialità della classificazione può portare a una certa ambiguità nell'applicazione della nozione di cluster: lo stesso gruppo di dati potrebbe essere in diviso, ad esempio, in sei, due o quattro cluster. D'altra parte, è proprio questa malleabilità dei cluster a rendere il metodo adatto ad essere applicato a più intuizioni.

Il clustering non è invece una *simple segmentation*, cioè una semplice creazione di gruppi (studenti ordinati, ad esempio, in ordine alfabetico). Il clustering si basa poi su **proprietà intrinseche ai dati**: i raggruppamenti non sono il risultato di una *external specification* (una query inserita dall'utente, ad esempio). Infine, il clustering cerca di trovare una struttura per l'intero data-set, non una semplice parte (si cerca una *global connection*, non una *local*).

Una distinzione importante è quella fra:

- **Partitional clustering:** ciascun oggetto appartiene a un cluster e ciascun cluster è separato l'un l'altro. Stiamo partizionando il set in sub-sets;
- **Hierarchical clustering[1]:** un cluster può essere parte di un altro cluster, con una struttura gerarchica. Si può rappresentare con una rappresentazione insiemistica o "ad albero". Piuttosto che un semplice *partitional clustering*, muovendosi livello su livello si hanno più cluster. Più si aumenta di livello, più i cluster diminuiscono (ma diventano più grandi, con più oggetti).

Altre distinzioni sono quelle fra:

- **Exclusive / Non-exclusive clustering**: lo stesso oggetto può appartenere a più cluster;
- **Fuzzy / Non-fuzzy clustering:** nel clustering fuzzy un oggetto appartiene a ogni cluster, con "peso" (weights) diverso da 0 a 1. La somma di tutti i pesi è 1. Similmente si comporta il **probabilistic clustering** (il "peso" è una probabilità[2]);

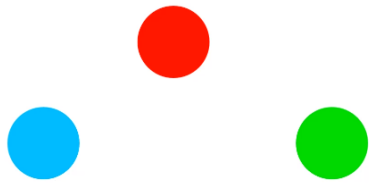
[1] Approfondiremo gli *hierarchical clustering* successivamente

[2] Per approfondire cfr. Teoria della probabilità.

- **Partial versus complete:** in alcuni casi vogliamo "clusterizzare" solo alcuni dati – gli altri potrebbero essere irrilevanti, o rumore[3];
- **Heterogeneous versus homogeneous:** nel primo caso i cluster possono avere differenti grandezze, forme e densità.

Esplicitiamo ora i criteri più diffusi per l'elaborazione di un buon cluster. Alcuni di tali metodi sono mutuamente esclusivi, altri no.

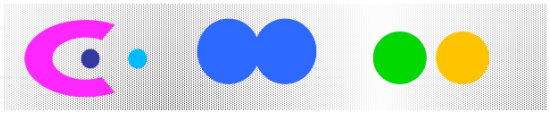
- **Well-separated clustering:** i cluster sono ben separati l'uno dall'altro. È importante che i diversi cluster non si confondino;
- **Center-based clustering:** un cluster forma una forma che può facilmente essere rappresentata con una forma con un centro (come un cerchio). È così palese che ci muoviamo attorno a tale centro (donde la già citata nozione di **centroid,** la media di tutto i punti di un cluster, a cui si accompagna il **medoid,** il "più rappresentativo" punto di un cluster). In tal modo è anche possibile visualizzare gli elementi più lontani dal centro e attigui a (e in taluni casi anche parte di) un altro cluster.
- **Contiguous Cluster (Nearest neighbor / Transitive):** se due oggetti (virtualizzati come punti) sono vicini l'un l'altro, appartengono allo stesso cluster – a prescindere da quanto hanno attorno. Ne consegue che **se abbiamo una sequenza di punti, ogni punto appartiene allo stesso cluster.** Il primo punto "attirerà" al suo cluster il secondo punto, che attirerà il terzo e così via – per quanto l'ultimo punto possa ben essere distante dal primo (i punti intermedi hanno così una funzione di "ponte"). Riprendendo l'esempio precedente degli "elementi parte di un altro clustering": se prima avevamo un cerchio rosso e uno blu, adesso i due cerchi sono entrambi blu. I *continguous clusters* sono così "merged together". **Nel caso si abbia molto rumore può venire a crearsi un "ponte di punti" che può portare a unire fra loro due cluster differenti;**
- **Density-based**: un metodo simile al precedente, che considera però non solo la vicinanza dei punti, ma anche la loro densità. La densità di un'area è più grande della densità di una semplice linea. La linea blu che formava un "ponte" nell'esempio precedente fra i due piccoli cluster, creando un unico cluster, non è qui presente[4] (e i due cluster sono quindi indipendenti e con diverso colore). In che modo è possibile definire l'ottimale densità? Non esistono metodi precisi (*ipse dixit:* "You have to guess!"), nondimeno alcuni punti posseggono molti "vicini", ma altri punti ne hanno di meno: questi pochi punti sono il rumore, che viene eliminato settando un'alta densità. A un certo punto apparirà un "jump" nella rappresentazione, in cui sarà palese l'eliminazione dei punti di rumore.



3 well-separated clusters

4 center-based clusters

8 contiguous clusters



6 density-based clusters

[3] Non è ciò in contrasto con il carattere *global* del clustering? Ad avviso dello sbobinatore, no: accettato che ogni dato non può essere escluso *a priori* o per ragioni *estrinseche*, appare altresì legittima la possibilità di escludere dati irrilevanti a una data intuizione *a posteriori.*

[4] Se un punto è la virtualizzazione di un dato e una linea un insieme di punti, non si rimuovono dei dati? È questo il "rumore" a cui si è precedentemente accennato e il rumore non rappresenta niente di significativo: **il metodo basato sulla densità permette così di identificarlo ed eliminarlo.** Un concetto analogo al "rumore" è quello dell'***outlier***.

Un cluster può anche essere definito da una **funzione matematica.** Parlare di "cluster che formano cerchi" è molto astratto, ma ciò può essere definito con una formula. Come fare? In primo luogo si cercano cluster che minimizzano o massimizzano una funziona obiettiva (*objective function*). Si enumerano così tutti i possibili clustering (cioè i modi di dividere i punti in cluster) e si valuta la "bontà" di ciascun set usando una funzione obiettiva (NP Hard), aiutandosi con algoritmi. Algoritmi di clustering gerarchici usano funzioni obiettive locali, mentre algoritmi di partizionali delle obiettive globali.

In chiusura, è bene ricordare l'importanza delle caratteristiche dei dati: la loro **dimensionalità,** la loro **distribuzioni,** i loro **attributi** e i loro **attributi spaziali** possono influenzare la virtualizzazione dei dati in punti. Si noti che alcune delle caratteristiche elencate implicano già una relazione fra un dato e l'altro. A seconda dei dati analizzati è anche importante decidere che tipo di misura di prossimità e densità utilizzare.

Metodo 1: K-means.

Andremo adesso nel concreto, descrivendo i tre metodi più comuni per formare clustering.

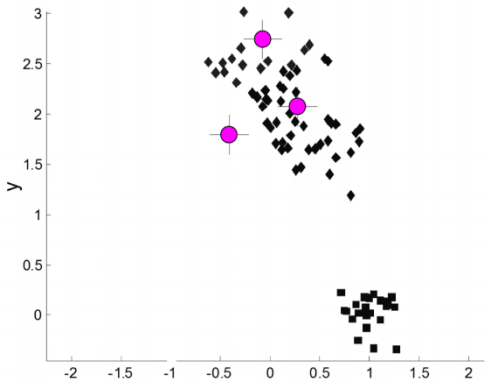
- K-means
- Hierarchical clustering
- Density-based clustering

Questa lezione si concentrare sul **K-means.** È il tipo di clustering più semplice ed quasi "archetipico" dell'intera metodologia. È un clustering di tipo **partitional** e ogni cluster **center-based** (abbiamo dunque un **centroid**): ogni punto è dunque assegnato al cluster che il centroid più vicino al punto stesso. La K nel nome significa che **il numero dei clusters** (K appunto) **deve essere definito all'inizio.**

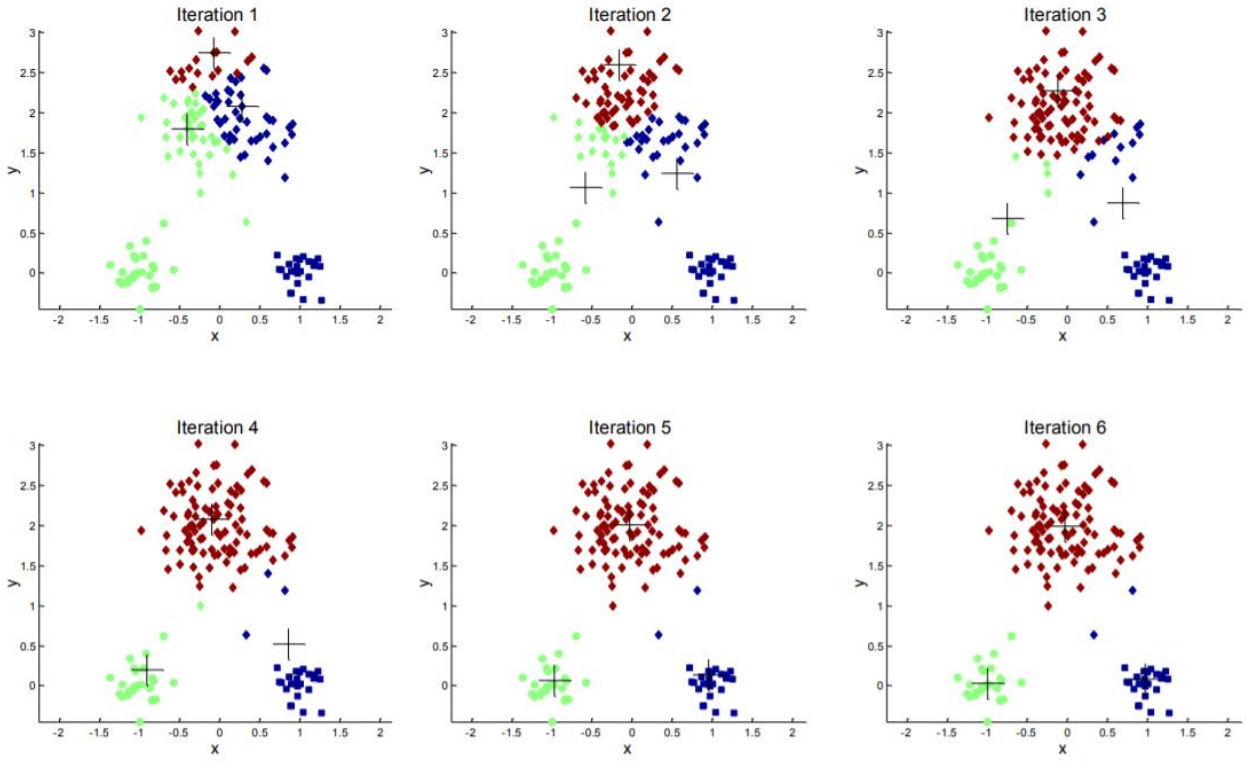
L'algoritmo dietro è di tipo interattivo:

1. Selezioniamo K punti come gli iniziali centroidi. Questa iniziale selezione è random.
2. Ripetiamo:
    a. Formiamo K clusters, assegnando tutti i punti al più vicino centroid.
        i. Assegnati tutti i punti a dei cluster, avremo però dei centroid poco rappresentativi di ogni classe perché – essendo stato selezionato a caso - non sarà il centro di ciascuno classe. Di conseguenza…
    b. … ricomputiamo i centroidi di tutti i cluster. Avremo così un nuovo set di centroidi.
3. I punti precedenti sono ripetuti per tutti i cluster finché i centroidi non cambiano più – cioè, finché non abbiamo trovato i migliori centroidi per ogni classe.[5]

Nell'immagine riportata a lato individuiamo tre centroid a caso, creando le iniziali classi (nell'immagine alla pagina successiva: rosse, verdi, blu). Notiamo in particolare che il centroide scelto per le classi verdi e blu non si trova davvero al loro centro e il (supposto) cluster più alto è "spezzato" in tre (quando intuitivamente sappiamo che dovrebbe essere tutto rosso). La divisione proposta è innaturale. **Ma la posizione dei vari centroidi verrà migliorata iterazione dopo interazione.** "Migliorata" non significa che attireranno più punti (spostando i centroidi verdi e blu verso il basso, anzi, perderanno dei punti), ma che sarà più rappresentativa della reale divisione dei dati.



[5] Ma se abbiamo costruito una classe attorno a un centroid, quello non sarà già il centro? No, a meno che non si abbia un **dataset infinito** con dati ovunque.



Notare che le ultime iterazione servono per un mero “fine-tuning” dei centri: **la convergenza reale avviene nelle prime interazioni.** Spesso, quindi, la condizione di stop non è “finché nessun punto cambia cluster”, ma “finché solo un numero relativamente basso di punti cambia clustering”.

Come detto l’iniziale set di centroidi è scelto a caso ed è solitamente il “mean” dei punti nel custer. Quando parliamo di “vicinanza” (“closeness”) pensiamo a una distanza euclidea misurata per “correlation” e “cosine similarity”. Sono queste le misure dietro la convergenza del metodo K-means. Esiste poi una formula matematica per calcolare a posteriori la complessità di ciascuna applicazione del K-means.

Complessità:
O * (n * K * I * d)

n = numero di punti; K = numero di cluster; I = numero di iterazioni; d = numero di attributi

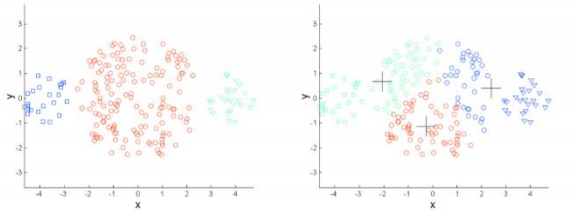
Un’altra misura da considerare è la **SSE: Sum of Squared Error**, l’equivalente di quella che in statistica è chiamata “variance” e ci permette di definire la qualità di un clustering.

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

x is a data point in cluster C_i and m_i is the representative point for cluster Ci

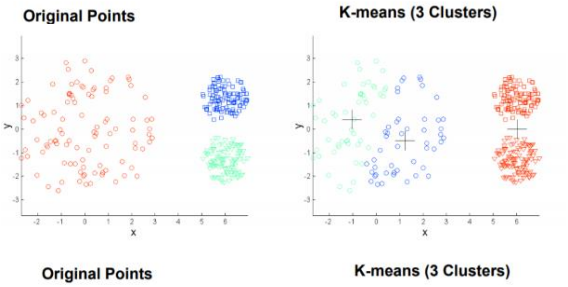
Per ciascun punto, l’errore è la distanza al più vicino centroide (perché più vicino è il punto al centroide, minore è l’errore). Eleviamo alla seconda questi errori e li sommiamo. Questo metodo è coerente con la logica del K-means: all’aumentare dei K diminuisce la somma degli errori, perché abbiamo dei centroidi più precisi. Tuttavia un buon clustering può avere, pur con un K più piccolo, un SSE più basso di un clustering realizzato male.

Generalmente il K-means è un metodo veloce ed efficiente per la misura dei cluster, pur non essendo perfetto. **Mostra infatti i suoi limiti con cluster con differenti grandezze, densità o forme che non sono cerchi.**

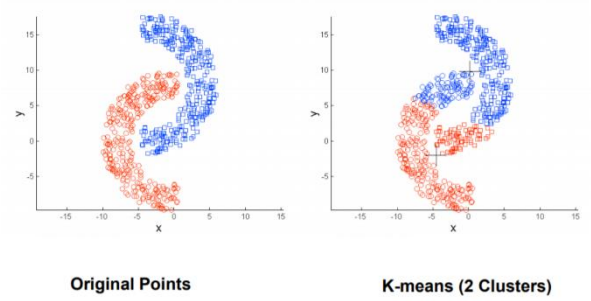


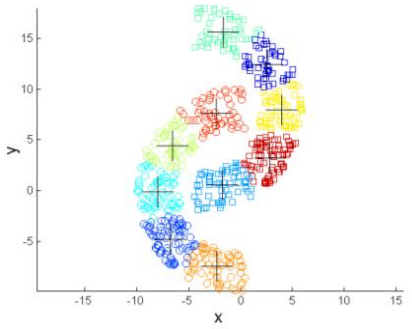
Nell'esempio riportato a lato si creano dei cluster troppo grandi, che comprendono dati che non dovrebbero avere. Una possibile soluzione è aumentare il numero dei K (e quindi dei cluster).

Nel secondo esempio i dati hanno densità differente: i cluster più densi sono più compatti e hanno meno centroidi dei dati meno densi. Anche qui la soluzione è aumentare il numero di K e poi unire più cluster: ma è un processo manuale.



Nel terzo esempio abbiamo i dati disposti in modo assolutamente non circolare: anche qui, la soluzione è dividere i dati in tanti cluster. Alleghiamo qui anche la soluzione corretta.

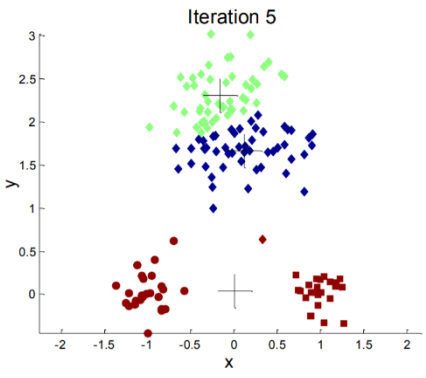




Il metodo K-means può anche generare dei cluster vuoti, se si prende come riferimento la semplice equidistanza. Per questo motivo solitamente si sceglie un punto (cioè un dato) che fa da centro, magari da un altro cluster con grande SSE.

Come "pre-processing" è anche importante normalizzare i dati, rimuovendo rumore e outliers. Nella fase "post-processing" si tende ad eliminare i cluster più piccoli (che rappresentano gli outliers), si dividono i cluster con SSE grande in cluster più piccoli o si uniscono fra loro cluster con basso SSE vicini l'un l'altro. **Sono dei processi manuali che accompagnano l'algoritmo: l'output del K means non deve essere usato "as is".**

Il processo interattivo che abbiamo visto può anche portare a dei "local optimum" poco utili. Selezionando inizialmente un centroide sbagliato, si può arrivare alla soluzione rappresentata nell'immagine a lato: **l'algoritmo non va più avanti, ma non è una divisione "corretta".** La probabilità di scegliere il giusto cluster è molto bassa: dipende dal numero di possibilità di selezionare il "giusto" centroide per ogni cluster e dal numero possibilità di selezionare la "giusta" quantità di K cluster. Talvolta la situazione si aggiusta da sola, altre no.



All'***Initial Centroid Problem*** la soluzione può essere un "metodo forza bruta", provando più volte. È il metodo più diffuso con il k-means. Il post-processing è un'altra opzione, così come l'utilizzo di clustering gerarchici per determinare il centroide iniziale: si tratta di un metodo meno efficiente del K-means, che viene focalizzato su set di data più piccoli semplicemente per determinare il centroide, che verrà poi usato dal K-means.

Una variante dal K-means è il **Bisecting K-means.**

1. Inizializziamo una lista di cluster che contenga un cluster che include tutti i punti.
2. Ripetiamo:
    a. Selezioniamo un cluster dalla lista dei cluster.
    b. For (i = 1, i < *number_of_iterations*) do {
        Bisect the selected cluster using basic K-means[6]};
    c. Aggiungiamo il migliore cluster (con l'SSE più basso) alla lista dei cluster.
3. Dopo ogni step abbiamo sempre un cluster un più e sono ripetuti finché la lista dei cluster contiene K clusters.

I risultati non sempre sono perfetti, ma nel complesso il Bisecting K-means fornisce una soluzione al Centroid Problem (ma non a *tutti* i limiti del K-means, che abbiamo già incontrato). Generalmente è consigliato iniziare a *splittare* i cluster più grandi.

Metodo 2: Hierarchical clustering.

Abbiamo già menzionato i clustering gerarchici. Verranno adesso approfonditi.

Il clustering gerarchico crea un set di **nested clusters** (una struttura a scatole cinesi o ad alberi gerarchici, che hanno il nome "tecnico" di dendogramma). A un primo livello ogni cluster contiene un oggetto, a un secondo livello un cluster contiene due di questi cluster – con somiglianza più generale di quella che hanno gli oggetti contenuti e così via. È bene precisare che un cluster di, ad esempio, livello 3 non necessariamente contiene solo cluster di livello 2: può contenere un cluster di livello 2 e un cluster di livello 1.

A differenza del K-means **non dobbiamo specificare un iniziale numero di cluster;** questo ci permette di postporre particolari decisioni riguardanti il numero di cluster richiesto. A colpo d'occhio il dendogramma visualizza inoltre il procedimento adottato dall'algoritmo: vediamo esattamente in che misura i diversi oggetti considerati hanno elementi in comune. La gerarchia offerta da questo tipo di clustering può inoltre corrispondere a **tassonomie esistenti in altre scienze** (si pensi al mondo degli animali).

Ci sono due tipi di clustering gerarchici:

- **Agglomerative:** Partiamo da punti individuali e ogni step la coppia più vicina (*closest pair*) di cluster è unita, finché non abbiamo un solo cluster rimasto (o in alternativa, k cluster).
- **Divisive:** Partiamo da un grande cluster con tanti punti, che l'algoritmo divide in due parti. Ogni cluster sarà sempre più piccolo: corrisponde a leggere il dendogramma dall'alto in basso.

Gli algoritmi usano una *similarity or distance matrix,* da computare prima dell'esperimento. Questo è un grande limite del metodo gerarchico (laddove il K-means era molto meno *resource-intensive*).

L'algoritmo dell'agglomerativo segue questi step:

1. È computata la matrice di prossimità.
2. Ogni punto è considerato un cluster.
3. Si ripete:
    a. I due cluster più vicini[7] sono uniti.
    b. È aggiornata la matrice di prossimità.
4. Finché non rimane un singolo cluster.

[6] Cosa significa "bisezionare un cluster con K-means"? K-means di per sé prende un set e lo divide in K cluster. In questo caso prende un cluster e lo divide in due parti (K = 2).

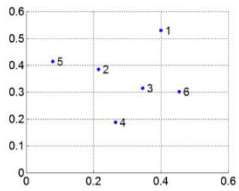
[7] È quindi molto importante il calcolo (la computazione) della prossimità / vicinanza dei due cluster. Approcci diversi (magari non euclidei) alla definizione della distanza possono generare algoritmi molto diversi fra loro.

Quando aggiorniamo la matrice decidiamo effettivamente che tipo di clustering vogliamo. Ma come definire la **Inter-Cluster Similarity,** la distanza fra più cluster? Ci sono più metodi.

Nel criterio **MIN,** o **Single Link,** dati due set di punti, la loro *similarity* è data dalla *similarity* dei loro punti più vicini. Nell'immagine riportata a lato, ad esempio, saranno prima uniti i cluster 3 e 6. Successivamente verrà considerato un unico cluster (3+6) e di questo cluster verrà considerato il punto più vicino agli altri cluster. Il cluster (3+6) verrà quindi unito al cluster 1, in quanto la distanza fra il punto 3 – inglobato in (3+6) – e il punto 1 è la più piccola.

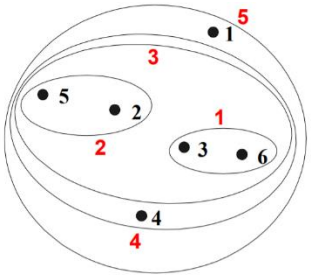
Nel dendogramma del risultato sull'asse y è riportato il valore di tali distanze minime (nel grafico il punto d'intersezione). Notiamo anche che l'ultimo merge ha un "salto" maggiore dei precedenti: è dunque meno naturale.

Il criterio MIN si utilizza nel caso di set di dati con strane forme, non ellittiche, che in comune non hanno la grandezza né la compattezza (K-means spezzerebbe i set di dati), ma **la vicinanza dei punti, che formano una grande "catena" di connessioni.** Il criterio MIN si basa su questa catena. Proprio per questo motivo, tuttavia, il criterio MIN fallisce nel caso si abbia molto rumore: il rumore crea una falsa catena, falsando il risultato finale unendo fra loro due cluster molto differenti o non integrando in un cluster punti più lontani.

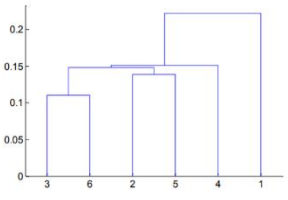


Distance Matrix:

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00



Nested Clusters



Dendrogram

Il criterio **MAX,** o **Complete Linkage,** è speculare al precedente: dati due set di punti, la loro *similarity* è data dalla *similarity* dei loro punti più lontani. Tale distanza è il **diametro** della somma dei due cluster. Questo criterio misura quindi quanto grande e *dispersive* sarebbe il cluster. L'algoritmo genera allora il cluster meno *dispersive* e cerca di avere cluster della stessa grandezza (ad esempio (5+2)+1) e ((3+6)+4)).

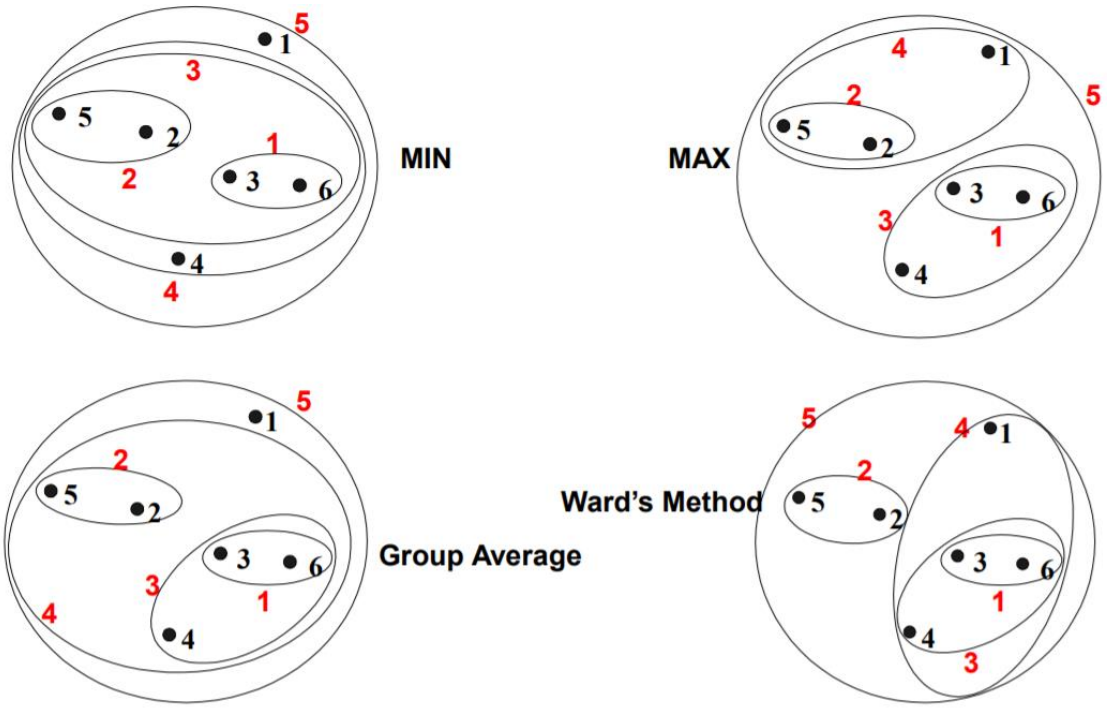
Rispetto al metodo MIN notiamo cluster più bilanciati di quelli rivisti con il MIN. Interessante notare anche che nell'esempio precedente i punti 3 e 2 venivano presto uniti, qua no. Il metodo MAX non ha le debolezze del MIN: è meno suscettibile a rumore e *outliers.* Tuttavia, **è propenso a rompere grandi cluster ed è *biased towards globular clusters*.** Da questo punto di vista risulta simile al K-means, pur per motivi diversi: alla fine avremo cluster di forma circolare, più o meno della stessa grandezza.

Il criterio **Group Average** è una sorta di via di mezzo: considerati tutte le coppie possibili di punti, piuttosto che scegliere MIN o MAX è scelta la media. Ne consegue che se molti punti sono vicini al punto MIN/MAX il risultato sarò simile al MIN/MAX. Tuttavia, complessivamente risulta avere pro e contro analoghi a quelli del metodo MAX. Un metodo simile è quello basato sulla **Distance Between Centroids:** sono prima computati i **centroidi**[8], cioè il "punto medio" (*the average of points*) di tutti i punti, ed è computata poi la distanza fra loro.

Si possono usare anche funzioni obiettive (**Objective Function**)**:** la distanza fra i due cluster è considerata un **costo** e viene formulata una funzione che li unisca al costo minore. In particolare, con il **Ward's Method** la

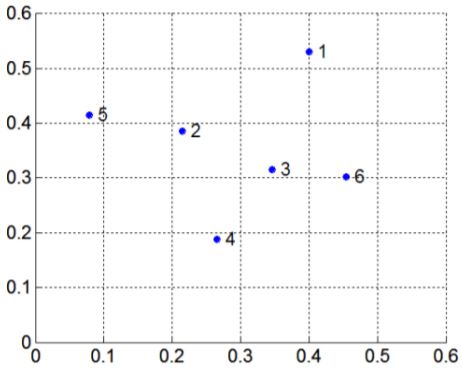
[8] In questo caso il centroide sembra un valore / oggetto non necessariamente esistente (non c'è un punto che ha il ruolo del centroide), ma un valore (punto) ipotetico che viene calcolato, esattamente come si calcola la distanza media fra due punti.

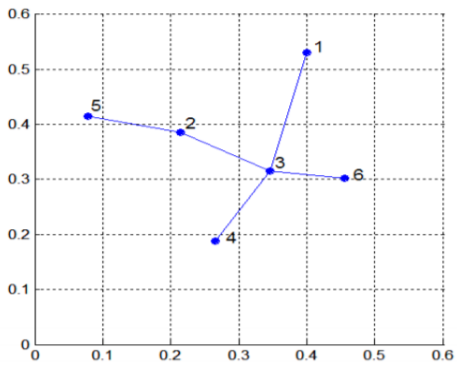
similarity fra i due cluster è *based on the increase in squared error when two clusters are merged.* Quando uniamo due cluster l'SSE totale cambia, perché due cluster spariscono lasciando spazio a un cluster: di conseguenza, l'SSE aumenterà.[9] Anche questo metodo ha pro e contro analoghi al MAX: poco suscettibile al rumore, biased verso cluster circolari. **Utilizza del resto una misura tipica del K-means (l'SSE), di cui nei fatti rappresenta la controparte gerarchica.**



Il metodo MIN è l'unico che *compute connection-based clusterings*; gli altri cercano di generare cluster più bilanciati.

Tutti i metodi visti formano clustering agglomerativi. Abbiamo quindi di base un grande cluster, che vogliamo dividere in gruppi più piccoli. Potremmo usare un metodo simile a quello di Ward, chiedendoci qual sia il metodo migliore per *splittare* un cluster, tenendo l'SSE come riferimento. Ogni volta dovremmo però computare tutti i possibili SSE: sarebbe un'operazione troppo *resource-intensive.* Useremo quindi un altro metodo: **l'MST (Minimum Spanning Tree).**





Partiamo da un solo punto e cerchiamo il punto più vicino ad esso. Questa coppia di punti è il primo ramo di un albero. Cerchiamo poi i rami successivi, prendendo come riferimento un punto appartenente all'albero e cercando un altro punto esterno all'albero il più vicino possibile. Pian piano si formerà l'intero albero. Nell'immagine, si è partiti da 1 e 3, primo ramo. 3 è stato poi unito a 6 e in seguito a 4. Poi a 2. 2 è stato poi unito a 5.

[9] Perché avremo, per tutti i dati del nuovo cluster, un centroide non ottimale in luogo dei più ottimali centroidi dei precedenti cluster. Più i due vecchi cluster erano lontani, più sensibile sarà l'aumento dell'SSE.

L'albero è il nostro cluster di partenza e la distanza fra i rami simboleggia la *similarity* fra i vari punti. L'albero è una struttura "di comodo", che andiamo poi a dividere.

L'algoritmo è:

1. Computare un *minimum spanning tree* per il grafico di prossimità.
2. Creare un nuovo cluster **rompendo il link corrispondente alla distanza più grande nell'albero** (i cluster con *smallest simlarity*).[10]
3. Ripetere il punto 2 finché non rimane un singolo cluster.

Chiudiamo con qualche considerazione sui clustering gerarchici. **Ogni metodo visto richiede grande spazio,** che aumenta esponenzialmente con l'aumentare dei punti perché dobbiamo costruire una *proximity matrix,* assolutamente richiesta. **Anche il tempo d'esecuzione è molto ampio** e aumenta anche *al cubo* con l'aumentare dei punti. Tuttavia, la complessità O(N^3) può essere ridotta a O(N^2 log (N)) in alcuni casi.

Oltre ad essere molto intensivi in termini di risorse, **i cluster gerarchici presentano una serie di decisioni che non possono essere *backtracked***: una volta uniti o divisi dei cluster, questa operazione non può essere annullata. Non c'è poi una *global objective function* che può essere minimizzata: si agisce sempre a livello *local*. Come visto, poi, tutte le metodologie hanno i loro contro.

Per tutte queste ragioni, i cluster gerarchici sono usati molto raramente. **Vengono però sfruttati come "base" per il K-means, usandoli per calcolare i centroidi inziali.**

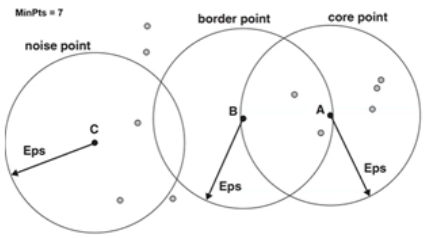
Metodo 3: Density Based.

Al K-means e ai clustering gerarchici si affianca un terzo metodo, basato sulla **densità dei dati nel *data space*.** Queste regioni particolarmente dense, separate l'un l'altra da regioni con minor densità. Questo metodo è particolarmente efficace per cluster di forma particolare.

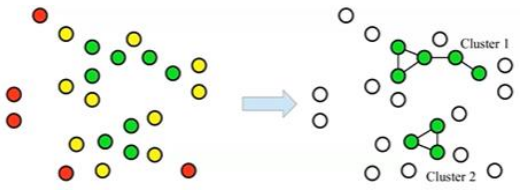
DBASCAN è un *density-based algorithm.*

- La **densità** è definita per il numero di punti all'interno di uno specifico raggio (*Epsilon,* o Eps);
- Un punto è un *core point* se ha uno specifico numero minimo di punti (MinPts) all'interno del raggio. Tali punti (incluso il core point) saranno quindi all'interno del cluster. I *core points* possono essere definiti anche *dense points*.
- Si parla invece di *border point* per i punti vicini a un core point. Infine, se un punto non è né core né border, è un *noise point*.

Nell'immagine a fianco solo il punto A è un *core point*: è l'unico che rispetta il *thresold*, la condizione minima dei punti interni (7, contando anche A stesso). B è comunque un *border point* di A. Definire il border, il confine, è molto importante: i punti fuori non saranno parte del cluster (nessuno dei punti inclusi in C fanno parte del cluster).

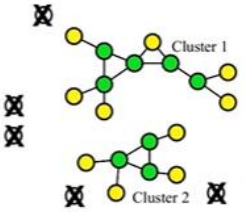


Definiti quali punti sono core, border e noise, lo *step 2* è **connettere i *core objects* che sono vicini** (*neighbors*) e **inserirli nello stesso cluster.** Nell'immagine a fianco i core object (definiti come precedentemente visto) sono rappresentati in verde. Il risultato saranno quindi due cluster.



[10] Notare che lo split dei cluster è fatto basandosi sulla *single connection*, non guardando il contenuto dei cluster; il metodo è quindi più simile al MIN che al K-means poiché si controlla la connessione, non la distribuzione dei punti.

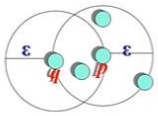
La distinzione fra *noise points* e *border points* ci permette inoltre di eliminare il rumore, associando ogni *border* al proprio core points. Al tempo stesso, i dati del rumor non sono perduti: accanto ai due cluster *main*, integrati con i *border points*, si ha un cluster-cestino (*kinda like a trash-bin*) a cui sono associati i *noise points.* È bene precisare che i *noise points* possono simboleggiare oggetti ad alta densità, ma spazialmente lontani dai dati usati come centro (*core*).



L'intero ragionamento può essere formalizzato parlando di oggetti *density-rechable* e possiamo formare **catene di punti.**

■ **Directly density-reachable**

❑ **An object q is directly density-reachable from object p if p is a core object and q is in p's ε-neighborhood.**

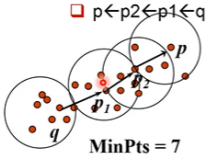


MinPts = 4

- **q is directly density-reachable from p**
- **p is not directly density- reachable from q?**
- **Density-reachability is asymmetric.**

■ Density-Reachable (directly and indirectly):

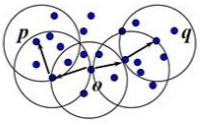
- ❑ A point p is directly density-reachable from p2;
- ❑ p2 is directly density-reachable from p1;
- ❑ p1 is directly density-reachable from q;
- ❑ p←p2←p1←q form a chain.



MinPts = 7

- **p is (indirectly) density-reachable from q**
- **q is not density- reachable from p?**

Notare che la catena può iniziare da un core points e finire su un border points (p -> q), ma non il contrario: non è una relazione simmetrica. Solitamente le relazioni che formano un cluster, invece, lo sono. Questo limite della *density-rechability* ci porta a formulare una nozione simile, ma simmetrica: *density-connectivity.* Nell'esempio, p e q sono connessi grazie al punto 0.



❑ **A pair of points p and q are density-connected if they are commonly density-reachable from a point o.**

Formalizzandolo, l'algoritmo DBSCAN è:

Input: The data set D

Parameter: ε, MinPts

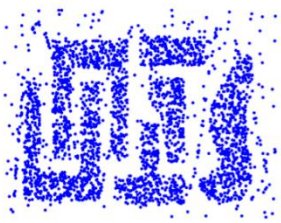
For each object p in D
  if p is a core object and not processed then
    C = retrieve all objects density-reachable from p
    mark all objects in C as processed
    report C as a cluster
  else mark p as outlier
  end if

End For

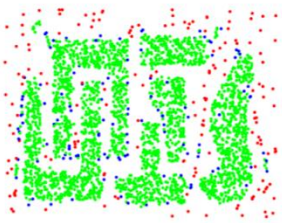
Scannerizziamo ogni punto e se il punto è un *core object,* lo usiamo come base per costruire una classe – cioè, cerchiamo di visitare *tutti i punti possibili* partendo da lì, e passando poi agli altri punti vicini.

Dopo aver visitato tutti i punti vicini (compatibilmente con i criteri prima visti), tutti gli oggetti sono *marked* come processati ed abbiamo così il nostro cluster. I restanti punti sono outlier.

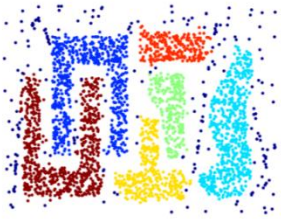
Sotto, le tre fasi del DBSCAN: i punti iniziali sono divisi in corder, border e noise e basandosi sui core sono generati i cluster. Il metodo è particolarmente efficace con queste forme irregolari.



Original Points

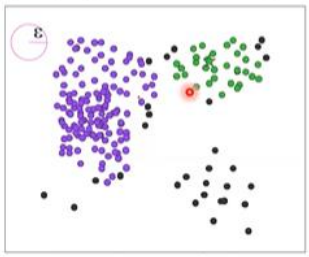


Point types: core, border and noise



Clusters

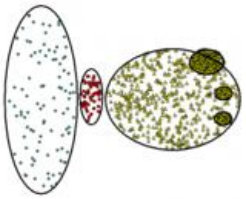
- **Radius ε as shown below (Euclidean distance).**
- **Minimum support m = 7.**
- **2 clusters in this example.**
- **Lower-right grouping not dense enough to form a cluster.**



Introduction to Data Mining, 2nd Edition

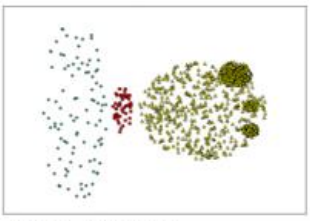
Notare che la connessione fra viola e verde è *solo fra i bordi*: in questo modo, rimangono due cluster separati. **I border point non portano due cluster a unirsi.** Se un border point è *density-rechable* da più di un cluster non è importante a quale venga assegnato (*it's a first come, first serve policy*) da un pdv matematico: *the backbone of the clusters is already decided.* Il gruppo in basso non è invece un cluster non essendo abbastanza denso. Tutti i punti neri sono considerati rumore.

Il DBSCAN tuttavia non è un metodo perfetto. Se funziona bene in caso di rumore e forme particolari, al tempo stesso **mostra tutti i suoi limiti con cluster di diversa densità.**[11] Nell'immagina a ci sono tre cluster grandi e tre piccoli: nel primo caso è data priorità ai grandi. Gli altri cluster sono considerati parte di uno dei tre cluster grandi. Riducendo l'EPS, si aumenta la densità: i tre cluster piccoli vengono così rilevati, a scapito degli altri (adesso sotto il *density thresold* imposto dal DBSCAN), considerati rumore – una perdita considerevole di informazioni importanti.

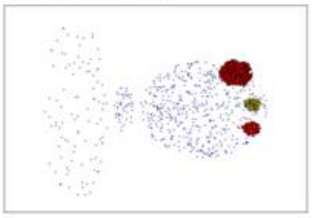


Original Points

- **Varying densities**
- **High-dimensional data**



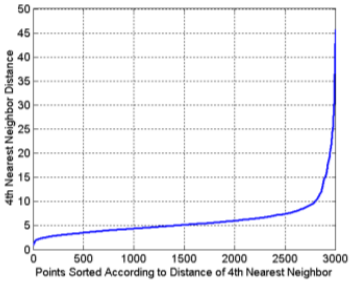
(MinPts=4, Eps=9.92)



(MinPts=4, Eps=9.75)

DBSCAN si basa su due importanti parametri: EPS e MinPts. Esistono delle strategie per suggerire all'utente quali valori utilizzare. Per ogni punto è computata la distanza non per il più vicino, ma per il k-simo punto più vicino: generalmente, i k-esimi punti sono tutti ad egual distanza e i**n questo modo sono isolati i *noise points.*** Nel grafico a fianco, è palese il valore in cui i *noise points* iniziano ad essere considerato: è presente un considerevole salto. In questo modo può essere formulato il valore di MinPoints da considerare.

- Idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance
- Noise points have the k^{th} nearest neighbor at farther distance
- So, plot sorted distance of every point to its k^{th} nearest neighbor



[11] Un altro problema è collegato a *high-dimensional data* che, per la loro natura, sono *sparsed* nello spazio. Un metodo basato sulla densità non funziona bene.

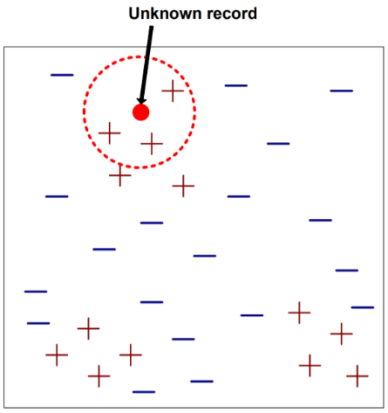
Con gli alberi decisionali, abbiamo provato a **costruire modelli.** Esistono metodi di classificazione che lo evitano. Di questo parleremo adesso.

Instance-Based Classifiers

Ogni qualvolta che abbiamo un record da classificare (cioè dobbiamo indovinarne la classe), piuttosto che costruire un modello **torniamo al training set e controlliamo quale sia la miglior decisioni che possiamo prendere**, comparando il record con il training set.

Ci sono più classificatori di questo tipo. Il ***rote-learner*** parte dal training data e classifica uno dei record **se e solo se nel training ne viene trovato uno perfettamente identico.** Questo metodo ovviamente fallisce con dataset particolarmente complessi.

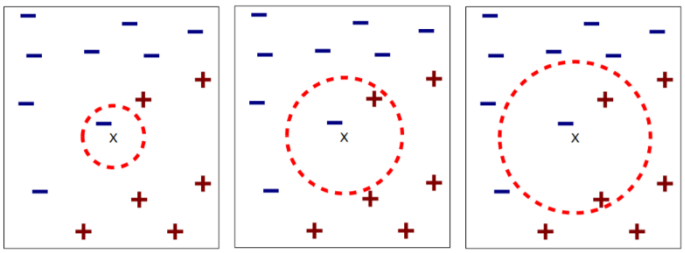
Un metodo più interessante è il **Nearest Neighbor Classifier,** che non cerca la corrispondenza 1:1 ma applica la filosofia del papero: *if it walks like a duck and quacks like a duck, it's probably a duck.* In altre parole, "Fra tutti i record, cerca quello più vicino". Si parla anche di **KNN,** quando il gruppo "dei record vicini" è formato da K elementi.



Nell'immagine, ad esempio, K = 3. I valori + e – nel quadrato rappresentano il training set. Il puntino rosso il record da classificare. Come si identificano i valori vicini? Come nei cluster, serve una **misura di vicinanza.**

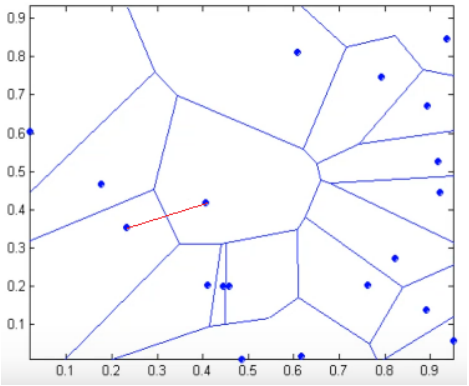
Ogni qualvolta che si ha un nuovo record, esso è quindi comparato agli altri. Si trovano i K record più vicini. Si prendono i loro label, nel caso facendo un "voto di maggioranza" (se fossero stati due + e un -, avremmo comunque deciso per il +). Per questo motivo solitamente **si scelgono numeri dispari**, per evitare situazioni di "parità".

Qual è il nostro margine di manovra? Possiamo modificare il numero di "K" da considerare e la misura di vicinanza. Notare che K e la misura sono direttamente proporzionabili: aumentando K, aumenta il raggio che analizziamo, alla ricerca di maggiori *samples*. **Scegliere K è quindi molto importante, non può essere scelto casualmente.**



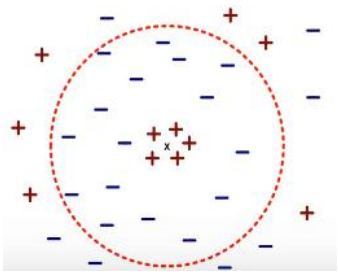
(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Nel Voronoi Diagram a fianco, visualizziamo cosa significa K = 1. Lo spazio viene partizionato e tutti i punti possibili in una regione prendono il valore del punto di quella regione. Come viene partizionato? Vengono tirate delle linee perpendicolari al punto medio di distanza fra un punto e un altro (cioè alla linea rossa nel grafico a fianco). In sostanza, **lo spazio è partizionato in basa alla posizione del singolo punto.** Negli alberi decisionali, invece, l'attenzione era su **una partizione (di più punti) del training set.**



La distanza fra i punti è la controparte dell'ipotenusa in uno spazio 3D.

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$



Per lo più si passa poi al "voto di maggioranza". Ma esiste anche la possibilità del **voto con weight factor:** alcuni punti hanno "più peso" di altri. La loro importanza diminuisce più aumenta la distanza. Nell'immagine, la maggioranza dei valori è " – ", ma i " + " hanno peso maggiore.

Fattore peso = 1/d^2

Se K è troppo piccolo, si fa affidamento soltanto sul ***single nearest object,*** che potrebbe essere **rumore**, frutto di errori di classificazione. D'altra parte, un K troppo grande potrebbe portare a "incontrare" i valori di altre classi, falsando così il risultato. Sempre nell'immagine di sopra, il punto che ci interessa è effettivamente all'interno della classe dei +. Aumentando troppo K (e non facendo affidamento sul fattore peso), esso verrebbe però classificato come -. Il caso estremo è un K così grande da catturare l'intero training set! I problemi che abbiamo con K basso (= 1) e K alto sono "paralleli" ai fenomeni di overfitting e underfitting: nel primo caso guardiano solo il caso particolare più vicino al nostro punto, nel secondo non partizioniamo proprio lo spazio e non cerchiamo di capire cosa il training set può dire. Solitamente, il valore ottimale è compreso fra 1 e il numero dei record.

Ci sono inoltre altri problemi, ad esempio quello legato allo **scaling.** Computando una distanza euclidea, si stanno mettendo insieme le **differenze fra ciascun attributo.** Una differenza, ad esempio, di altezza (1.8, 1.7) sarà molto bassa. Una di stipendio potenzialmente altissima. La misura dello stipendio è di **ordine di grandezza superiore** e "domina" la misurazione della distanza. **Gli attributi devono quindi essere normalizzati.** Un altro problema si ha con la **distanza euclidea,** sia per la ***Curse of Dimensionality*** intrinseca a **dataset multidimensionali**, sia per la diversa **semantica dei dati** (si pensi ai valori categorici "numerizzati" come 0).

I KNN sono poi "lazy lerner": non costruiscono modelli veri e proprio e non "imparano" fino alla fine: ogni volta che vengono lanciati, tutto il processo viene quindi ripetuto dall'inizio. Nessuna informazione viene memorizzata, laddove nel Decision Tree si ha ad esempio una *learning phase*, dopo di ché per classificare record è sufficiente "ripassare" dall'albero – che continua ad esistere. **I KNN hanno quindi un grande costo.**

La maggior parte dei KNN usano valori numerici, distanze numeriche e "voti di maggioranza". L'implementazione **PEBLS** (Parallel Exemplar-Based Learning System) è interessante perché **funziona sia con feature numeriche che continue** ed assegna di default il fattore peso e K = 1. La distanza è misurata in modo molto particolare (Modified Value Distance Metric), considerando **la distribuzioni dei valori nelle classi.** Le classi distribuite nello stesso modo sono **considerate equivalenti** (= 0, Single e Divorced)

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Distance between nominal attribute values:

d(Single,Married) |Yes| + |No|
= | 2/4 – 0/4 | + | 2/4 – 4/4 | = 1
d(Single,Divorced)
= | 2/4 – 1/2 | + | 2/4 – 1/2 | = 0
d(Married,Divorced)
= | 0/4 – 1/2 | + | 4/4 – 1/2 | = 1
d(Refund=Yes,Refund=No)
= | 0/3 – 3/7 | + | 3/3 – 4/7 | = 6/7

Class	Marital Status		
	Single	Married	Divorced
Yes	2	0	1
No	2	4	1

Class	Refund	
	Yes	No
Yes	0	3
No	3	4

$$d(V_1,V_2) = \sum_i \left| \frac{n_{1i}}{n_1} - \frac{n_{2i}}{n_2} \right|$$

La distanza fra un punto P e un punto D è calcolata sia tenendo conto della **somma quadratica delle distanze singole,** sia del fattore peso w. Il w di qualsiasi record dipende dal **numero di volte in cui il record è usato per la predizione** e dal **numero di volte in cui è stato predetto correttamente.** È una forma elementare di

Distance between record X and record Y:

$$\Delta(X,Y) = w_X w_Y \sum_{i=1}^{d} d(X_i,Y_i)^2$$

where:

$$w_X = \frac{\text{Number of times X is used for prediction}}{\text{Number of times X predicts correctly}}$$

$w_X \cong 1$ if X makes accurate prediction most of the time

$w_X > 1$ if X is not reliable for making predictions

self-adjustement, in base a come la predizione avviene. Più il numero di volte usato per la predizione si avvicina a quante volte il record è stato usato, più il peso si avvicina a 1. Più il peso è maggiore di 1 (cioè più è alta la discrasia fra quante volte X è stato usato e le giuste predizioni), più la distanza fra i due punti P e D è amplificata.

sim_F

q \ c	no	avg	high
no	1.0	0.7	0.2
avg	0.5	1.0	0.8
high	0.0	0.3	1.0

In alcuni campi specifici ci riferiamo al parere di un esperto per calcolare la "distanza" fra i parametri di alcune feature, soprattutto se non numeriche. Si parla allora di **somiglianza.** Nella tabella a lato, vediamo la somiglianza relativa alla **febbre di un possibile paziente,** che può non esserci (*no*), esserci (*high*) o esserci in forma media (*average*). Su x, c = i valori che abbiamo; su y, q = i valori che vogliamo classificare. Se vogliamo classificare la febbre del paziente come *no*, è utile comparare i valori a quelli di un paziente con febbre *average* (somiglianza = 0.7). Ma il contrario non è vero nella stessa misura: se vogliamo classificare la febbre come *average*, è più utile paragonarla ad *high* (0.8) che a *no* (0.5). **Non c'è dunque simmetria.** Sempre in questi casi, le varie feature possono avere dei pesi *ad hoc,* per cui moltiplichiamo la somiglianza. La somiglianza totale sarà data dalle somme delle varie somiglianze * il rispettivo peso dei parametri.

Passiamo ora al **Pattern Discovering** e le conseguenti **Regole di Associazione,** il "cuore" del Data Mining – pre-esistente agli algoritmi di machine learning e ai cluster e totalmente **data-driven,** il cui uso paradigmatico è quello del **Market Basket Analysys.** Cosa c'è nel carrello della spesa? Ma è comunque applicabile in ogni contesto dove i "clienti" comprano più cose in prossimità (carte di credito, servizi di telecomunicazioni, servizi bancari, trattamenti medici… e in scenari scientifici, si usa per **esplorare possibili correlazioni di eventi**).

Given a database of customer transactions, where each one is a set of items, ***find groups of items which are frequently purchased together.***

È quindi necessario analizzare **ogni combinazione possibile?** A livello computazionale questo sarebbe molto difficile, soprattutto se guardiamo non solo coppie, ma triple, quintuple… Da qui le regole di associazioni, che esprimono come un oggetto è correlato con un altro e li raggruppano assieme. "Se un cliente compra …, allora comprerà anche …".

Questa informazione è **actionable:** porta direttamente ad azioni concrete nel mondo. E fornisce anche informazioni su una **parte precisa della popolazione analizzata:** le regole nello specifico non si applicano a tutti ("… *tutti* compreranno anche un birra"), ma a parte specifica della popolazione ("*chi compra pannolini,* compreranno anche una birra"). Le regole non creano un grande modello per descrivere il dataset, si limitano a **descrivere una sotto-popolazione,** che ha delle proprietà forti e particolari. Le informazioni che ricaviamo possono essere **di poco conto** o **inspiegabili,** e solo talvolta **utili** ("il giovedì vengono di solito comprati pannolini e birre insieme"). Notare che **la freccia indica co-occorrenza,** non implicazione. Inoltre, guardiamo appunto solo la co-occorrenza degli elementi, non la quantità specifica.

Market-Basket transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

Si parla spesso di (k-)**itemset,** ma cos'è? È un insieme di *k* oggetti (set of items). Si procede poi a contare **quante volte un'occorrenza di quel dato itemset appare nel database:** è il **Support Count**, cioè la frequenza assoluta. Si parla anche di **Support**, che è la frequenza relativa. Un itemset è ***frequente*** se il suo supporto è superiore a ***minsup***, il threshold minimo di supporto.

L'**Association Rule** è un'espressione di implicazione X -> Y, con X e Y che sono itemset. Ha come misura il **support** e la **confidence.** Entrambe sono formule di **probabilità.**

Prendendo l'esempio a lato:

Support: Quando Milk, Diaper & Beer possono essere osservati nell'intero set di transizioni (= nell'intero dataset).

Example:

$$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$$

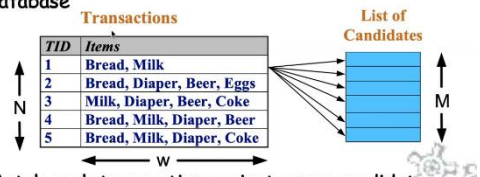
$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

Confidence: Quando Milk, Diaper & Beer possono essere osservati all'interno delle transizioni che hanno Milk & Diaper.

Cerchiamo poi le regole di associazioni che hanno suport e confidence superiori a un ***minsup threshold.*** Un approccio efficace è il **metodo forza bruta,** computando support e confidence di tutte le possibili regole di associazione ed eliminando quelle che non rispettano i requisiti di threshold. Ma è computazionalmente proibitivo.

- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database



  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since M = 2^d !!!

Per questo motivo si usa un approccio a due step: prima vengono computati tutti i support. Se è > minsup, si procede con la confidence. Il primo step rimane però computazionalmente dispendioso.

- Reduce the number of candidates (M)
  - Complete search: M=2^d
  - Use pruning techniques to reduce M
- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases
  - Used by DHP and vertical-based mining algorithms
- Reduce the number of comparisons (NM)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Il problema del metodo forza bruta è che confronta ogni possibile candidato M con ogni transazione N. Per essere più "parsimoniosi", o si riducono gli M, o si riducono le N (o il numero di comparazioni).

- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

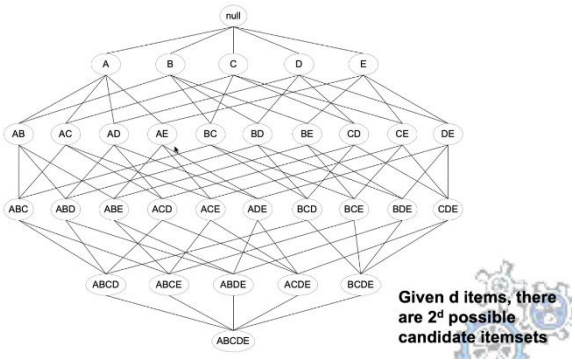
$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

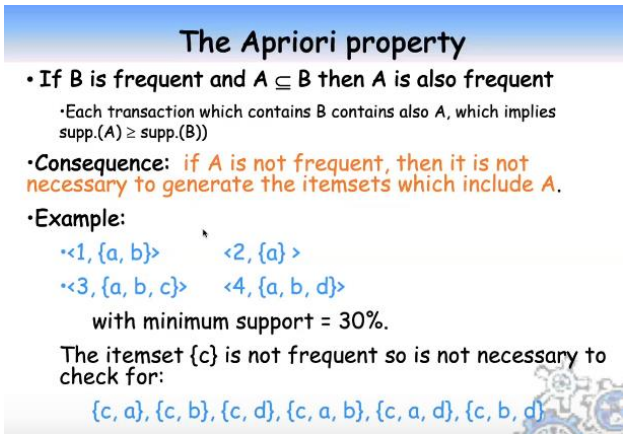
Il principio a priori riduce il numero di N. L'**algoritmo a priori** si basa su un assunto: **un subset di un itemset frequente è esso stesso un itemset frequente.** Se {A, B} è un itemset frequente, allora lo sono anche {A} e {B}. In modo iterativo possiamo così trovare gli intemset frequenti di cardinalità da 1 a k, senza computare ogni singola combinazione.

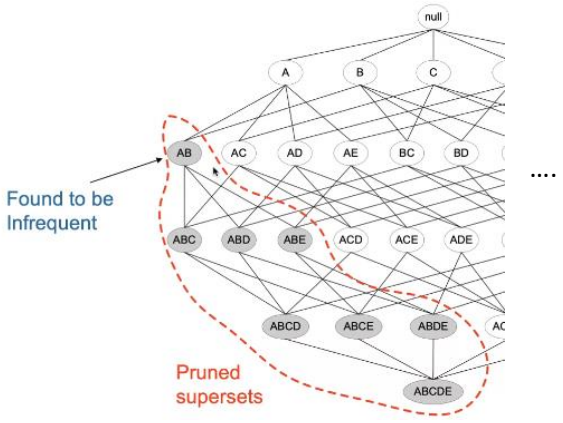
- $I=\{x_1, ..., x_n\}$ set of distinct literals (called items)
- $X \subseteq I$, $X \neq \varnothing$, $|X| = k$, X is called k-itemset
- A transaction is a couple $\langle tID, X \rangle$ where X is an itemset
- A transaction database TDB is a set of transactions
- An itemset X is contained in a transaction $\langle tID, Y \rangle$ if $X \subseteq Y$
- Given a TDB the subset of transactions of TDB in which X is contained is named TDB[X].
- The support(COUNT) of an itemset X, written $supp_{TDB}(X)$ is the cardinality of TDB[X].
- The support(relative) of an itemset X, written supp(X) is the cardinality of TDB[X]/ cardinality of TDB.
- Given a user-defined min_sup threshold an itemset X is frequent in TDB if its support is no less than min_sup.
- Given a user-defined min_sup and a transaction database TDB, the Frequent Itemset Mining Problem requires to compute all frequent itensets in TDB w.r.t min_sup.

Tutti i possibili "split":

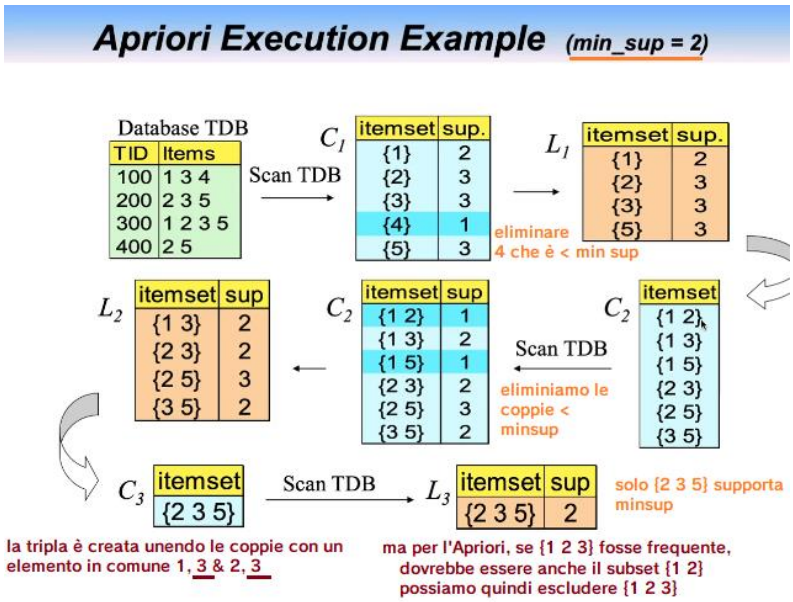


Una conseguenza dell'Algoritmo A Priori è che **il support di un itemset non è mai > di quello dei suoi subset.** Questo ci permette di fare economia di calcolo: se un itemset "padre" non è frequente, non lo saranno neppure i suoi subset.





Vediamo adesso l'Apriori in esecuzione. Se avessimo avuto più triple, avremmo potuto continuare. In sostanza, si "filtrano" subito le possibili triple (o quadruple ecc.) controllando il minsup degli elementi che le compongono, dei loro subset. L'output finale sarà la somma di tutti gli L, tutti gli elementi del dataset con un certo support. La scelta del minsup è molto importante: con uno troppo basso, si avrebbero tutte (o quasi) le combinazioni possibili – anche quelle date dal caso. **A causa della sparsità dei dai, si tende a prendere un minsup molto piccolo (ma non pari a 0): se si hanno tanti oggetti, ma transizione di piccole dimensioni, si hanno buoni risultati anche con un minsup basso.**



- **Join Step**: C_k is generated by joining L_{k-1} with itself
- **Prune Step**: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- **Pseudo-code**:

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) do begin
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
    increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
return $\cup_k L_k$;

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into C_k
  select $p.item_1$, $p.item_2$, …, $p.item_{k-1}$, $q.item_{k-1}$
  from L_{k-1} p, L_{k-1} q
  where $p.item_1=q.item_1$, …, $p.item_{k-2}=q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$
- Step 2: pruning

  forall *itemsets c in* C_k do
    forall *(k-1)-subsets s of c* do
      if *(s is not in* L_{k-1}*)* then delete *c* from C_k

- L_3={abc, abd, acd, ace, bcd}
- Self-joining: L_3*L_3
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*
- Pruning:
  - *acde* is removed because *ade* is not in L_3
- C_4={abcd}

Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Funziona anche su grandi dataset… grazie alla Course of Dimensionality! Anche con un dataset di centinaia di oggetti, nel carrello ce ne saranno una manciata. Questo produce un *sparsity effect.*

Come generare regole di associazioni forti? Vogliamo che soddisfino un *minimum confidence threshold*

- $confidence(A \Longrightarrow B) = \Pr(B \mid A) =$

```
For each frequent itemset, f, generate all non-empty subsets of f
For every non-empty subset s of f do
   if support(f)/support(s) ≥ min_confidence then
      output rule s ==> (f-s)
end
```

Alto dispendio computazionale, ma c'è qualche trucchetto.

- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
  - If {A,B,C,D} is a frequent itemset, candidate rules:

ABC →D,	ABD →C,	ACD →B,	BCD →A,
A →BCD,	B →ACD,	C →ABD,	D →ABC
AB →CD,	AC → BD,	AD → BC,	BC →AD,
BD →AC,	CD →AB,		

- If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

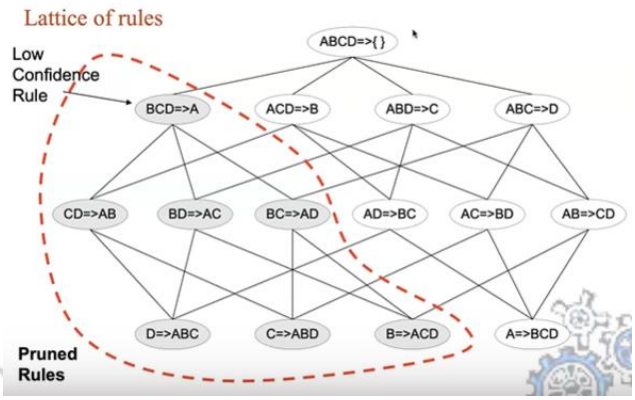
ATTENZIONE! La confidence non gode dell'anti-monotonicità. Tuttavia, confidence di regole generate **dallo stesso itemset** sì: cioè **muovendo un item dall'antecedente al conseguente**

ABC -> D e **AB -> D** non hanno alcuna relazione, non è lo stesso itemset. Ma c(ABC -> D) > c(AB -> CD). Più l'antecedente è grande, più è grande la confidence.

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property
    $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$
  - But confidence of rules generated from the same itemset has an anti-monotone property
  - e.g., L = {A,B,C,D}:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

✓ Confidence is anti-monotone w.r.t. number of items on the RHS of the rule



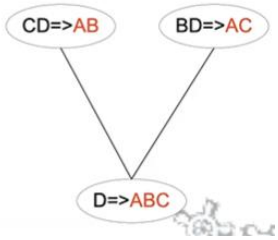
Si possono anche generare delle regole. Questo ci permette di "tagliare" più facilmente l'albero di regole.

Contingency table for $X \rightarrow Y$

	Y	$\overline{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\overline{X}$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	|T|

f_{11}: support of X and Y
f_{10}: support of X and $\overline{Y}$
f_{01}: support of $\overline{X}$ and Y
f_{00}: support of $\overline{X}$ and $\overline{Y}$

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent
- join(CD=>AB,BD=>AC) would produce the candidate rule D => ABC
- Prune rule D=>ABC if its subset AD=>BC does not have high confidence



Vediamo la tabella di esempio. La confidence è data da tutte le transazioni con caffè e tè (20) / tutte le transizioni con il tè in generale (25).

C (tea -> coffee) = 20/25 = 0.8 => 80%

Ma la regola ***tè -> caffè*** ci dice qualcosa di interessante? **No,** perché il caffè in generale ha probabilità del 90%. **La confidence di "assenza di tè" (not tea) -> caffè è superiore a quella di tè -> caffè.**

C (not tea -> coffee) = 70/75 = 0.93 => 93%

Il caffè è così popolare che il tè non lo determina.

Come sapere allora se due eventi sono **correlati?** Dobbiamo studiarne l'**indipendenza statistica,** attraverso il prodotto delle loro probabilità.

- Example 1: (Aggarwal & Yu, PODS98)

	coffee	not coffee	sum(row)
tea	20	5	25
not tea	70	5	75
sum(col.)	90	10	100

- {tea} => {coffee} has high support (20%) and confidence (80%)
- However, a priori probability that a customer buys coffee is 90%
  - A customer who is known to buy tea is less likely to buy coffee (by 10%)
  - There is a negative correlation between buying tea and buying coffee
  - {~tea} => {coffee} has higher confidence(93%)

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 420 students know how to swim and bike (S,B)
  - $P(S \wedge B) = 420/1000 = 0.42$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$
  - $P(S \wedge B) = P(S) \times P(B)$ => Statistical independence
  - $P(S \wedge B) > P(S) \times P(B)$ => Positively correlated
  - $P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated

Il **lift** permette poi di **normalizzare la probabilità di X dato Y considerando l'alto Y.** Nell'esempio di prima, **si considera così l'alta popolarità del caffè.** L'Interest è la stessa cosa, solo definita in modo diverso senza "passare" dalla Confidence

$$Lift = \frac{P(Y \mid X)}{P(Y)}$$

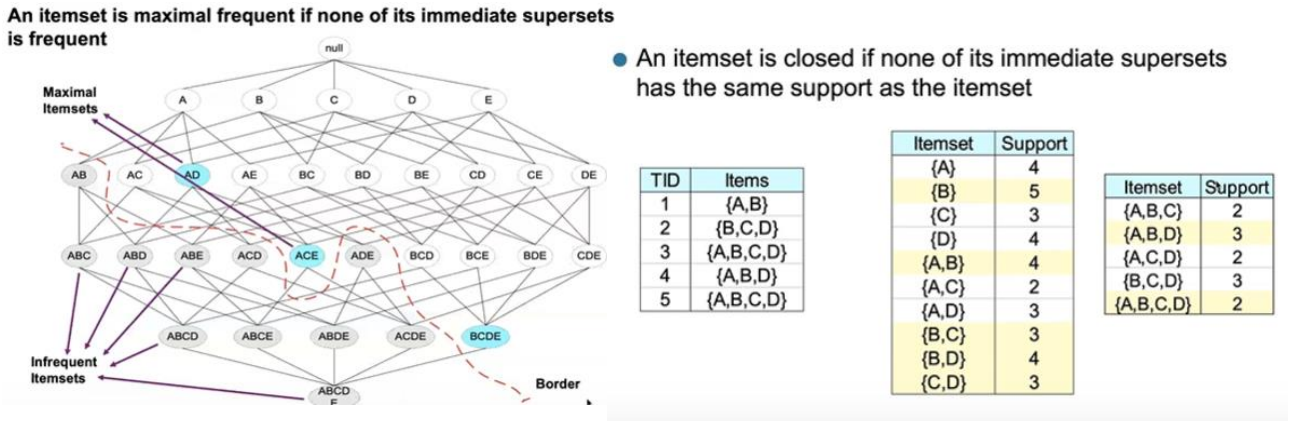
P(X,Y) / P(X) = P(Y|X) = Confidence

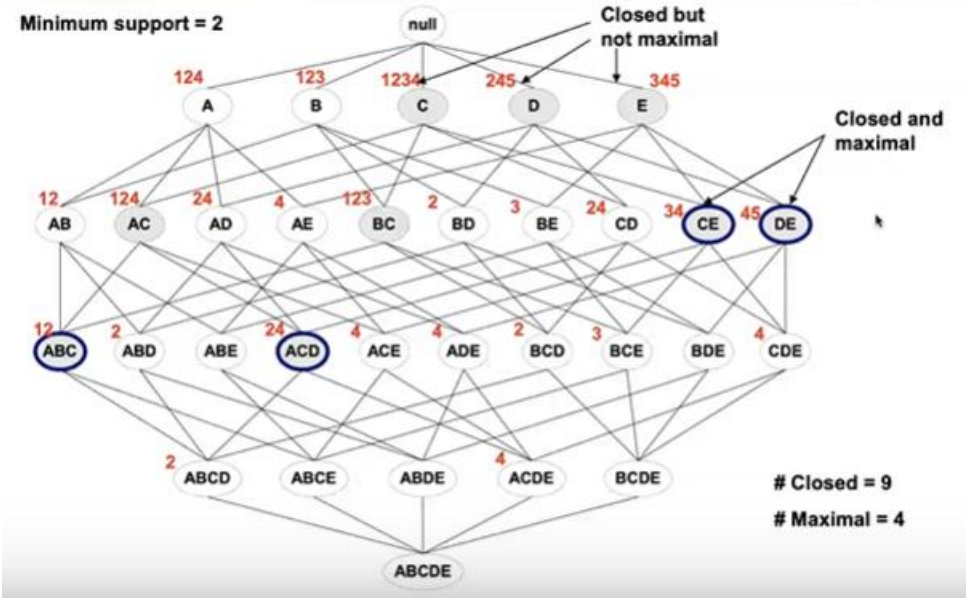
$$Interest = \frac{P(X,Y)}{P(X)P(Y)}$$

Perché non usiamo il Lift al posto della Confidence? **Perché non godono dell'Anti-monotonicità.** Non sarebbe quindi possibile alleggerire il calcolo computazionale con l'Apriori.

Le combinazioni interessanti sono quelle in cui X e Y avvengono insieme più volte di quando X e Y avvengono random.

Per quanto detto prima, possiamo considerare solo i **Maximal Frequent Itemset.** Ma è possibile anche un approccio più "soft": i **Closed Itemset** (vi sono inclusi anche gli itemset di "grado massimo", come {A B C D}). Solitamente un Max Itemset è anche Closed, mentre il contrario non è sempre vero. Grazie ai Closed possiamo ricostruire anche il support degli itemset che contengono (???).





Applicare le regole a dataset pluridimensionale necessita di **discretizzazione.**

Multi-dimensional → **Single-dimensional**

<1, Italian, 50, low> → <1, {nat/Ita, age/50, inc/low}>

<2, French, 45, high> → <2, {nat/Fre, age/45, inc/high}>

Schema: <ID, a?, b?, c?, d?>

<1, yes, yes, no, no> ← <1, {a, b}>

<2, yes, no, yes, no> ← <2, {a, c}>

- Quantitative attributes (e.g. age, income)
- Categorical attributes (e.g. color of car)

CID	height	weight	income
1	168	75,4	30,5
2	175	80,0	20,3
3	174	70,3	25,8
4	170	65,2	27,0

Problem: too many distinct values

Solution: transform quantitative attributes in categorical ones via discretization.

Solution: each value is replaced by the interval to which it belongs.

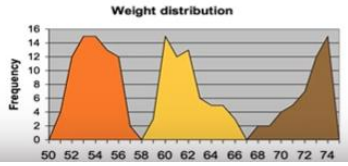
height: 0-150cm, 151-170cm, 171-180cm, >180cm
weight: 0-40kg, 41-60kg, 60-80kg, >80kg
income: 0-10ML, 11-20ML, 20-25ML, 25-30ML, >30ML

CID	height	weight	income
1	151-171	60-80	>30
2	171-180	60-80	20-25
3	171-180	60-80	25-30
4	151-170	60-80	25-30

Problem: the discretization may be useless (see weight).

How to choose intervals?

1. Interval with a fixed "reasonable" granularity
   Ex. intervals of 10 cm for height.
2. Interval size is defined by some domain dependent criterion
   Ex.: 0-20ML, 21-22ML, 23-24ML, 25-26ML, >26ML
3. Interval size determined by analyzing data, studying the distribution or using clustering



50 - 58 kg
59-67 kg
> 68 kg