1. Clustering by partitioning.

a. n: # data objects, k: # clusters

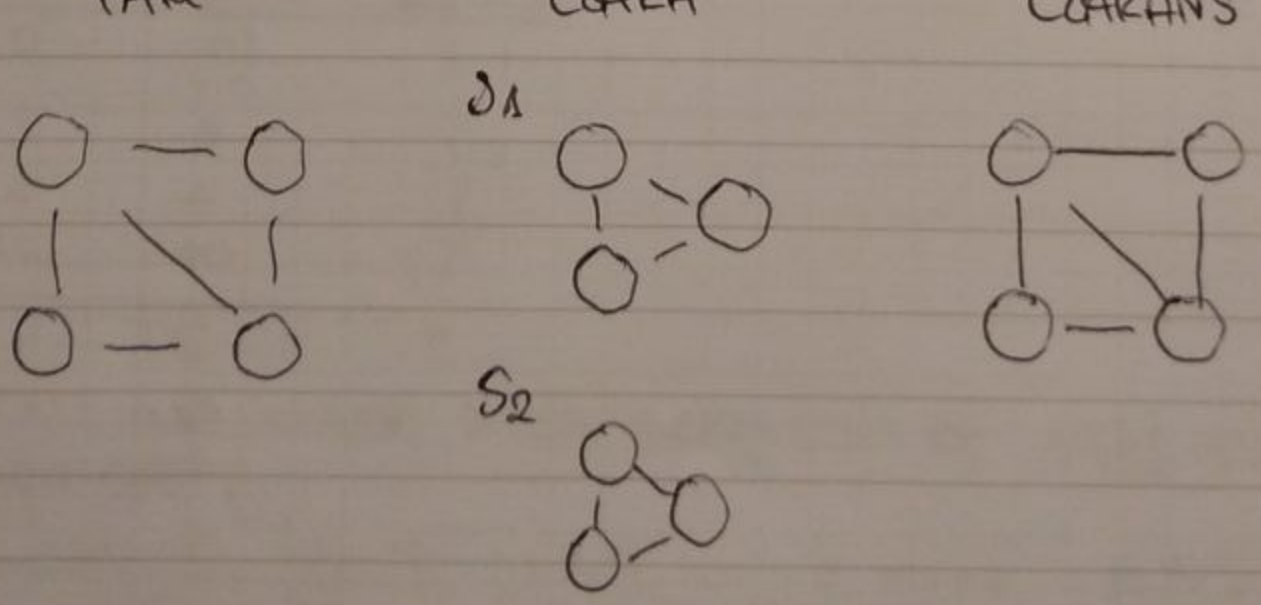
i. A node in the graph represent a combination of k objects which are the medoids so it represents a possible solution for the clustering problem.

ii. Two nodes are neighbors if they have k-1 same object so if they only differ by one element:

ex. {0,3} and {0,4} where k=2

iii. Considering PAM, CLARA and CLARANS algorithm we can say that PAM and CLARANS have same number of nodes because they use all data object in their algorithm to find the best solution. CLARA, instead, select a sample from the dataset and on that sample it builds a subgraph (in each iteration - n iterations). This means that CLARA will probably have more nodes.

PAM CLARA CLARANS



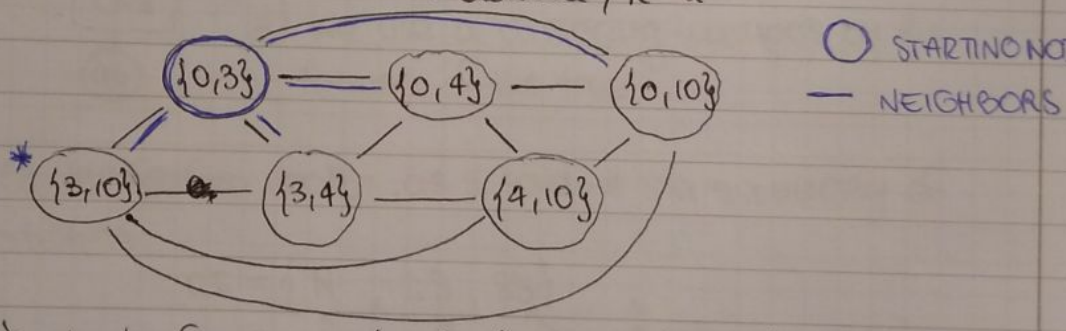
the difference is in how the algorithms acts on their neighborhood: PAM check all neighbors, CLARA only some neighbors in the subset, CLARANS select randomly MAXNEIGHBORS nodes to check on.

Node that make the second iteration

iv. PAM, CLARA and CLARANS find local optima in their algorithm. None of them guarantee to find the global optimum. CLARANS depends on the random selection of neighbors, CLARA is limitated to its subgraph at every iteration so it's even harder to find the best solution when not all objects are included in the subgraph. These partitioning methods are very efficient for a fast solution but they don't give the global optima.

2

b. $\{0,3,4,10\}$ Euclidean distance, $k=2$



Let's start from node $\{0,3\}$ so we know that our medoids are $m_1 = 0$, $m_2 = 3$. Let's calculate the total swapping cost of TC_{ih} where $i = 0,3$ and h is every not medoid.

	$m_1 = 0$	$m_2 = 3$
0	(0)	3
3	3	(0)
4	4	(1)
10	10	(7)
	TC =	8

3

Now we check each neighbors of $\{0,3\}$ and we calculate their cost.

$\{0,10\}$

	$m_1 = 0$	$m_2 = 10$
0	(0)	10
3	(3)	7
4	(4)	6
10	10	(0)
	TC =	7

$\{0,4\}$

	$m_1 = 0$	$m_2 = 4$
0	(0)	4
3	3	(1)
4	4	(0)
10	10	(6)
	TC =	7

$\{3,4\}$

	$m_1 = 3$	$m_2 = 4$
3	(3)	4
0	(0)	1
1	1	(0)
7	7	(6)
	TC =	9

$\{3,10\}$

	$m_1 = 3$	$m_2 = 10$
3	(3)	10
0	(0)	7
1	(1)	6
7	7	(0)
	TC =	4

The minimum TC found is 4 that belongs to $\{3,10\}$ so we start from that node the second iteration. *

2. Hierarchical clustering.

a) BIRCH algorithm is a hierarchical clustering method based on some important definition such as clustering feature vector and clustering feature tree. Like most of the hierarchical methods it builds a tree of cluster where in each parent node we find a summary of the most important statistics of that cluster and on the leaves nodes we find the objects which belong to that cluster. It's divided into two steps

CF1 (a6)

1) We build the CF Tree. ~~of all~~
2) We use a clustering methods on the leaves of the tree.

ii. A cluster feature vector, as I said, it's a summary of statistics:

$$CF = \{N, LS, SS\}$$

N: # of objects in a cluster

$LS = \sum_{i=1}^{N} x_i$

$SS = \sum_{i=1}^{N} x_i^2$

Using CF is very useful because it let's us understand if an object belong to a cluster very easily, for example we can easily find the centroid: $\frac{LS}{N}$

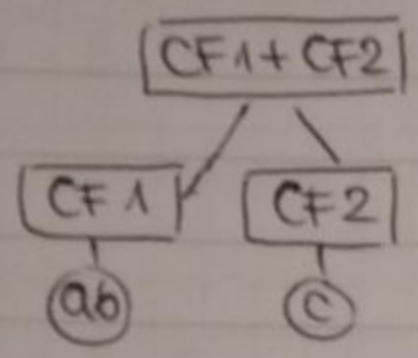
If we have (1,2), (1,3) and (2,2) the cluster feature vector is: $CF = \{3, (4,7), (6,17)\}$

$1+1+2$ $2+3+2$ $1^2+1^2+2^2$ $2^2+3^2+2^2$

When two clusters are merged their CF are summed up.

iii. A CF-tree is a tree of cluster where in each parent node is saved the CF of their children so the CF of the cluster and on the leaves node we find the objects which belong to that cluster. The three is height balanced so all the leaves are on the same line.

EXAMPLE :



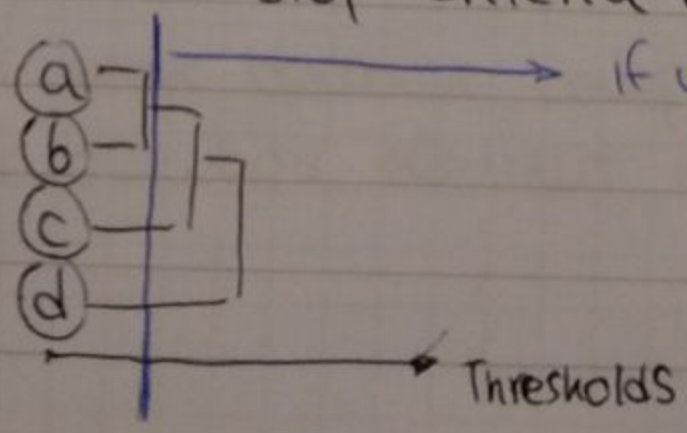
height balanced tree

In Birch is used in the second phase to assign an object to a cluster. When we have a new object we start from the root and we go through all the tree looking for the cluster which best represent our new object (ex. the one which has the closest centroid.)

Once we found the cluster we see if we can accomodate that object in that cluster, if it doesn't satisfy the threshold given we split the entry, if it doesn't satisfy the max diameter we split the leaf node, if its parent node is full we split that and so on. Once we ~~assessed the~~ accomodate the object we update all cluster features.

iv. The input parameters in BIRCH are T: threshold which indicates the max number of children a node can have and the branching factor (B) which indicates the maximum diameter.

b) Agglomerative ~~cluste~~ hierarchical clustering is a bottom up method of clustering. It starts with the idea that each object represents a cluster and keep merging clusters which are "more similar" base on distance matrix until all objects belong to the same cluster. As required parameter we need a threshold which represents the stop criteria of merging.



if we choose this we will have (ab), (c) and (d) as clusters.

	1	2	3	4	5
1	0				
2	5	0			
3	9	10	0		
4	3	2	6	0	
5	7	(1)	4	8	0

→ we use complete link clustering on the dissimilarity matrix

1) we check for the minimum in the matrix and we find that 1 is the minimum so we merge 2 and 5

	1	(2,5)	3	4
1	0			
(2,5)	7	0		
3	9	10	0	
4	3	8	6	0

$dist((2,5),1) = max\{dist(2,1), dist(5,1)\} =$
$= max(5,7) = 7$
$dist((2,5),3) = max\{dist(2,3), dist(5,3)\} =$
$= max(10,4) = 10$
$dist((2,5),4) = max\{dist(2,4), dist(5,4)\} =$
$= max(2,8) = 8$

2) we look for the minimum again and we find 3 so we merge 3 and 4.

	(1,4)	(2,5)	3
(1,4)	0		
(2,5)	8	0	
3	9	10	0

$dist((1,4),(2,5)) = max\{dist(1,(2,5)), dist(4,(2,5))\} =$
$= max(7,8) = 8$
$dist((1,4),3) = max\{dist(1,3), dist(4,3)\} =$
$= max(9,6) = 9$

3) Again the minimum is 8 so we merge (1,4) and (2,5)

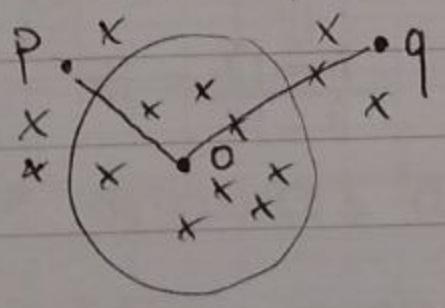
	(1,2,4,5)	3
(1,2,4,5)	0	10
3	10	0

$dist((1,2,4,5),3) = max(dist((1,4),3), dist((2,5),3)) =$
$= 10$

We used max because complete link finds the maximum distance between element in two clusters.

3. Density based clustering:
DBSCAN algorithm is a density based clustering method which creates cluster with high density objects and "isolate" objects which are not in an density area solving the outliers problem. This algorithm is based on some important definitions.

- The neighborhood of a point p is defined as
  $N_{Eps}(p) = \{q \text{ in } D \text{ such that } dis(p,q) \leq \varepsilon\}$ where ε is a given parameter.
- A core point is such that $|N_{EPS}(p)| \geq MinPts$ where MinPts is a given parameter indicating the minimum number of objects need around a point to make it a CORE POINT.
- A point p is direct density reachable from q if $p \in N_{Eps}(q)$ and $|Neps(q)| \geq minPts$ so q is a core point
- A point p is density reachable from q if exists a set of points $p_1, .. p_m$ $p_1 = q$ and $p_m = p$, such that p_{i+1} is directly density reachable from p_i where $i = 1, .. m-1$.
- A point is density connected to q if exist a point o such that both p and q are density reachable from o.



The algorithm creates cluster with density reachable points from core points. The main weakness of this method is the sensitivity to parameters. Setting ε is an important task to define density in a cluster.

4. Distance measures:

	A	B	C	D	E	F	G
K	(10,500)	(2,1,1)	1	0	1	0	8
L	(10,505)	(1,3,1)	1	1	0	0	-
distances:	5	1+2=3	0	1	1	0	-
delta:	1	1	1	1	1	0	0

$= \sum \delta^f = 5$

$$dist(K,L) = \frac{\sum_{f=1}^{7} \delta_{KL}^{(f)} d_{KL}^{(f)}}{\sum_{f=1}^{7} \delta_{KL}^{(f)}} = \frac{5+3+1+1}{5} = 2$$

- d_{KL} is the distance calculated for different variables measures
- δ_{KL} is the delta which is 0 if we have missing values or if we have $x=0$ for asymmetric binary variables.

b. Ordinal variables can be expressed as interval-based measure:

STEP I: we assign to each x_i the value of its rank in $r_i \in \{1, \dots M\}$

STEP II: we map those values in $[0,1]$ interval with the following formula:

$$z_i = \frac{r_i - 1}{M - 1}$$

We calculate distances as those variables were interval-based

EX: {short, Medium, Large} $M=3$

short → 1, Medium → 2, Large → 3

$z_1 = \frac{1-1}{3-1} = 0$ so short will be $z_1 = 0$

$z_2 = \frac{2-1}{3-1} = \frac{1}{2}$ so Medium will be $z_2 = 1/2$

$z_3 = 1$ so Large will be $z_3 = 1$

5.

(a) min supp = 1

Transaction Id	Items
1	A B C
2	A C D
3	A B
4	A B
5	A D
6	A D

(1) The items in the database are ordered in alphabetical order so we don't need to order them.

(2) Let's calculate

C_1

Items	support
A	6
B	3
C	2
D	3

all items are frequent because their support ≥ min support so they all go in the OUTPUT

L_1

A
B
C
D

C_2

Items	support
AB	3
AC	1
AD	2
BC	1
~~BD~~	—
CD	1

this item is infrequent so we PRUNE IT

L_2

AB
AC
AD
BC
CD

C_3

Items	support
ABC	1
~~ABD~~	
ACD	1

L_3

ABC
ACD

we don't have C_4 because ABC and ACD don't share prefix.

before checking the support we prune ABD because it doesn't pass the subset check. Its subset BD has been pruned in the previous step so it won't be frequent (A PRUNE PROPERTY)

OUTPUT : {A, B, C, D, AB, AC, AD, BC, CD, ABC, ACD}

2

6. Additional constraint: Find the frequent itemsets that do not contain the itemset CD
The constraint is antimonotone so we will prune the itemset which will contain CD because superset of them will not satisfy the constraint either.

C_1	Items	support		L_1
	A	6	✓	A
	B	3	✓	B
	C	2	✓	C
	D	3	✓	D

↑ all of them satisfy the constraint

C_2	Items	support	Additional constraint	L_2
	AB	3	✓ ok	AB
	AC	1	✓ ok	AC
	AD	2	✓ ok	AD
~~✗~~	~~BC~~	~~1~~	~~✗~~ ok	~~BC~~
	BD	–	INFREQUENT	BC
~~✗~~	~~CD~~	1	✗	

↓ does not satisfy additional constraint

C_3	Items	support	Additional constraint	L_3
	ABC	1	ok ✓	ABC
~~✗~~	~~ACD~~	~~1~~	~~✗ NO~~	
~~✗~~	~~ABD~~			

↓ subset check

↓ does not satisfy add. constraint

OUTPUT: { A, B, C, D, AB, AC, AD, BC, ABC }

C. Rule generation algorithm to the frequent itemset ABC confidence ≥ 50%

$C(AB \rightarrow C) = sup(ABC)/sup(AB) = 1/3 \leq 50\%$

$C(BC \rightarrow A) = sup(ABC)/sup(BC) = 1/1 = 100\%$ RULE

$C(AC \rightarrow B) = sup(ABC)/sup(AC) = 1/1 = 100\%$ RULE

we don't check rules $A \rightarrow BC$, $B \rightarrow AC$ because of APRIORI PROPERTY

$C(B \rightarrow AB) = 1/2 \geq 50\%$ RULE

6) minSupp = 1

(a)

ID	Items
1	CBA
2	DCA
3	AB
4	AB
5	AD
6	AD

1) check if they are frequent

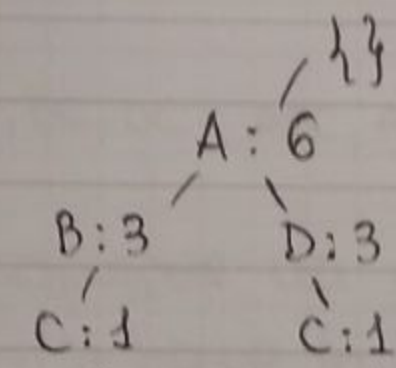
	sup.
A	6
B	3
C	2
D	3

2) order in decreasing support order

A B D C

ID	Items
1	ABC
2	ADC
3	AB
4	AB
5	AD
6	AD

3) Build a FP Tree



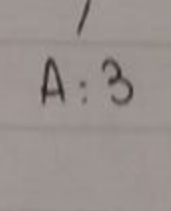
2

4) Conditional database : Given x at the left I report all items ending with x

A	-
B	A:3
D	A:3
C	AB:1, AD:1

(B) conditional database has {A:3}

Items : ∅ A
Support : 3
Order : A
output : AB

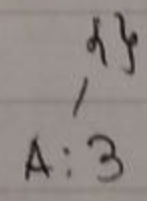


(AB) conditional database

Items : ∅ so BackTrack

(D) conditional database :

Items : A
Support : 3
Order : A
output : AD

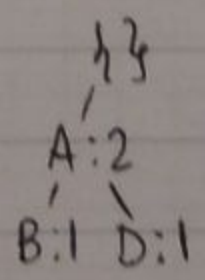


(AD) conditional database :

Items : ∅
so Back track

(C) conditional database :

Items : A, B, D
Support : 2, 1, 1
Order : ABD
Outputs : AC, BC, DC



(AC) conditional database

Items : ∅
back track

(BC) conditional database:
Items : A
support : 1
order : A
output : ABC

A:1, 1

(DC) conditional database:
Items : A
support : 1
order : A
output : ADC

A:1, 1

recursively I create all conditional databases

OUTPUTS : { AB, AD, AC, BC, DC, ABC, ADC }
A, B, C, D

b) If we want to incorporate an ANTIMONOTONE CONSTRAINT in the algorithm we first have to check, during the building of the tree, if it's satisfied by all items. We prune the items which don't satisfy the constraint in the first step. In the second step if α doesn't satisfy the constraint we don't report it's conditional database because their element won't satisfy it either. If α satisfy the constraint then we have to check if also the ~~items~~ do.

→ For MONOTONE constraint, instead, we can't ~~apply~~ prune items while building the tree but when we will valutate the conditional database if α satisfy the constraint we don't need to check it's conditional database, on the contrary if α doesn't then we need to check.

c. The FP grow algorithm is faster, save more time and save memory compared to the APRIORI because while APRIORI generate all possible candidates and then it valutate if they satisfy the properties, the FP algorithm generate a lower number of them based on the frequent patterns.

7. Constraints and causality:

a) CONVERTIBLE MONOTONE constraint:

$\underline{avg(S) \geq V}$ when the items are in decreasing order

CONVERTIBLE ANTIMONOTONE constraint:

$\underline{avg(S) \geq V}$ when items are in increasing order

CONVERTIBLE MONOTONE: if given an itemset A C(A) true then respecting a certain order R, C(B) true where A is a prefix of B.

We know that if A is a prefix of B and we are in decreasing order the constraint for A was lower than the one for B so if the average constraint was valid before will still be valid for B.

CONVERTIBLE ANTIMONOTONE, instead, B is a prefix for A but in this case with increasing order so if the average was $\geq v$ for A, then will still be valid for B (because $\underline{BA}$).

3

b) An association rule is casual: buying contact lens $\Rightarrow$ buying liquid to clean them

is casual because increasing the antecedent will increase the "buying" liquid" to clean them because it's the only solution we have to still using them. (ASSUMPTION and the only way to clean them.

0