SS-ZG548: ADVANCED DATA MINING

Lecture-06: Incremental Clustering

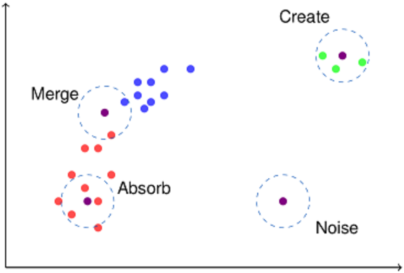
Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems Engineering,
BITS Pilani, Rajasthan-333031 INDIA

Jan 27, 2018 (WILP @ BITS-Pilani Jan-Apr 2018)

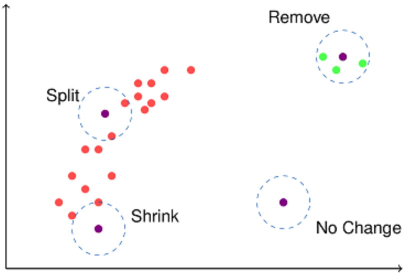
Recap: Incremental DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a spatial clustering algorithm of KDD96

- Parameters (Eps/MinPts) and points (core/border/noise)
- Uses DFS



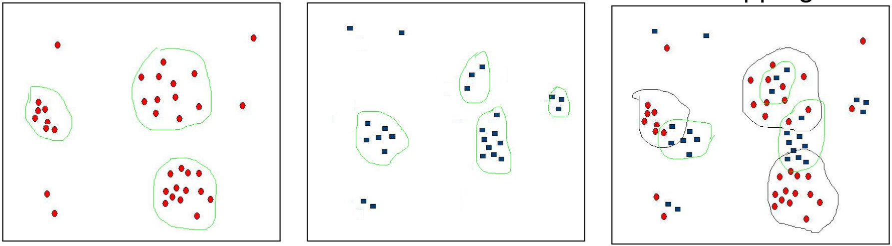
Insertion



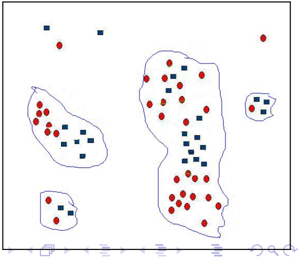
Deletion

Recap: Incremental DBSCAN

Initial Database | New Date | Overlapping



- Set of intersection I' contains those point $p \in \Delta$ for which $\exists$ a neighbor $p' \in D$
- It is necessary an sufficient to process all $p \in I'$
- How to efficiently compute I'? R-Tree



A question

Example: Assume you have a database that contains coordinates of points in a 2D plane. Now I give you one more point P and ask you the following question.

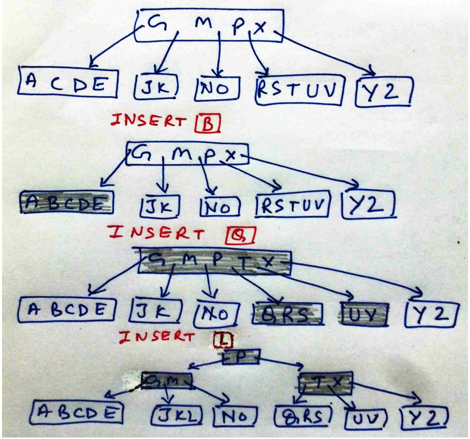
Give me the point from database which is less than 3cm away from P.

- What approach you would follow?
- Evaluate distance from all points in database and sort
- Evaluate distance from all points in database and take minimum
- some thing else?

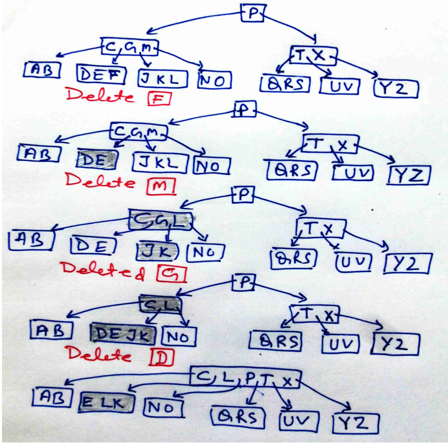
Recap: B-tree

Parameter k=2 specifies # keys a node can have, it is k-1 to 2k-1

Insertion



Deletion

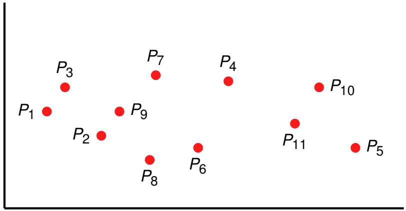


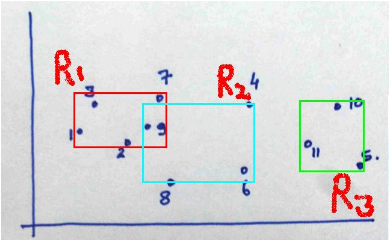
Height of tree $h \leq \log_t \frac{n+1}{2}$

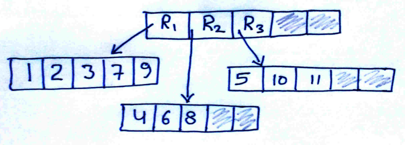
R-tree date structure

Consider arrival of

- P_1, P_2, P_3, P_4, P_5
- P_6 (split and region formation)
- P_7 (R1 expands)
- P_8 (R2 expands)
- P_9, P_{10}
- P_{11} (Split in R2)







R-tree an efficient date structure

- R-tree is an indexing approach to multidimensional spatial data
- Object near to a current location is easy to locate using R-tree
- Or finding all objects in vicinity
- Uses a minimum bounding rectangle (MBR) that is a smallest rectangle containing specified object
- Each node in the index contains its children
- Leaves of the tree points the actual objects
- R-Tree node usually corresponds to database points
- Similar to B-Tree
- Tree is height-balanced, so the height is $O(\log n)$

Incremental AR Mining Without Candidate Generation

- Two key issues with Apriori algorithms are as below
  - Costly to handle huge candidate sets
  - Tedious to repeatedly scan database
- Frequent Pattern tree (FP-tree) structure[1], which is an extended prefix tree structure for storing information about frequent pattern
- Three key things: (1) compression (2) avoid generation of candidate sets, and (3) decompose mining
- Method is efficient and scalable
- Other methods are CATS-tree, CP-tree

[1] Han, Jiawei and Pei, Jian and Yin, Yiwen, “Mining Frequent Patterns without Candidate Generation”, in ACM Sigmod 29(2) pages 1-12, ACM 2000

FP-tree at work

Tr ID	=	**{items}**
T_1	=	{A,B,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{A,B,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{A,B,C,E}
T_9	=	{A,B,C}

Let minimum support be 22%

FP-tree at work

Tr ID	=	{items}
T_1	=	{A,B,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{A,B,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{A,B,C,E}
T_9	=	{A,B,C}

Let minimum support be 22%,
Minimum support count is
9*22/100 = 1.98

FP-tree at work

Tr ID	=	{items}
T_1	=	{A,B,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{A,B,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{A,B,C,E}
T_9	=	{A,B,C}

Item	Frequency	Priority
A	6	
B	7	
C	6	
D	2	
E	2	

Let minimum support be 22%,
Minimum support count is
9*22/100 = 1.98

FP-tree at work

Tr ID	=	{items}
T_1	=	{A,B,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{A,B,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{A,B,C,E}
T_9	=	{A,B,C}

Item	Frequency	Priority
A	6	
B	7	
C	6	
D	2	
E	2	

Priority order: B, A, C, D, E

Let minimum support be 22%,
Minimum support count is
9*22/100 = 1.98

FP-tree at work

This would lead to a reordering

Tr ID	=	{items}
T_1	=	{A,B,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{A,B,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{A,B,C,E}
T_9	=	{A,B,C}

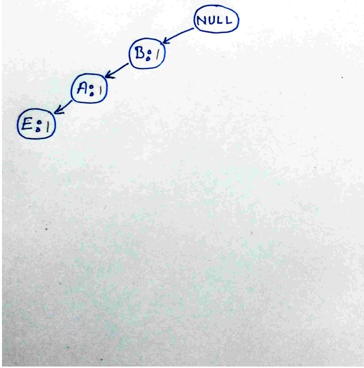
Priority order: B, A, C, D, E

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {B,A,E}

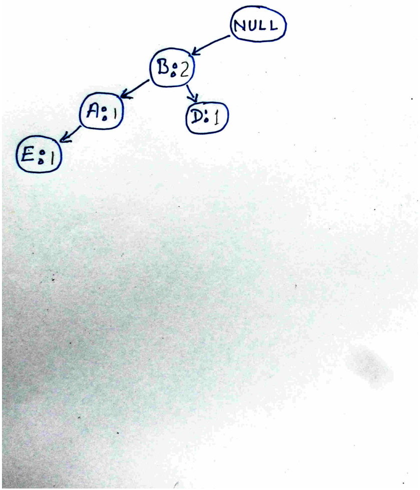


Next transaction {B,D}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {B,D}

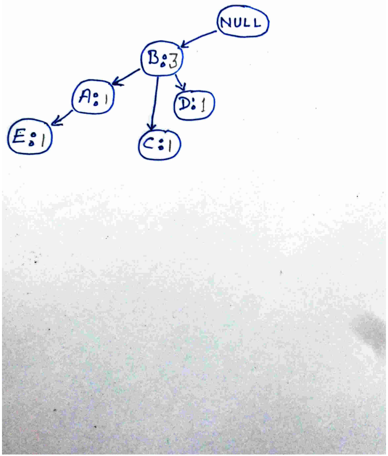


Next transaction {B,C}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {B,C}

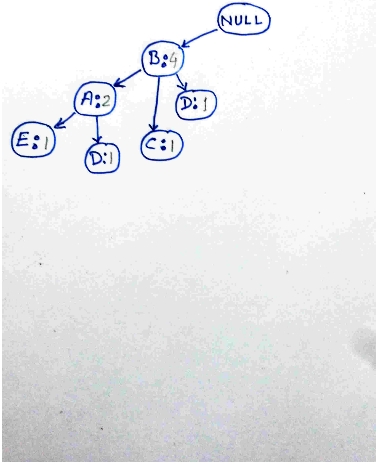


Next transaction {B,A,D}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {B,A,D}

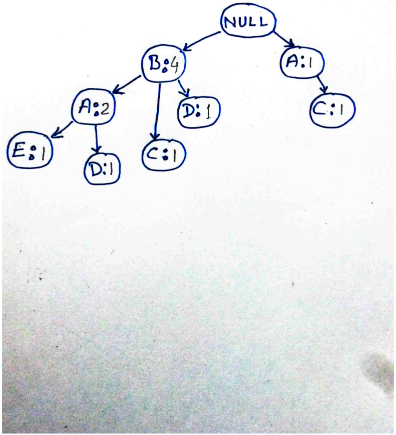


Next transaction {A,C}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {A,C}

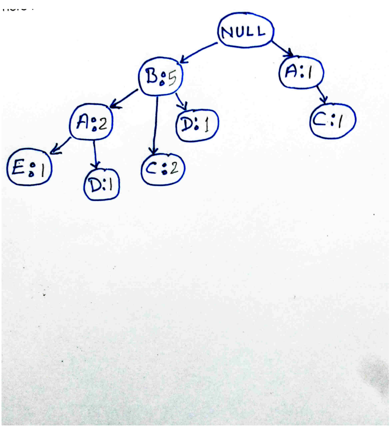


Next transaction {B,C}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {B,C}

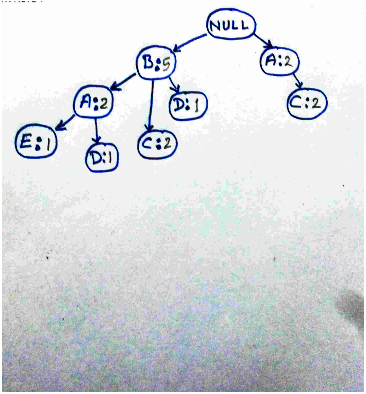


Next transaction {A,C}

FP-tree at work

Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {A,C}

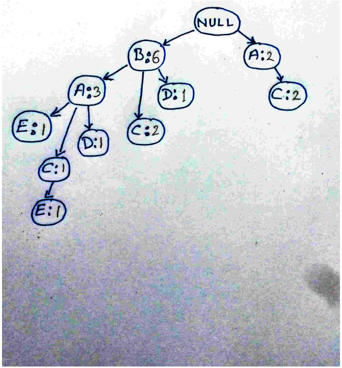


Next transaction {B,A,C,E}

FP-tree at work

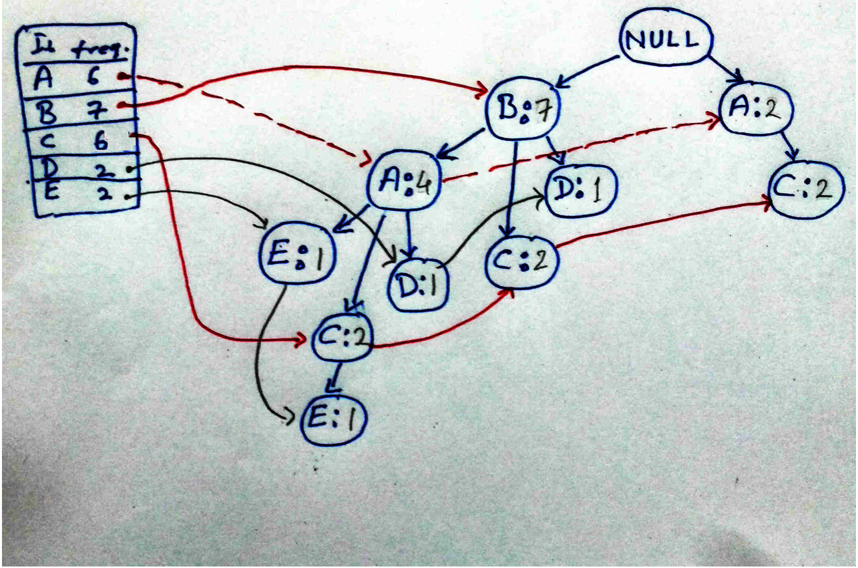
Tr ID	=	{items}
T_1	=	{B,A,E}
T_2	=	{B,D}
T_3	=	{B,C}
T_4	=	{B,A,D}
T_5	=	{A,C}
T_6	=	{B,C}
T_7	=	{A,C}
T_8	=	{B,A,C,E}
T_9	=	{B,A,C}

FP-tree with transaction {B,A,C,E}



Next transaction {B,A,C}

FP-tree at work



Create links

FP-tree at work

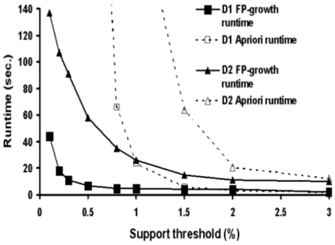
Item	Conditional Pattern Base	Cond. FP-Tree
E	{B,A:1}, {B,A,C:1}	(<B:2,A:2>)
D	{B,A:1}, {B:1}	(<B:2>)
C	{B,A:2}, {B:2}, {A:2}	(<B:4,A:2>, <A:2>)
B	{}	(Null)
A	{B:4}	(<B:4>)

FP-tree at work

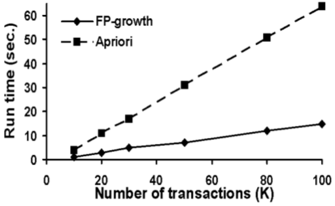
Item	Conditional Pattern Base	Cond. FP-Tree
E	{B,A:1}, {B,A,C:1}	(<B:2,A:2>)
D	{B,A:1}, {B:1}	(<B:2>)
C	{B,A:2}, {B:2}, {A:2}	(<B:4,A:2>, <A:2>)
B	{}	(Null)
A	{B:4}	(<B:4>)

- **E:** {B,E}, {A,E}, {A,B,E}
- **D:** {B,D}
- **C:** {B,A,C}, {B,C}, {A,C}
- **A:** {B,A}

FP-tree at work



Scalability with threshold.



Scalability with number of transactions.

CATS Tree

- CATS (Compressed and Arranged Transaction Sequences Tree) Tree [2] extends the idea of FP-Tree to improve storage compression and allow frequent pattern mining without generation of candidate itemsets.
- The proposed algorithms enable frequent pattern mining with different supports without rebuilding the tree structure
- Algorithms allow mining with a single pass over the database
- Handles insertion or deletion of transactions
- CATS Tree is a prefix tree and it contains all elements of FP-Tree including the header, the item links etc.

[2] Cheung, William and Zaiane, Osmar R, "Incremental mining of frequent patterns without candidate generation or support constraint", in Seventh International Database Engineering and Applications Symposium, pages 111–116, IEEE, 2003

Thank You!

Thank you very much for your attention!

Queries ?