Unsupervised learning: Data Mining

Association rules and frequent itemsets mining

Data Mining concepts

- Is the computational process of discovering patterns in very large datasets (rules, correlations, clusters)
- **Untrained** process: no previous knowledge is provided. Rules are learned from available data
- Will see two methods: rule learning and clustering

Examples

- Profiling buyers: “Ninety-percent of transactions that purchase bread and butter also purchase milk” (**IF** BREAD **and** BUTTER **then** Pr(MILK)=0.9)
- Medical domain: **IF** AGE>=70 **and** Smoke=Yes **and** Gender=M **then** RCA>=50 (Right Coronary Artery)
- Etc.
- Note that task is **unsupervised**: system provides SUPPORT and CONFIDENCE (see Decision Trees) for each learned rule, based on untagged data, but precision must be assessed by human experts

Association Rules

- Model and terminology is influenced by the main initial application domain: analyzing buyers transactions
- Input to the learner is a set of “transactions” consisting in a list of “purchased items”
- This model has a **general validity**: e.g. in the medical domain items are medical conditions and lifestyles (blood pressure, somoking, obesity..); in web users’ profiling items are visited web pages, in movie recommendation they are movies seen by users, etc.

Association Rule Mining

- **Definition**

  Finding frequent patterns, associations, correlations, or causal structures among sets of **items** or objects in **transaction** databases, relational databases, and other information repositories.

- **Applications**

  Basket data analysis, medical domain, cross-marketing, catalog design, clustering user profiles (e.g. Twitter Friends) , etc.

- **Rule form:**
  - Rule form: "Body → Head [support, confidence]".

  buys(x, "diapers") → buys(x, "beers") [0.5%, 60%]

  follows(x,"KatyPerry")^follows(x,"LadyGaga")➔>follows(x," BritneySpears") [0.7%,70%]

Overview

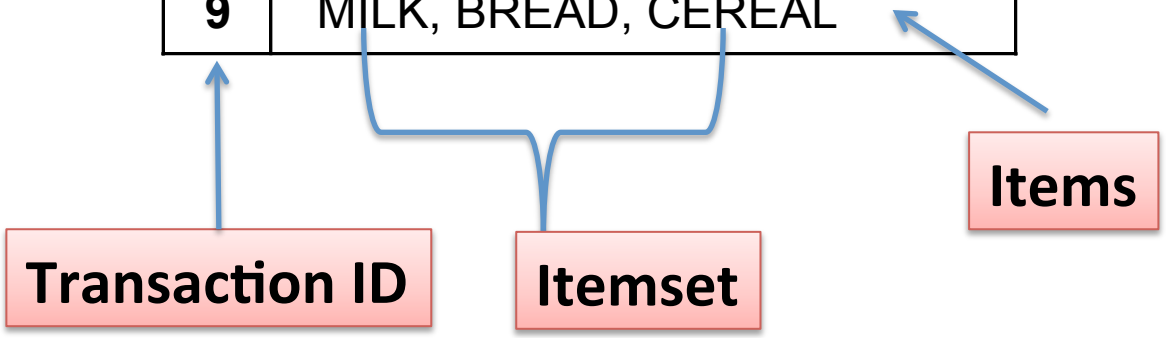
- **Basic Concepts of Association Rule Mining**
- The Apriori Algorithm (Mining single dimensional boolean association rules)
- Methods to Improve Apriori's Efficiency
- Frequent-Pattern Growth (FP-Growth) Method
- From Association Analysis to Correlation Analysis
- Summary

Association Model: Problem Statement

- **I** =$\{i_1, i_2, ..., i_n\}$ a set of **items**
- **J** = P(**I**)= $\{j_1, j_2, ..., j_N\}$ (powerset) set of all possible subsets of the set of items, $N=2^n$ (e.g: $j_l=\{i_k i_n i_m\}$)
- Elements of **J** are called **itemsets** (e.g: j_l={beer coke chips})
- **Transaction T_k (also called tuple): <*tid*,j>** *tid* is a transaction identifier, j is an element of **J**
- **Data Base: D** = set of transactions
- An **association rule** is an **implication** of the form : **X-> Y,** where **X, Y are disjoint** subsets of **I** (elements of **J**)

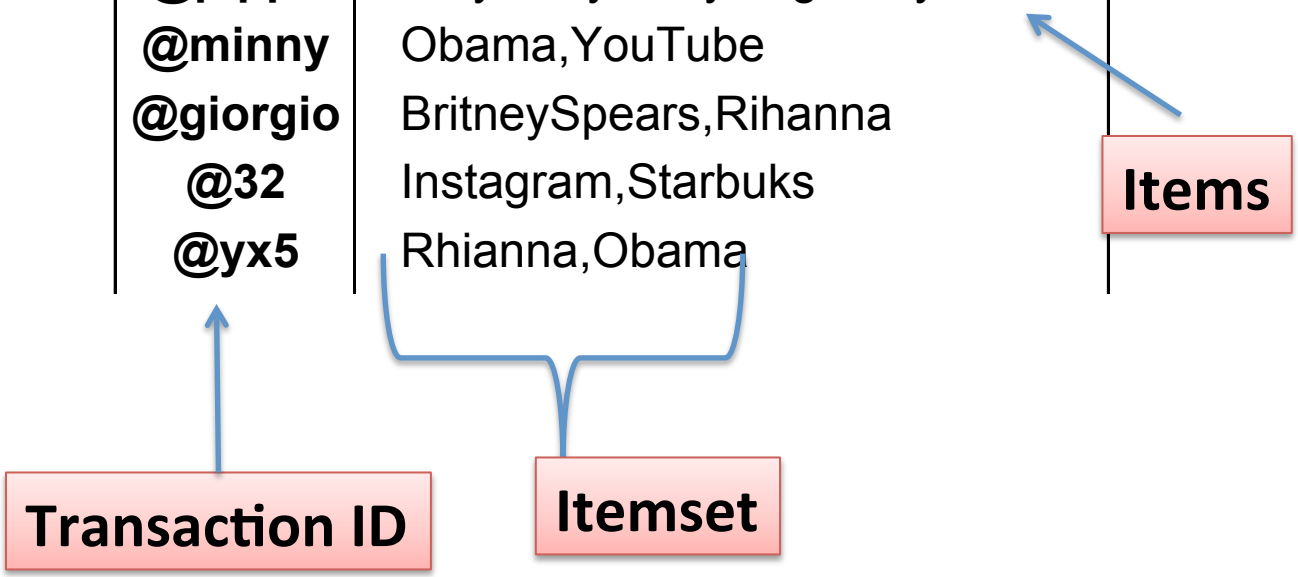
Transactions Example

TID	Purchased Product
1	MILK, BREAD, EGGS
2	BREAD, SUGAR
3	BREAD, CEREAL
4	MILK, BREAD, SUGAR
5	MILK, CEREAL
6	BREAD, CEREAL
7	MILK, CEREAL
8	MILK, BREAD, CEREAL, EGGS
9	MILK, BREAD, CEREAL



Another Transactions Example

USER	Likes
@pippo	KatyPerry,LadyGaga,TaylorSwift
@minny	Obama,YouTube
@giorgio	BritneySpears,Rihanna
@32	Instagram,Starbuks
@yx5	Rhianna,Obama



Transaction database: Example (neutral)

TID	Items
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

Transaction database: binary representation

Attributes converted to binary flags

TID	Items
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

TID	A	B	C	D	E
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

"1" in cell (i,j) means that the attribute j is included in the itemset of transaction i

Simpler ways to express rules

- First Order Logic:

follows(x,"KatyPerry")^follows(x,"LadyGaga")➔follows(x,"BritneySpears")

- Synthetic representation:

KatyPerry,LadyGaga➔BritneySpears

- Formally:

X➔Y $X, Y \subseteq J$

Rules accuracy estimates: Support & Confidence

- Simple Formulas:
  - Confidence (X→Y) = #tuples (transactions) containing both X & Y / #tuples containing only X , e.g,: **Pr(Y | X) = Pr(X U Y) / Pr (X)**
  - Support (X→Y) = #tuples containing both X & Y/ total number of tuples in the DB Pr(X U Y) (you can also specify absolute support, only denominator)
  - **Note** X, Y either items or itemsets (an item is a 1-dimensional itemset)
  - **Note** same definitions as for Dtrees

Support & Confidence

- How do we use support and confidence?
- Contrary to Dtrees, they are used for **seaching a solution**, rather than for evaluation:

  Find all the rules $X \rightarrow Y$ with confidence and support >= threshold
  - support, *s*, is the probability that a transaction contains {X, Y}
  - confidence, *c*, is the conditional probability that a transaction having {X} also contains Y
  - *Rules with support and confidence over a given threshold θ are called **strong rules***
  - *θ is a parameter*

Association Rules: finding "strong" rules

Q: Given a set of transactions, what association rules have min_sup = 2 and min_conf= 50% ?

TID	List of items
1	**A, B, E**
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	**A, B, C, E**
9	A, B, C

A, B => E supp=2 (or 2/9)

There are **2** transactions (support) with A and B and E (transactions 1 and 8)

A, B => E : conf=2/4 = 50%

(4 transactions include A,B: 1,4,8,9, of which only 2 also include E)

Association Rules Example (2)

- ***There are several rules that can be derived from itemset {A,B,E}, but only few meet the requirement minconf=50%***

  A, B => E : conf=2/4 = 50% (A,B: 1,4,8,9)

  A, E => B : conf=2/2 = 100% (A,E:1,8)

  B, E => A : conf=2/2 = 100% (B,E:1,8)

  E => A, B : conf=2/2 = 100% (E:1,8)

Other rules don't qualify (since confidence <50%)

A =>B, E : conf=2/6 =33%< 50% (A:1,4,5,7,8,9)

B => A, E : conf=2/7 = 28% < 50% (B:1,2,3,4,6,8,9)

⃠=> A,B,E : conf: 2/9 = 22% < 50% ({} : 1,2,3,4,5,6,7,8,9)

TID	List of items
1	**A, B, E**
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	**A, B, C, E**
9	A, B, C

Generating association rules from a set of transactions T

Usually consists of two subproblems :

1) **Finding frequent itemsets** whose occurences exceed a predefined minimum support threshold
2) **Deriving association rules** from those frequent itemsets (with the constrains of minimum confidence threshold)

- These two sub-problems are soleved iteratively until new rules no more emerge
- The second subproblem is quite straight- forward and most of the research focus is on the first subproblem
- Note that in real domains the main problem is computability over very large (big) data – parallelizable algorithms are preferred (e.g. http://dmkd.cs.vt.edu/papers/ICDMW10.pdf presents a MapReduce implementation)

Overview

- Basic Concepts of Association Rule Mining
- The Apriori Algorithm (Mining single dimensional boolean association rules)
- Methods to Improve Apriori's Efficiency
- Frequent-Pattern Growth (FP-Growth) Method
- From Association Analysis to Correlation Analysis
- Summary

The Apriori Algorithm: Basics

The Apriori Algorithm is an influential algorithm for mining frequent itemsets for boolean association rules.

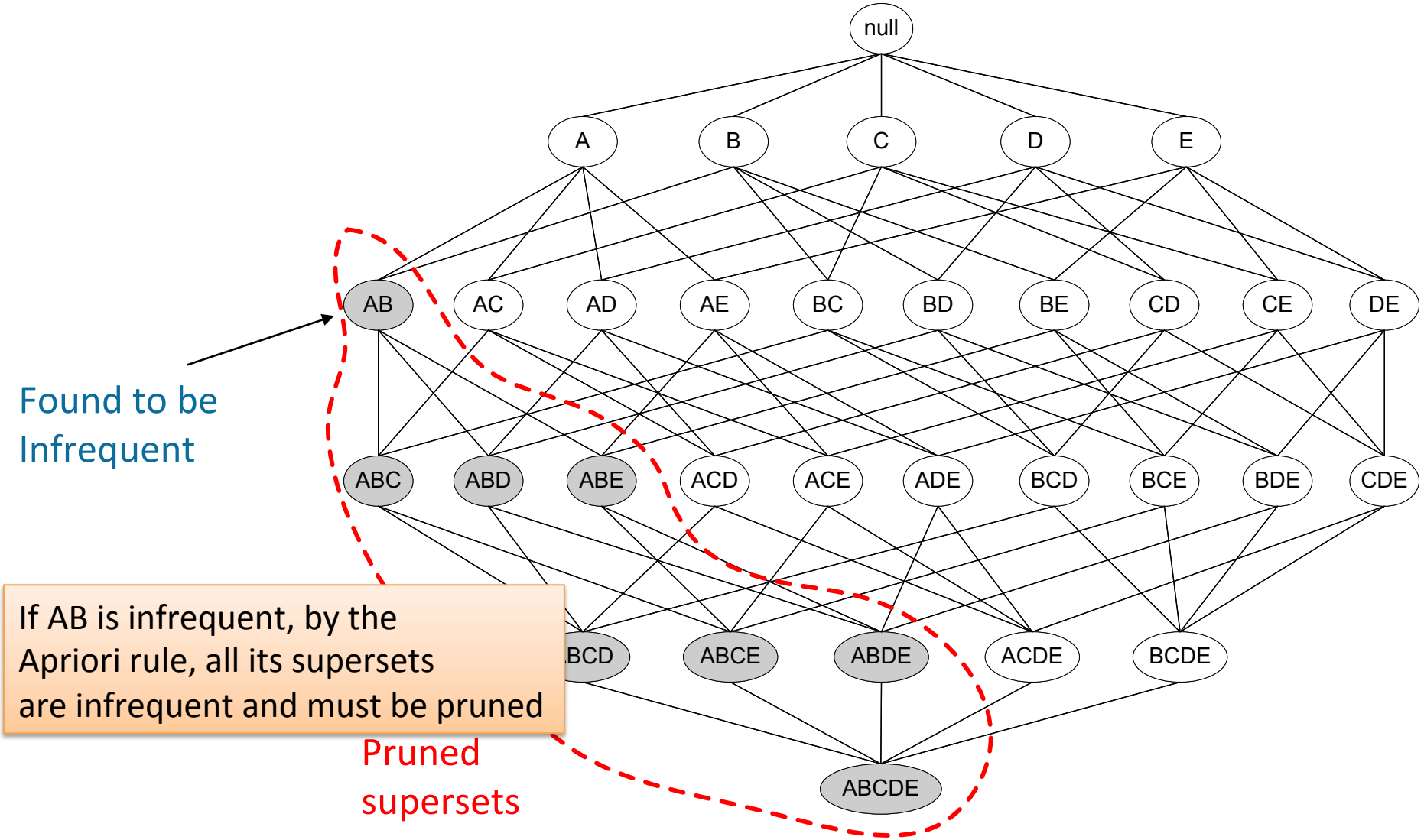
Key Concepts :

- Frequent Itemsets: The sets of items which meet the minimum support requirement (denoted by $\mathbf{L_k}$ for itemsets of **k** elements).
- Apriori Property: **Any subset of a frequent itemset must be frequent.** (if ABC is "frequent" according to a given min-supp constraint, clearly also AB, BC, AC, A, B and C must be)
- Join Operation: To find L_k , a set of **candidate** k-itemsets is generated by joining L_{k-1} with itself (e.g. if L_2 includes all and only item pairs with *freq*>=*min-supp*, then L_3 – because of the Apriori property- can only include triples generated by joining elements of L_2).

The Apriori Algorithm: 2 phases

1. **Find** the *frequent itemsets*: the sets of items that have minimum support
   - Sice a subset of a frequent itemset must also be a frequent itemset (Apriori property), **iteratively find frequent itemsets with cardinality from 1 to *k* (*k*-itemset)**
2. **Use** the frequent itemsets to generate association rules.

Illustrating Apriori Principle



The Apriori algorithm: why the name

- Initial information: transaction database D and user-defined numeric minimun support threshold *min_sup*
- Algorithm uses knowledge from previous iteration phase to produce frequent itemsets
- This is reflected in the Latin origin of the word *apriori* that means ”from what comes before”

The Apriori Algorithm: **Find** phase

- Let's define:

  C_k as a candidate itemset of size k

  L_k as a **frequent** itemset of size k
- Main steps of iteration are:

  To Find frequent itemset L_{k-1} (starting from L_1)

  1)Join step: C_k is generated by joining L_{k-1} with itself (cartesian product L_{k-1} x L_{k-1})

  2)Filter (Prune) step (Apriori property): Any (k − 1) size itemset that is not frequent cannot be a subset of a frequent k size itemset, hence should be removed from C_k

  Iterate: Frequent set L_k has been achieved, k:=k+1

The Apriori Algorithm: **Find** phase (2)

- Algorithm uses **breadth-first** search and a hash tree structure to handle candidate itemsets efficiently
- Then frequency for each candidate itemset is counted
- Those candidate itemsets that have frequency higher than minimum support threshold are qualified to be frequent itemsets

Apriori algorithm in pseudocode (Find phase)

Apriori Pseudocode

Apriori (T, ε)

$L_1 \leftarrow \{$ *large 1-itemsets that appear in more than* ε *transactions* $\}$

$k \leftarrow 2$

while $L_{k-1} \neq \varnothing$

$C_k \leftarrow$ *Generate*(L_{k-1})

for transactions $t \in T$

$C_t \leftarrow$ *Subset*(C_k, t)

for candidates $c \in C_t$

$\text{count}[c] \leftarrow \text{count}[c] + 1$

$L_k \leftarrow \{c \in C_k | \ \text{count}[c] \geq \varepsilon\}$

$k \leftarrow k + 1$

return $\bigcup L_k$

← **Join** step: create candidates from Lk_{-1}

← **Filter (prune)** step: check min_supp

The Apriori Algorithm: Example

TID	List of Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2 ,I3, I5
T108	I1, I2, I3

- Consider a database, D , consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. min_sup =ε= 2/9 = 22 %)
- Let minimum confidence required be 70%.
- We have first to find out the frequent itemsets using Apriori algorithm (**find** phase).
- Then, Association rules will be generated using min. support & min. confidence. (**use** phase)

FIND: Iteration 1- Generating 1-itemset Frequent Pattern

Scan D for count of each candidate

Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_1

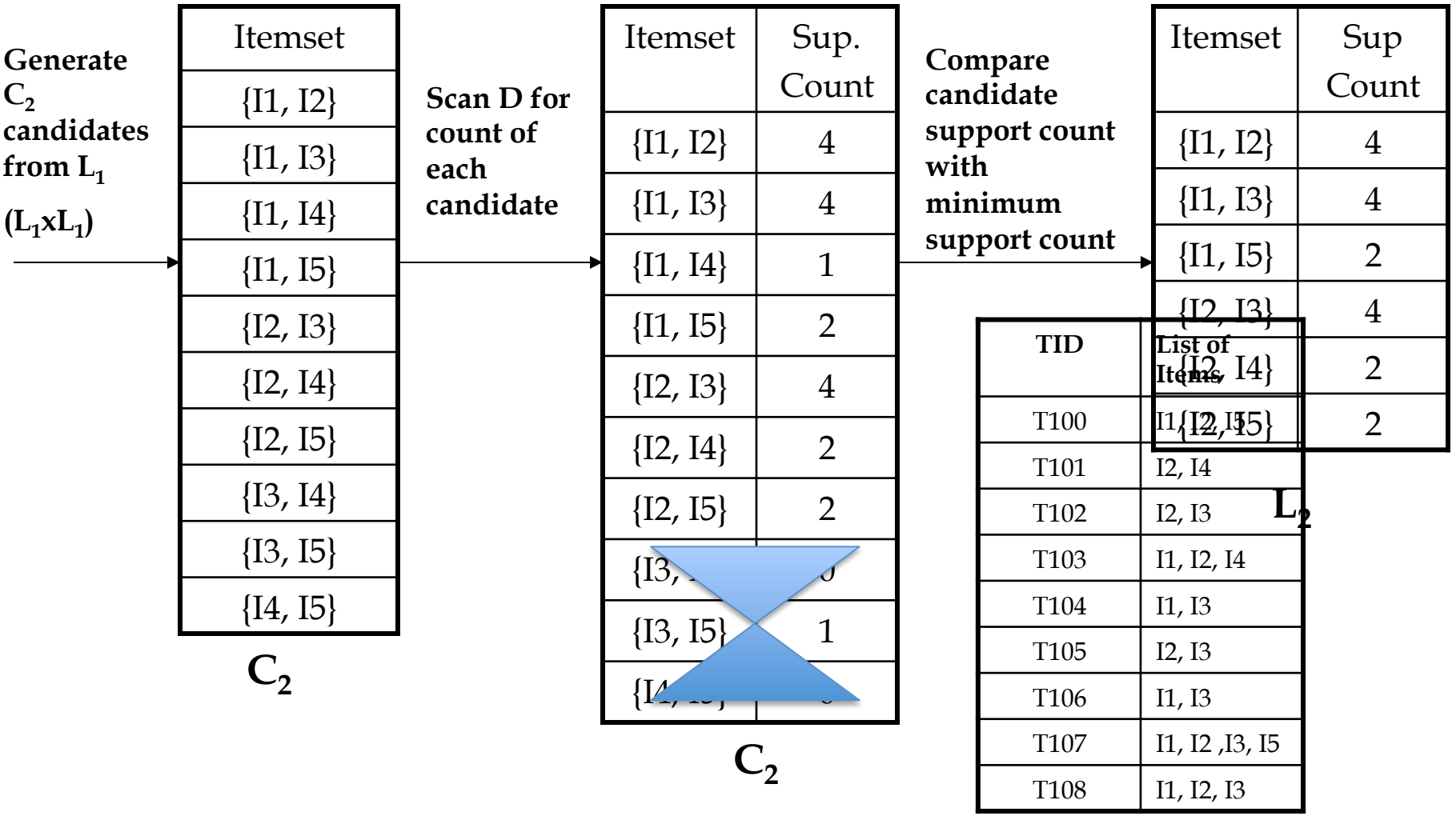
Compare candidate support count with minimum support count

Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

- According to the algorithm, the set of frequent 1-itemsets, L_1, consists of the candidate 1-itemsets satisfying minimum support.
- In this first iteration, all items in D are members of the set of candidate (they all meet the min-supp).

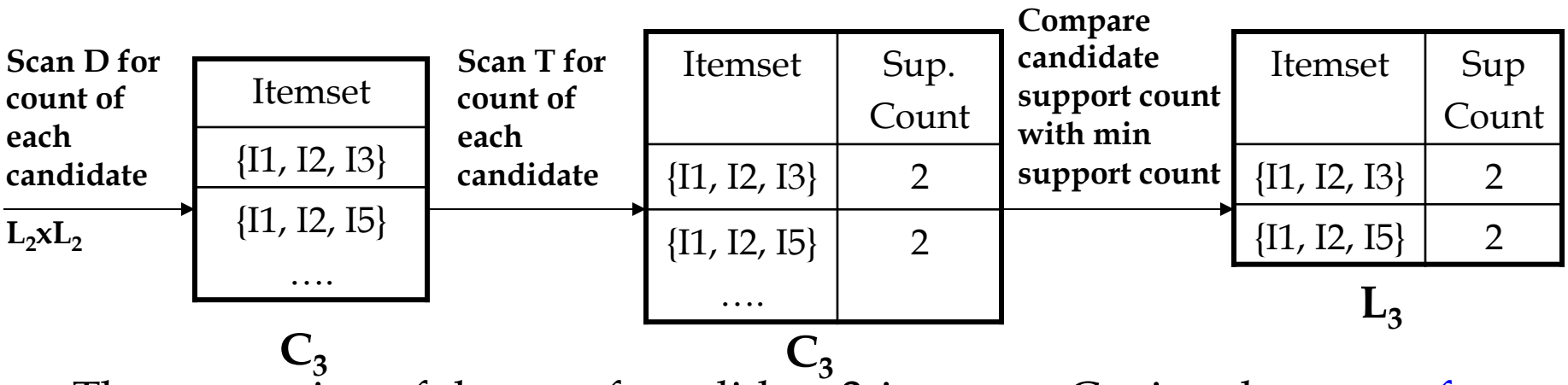
It 2: Generating 2-itemset Frequent Pattern



It 2: Generating 2-itemset Frequent Pattern [Cont.]

- To discover the set of frequent 2-itemsets, L_2 , the algorithm uses L_1 *Join* L_1 to generate a candidate set of 2-itemsets, C_2.
- Next, the transactions in D are scanned and the support count for each candidate itemset in C_2 is accumulated (as shown in the middle table).
- The set of frequent 2-itemsets, L_2 , is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.
- Note: **We haven't used Apriori Property yet.**

It 3: Generating 3-itemset Frequent Pattern



- The generation of the set of candidate 3-itemsets, C_3 , involves use of the Apriori Property (see next slide).
- In order to find candidates C_3, we first compute L_2 *Join* L_2.
- C_3 = L2 *Join* L2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}.
- Then, Join step is complete and Prune step will be used to reduce the size of C_3. Prune step helps to avoid heavy computation due to large C_k.

It 3: Generating 3-itemset Frequent Pattern [Cont.]

C_3= {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}.

- Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that four candidates in C_3 cannot possibly be frequent. How ?
- For example , lets take {I1, I2, I3}. The 2-item subsets of it are {I1, I2}, {I1, I3} & {I2, I3}. Since all 2-item subsets of {I1, I2, I3} **are members of** L_2, we will keep {I1, I2, I3} in C_3.
- Lets take another example of {I2, I3, I5} which shows how the pruning is performed. The 2-item subsets are {I2, I3}, {I2, I5} & {I3,I5}.
- BUT, {I3, I5} **is not a member of L_2** and hence it is not frequent violating Apriori Property. Thus we will have to remove {I2, I3, I5} from C_3.
- Eventually, we get C_3 = {{I1, I2, I3}, {I1, I2, I5}} after checking for all members of result of Join operation for Pruning.
- Finally, the transactions in D are scanned again in order to determine L_3 from C_3, consisting of those (survived) candidates 3-itemsets in C_3 having minimum support.

It 4: Generating 4-itemset Frequent Pattern

- The algorithm uses L_3 *Join* L_3 to generate a candidate set of 4-itemsets, C_4. Although the join results is {{I1, I2, I3, I5}}, this itemset is pruned since its subset {{I2, I3, I5}} is not frequent, thus violating the Apriori property.
- Thus, $C_4 = \phi$, and algorithm terminates, having found all of the frequent items. This completes our Apriori Algorithm.
- What's Next ?

  Phase 2 (use): These frequent itemsets will be **used** to generate strong association rules (where strong association rules satisfy both minimum support & minimum confidence).

USE phase: Generating Association Rules from Frequent Itemsets

- Procedure:
  - Let $L=UL_k$ be the set of frequent itemsets derived by Apriori during iterations k=1..n
  - For each frequent itemset ***"l"*** in L , generate **all** nonempty subsets of ***l.***
  - For **every** nonempty subset ***s*** of *l*, output the rule **"s → (l-s)"** for all subsets *l* and *s* such that:

    support_count(l) / support_count(s) >= min_confidence

- Back To Example:

  After phase 1 of the algorithm we obtain: **L** = {{I1}, {I2}, {I3}, {I4}, {I5}, {I1,I2}, {I1,I3}, {I1,I5}, {I2,I3}, {I2,I4}, {I2,I5}, {I1,I2,I3}, {I1,I2,I5}}.
  - Lets consider ***l*** = {I1,I2,I5}.
  - All its non-empty subsets are {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2}, {I5}.

USE phase: Generating Association Rules from Frequent Itemsets [Cont.]

l = {I1,I2,I5}, **s**: {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2}, {I5}

- Let minimum confidence threshold is , say 70% (means that, given rule *s*->*(l-s)* sc(l)/sc(s)>=0.7 where sc=support-count)
- The resulting association rules are shown below, each listed with its confidence.
  - R1: **I1 ^ I2 → I5 s → (l-s)**
    - Confidence = sc{I1,I2,I5}/sc{I1,I2} = 2/4 = 50%
    - **R1 is Rejected**.
  - R2: I1 ^ I5 → I2
    - Confidence = sc{I1,I2,I5}/sc{I1,I5} = 2/2 = 100%
    - R2 is Selected.
  - R3: I2 ^ I5 → I1
    - Confidence = sc{I1,I2,I5}/sc{I2,I5} = 2/2 = 100%
    - R3 is Selected.

Use phase: Generating Association Rules from Frequent Itemsets [Cont.]

- R4: I1 → I2 ^ I5
  - Confidence = sc{I1,I2,I5}/sc{I1} = 2/6 = 33%
  - R4 is Rejected.
- R5: I2 → I1 ^ I5
  - Confidence = sc{I1,I2,I5}/{I2} = 2/7 = 29%
  - R5 is Rejected.
- R6: I5 → I1 ^ I2
  - Confidence = sc{I1,I2,I5}/ {I5} = 2/2 = 100%
  - R6 is Selected.

  In this way, we have found three strong association rules: R2, R3 and R6

The process is repeated for all elements in L

Overview

- Basic Concepts of Association Rule Mining
- The Apriori Algorithm (Mining single dimensional boolean association rules)
- Methods to Improve Apriori's Efficiency
- Frequent-Pattern Growth (FP-Growth) Method
- From Association Analysis to Correlation Analysis
- Summary

Problems of Apriori

- To discover a 100-itemset
- 2^{100}-1 candidates have to be generated

$$2^{100}-1=1.27*10^{30}$$

(Do you know how big this number is?)

To improve efficiency:

- parallelization techniques (your Big Data course!)
- more efficient algorithms (next)

Efficiently Mining Frequent Patterns Without Candidate Generation: FP Growth

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - highly condensed, but complete for frequent pattern mining
  - avoid costly database scans
- **FP-Growth**: allows frequent itemset discovery without candidate itemset generation. Two-phase approach:
  - Phase 1: Build a compact data structure called the FP-tree
    - Built using 2 passes over the data-set.
  - Phase 2: Extracts frequent itemsets directly from the FP-tree

FP-Growth Method: Construction of FP-Tree

- First, create the root of the tree, labeled with "null".
- Scan D to create 1-itemsets and then L_1 (list of single items ordered by frequency)
- Scan the database D a second time.
- Re-order the items in each transaction **according to their frequency in D**. E.g., if t_j=abc and freq(b)>freq(c)>freq(a) then re-order as t_j=*bca*
- A branch of the tree is created for each (re-ordered) transaction: nodes in the branch are in decreasing frequency order, as for the transaction string, and each node of the branch is labeled with an item having its support count separated by colon (e.g.: b:6➔c:4➔a:3).
- Any new transaction (and its related branch) is first overlapped with existing branches (starting from the prefix of the sequence), and if **no overlapping** is possible, **a new branch is created**. Whenever an overlapping is possible, we just increment the support count **of the common nodes**.
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.
- Now, the problem of mining frequent patterns in database is transformed to that of mining the FP-Tree (step 2 of algorithm, see later).

Kind of confused?

Example FP-Growth (step 1)

Suppose min-support=3

TID	Items
1	E, A, D, B
2	D, A, C, E, B
3	C, A, B. E
4	B, A, D
5	D
6	D,B
7	A,D,E
8	B,C

Item	Frequency	
A	5	3
B	6	1
C	3	5
D	6	2
E	4	4

Red numbers are the order

Step 1. Count item frequency and order items by support

Red numbers are the order of items (by support)
The order is then **B->D->A->E->C**

Example (step 2)

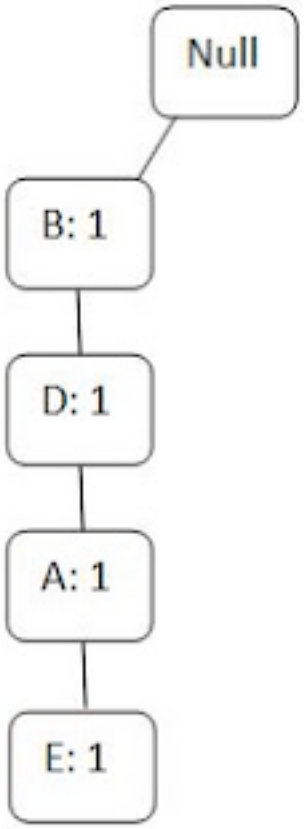
TID	Items	Ordered Items
1	E, A, D, B	B,D,A,E
2	D, A, C, E, B	B,D,A,E,C
3	C, A, B. E	B,A,E,C
4	B, A, D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C

Step2. Items are ordered in each transaction according to their support

(remember from previous slide: frequency order is **BDAEC**)

Example (step 3): create FP-tree

TID	Items	Ordered Items
1	E, A, D, B	B,D,A,E
2	D, A, C, E, B	B,D,A,E,C
3	C, A, B. E	B,A,E,C
4	B, A, D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C



- The root node is a **null** node
- A branch is created for each transaction with items having their support count separated by colon.
- We start with T1: every node is an item with its count, in order of decreasing support

Example (Step 3 sub-example)

Suppose we have 3 ordered transactions. Each transaction is a branch, as shown.

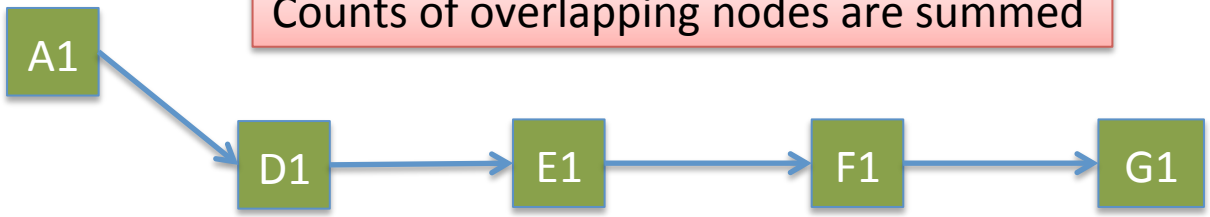
The orange branch has been overlapped with **prefix** of blue branch, and counts are updated

Prefix: the **head** of a string Suffix: the **tail** of a string (lenght of head is any value >= than the string length)

Example

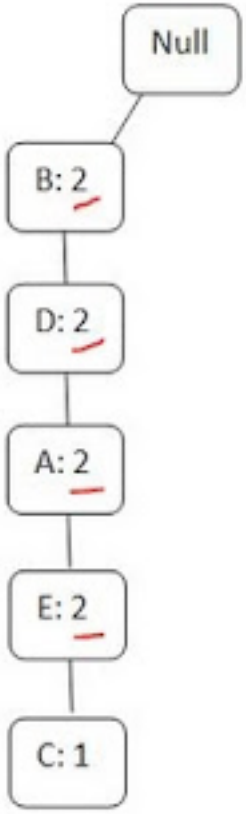
Now we try to overlap the blue and green branches. Note that overlapping is only possible if there is a common PREFIX between the branches

Counts of overlapping nodes are summed



Back to previous example (step 4): create FP-tree

TID	Items	Ordered Items
1	E, A, D, B	B,D,A,E
2	D, A, C, E, B	B,D,A,E,C
3	C, A, B. E	B,A,E,C
4	B, A, D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C



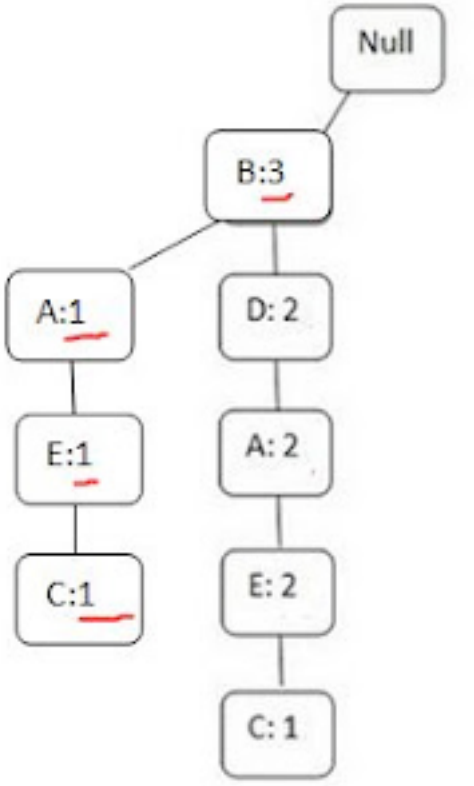
Now consider **T2:**

First 4 items are in the same order as T1, so we can overlap the **prefix** BDAE of T2 with the existing branch, incrementing counts, and adding C at the end.

Note: we check for prefix overlapping!! E.g. if T2 was DAEC we could NOT overlap.

Example (5) : create FP-tree

TID	Items	Ordered Items
1	E, A, D, B	B,D,A,E
2	D, A, C, E, B	B,D,A,E,C
3	C, A, B. E	B,A,E,C
4	B, A, D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C



In **T 3** the order is BAEC

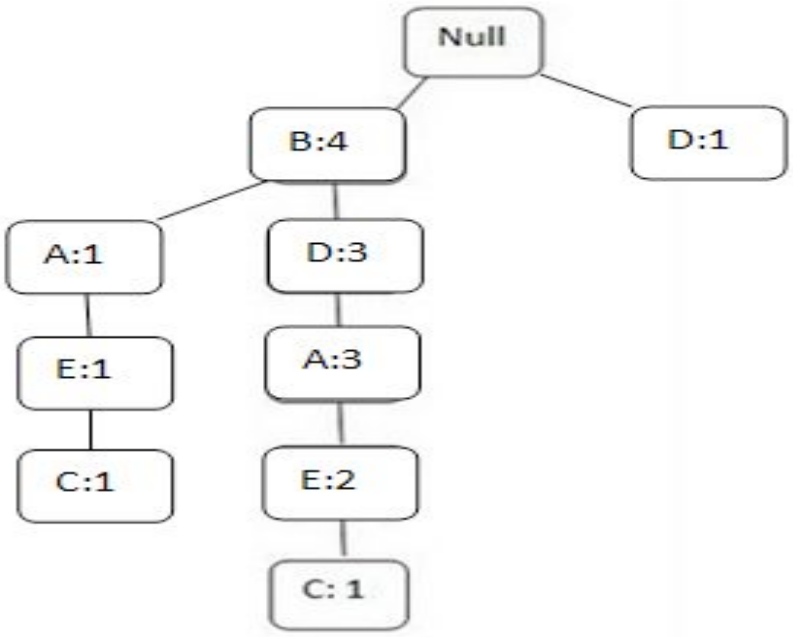
But can't overlap AEC to existing branch overtaking D (which is morte frequent)!

So the only prefix overlap is B

As a result we draw another branch from B, where we add A and then connect new E to that A and new C to new E.

Example (6) : create FP-tree

TID	Items	Ordered Items
1	E, A, D, B	B,D,A,E
2	D, A, C, E, B	B,D,A,E,C
3	C, A, B. E	B,A,E,C
4	B, A, D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C



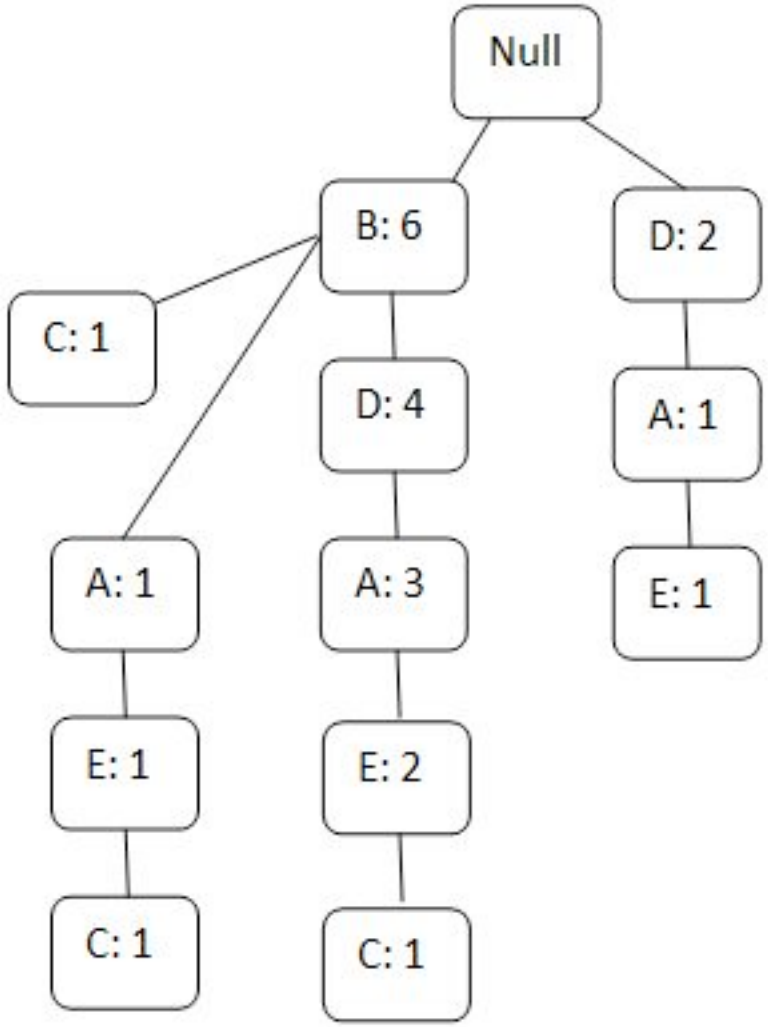
T 4 (BDA) fully overlaps with prefix of our first created branch, so we only need to update counts

With T5 (D) we need to add a third branch (D:1)

Example (7): Final FP-tree

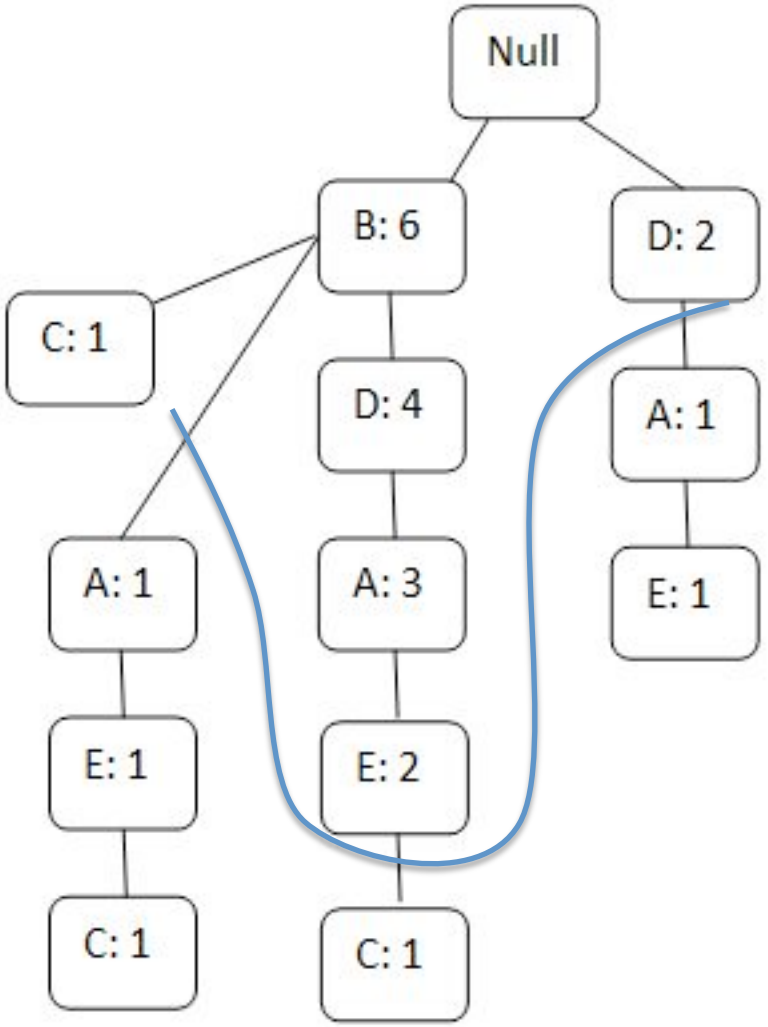
TID	Items	Ordered Items
1	E, A, D, B	B,D,A,E
2	D, A, C, E, B	B,D,A,E,C
3	C, A, B. E	B,A,E,C
4	B, A, D	B,D,A
5	D	D
6	D,B	B,D
7	A,D,E	D,A,E
8	B,C	B,C

T6 , T7 and **T8** follow the same procedure



FP-Tree properties

- Note that we can generate itemsets scanning branches bottom up!
- **The frequency of an itemset is determined by its lowest frequency item in a tree brunch prefix**
- E.g. CEADB:1 EADB:2 ADB:3 ecc
- Furthermore itemset with min_supp are easily identified (e.g. min_supp 2 are EADB, ADB, DB, B and D)



Now the first phase is concluded

- Remember: in Phase 1 we rearrange all itemsets creating the FP-tree
- We did this scanning the itemsets twice:
  - First, to compute the frequency of single items and reordering transactions
  - Second, to create the FP-tree by incrementally adding transactions

Phase 2: Mining the FP-Tree by Creating Conditional (sub) pattern bases

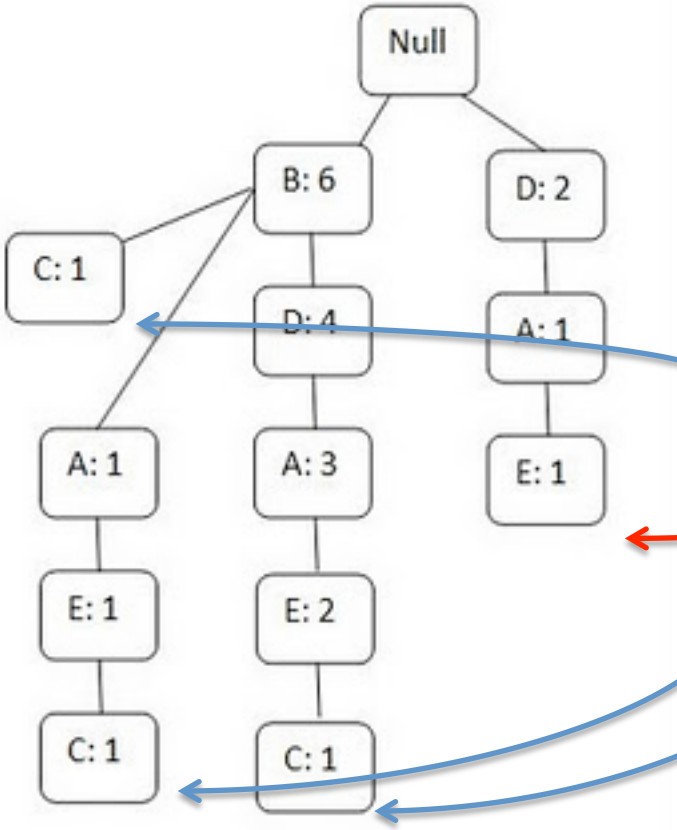
Prefix: the **head** of a string **Suffix**: the **tail** of a string

Steps:

1. We now consider suffix patterns in the FPT: we start from frequent length-1 suffix patterns.
2. For any suffix of length k, construct its conditional pattern base CPB which consists of the set of prefix paths in the FP-Tree co-occurring with that suffix pattern (e.g. given branches a➔ b➔ **c**, a➔ d➔ e➔ **c** if **c** is the suffix and its CPB is **ab, ade**.)
3. Build the **F-list** (list of all single items in CPB with their counts) and remove those with supp<min_supp
4. Use the F-list to construct a conditional F-Tree for the suffix pattern, using CPB and F-list, and perform mining on such a tree.
5. The pattern growth is achieved by concatenation of the initial suffix with its F-Tree.
6. The union of all frequent patterns (generated by step 5) gives the required frequent itemset.
7. Repeat for all suffixes

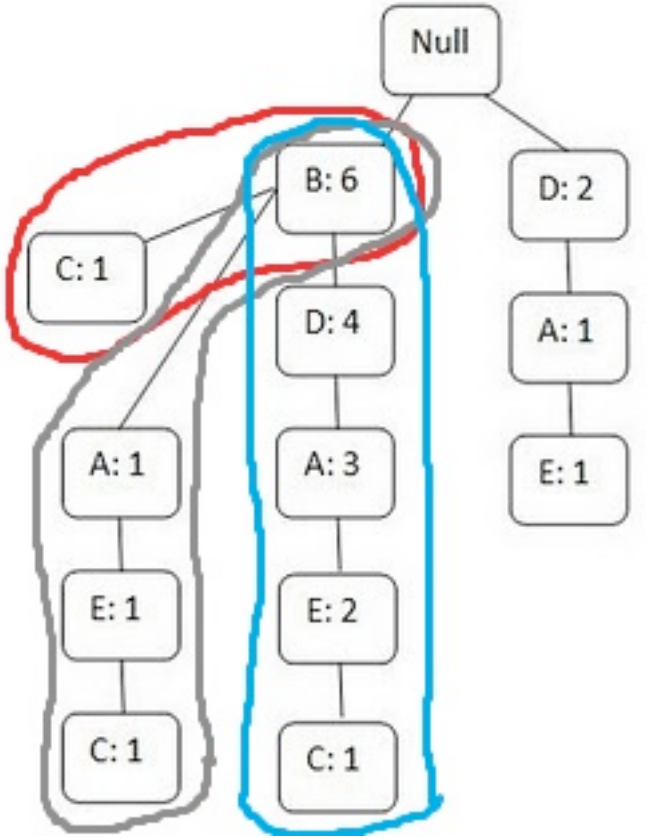
Kind of (even more) confused?

Example : 1. Start from each frequent length-1 suffix pattern.



The L1 Items and their frequency of occurrences are:
B:6 D:6 A:5 E:4 C:3
Minimum support count is 3 (as previously stated) so all can be considered as frequent lenght-1 suffixes;
Start with the leaves of tree (**initial suffix patterns**), eg. C, E

Example :2. Build Conditional Pattern Bases



We start with the first suffix item, C

C:3 There are 3 patterns with C (gray, red and blue)

The conditional pattern base (CPB) for C are: BDAE:1, B:1 BAE:1

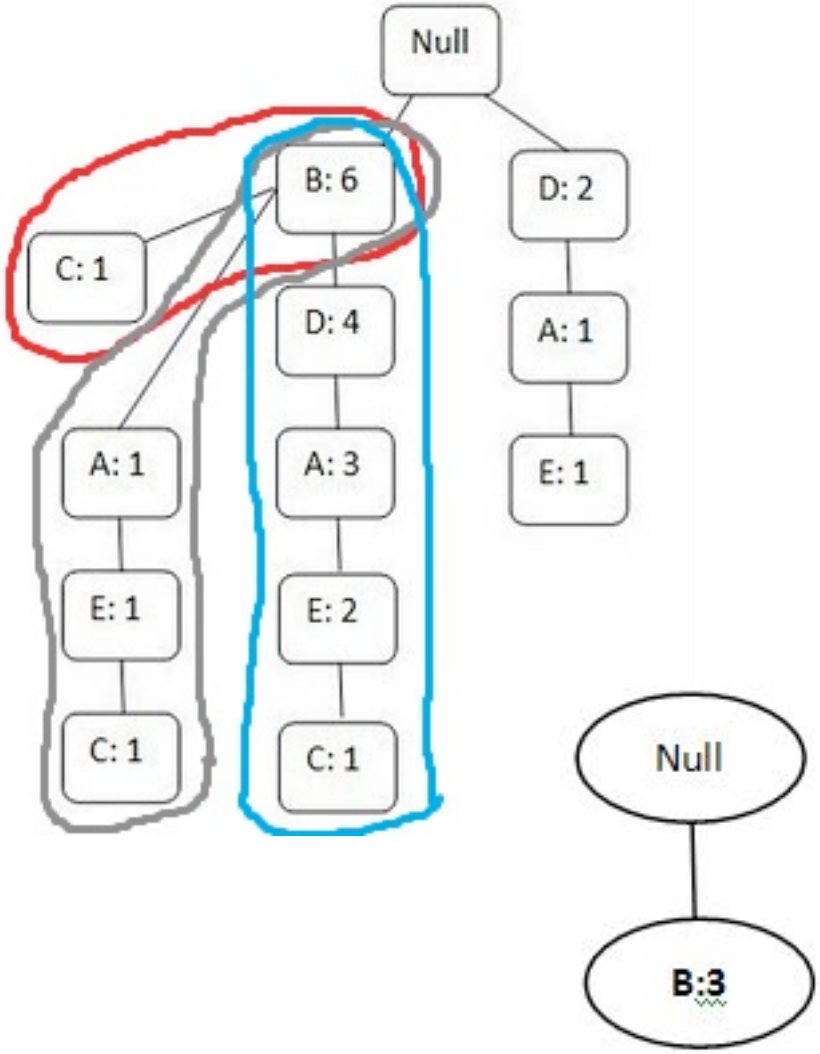
(CPB of X is the set of prefix paths in the FP-Tree with suffix X, and their frequency is determined by the frequency of X)

Note that:

1)the CPB of a suffix does not include the suffix

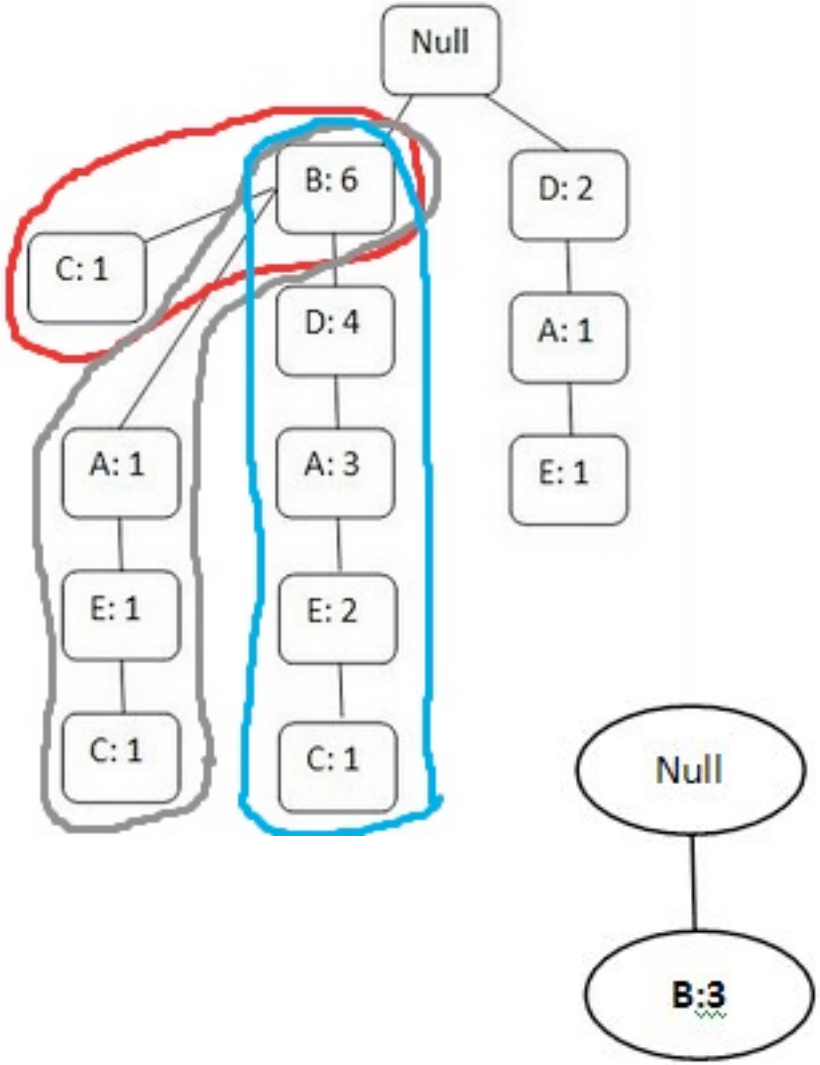
2)Single items in CBP are again listed in order of decreasing relevance (support), as in the FP tree

Example : 3. Build conditional FP-tree for an item



- (CPB) for C : BDAE:1, B:1 BAE:1
- Then we have to create the F-tree for C using its Conditional Pattern Base.
- The F-list (list of items in CPB with counts) is B:3,D:1,A:2,E:2 but **B:3 only** eligible for sitting on the F-tree due to fulfill the Minimum Support Count.
- So the F-tree of C is just **B:3**
- **F-tree of a suffix item X is used to find the set of itemsets including X and with min_supp>=threshold**

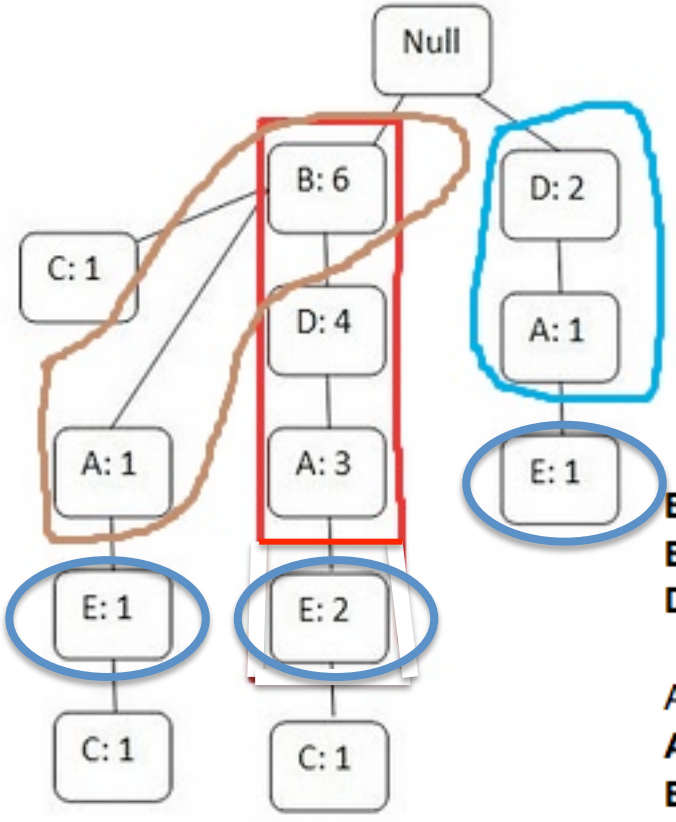
Example : 4. Extract frequent patterns FP from conditional F-tree



All frequent patterns corresponding to suffix C are generated by considering all possible combinations of C and F-Tree patterns.

Now we can find the frequent patterns from the F-tree of suffix C: they are only C:3, BC:3 since either we combine C with null node or with B

Example (cont'd) : Build Conditional Pattern Base for E



BDA: 2 - Red line
BA:1 - Brown line
DA:1 - Blue line

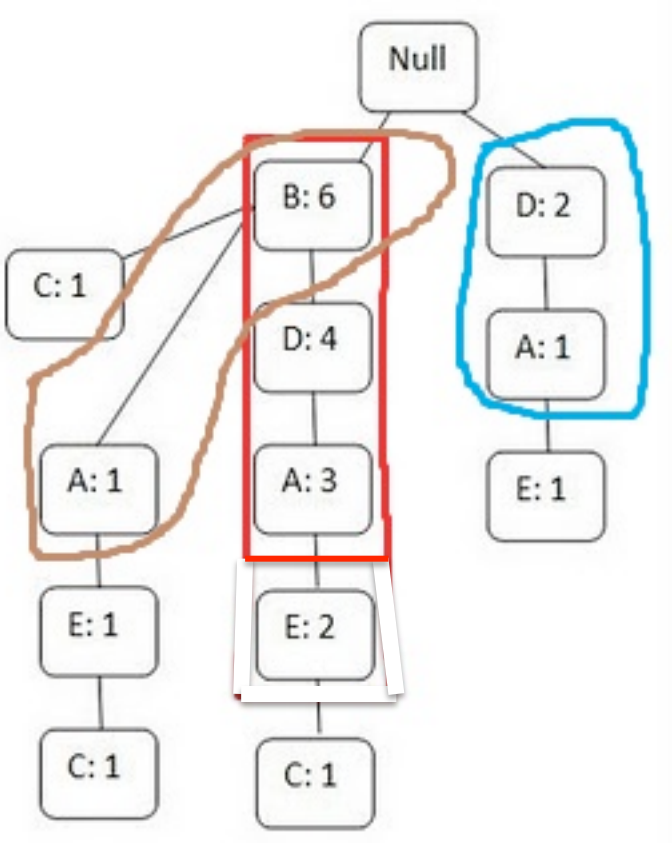
And F-list will be
A:4
B:3
D:3

Now we consider the lenght 1 suffix **E**

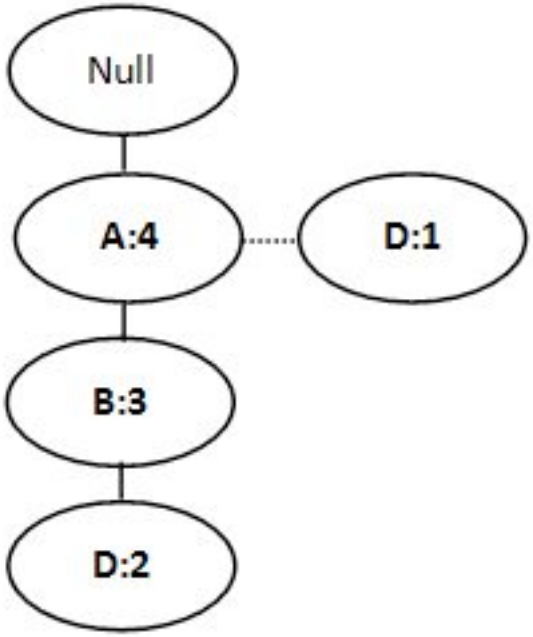
The conditional pattern base (CPB) for E is:
BDA:2, BA:1 DA:1
And the F-list (item counts from CPB) is:
A:4 B:3 D:3
all have the count>=min_count

Note that in F-list the frequency of items is determined by that of E. Two counts are taken from the red pattern, one from the brown and one from the blue. E.g., A appears in all patterns, therefore its count is 4; D appears in red and blue, so its count is 3, etc.

Example : Build F-tree for E

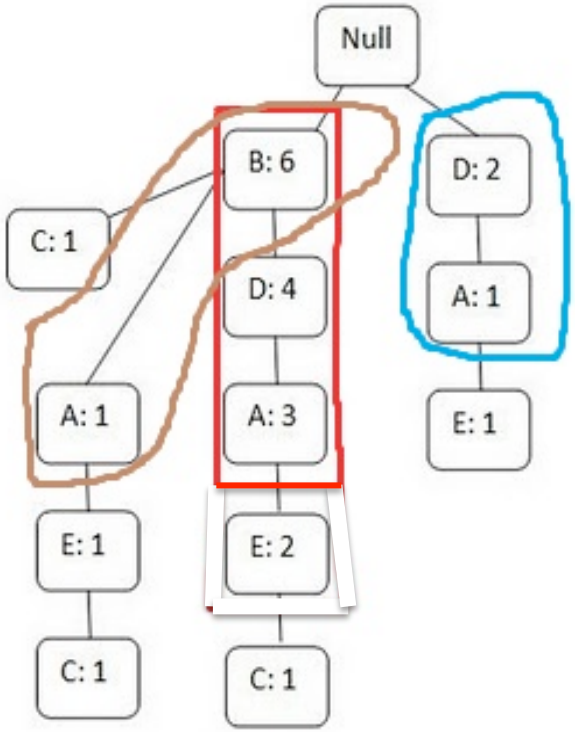


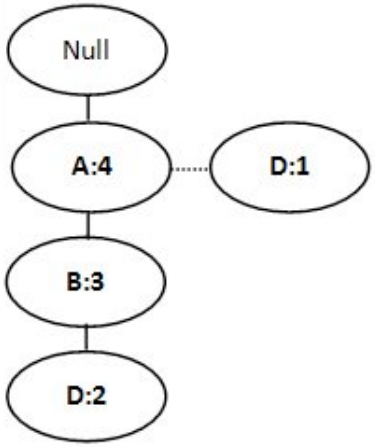
Frequency-ordered F-list for E is then A:4 B:3 D:3 and frequency re-ordered CPF patterns are ABD:2, AB:1, AD:1
F-tree is built from these patterns (same procedure as for the FP-tree)



Note in F-tree of E all branch items ordered by counts in F-list!!

Example : Build FPs for E





Remember: the lowest frequency item determines the support of an itemset!!

All frequent pattern corresponding to suffix E are generated by considering all possible combinations of E and frequent itemsets extracted from its conditional F-Tree.

F-list: A:4 B:3 D:3

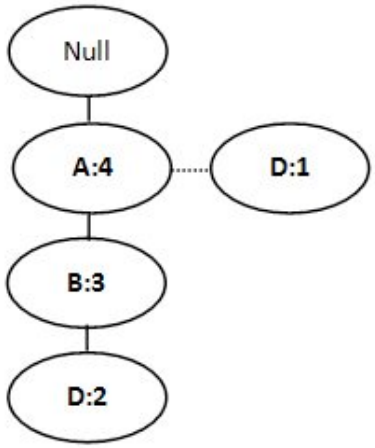
FPs:

DE:3 ABE:3, ADE:3

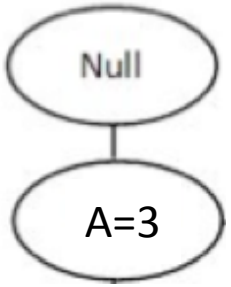
BE:3, ABE:3 AE:4

Note no ABDE since from tree you see it hasn't necessary support (=2)

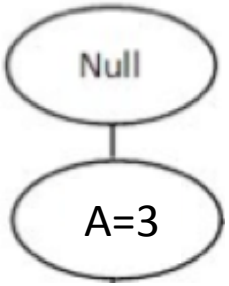
Recursion step: consider length-2 suffixes, then length 3..



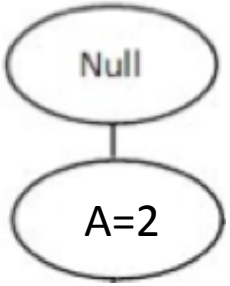
E conditional FP-tree



ED conditional FP-tree

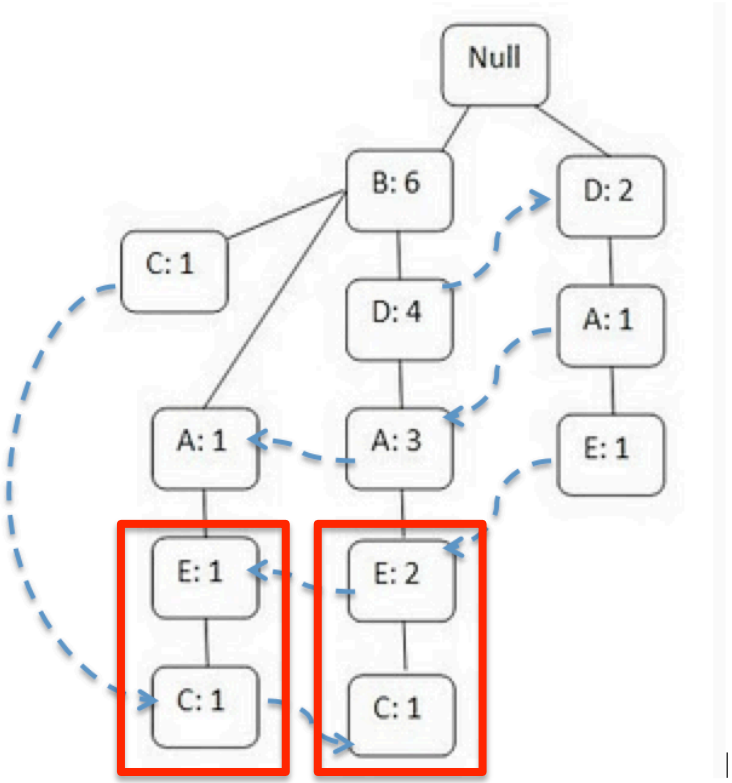


EDB conditional FP-tree



EB conditional FP-tree

Can also use FP-tree with pointers to build Conditional patterns

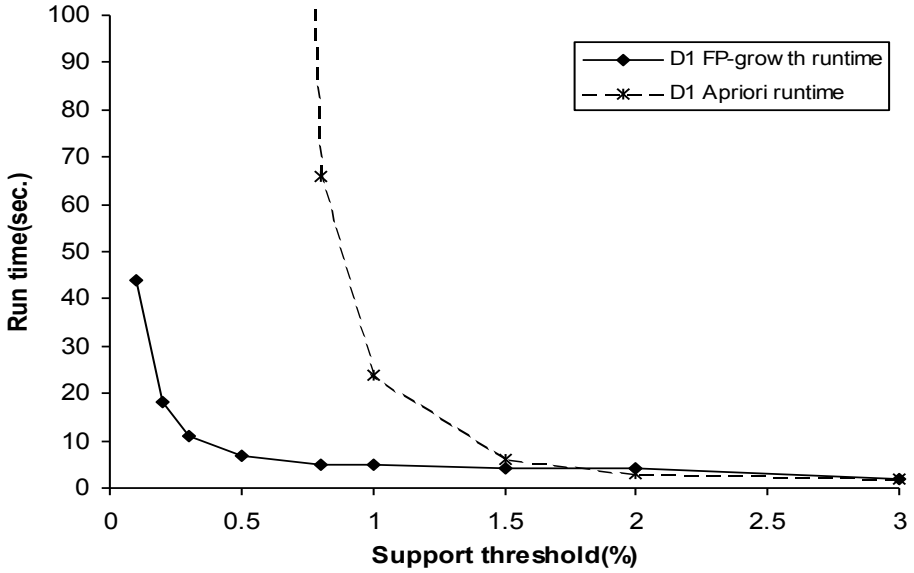


- C : BDAE:1, B:1 BAE:1
- E: DA:1 BDA:2 BA:1
- **CE: AB:1 ADB:1**

- ecc

Benefits of the FP-tree Structure

- Performance study shows
  - FP-growth is an order of magnitude faster than Apriori
- Reason
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building



Overview

- Basic Concepts of Association Rule Mining
- The Apriori Algorithm (Mining single dimensional boolean association rules)
- Methods to Improve Apriori's Efficiency
- Frequent-Pattern Growth (FP-Growth) Method
- From Association Analysis to Correlation Analysis
- Summary

Association & Correlation

- Correlation Analysis provides an alternative framework for finding interesting relationships, or to improve understanding of meaning of some association rules (a lift of an association rule).

Correlation Concepts

- Two itemsets X and Y are independent (the occurrence of X is independent of the occurrence of item set Y) iff

  $P(X \cup Y) = P(X) \cdot P(Y)$

- Otherwise X and Y are dependent and correlated

- The measure of correlation, or correlation between X and Y is given by the formula:

  $Corr(X,Y) = P(X \cup Y) / P(X) \,.\, P(Y)$

Correlation Concepts [Cont.]

- **corr(X,Y) >1** means that X and Y are positively correlated i.e. the occurrence of one implies the occurrence of the other.

- **corr(X,Y) < 1** means that the occurrence of X is negatively correlated with (or discourages) the occurrence of Y.

- **corr(X,Y) =1** means that X and Y are independent and there is no correlation between them.

Association & Correlation

- The correlation formula can be re-written as
  - Corr(X,Y) = P(Y | X) / P(Y)

- We already know that
  - Support(X →Y)= P(XUY)
  - Confidence(X → Y)= P(Y | X)
  - That means that, Confidence(X →Y)= corr(X,Y) P(Y)

- So correlation, support and confidence are all different, but the correlation provides **an extra information** about the association rule (X →Y).

- We say that the correlation corr(X,Y) provides the LIFT of the association rule (X=>Y), i.e. X is said to increase (or LIFT) the likelihood of Y by the factor of the value returned by the formula for corr(X,Y).

Summary

- Association Rule Mining
  - Finding interesting association or correlation relationships.
- Association rules are generated from frequent itemsets.
- Frequent itemsets are mined using Apriori algorithm or Frequent-Pattern Growth method.
- Apriori property states that all the subsets of frequent itemsets must also be frequent.
- Apriori algorithm uses frequent itemsets, join & prune methods and Apriori property to derive strong association rules.
- Frequent-Pattern Growth method avoids repeated database scanning of Apriori algorithm.
- FP-Growth method is faster than Apriori algorithm.