Mining Frequent Patterns, Association and Correlations:

Efficient & Scalable Frequent Itemset Mining Methods

Continued from previous slides unit-II-part 2.1

Dr. K. Raghava Rao
Professor of CSE ,Dept. of MCA
KL University
krraocse@gmail.com
http://datamining.blog.com

Apriori Algorithm: Finding Frequent itemsets Using Candidate Generation

- The Apriori Algorithm (Mining single dimensional boolean association rules)
- Proposed by R.Agarwal & Srikanth in 1994 for mining frequent itemsets for boolean assoication rules.

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 item which has minimum support (denoted by L_i for i^{th}-Itemset).
- Apriori Property: Any subset of frequent itemset must be frequent.
- Join Operation: To find L_k , a set of candidate k-itemsets is generated by joining L_{k-1} with itself.

The Apriori Algorithm in a Nutshell

- Find the *frequent itemsets*: the sets of items that have minimum support
- Apriori property:
  - All nonempty subsets of a frequent itemset must also be frequent.
    - i.e., if {*AB*} is a frequent itemset, both {*A*} and {*B*} should be a frequent itemset
  - Iteratively find frequent itemsets with cardinality from 1 to *k* (*k*-itemset)
- Use the frequent itemsets to generate association rules.

The Apriori Algorithm : Pseudo code

- 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 ∪_k L_k;
```

The Apriori Algorithm: Example

TID	List of Items
T100	I1, I2, I5
T100	I2, I4
T100	I2, I3
T100	I1, I2, I4
T100	I1, I3
T100	I2, I3
T100	I1, I3
T100	I1, I2 ,I3, I5
T100	I1, I2, I3

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

Step 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

- In the first iteration of the algorithm, each item is a member of the set of candidate.
- The set of frequent 1-itemsets, L_1 , consists of the candidate 1-itemsets satisfying minimum support.

Step 2: Generating 2-itemset Frequent Pattern

Generate C_2 candidates from L_1

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Scan D for count of each candidate

Itemset	Sup. Count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

C_2

Compare candidate support count with minimum support count

Itemset	Sup Count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

L_2

Step 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.

Step 3: Generating 3-itemset Frequent Pattern

Scan D for count of each candidate

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Scan D for count of each candidate

Itemset	Sup. Count
{I1, I2, I3}	2
{I1, I2, I5}	2

C_3

Compare candidate support count with min support count

Itemset	Sup Count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

- The generation of the set of candidate 3-itemsets, C_3 , involves use of the Apriori Property.
- In order to find C_3, we 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}}.
- Now, 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.

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

- Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that four latter candidates 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.
- Therefore, C_3 = {{I1, I2, I3}, {I1, I2, I5}} after checking for all members of result of Join operation for Pruning.
- Now, the transactions in D are scanned in order to determine L_3, consisting of those candidates 3-itemsets in C_3 having minimum support.

Step 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 in {{I1, I2, I3, I5}}, this itemset is pruned since its subset {{I2, I3, I5}} is not frequent.
- Thus, $C_4 = \varphi$, and algorithm terminates, having found all of the frequent items. This completes our Apriori Algorithm.
- What's Next ?

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

Step 5: Generating Association Rules from Frequent Itemsets

- Procedure:
  - For each frequent itemset ***"l",*** generate all nonempty subsets of ***l.***
  - For every nonempty subset ***s*** of ***l***, output the rule **"s → (l-s)"** if **support_count(l) / support_count(s) >= min_conf** where min_conf is minimum confidence threshold.

- Back To Example:

  We had 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 take ***l*** = {I1,I2,I5}.
  - Its all nonempty subsets are {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2}, {I5}.

Step 5: Generating Association Rules from Frequent Itemsets [Cont.]

- Let minimum confidence threshold is , say 70%.
- The resulting association rules are shown below, each listed with its confidence.
  - R1: I1 ^ I2 → I5
    - 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.

Step 5: 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.

Methods to Improve Apriori's Efficiency

- Hash-based itemset counting: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.
- Transaction reduction: A transaction that does not contain any frequent k-itemset is useless in subsequent scans.
- Partitioning: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB.
- Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness.
- Dynamic itemset counting: add new candidate itemsets only when all of their subsets are estimated to be frequent.

For more details go to slides unit-II part-2.2 later com back to here.

Mining Frequent Itemsets without Candidate Generation.

- >-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

- >-Develop an efficient, FP-tree-based frequent pattern mining method
  - -A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - -Avoid candidate generation: sub-database test only!

FP-Growth Method : An Example

TID	List of Items
T100	I1, I2, I5
T100	I2, I4
T100	I2, I3
T100	I1, I2, I4
T100	I1, I3
T100	I2, I3
T100	I1, I3
T100	I1, I2 ,I3, I5
T100	I1, I2, I3

- Consider the same previous example of a database, D , consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. min_sup = 2 / 9 = 22 %)
- The first scan of database is same as Apriori, which derives the set of 1-itemsets & their support counts.
- The set of frequent items is sorted in the order of descending support count.
- The resulting set is denoted as L = {I2:7, I1:6, I3:6, I4:2, I5:2}

FP-Growth Method: Construction of FP-Tree

- -First, create the root of the tree, labeled with "null".
- -Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).

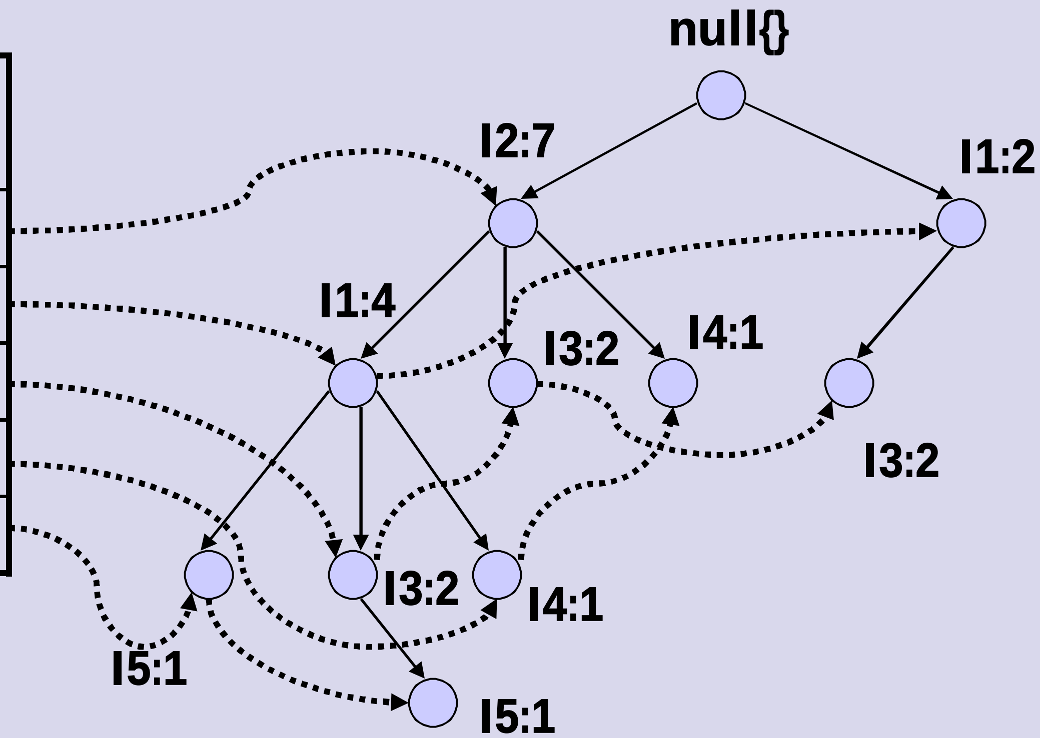
- -The items in each transaction are processed in L order (i.e. sorted order).
- -A branch is created for each transaction with items having their support count separated by colon.

- -Whenever the same node is encountered in another transaction, we just increment the support count of the common node or Prefix.

- -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.

FP-Growth Method: Construction of FP-Tree

Item Id	Sup Count	Node-link
I2	7	
I1	6	
I3	6	
I4	2	
I5	2	



An FP-Tree that registers compressed, frequent pattern information

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

Steps:

1. Start from each frequent length-1 pattern (as an initial suffix pattern).

2. Construct its conditional pattern base which consists of the set of prefix paths in the FP-Tree co-occurring with suffix pattern.
3. Then, Construct its conditional FP-Tree & perform mining on such a tree.

4. The pattern growth is achieved by concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-Tree.

5. The union of all frequent patterns (generated by step 4) gives the required frequent itemset.

FP-Tree Example Continued

Item	Conditional pattern base	Conditional FP-Tree	Frequent pattern generated
I5	{(I2 I1: 1),(I2 I1 I3: 1)}	<I2:2 , I1:2>	I2 I5:2, I1 I5:2, I2 I1 I5: 2
I4	{(I2 I1: 1),(I2: 1)}	<I2: 2>	I2 I4: 2
I3	{(I2 I1: 1),(I2: 2), (I1: 2)}	<I2: 4, I1: 2>,<I1:2>	I2 I3:4, I1, I3: 2 , I2 I1 I3: 2
I2	{(I2: 4)}	<I2: 4>	I2 I1: 4

Mining the FP-Tree by creating conditional (sub) pattern bases

Now, Following the above mentioned steps:

- Lets start from I5. The I5 is involved in 2 branches namely {I2 I1 I5: 1} and {I2 I1 I3 I5: 1}.
- Therefore considering I5 as suffix, its 2 corresponding prefix paths would be {I2 I1: 1} and {I2 I1 I3: 1}, which forms its conditional pattern base.

FP-Tree Example Continued

- -Out of these, Only I1 & I2 is selected in the conditional FP-Tree because I3 is not satisfying the minimum support count.

  -For I1 , support count in conditional pattern base = 1 + 1 = 2

  -For I2 , support count in conditional pattern base = 1 + 1 = 2

  -For I3, support count in conditional pattern base = 1

  -Thus support count for I3 is less than required min_sup which is 2 here.

- -Now , We have conditional FP-Tree with us.

- -All frequent pattern corresponding to suffix I5 are generated by considering all possible combinations of I5 and conditional FP-Tree.

- -The same procedure is applied to suffixes I4, I3 and I1.
- Note: I2 is not taken into consideration for suffix because it doesn't have any prefix at all.

FP-Tree algorithm

Algorithm: FP_growth. Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database;
- *min_sup*, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

1. The FP-tree is constructed in the following steps:
   (a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.
   (b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following:
   Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in the *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call `insert_tree`$([p|P], T)$, which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call `insert_tree`(P, N) recursively.
2. The FP-tree is mined by calling `FP_growth`(*FP_tree, null*), which is implemented as follows.

procedure `FP_growth`$(Tree, \alpha)$

(1) `if` *Tree* contains a single path P `then`
(2) `for each` combination (denoted as β) of the nodes in the path P
(3) generate pattern $\beta \cup \alpha$ with *support_count = minimum support count of nodes in* β;
(4) `else for each` a_i in the header of *Tree* {
(5) generate pattern $\beta = a_i \cup \alpha$ with $support\_count = a_i.support\_count$;
(6) construct β's conditional pattern base and then β's conditional FP_tree $Tree_\beta$;
(7) `if` $Tree_\beta \neq \emptyset$ `then`
(8) call `FP_growth`$(Tree_\beta, \beta)$; }

Why Frequent Pattern Growth Fast ?

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

Mining Frequent Itemsets Using Vertical Data Format

- Vertical format: $t(AB) = \{T_{11}, T_{25}, \ldots\}$
  - tid-list: list of trans.-ids containing an itemset
- Deriving frequent patterns based on vertical intersections
  - $t(X) = t(Y)$: X and Y always happen together
  - $t(X) \subset t(Y)$: transaction having X always has Y
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - $t(X) = \{T_1, T_2, T_3\}$, $t(XY) = \{T_1, T_3\}$
  - Diffset $(XY, X) = \{T_2\}$
- Eclat (Zaki et al. @KDD'97) Mining Closed patterns using vertical format: CHARM (Zaki & Hsiao@SDM'02)

Mining Frequent Itemsets Using Vertical Data Format

Table 5.3

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Mining Frequent Itemsets Using Vertical Data Format

Table 5.3

itemset	*TID_set*
{I1,I2}	{T100, T400, T800, T900}
{I1,I3}	{T500, T700, T800, T900}
{I1,I4}	{T400}
{I1,I5}	{T100, T800}
{I2,I3}	{T300, T600, T800, T900}
{I2,I4}	{T200, T400}
{I2,I5}	{T100, T800}
{I3,I5}	{T800}

Mining Frequent Itemsets Using Vertical Data Format

Table 5.5

itemset	*TID_set*
{I1,I2,I3}	{T800, T900}
{I1,I2,I5}	{T100, T800}

Mining Closed frequent Itemsets

>-To mine closed frequent itemsets, first mine complete set of frequent itemsets and then remove every frequent itemsets that is a proper subset of , and carries the same support as, an existing frequent itemset.

>-above method is prohibitively expansive .

>-a recommended methodology search for closed frequent itemsets directly during the mining process.

>-this requires to prune the search space as soon as identified case of close itemsets during mining.

>-Pruning strategies are given in next slide.

Mining Closed frequent Itemsets

- Pruning strategies include the following:
- **>-Item merging:** If every transaction containing a frequent itemset X also contains an itemset Y but not any proper superset of Y , then X ∪Y forms a frequent closed itemset and there is no need to search for any itemset containing X but no Y .

Example:

--In FP-Tree example prefix itemset {I5:2} is {{I2,I1},{I2,I1,I3}}, from which we can see that each its transaction contains itemset {I2,I1} but no proper superset of {I2,I1}.

--Itemset {I2,I1} can be merged with {I5} to form the closed itemset, {I5,I2,I1:2} , and we do not need to mine for closed itemsets that contain I5 but not {I2,I1}.

Mining Closed frequent Itemsets

- Pruning strategies include the following:
- >-**Sub-itemset pruning**: If a frequent itemset X is a proper subset of an already found frequent closed itemset Y and support count(X) = support count(Y), then X and all of X's descendants in the set enumeration tree cannot be frequent closed itemsets and thus can be pruned.
- Example:
- -- A transaction database has only two transactions: {<a1,a2…,a100>,<a1,a2,….a50>}
- and minimum support count=2. The projection on fist item a1,derived frequent
- itemset, {a1,a2,..,a50:2},based itemset merging optimization.

- --Because support (a2)=support({a1,a2,…a50})=2 and a2 is a proper subset of
- {a1,a2,…a50}, there is no to examine a2 and its projected database.

- Similar pruning can be done for a3..a50 as well.

- --This mining closed frequent itemsets in this data set terminates after mining a1's
- project database.

Mining Closed frequent Itemsets

- Pruning strategies include the following:

- **>-Item skipping:** In the depth-first mining of closed itemsets, at each level, there will be a prefix itemset X associated with a header table and a projected database. If a local frequent item p has the same support in several header tables at different levels, we can safely prune p from the header tables at higher levels.

- Example:

--transaction database having only two transactions:

{<a1,a2,...,a100>,<a1,a2,...a50>}, where min_sup=2.

--Because a2 in a1's projected database has same support as a2 in global header table, a2 can be pruned from global header table.

--Similar pruning can be done for a3,...,a50.

--There is no need to mine anything more after mining a1's projected database.

- Next slide you may skip, not in syllabus.

Mining Closed frequent Itemsets

- Flist: list of all frequent items in support ascending order
  - Flist: d-a-f-e-c
- Divide search space
  - Patterns having d
  - Patterns having d but no a, etc.
- Find frequent closed pattern recursively
  - Every transaction having d also has *cfa* → *cfad* is a frequent closed pattern
- J. Pei, J. Han & R. Mao. “CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", DMKD'00.

TID	Items
10	a, c, d, e, f
20	a, b, e
30	c, e, f
40	a, c, d, f
50	c, e, f

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.

-Correlation concepts & rules can be used to further support our derived association rules.

Go to Unit-2 part 2.4 slides