

UNIT-V

UNIT – V SYLLABUS

Association Analysis: Basic Concepts and Algorithms: Problem Defecation, Frequent Item Set generation, Rule generation, compact representation of frequent item sets, FP-Growth Algorithm. **(Tan & Vipin)**

Association Analysis

Association mining aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items or objects in transaction databases, relational database or other data repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.

EXAMPLES:

Rule Form: Body → Head [Support, confidence]

Buys (X, “Computer”) → Buys (X, “Software”) [40%, 50%]

ASSOCIATION RULE: BASIC CONCEPTS:

- Given: (1) database of transaction, (2) each transaction is a list of items (purchased by a customer in visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items.
 - E.g., 98% of people who purchase tires and auto accessories also get automotive—services done.
 - E.g., Market Basket Analysis — This process analyzes customer buying habits by finding associations between the different items that customers place in their “Shopping Baskets”. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customer.

APPLICATIONS:

- Maintenance agreement (what the store should do to boost maintenance agreement sales)
- Home Electronics (what other products should the store stocks up?)
- Attached mailing in direct marketing

ASSOCIATION RULE:

An association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in transactions that contain X . The formal definition of these metrics are

$$\text{Support, } s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

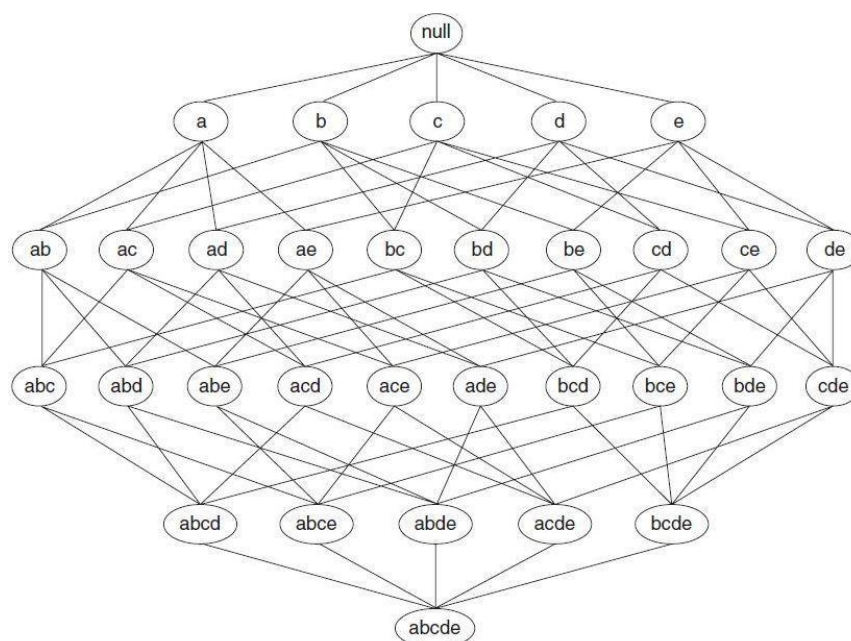
$$\text{Confidence, } c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Why Use Support and Confidence? Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together. For these reasons, support is often used to eliminate uninteresting rules.

Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X . Confidence also provides an estimate of the conditional probability of Y given X .

Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. Frequent Itemset Generation, whose objective is to find all the item-sets that satisfy the *minsup* threshold. These itemsets are called frequent itemsets.
2. Rule Generation, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.



Frequent Itemset Generation:

A lattice structure can be used to enumerate the list of all possible itemsets. Above Figure shows an itemset lattice for $I = \{a, b, c, d, e\}$. In general, a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.

Data Warehousing and Data Mining

To find frequent itemsets we have two algorithms,

- a) Apriori Algorithm
- b) FP-Growth

a) APRIORI ALGORITHM:

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see later. Apriori employs an iterative approach known as a *level-wise* search, where k -itemsets are used to explore $(k+1)$ -itemsets.

First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted by L_1 . Next, L_1 is used to find L_2 , the set of frequent 2-itemsets, which is used to find L_3 , and so on, until no more frequent k -itemsets can be found. The finding of each L_k requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property is used to reduce the search space.

Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, $min\ sup$, then I is not frequent, that is, $P(I) < min\ sup$. If an item A is added to the itemset I , then the resulting itemset (i.e., IUA) cannot occur more frequently than I . Therefore, IUA is not frequent either, that is, $P(IUA) < min\ sup$.

This property belongs to a special category of properties called **antimonotonicity** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotonicity* because the property is monotonic in the context of failing a test.

A two-step process is followed, consisting of **join** and **prune** actions.

- 1. The join step:** To find L_k , a set of **candidate** k -itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k .
- 2. The prune step:** C_k is a superset of L_k , that is, its members may or may not be frequent, but all of the frequent k -itemsets are included in C_k . A database scan to determine the count of each candidate in C_k would result in the determination of L_k .

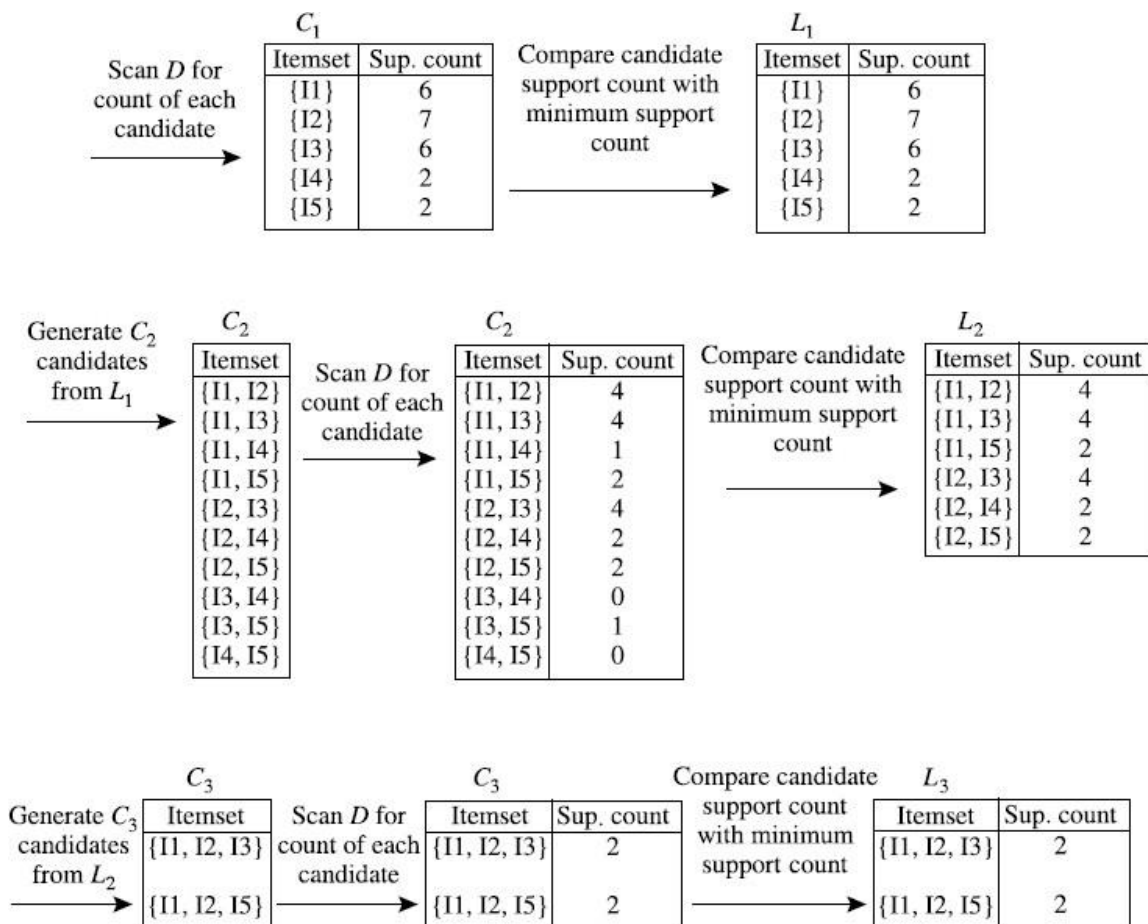
Data Warehousing and Data Mining

EXAMPLE:

Transactional Data for an *AllElectronics* Branch

<i>TID</i>	<i>List of item_IDs</i>
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1 . The algorithm simply scans all of the transactions to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2, that is, $\min \text{sup} = 2$. (Here, we are referring to *absolute* support because we are using a support count. The corresponding relative support is $2/9 = 22\%$.) The set of frequent 1-itemsets, L_1 , can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C_1 satisfy minimum support.
3. To discover the set of frequent 2-itemsets, L_2 , the algorithm uses the join $L_1 \bowtie L_1$ to generate a candidate set of 2-itemsets, C_2 . C_2 consists of 2-itemsets. Note that no candidates are removed from C_2 during the prune step because each subset of the candidates is also frequent.
4. Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown in the middle table of the second row in Figure
5. The set of frequent 2-itemsets, L_2 , is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.



6. The generation of the set of the candidate 3-itemsets, C_3 , is detailed in Figure From the join step, we first get $C_3 = L_2 \bowtie L_2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$ Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from C_3 , thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3 .
7. The transactions in D are scanned to determine L_3 , consisting of those candidate 3-itemsets in C_3 having minimum support.
8. The algorithm uses $L_3 \bowtie L_3$ to generate a candidate set of 4-itemsets, C_4 . Although the join results in $\{I1, I2, I3, I5\}$, itemset $\{I1, I2, I3, I5\}$ is pruned because its subset $\{I2, I3, I5\}$ is not frequent. Thus, $C_4 \neq \emptyset$, and the algorithm terminates, having found all of the frequent itemsets.

b) FP-GROWTH:

FP-growth (finding frequent itemsets without candidate generation). We reexamine the mining of transaction database, D , of Table in previous Example using the frequent pattern growth approach.

Transactional Data for an *AllElectronics* Branch

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

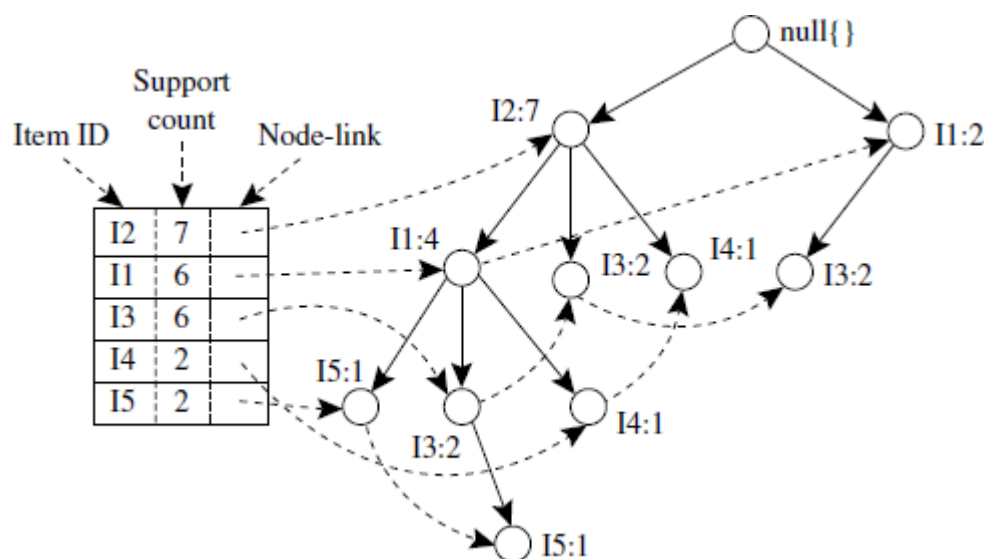
The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be

2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted by L . Thus, we have $L = \{\{I2:7\}, \{I1:6\}, \{I3:6\}, \{I4:2\}, \{I5:2\}\}$

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with "null." Scan database D a second time. The items in each transaction are processed in L

Data Warehousing and Data Mining

order (i.e., sorted according to descending support count), and a branch is created for each transaction.



The FP-tree is mined as follows. Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a “sub-database,” which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern), then construct its (*conditional*) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	$\langle I2: 2, I1: 2 \rangle$	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	$\langle I2: 2 \rangle$	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	$\langle I2: 4, I1: 2 \rangle, \langle I1: 2 \rangle$	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	$\langle I2: 4 \rangle$	{I2, I1: 4}

Finally, we can conclude that frequent itemsets are {I2, I1, I5} and {I2, I1, I3}.

GENERATING ASSOCIATION RULES FROM FREQUENT ITEMSETS

Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them (where *strong* association rules satisfy both minimum support and minimum confidence). This can be done using Eq. for confidence, which we show again here for completeness:

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support_count}(A \cup B)}{\text{support_count}(A)}.$$

The conditional probability is expressed in terms of itemset support count, where $\text{support_count}(A \cup B)$ is the number of transactions containing the itemsets $A \cup B$, and $\text{support_count}(A)$ is the number of transactions containing the itemset A . Based on this equation, association rules can be generated as follows:

- For each frequent itemset l , generate all nonempty subsets of l .
- For every nonempty subset s of l , output the rule " $s \Rightarrow (l - s)$ " if $\frac{\text{support_count}(l)}{\text{support_count}(s)} \geq \text{min_conf}$, where min_conf is the minimum confidence threshold.

Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

Generating association rules. Let's try an example based on the transactional data for *AllElectronics* shown before in Table 6.1. The data contain frequent itemset $X = \{I1, I2, I5\}$. What are the association rules that can be generated from X ? The nonempty subsets of X are $\{I1, I2\}$, $\{I1, I5\}$, $\{I2, I5\}$, $\{I1\}$, $\{I2\}$, and $\{I5\}$. The resulting association rules are as shown below, each listed with its confidence:

$\{I1, I2\} \Rightarrow I5$,	$\text{confidence} = 2/4 = 50\%$
$\{I1, I5\} \Rightarrow I2$,	$\text{confidence} = 2/2 = 100\%$
$\{I2, I5\} \Rightarrow I1$,	$\text{confidence} = 2/2 = 100\%$
$I1 \Rightarrow \{I2, I5\}$,	$\text{confidence} = 2/6 = 33\%$
$I2 \Rightarrow \{I1, I5\}$,	$\text{confidence} = 2/7 = 29\%$
$I5 \Rightarrow \{I1, I2\}$,	$\text{confidence} = 2/2 = 100\%$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules are output, because these are the only ones generated that are strong. Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right side of the rule. ■