

APRIORI Algorithm

Professor Anita Wasilewska

Book slides

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
 - A subset of a frequent itemset must also be a frequent itemset
 - 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

$L_1 = \{\text{frequent items}\};$

for ($k = 1; L_k \neq \emptyset; k++$) **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

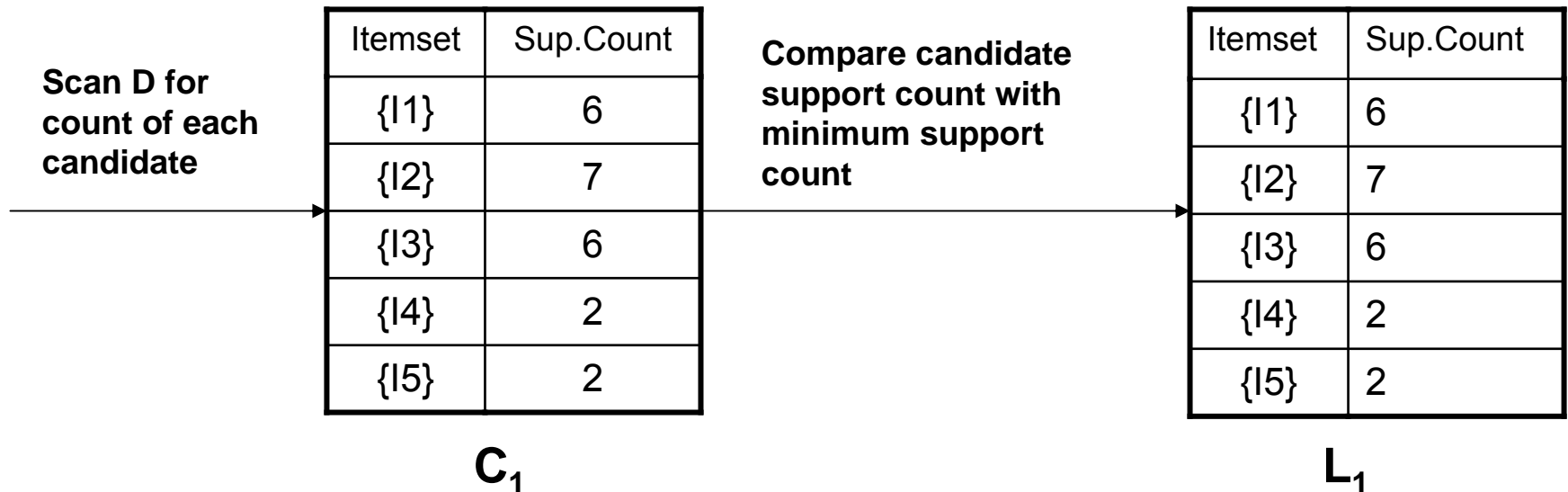
return $\cup_k L_k$;

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. $\text{min_sup} = 2/9 = 22\%$)
- 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



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

Step 2: Generating 2-itemset Frequent Pattern

- To discover the set of frequent 2-itemsets, L_2 , the algorithm uses $L_1 \text{ Join } L_1$ 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 2: Generating 2-itemset Frequent Pattern

Generate C_2 candidates from L_1

| Itemset |
|---------|
| {1, 12} |
| {1, 13} |
| {1, 14} |
| {1, 15} |
| {2, 13} |
| {2, 14} |
| {2, 15} |
| {3, 14} |
| {3, 15} |
| {4, 15} |

C_2

Scan D for count of each candidate

| Itemset | Sup. Count |
|---------|------------|
| {1, 12} | 4 |
| {1, 13} | 4 |
| {1, 14} | 1 |
| {1, 15} | 2 |
| {2, 13} | 4 |
| {2, 14} | 2 |
| {2, 15} | 2 |
| {3, 14} | 0 |
| {3, 15} | 1 |
| {4, 15} | 0 |

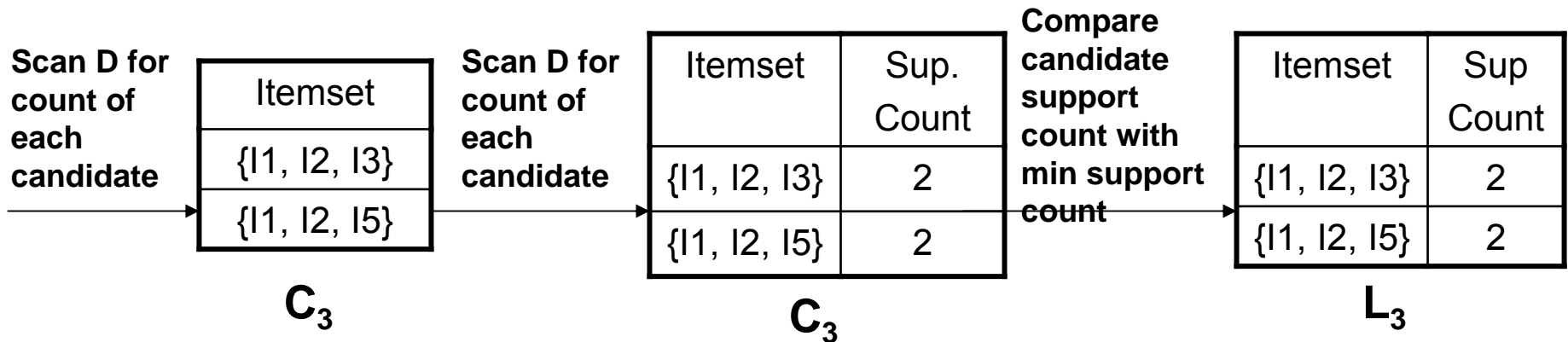
C_2

Compare candidate support count with minimum support count

| Itemset | Sup Count |
|---------|-----------|
| {1, 12} | 4 |
| {1, 13} | 4 |
| {1, 15} | 2 |
| {2, 13} | 4 |
| {2, 14} | 2 |
| {2, 15} | 2 |

L_2

Step 3: Generating 3-itemset Frequent Pattern



- 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 $L_2 \text{ Join } L_2$.
- $C_3 = L_2 \text{ Join } L_2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$.
- 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

- 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, $C_4 = \emptyset$, 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 I , generate all nonempty subsets of I .
- For every nonempty subset s of I , output the rule " $s \rightarrow (I-s)$ " if $\text{support_count}(I) / \text{support_count}(s) \geq \text{min_conf}$ 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 $I = \{I1, I2, I5\}$.
- Its all nonempty subsets are $\{I1, I2\}, \{I1, I5\}, \{I2, I5\}, \{I1\}, \{I2\}, \{I5\}$.

Step 5: Generating Association Rules from Frequent Itemsets

- Let **minimum confidence threshold** is , say 70%.
- The resulting association rules are shown below, each listed with its confidence.
 - R1: $I1 \wedge I2 \rightarrow I5$
 - Confidence = $sc\{I1,I2,I5\}/sc\{I1,I2\} = 2/4 = 50\%$
 - R1 is Rejected.
 - R2: $I1 \wedge I5 \rightarrow I2$
 - Confidence = $sc\{I1,I2,I5\}/sc\{I1,I5\} = 2/2 = 100\%$
 - **R2 is Selected.**
 - R3: $I2 \wedge I5 \rightarrow I1$
 - Confidence = $sc\{I1,I2,I5\}/sc\{I2,I5\} = 2/2 = 100\%$
 - **R3 is Selected.**

Step 5: Generating Association Rules from Frequent Itemsets

- R4: $I1 \rightarrow I2 \wedge I5$
 - Confidence = $sc\{I1, I2, I5\} / sc\{I1\} = 2/6 = 33\%$
 - R4 is Rejected.
- R5: $I2 \rightarrow I1 \wedge I5$
 - Confidence = $sc\{I1, I2, I5\} / \{I2\} = 2/7 = 29\%$
 - R5 is Rejected.
- R6: $I5 \rightarrow I1 \wedge I2$
 - 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.

Mining Frequent Patterns 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

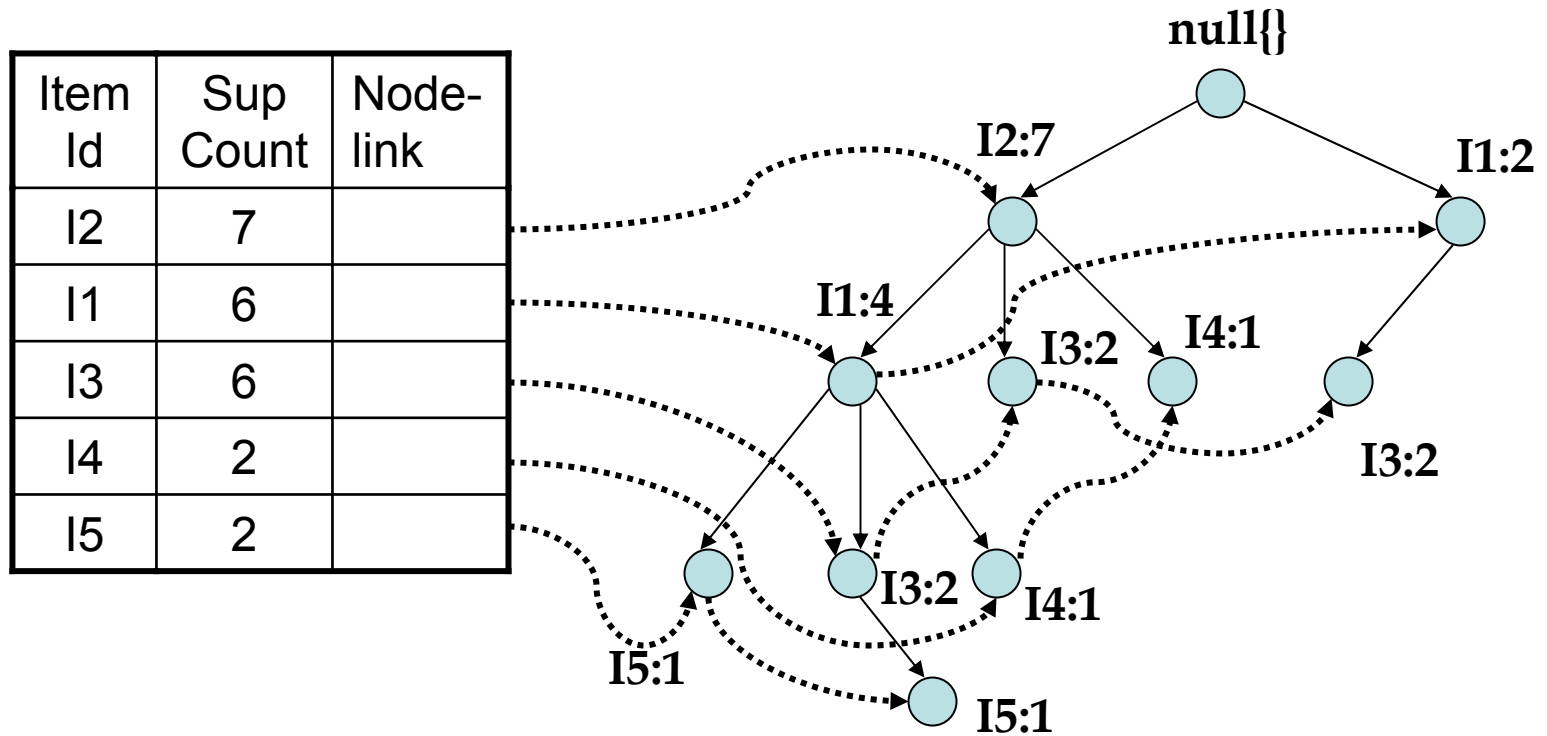
| TID | List of Items |
|------|----------------|
| T100 | I1, I2, I5 |
| T100 | I2, I4 |
| T100 | I2, I3 |
| T100 | I1, I2, I4 |
| T100 | I1, I3 |
| T100 | I2, I3 |
| T100 | I1, I3 |
| T100 | I1, I2, I3, I5 |
| T100 | I1, I2, I3 |

- Consider the same previous example of a database, D , consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. $\text{min_sup} = 2/9 = 22\%$)
- 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



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.

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