Data Science – Fall 2025/2026

INTRODUCTION TO Machine Learning

Lecture 2
Association Rules

Topics

- Introduction
- Formal Definitions
- Lift Analysis
- Closed & Maximal Itemsets
- Monotonicity
- Apriori Algorithm
- FP Growth Algorithm

Bibliography

Oded Maimon, Lior Rokach (Editors) - Data Mining and Knowledge Discovery Handbook, Second Edition, Springer 2010

Jiawei Han, Micheline Kamber - Data Mining : Concepts and Techniques, Morgan Kaufmann Publication, 2006

Introduction

Association rules are a fundamental concept in data mining used to discover i nteresting relationships between variables in large dataset,often applied in ma rket basket analysis.

Association rules are a rule-based machine learning method that identifies relationships between items in large databases. They are typically expressed in the form of $X \Rightarrow Y$, where X and Y are itemsets. For example, in a supermarket, an association rule might indicate that if a customer buys bread (X), he is likely to by also butter (Y).

Association rules are used for marketing strategies, product placements and inventory management.

Example

$$computer \Rightarrow antivirus\_software\ [support = 20\%, confidence = 60\%]$$

Significance:

- 20% of purchases contain both "computer" and "antivirus_software".

- 60% of the purchases that contain "computer" also contain "antivirus_software".

An association is interesting if its **support** and **confidence** exceed, resepectiveley a predefined confidence threshold (denoted **min_support)** and a predefined confidence threshold (denoted **min_confidence)**.

Formal Definitions

- Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finit set of items (called the universe of items)

- A transaction t is a subset of I, i.e., $t \subseteq I$

- A transaction database $D = \{ t_1, t_2, \ldots, t_n \}$ is a finit multiset of such transactions

- If a is an itemset such that $a \subseteq I$, a is called a k-itemset if $card(a) = k$

- We define $support\_count(a) = card(\{ t \in D \mid a \subseteq t \})$

$\{ t \in D \mid a \subseteq t \}$ is the set of all transactions t in the database D such that t contains the itemset a

- We define $support(a) = \dfrac{support\_count(a)}{card(D)}$

Formal Definitions

-An association rule is of the form: $a \Rightarrow b$, where a and b are disjoint sets of itemsets, i.e., $(a, b) \subseteq I$ and $a \cap b = \emptyset$

-An association rule $a \Rightarrow b$ is said to be supported by a transaction t if the union of a and b is contained in t, i.e., $a \cup b \subseteq I$

$a \cup b \subseteq I$: means all items in both a and b appear together in transaction t

- We define $support(a \Rightarrow b) = p(a \cup b) = \dfrac{support\_count\ (a \cup b)}{card(D)}$

- We define $confidence(a \Rightarrow b) = \dfrac{p(a \cup b)}{p(a)} = \dfrac{support\_count\ (a \cup b)}{support\_count\ (a)}$

- In addition to support and confidence, we can also consider the ***lift*** for an association rule. We define the lift by the following equation:

$$lift(a \Rightarrow b) = \frac{p(b \mid a)}{p(b)} = \frac{support(a \cup b)}{suppor(a) \times support(b)} = \frac{conf(a \Rightarrow b)}{support(b)}$$

Lift Analysis

The lift of an association rule $A \Rightarrow B$ measures how much more likely *A* and *B* occur together compared to what we would expect if they were independent.

If **Lift < 1,** then the occurrence of A is **negatively correlated** with the occurrence of B: meaning that the occurrence of one likely leads to the absence of the other one)

If **Lift > 1,** then A and B are **positively correlated,** meaning that the occurrence of one implies the occurrence of the other

If **Lift = 1,** then A and B are **independent** and there is no correlation between them

- Example: $lift(Cigarettes \Rightarrow Lighter) = \dfrac{2\%}{1\%} = 2$

- Signification: Buying cigarettes increases the probability of buying a lighter by 2 times

Definition of a Strong Rule

- In association rule mining, a rule of the form $A \Rightarrow B$ is called a **strong rule**, if it satisfies the two following conditions:

1) Minimum Support : The rule apperas often enough in the dataset, i.e.:

$$support(A \cup B) \geq minimum\ support\ threshold$$

2) Minimum Confidence: The rule is reliable enough (not just coincidence), i.e.:

$$confidence(A \Rightarrow B) \geq minimum\ confidence\ threshold$$

- The discovery of association rules can be seen as a two-step process:

Step 1: Finding frequent itemsets: identify all itemsets X such that support$(X) \geq min_support$. This step is computationally **difficult**.

Step 2: Generating strong association rules: generate all rules R corresponding to the identified frequent itemsets, for which confidence$(R) \geq min_conf$. This step is **easier** once the frequent itemsets have been found.

Combinatorics

- For any number of transactions using d items, there are $\mathbf{2^d}$ itemsets.

- The number of possible association rules with non-empty left-hand side and right-hand side is: $\mathbf{3^d - 2^{d+1} - 1}$

- A greedy strategy (step-by-step algorithm) for generating all itemsets is not feasible This is why we need an efficient pruning algorithm such as the Apriori algorithm or others like FP-Growth

- These algorithms allow us to **identify only the frequent itemsets**, avoiding the need to generate and test all possible combinations of itemsets.

Mining for Association Rules

Deriving rules from itemsets:

1) Take each frequent itemset L (support(L) $\geq$ $min\_support$). Decompose L in all possible ways such that : $L = X \cup Y$ with $X \cap Y = \emptyset$

2) Compute the confidence of $X \Rightarrow Y$ and $Y \Rightarrow X$. Keep the rules that are strong.

Remark: If the itemset $X \cup Y$ is frequent, then both X and Y are also frequent itemset

Closed Itemsets

A frequent itemset *X* is **closed** *if and only if* it satisfies this condition:

$$\forall\ Y\ /\ X \subset Y,\ support\ (Y) < support\ (X)$$

In other words, you cannot add an item to a closed frequent itemset without **decreasing its support**.

✅ **Use: Frequent closed itemsets reduce redundancy** while still keeping support information. So, instead of listing many similar itemsets, we keep only the most “informative” ones.

When identifying frequent itemsets, we can ignore the non-closed ones and keep only the closed frequent itemsets because the closed ones already contain all the necessary information.

Example of Closed Itemsets

min_support = 2

🛒 Example: Transactions

Transaction ID	Items bought
T1	{milk, bread, butter}
T2	{milk, bread}
T3	{milk, bread, butter}
T4	{bread, butter}

Itemset	Appears in	Support	Frequent?
{milk}	T1, T2, T3	3	✅
{bread}	T1, T2, T3, T4	4	✅
{butter}	T1, T3, T4	3	✅
{milk, bread}	T1, T2, T3	3	✅
{milk, butter}	T1, T3	2	✅
{bread, butter}	T1, T3, T4	3	✅
{milk, bread, butter}	T1, T3	2	✅

A **closed frequent itemset** has **no superset with the same support.**

Itemset	Support	Superset with same support?	Closed?
{milk}	3	✅ Yes, {milk, bread} = 3 → ❌	**Not closed**
{bread}	4	No superset with 4 → ✅	**Closed**
{butter}	3	✅ Yes, {bread, butter} = 3 → ❌	**Not closed**
{milk, bread}	3	No superset with 3 → ✅	**Closed**
{milk, butter}	2	✅ Yes, {milk, bread, butter} = 2 → ❌	**Not closed**
{bread, butter}	3	No superset with 3 → ✅	**Closed**
{milk, bread, butter}	2	No superset → ✅	**Closed**

Maximal Itemsets

A frequent itemset X is **maximal** if and only if it satisfies this condition:

$$\forall\ Y / X \subset Y,\ support\ (Y)\ <\ min\_support$$

In simple terms, a maximal frequent itemset is a frequent itemset that cannot be extended (by adding any single item) without making the resulting new itemset infrequent

Why we care about maximal frequent itemsets ?

To reduce the number of patterns

1) In real datasets, there can be millions of frequent itemsets.
2) Many of them are just subsets of larger ones
3) Gives a **compact summary:** only the largest frequent patterns

Example of Maximal Itemsets

min_support = 3

🛒 Example: Transactions

Transaction ID	Items bought
T1	{milk, bread, butter}
T2	{milk, bread}
T3	{milk, butter}
T4	{bread, butter}
T5	{milk, bread, butter}

Itemset	Appears in	Support	Frequent?
{milk}	T1, T2, T3, T5	4	✅
{bread}	T1, T2, T4, T5	4	✅
{butter}	T1, T3, T4, T5	4	✅
{milk, bread}	T1, T2, T5	3	✅
{milk, butter}	T1, T3, T5	3	✅
{bread, butter}	T1, T4, T5	3	✅
{milk, bread, butter}	T1, T5	2	❌ (below 3)

Itemset	Support	Any frequent superset?	Maximal?
{milk}	4	Yes, {milk, bread} and {milk, butter} are frequent → ❌	Not maximal
{bread}	4	Yes, {milk, bread} and {bread, butter} are frequent → ❌	Not maximal
{butter}	4	Yes, {milk, butter} and {bread, butter} are frequent → ❌	Not maximal
{milk, bread}	3	No frequent superset → ✅	**Maximal**
{milk, butter}	3	No frequent superset → ✅	**Maximal**
{bread, butter}	3	No frequent superset → ✅	**Maximal**

Monotonicity

Fundamental Property: if an itemset A is contained within an itemset B, then:

$$support(A) \geq support(B).$$

- Monotonicity: If X is a frequent itemset and $Y \subseteq X$ (Y is a subset), then Y is a frequent itemset
- Anti - Monotonicity: If X is not a frequent itemset and $X \subseteq Y$, then Y is not a frequent itemset

Anti-monotonicity Property of Strong Association Rules:

If the rule $R = X \Rightarrow Y$ is not strong and $X \subset X'$, $Y \subset Y'$ with $X \cap Y = \emptyset$ and $X' \cap Y' = \emptyset$, then, $R' = X' \Rightarrow Y'$ is not strong either

The **Anti-monotonicity Property** enables faster generation of strong association rules from a frequent itemset.

Determination of Association Rules

The ideal approach is to first find all the frequent itemsets and then deduce the strong association rules from them.

Remark : The number *k*-itemsets that we can retrieve from *I* (containing *m* items) is:

$$\binom{m}{k} = \frac{m!}{k!\,(m-k)!}$$

The total number of itemsets is: $\sum_{k=1}^{k=m} \binom{m}{k} = 2^m - 1$

Generating all possible itemsets (whether frequent or not) exhaustively and then checking them is computationally prohibitive because the number of potential itemsets grows exponentially with the number of items, a phenomenon known as the **combinatorial explosion**.

This is precisely why specialized algorithms like **Apriori** and **FP-Growth** are essential. They avoid this exhaustive search through intelligent strategies

Apriori Algorithm

1994 : Agrawal and his team

Basic Idea:

- Generate iteratively all the frequent k-itemsets
- Starting from $k = 1$ up to a value k' or which there are no more k' frequent itemsets
- For each level k, we rely on the calculations made at level $k - 1$
- We use of the **monotonicity** and **anti-monotonicity** properties.

Apriori Algorithm

Notations :

- L_k: all frequent k-itemsets
- C_k: a set of candidate k-itemsets ($L_k \subset C_k$)

Apriori Algorithm :

- Step 1 (k =1): Generate L_1 (all frequent 1-itemsets)
- Step k: While $L_{k-1} \neq \emptyset$ do:

  - Generate C_k by join $L_{k-1} \bowtie L_{k-1}$

  - Scan the transaction database D in order to calculate the support of each itemset in C_k

  - Prune from C_k the itemsets that are not frequent to identify L_k

  $$L_k \leftarrow \text{Pruning}(C_k)$$

  - Increment k: $k \leftarrow k + 1$

Apriori Algorithm: The Join

We assume that the elements of ***I*** are ordered, and that in each transaction and each computed itemset, the items appear in increasing order.

The join $L_{k-1} \bowtie L_{k-1}$ is calculated as follow:

Apriori Algorithm :

- We take all pairs P, Q from L_{k-1} that have $k-2$ elements in common and such that $p[k-1] < q[k-1]$.

$$P = p[1], p[2], \dots, p[k-1]$$
$$Q = p[1], p[2], \dots, p[k-2], q[k-1]$$

- We construct PQ : $p[1], p[2], \dots, p[k-1], q[k-1]$ and we include PQ in C_k

The **anti-monotonicity property** ensures that this procedure is **correct and complete**: every frequent ***k*-itemset** is included in C_k

Apriori Algorithm: The Pruning

- For each T a transaction in D
  - For each C (a k itemset) in C_k:
    if $C \subset T$, then *counter(C)* = *counter(C)* +1

During pruning, remove from C_k all itemsets whose count is less than the minimum support.

Apriori Algo : Example

T ID	Itemset
0	I1, I2, I5
1	I2, I4
2	I2, I3
3	I1, I2, I4
4	I1, I3
5	I2, I3
6	I1, I3
7	I1, I2, I3, I5
8	I1, I2, I3

order : i1, i2, i3, i4, i5

min supp : 2/9

$C_1 : \{ \underset{6}{i1}, \underset{7}{i2}, \underset{6}{i3}, \underset{2}{i4}, \underset{2}{i5} \}$

$L_1 := C_1$

$C_2 : L_1 \bowtie L_1$

We need to put together, *in the lexical order*, all the items where the second one is highest than the first one

$C_2 = \{ \{i1, i2\} \{i1, i3\} \{i1, i4\} \{i1, i5\} \{i2, i3\} \{i2, i4\} \{i2, i5\} \{i3, i4\} \{i3, i5\} \{i4, i5\} \}$

$C_2 = \{ \underset{4}{\{i1, i2\}} \underset{4}{\{i1, i3\}} \underset{1}{\{i1, i4\}} \underset{2}{\{i1, i5\}} \underset{4}{\{i2, i3\}} \underset{2}{\{i2, i4\}} \underset{2}{\{i2, i5\}} \underset{0}{\{i3, i4\}} \underset{1}{\{i3, i5\}} \underset{0}{\{i4, i5\}} \}$

$C_2 = \{ \underset{4}{\{i1, i2\}} \underset{4}{\{i1, i3\}} \underset{1}{\{\cancel{i1, i4}\}} \underset{2}{\{i1, i5\}} \underset{4}{\{i2, i3\}} \underset{2}{\{i2, i4\}} \underset{2}{\{i2, i5\}} \underset{\cancel{0}}{\{i3, i4\}} \underset{\cancel{1}}{\{i3, i5\}} \underset{\cancel{0}}{\{\cancel{i4, i5}\}} \}$

Apriori Algo : Example

$L_2 = \{ \{i_1, i_2\} \{i_1, i_3\} \{i_1, i_5\} \{i_2, i_3\} \{i_2, i_4\} \{i_2, i_5\} \}$

$C_3 = L_2 \bowtie L_2$

$L_2 = \{ \{i_1, i_2\} \{i_1, i_3\} \{i_1, i_5\} \{i_2, i_3\} \{i_2, i_4\} \{i_2, i_5\} \}$

$L_2 = \{ \{i_1, i_2\} \{i_1, i_3\} \{i_1, i_5\} \{i_2, i_3\} \{i_2, i_4\} \{i_2, i_5\} \}$

Join rule : The first item should be the same, and the last item from the set below should be higher than the last item from the set above

$\{i_1, i_2\} \bowtie \{i_1, i_3\} \Rightarrow \{i_1, i_2, i_3\}$

$\{i_1, i_2\} \bowtie \{i_1, i_5\} \Rightarrow \{i_1, i_2, i_5\}$

$\{i_1, i_3, i_5\}$

$\{i_2, i_3, i_4\}$ $\{i_2, i_3, i_5\}$

$\{i_2, i_4, i_5\}$

$C_3 = \{ \{i_1, i_2, i_3\} \{i_1, i_2, i_5\} \{i_1, i_3, i_5\} \{i_2, i_3, i_4\} \{i_2, i_3, i_5\} \{i_2, i_4, i_5\} \}$

Apriori Algo : Example

$$C_3 = \{ \{i_1, i_2, i_3\} \; \{i_1, i_2, i_5\} \; \{i_1, i_3, i_5\} \; \{i_2, i_3, i_4\} \; \{i_2, i_3, i_5\} \; \{i_2, i_4, i_5\} \}$$

Before we naively count the frequency, we must check for each of this itemsests which one contains a subset that was previously discarded

Remember Anti - Monotonicity: If *X* is not a frequent itemset and *X* ⊆ *Y*, then *Y* is not a frequent itemset

$$C_2 = \{ \underset{4}{\{i_1, i_2\}} \; \underset{4}{\{i_1, i_3\}} \; \underset{1}{\cancel{\{i_1, i_4\}}} \; \underset{2}{\{i_1, i_5\}} \; \underset{4}{\{i_2, i_3\}} \; \underset{2}{\{i_2, i_4\}} \; \underset{2}{\{i_2, i_5\}} \; \underset{0}{\cancel{\{i_3, i_4\}}} \; \underset{1}{\cancel{\{i_3, i_5\}}} \; \underset{0}{\cancel{\{i_4, i_5\}}} \}$$

$$C_3 = \{ \overset{2}{\{i_1, i_2, i_3\}} \; \overset{2}{\{i_1, i_2, i_5\}} \; \{i_1, i_3, i_5\} \; \{i_2, i_3, i_4\} \; \{i_2, i_3, i_5\} \; \{i_2, i_4, i_5\} \}$$

$$L_3 = \{ \{i_1, i_2, i_3\} \; \{i_1, i_2, i_5\} \}$$

Apriori Algo : Example

$C_4 = L_3 \bowtie L_3$

$L_3 = \{\{i_1, i_2, i_3\}\ \{i_1, i_2, i_5\}\}$

$L_3 = \{\{i_1, i_2, i_3\}\ \{i_1, i_2, i_5\}\}$

Join condition: Except the last one, all items should be the same , and the last one down is higher than the last one up

$C_4 = L_3 \bowtie L_3 = \{i_1, i_2, i_3, i_5\}$

$L_4 = \phi$

Frequent Itemsets

$C_1 : \{i_1, i_2, i_3, i_4, i_5\}$

i_1	i_2	i_3	i_4	i_5
6	7	6	2	2

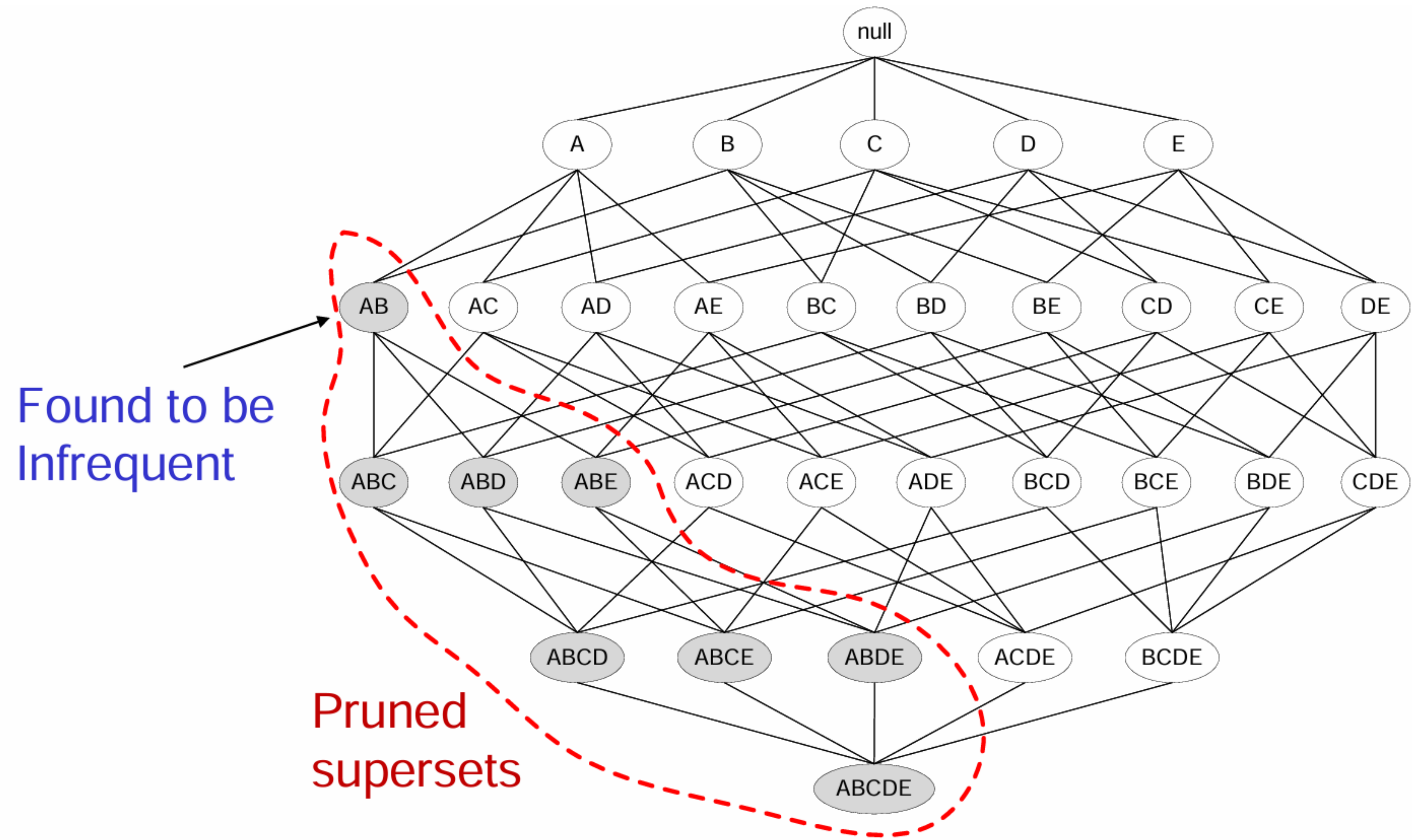
$L_1 : = C_1$

$L_2 = \{\{i_1, i_2\}\{i_1, i_3\}\{i_1, i_5\}\{i_2, i_3\}\{i_2, i_4\}\{i_2, i_5\}\{$

$L_3 = \{\{i_1, i_2, i_3\}\ \{i_1, i_2, i_5\}\}$

$L_4 = \phi$

The Apriori Principle



FP Growth Algorithm

- FP-Growth stands for **Frequent Pattern Growth**.

- It’s an algorithm used in association rule mining to find frequent itemsets

- It’s an improvement over Apriori, because it avoids generating too many candidate sets.

“Instead of generating and testing combinations, FP-Growth builds a compact structure to find frequent patterns efficiently*”*

FP-Growth is the faster frequent pattern mining using tree compression.

What is a Compact Structure ?

In the context of computer science and data mining, a **compact structure** refers to a way of organizing data that:

1. Reduces Redundancy: It avoids storing duplicate information.

2. Minimizes Memory Usage: It uses significantly less memory (RAM) than a naive representation of the data.

3. Preserves Essential Information: Despite being smaller, it contains all the necessary data to perform required operations without needing to go back to the original, larger dataset.

Imagine you have a huge book and you want to analyze how often certain words appear together. The **compact structure way** (like FP-Growth) is to create a single, master **index** for the book. This index is much smaller than the book itself, but it contains all the information about where words appear. You then only need to work with this efficient index, making the process incredibly fast.

Why FP Growth ?

- Apriori scans the database many times and generates many candidates.

- FP-Growth needs only two scans of the database.

- It uses a special data structure called an FP-tree (Frequent Pattern Tree).

FP-Growth is faster because it compresses the data and mines directly from the tree.

Main Steps of FP Growth ?

Step 1: Scan the database

- Count the frequency (support) of each item.
- Remove infrequent items.
- Order the frequent items in **descending frequency**.

Step 2: Build the FP-Tree

- Insert transactions into the tree using the ordered frequent items.
- Shared prefixes of transactions are **merged**. For example, if different shopping lists start the same way, FP-Growth keeps that shared beginning only once in the tree.

Step 3: Mine the FP-Tree

- Start from the least frequent items (bottom of the header table).
- Build conditional FP-trees for each item.
- Extract frequent itemsets from these conditional trees

FP Growth: Advantages and Limitations

Advantages of FP-Growth

✅ Faster than Apriori (no candidate generation)

✅ Fewer database scans (only two)

✅ Compact data representation (FP-tree)

✅ Works well on large datasets

Limitations

⚠️ FP-tree can still become large for dense datasets

⚠️ Harder to implement than Apriori

FP Growth: Advantages and Limitations

Advantages of FP-Growth

✅ Faster than Apriori (no candidate generation)

✅ Fewer database scans (only two)

✅ Compact data representation (FP-tree)

✅ Works well on large datasets

Limitations

⚠️ FP-tree can still become large for dense datasets

⚠️ Harder to implement than Apriori

FP-Tree Construction

Example

Transaction ID	Items
T1	{E,K,M,N,O,Y}
T2	{D,E,K,N,O,Y}
T3	{A,E,K,M}
T4	{C,K,M,U,Y}
T5	{C,E,I,K,O,O}

Step 1 : Identify the frequent Pattern Set

First **scan** and **count** the support of each item

Item	Frequency
A	1
C	2
D	1
E	4
I	1
K	5
M	3
N	2
O	3
U	1
Y	3

A **Frequent Pattern set (L)** is built which will contain all the elements whose frequency is greater than or equal to the minimum support.

As minimum support be 3.

These elements are stored in descending order of their respective frequencies.

After insertion of the relevant items, the set L looks like this:- L = {K : 5, E : 4, M : 3, O : 3, Y : 3}

Step 2 : Oredered-Itemset

Now for each transaction , the respective **Ordered-Itemset** is built

L = {K : 5, E : 4, M : 3, O : 3, Y : 3}

Transaction ID	Items	Ordered-Item Set
T1	{E,K,M,N,O,Y}	{K,E,M,O,Y}
T2	{D,E,K,N,O,Y}	{K,E,O,Y}
T3	{A,E,K,M}	{K,E,M}
T4	{C,K,M,U,Y}	{K,M,Y}
T5	{C,E,I,K,O,O}	{K,E,O}

Step 3 : Tree Data Structure

Create a tree data structure

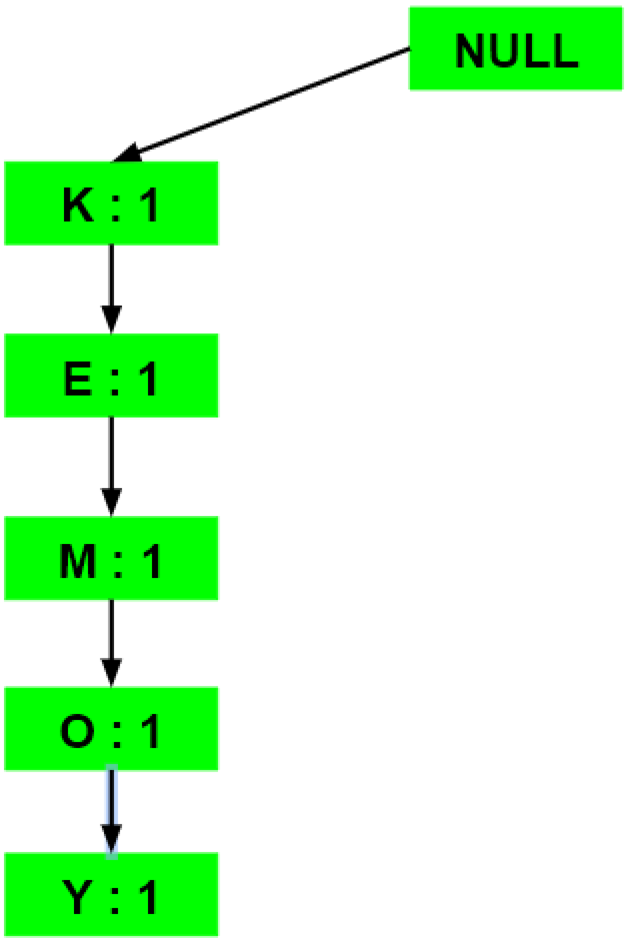
- create the root, label it as "*null*"
- for each Ordered Itemset denoted as *OrdItem*, do:
  - select and sort the *OrdItem*
  - increase nodes count or create new nodes

    If prefix nodes already exist, increase their counts by 1; If no prefix nodes, create it and set count to 1.

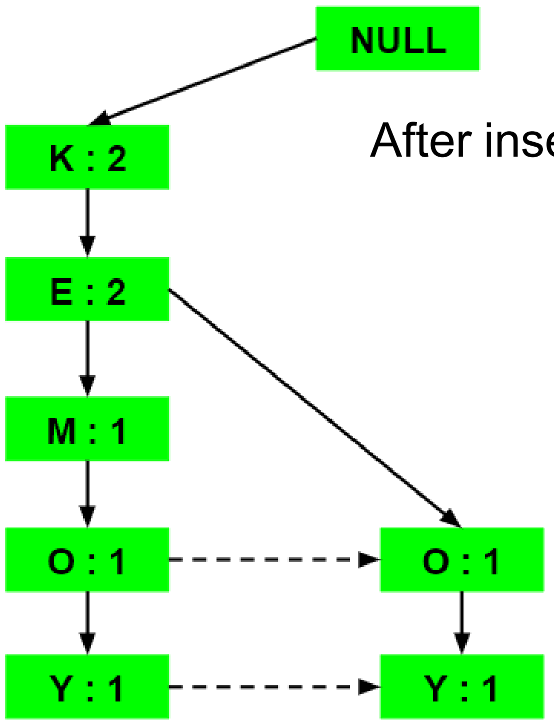
Step 3 : Tree Data Structure

Transaction ID	Items	Ordered-Item Set
T1	{E,K,M,N,O,Y}	{K,E,M,O,Y}
T2	{D,E,K,N,O,Y}	{K,E,O,Y}
T3	{A,E,K,M}	{K,E,M}
T4	{C,K,M,U,Y}	{K,M,Y}
T5	{C,E,I,K,O,O}	{K,E,O}

After inserting {K,E,M,O,Y}



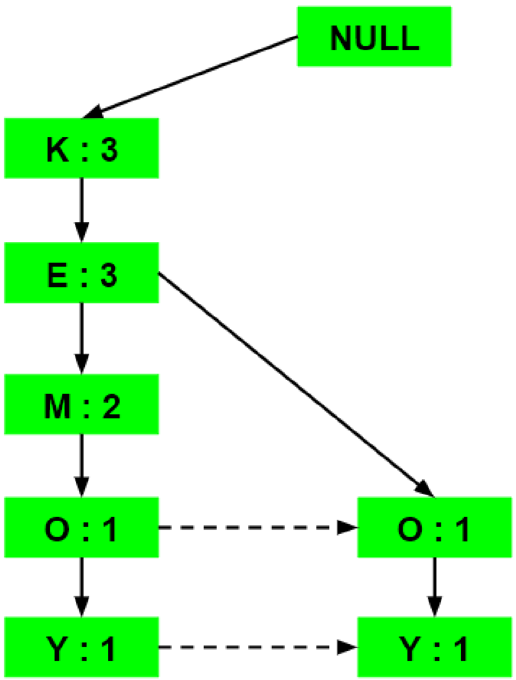
After inserting {K,E,O,Y}



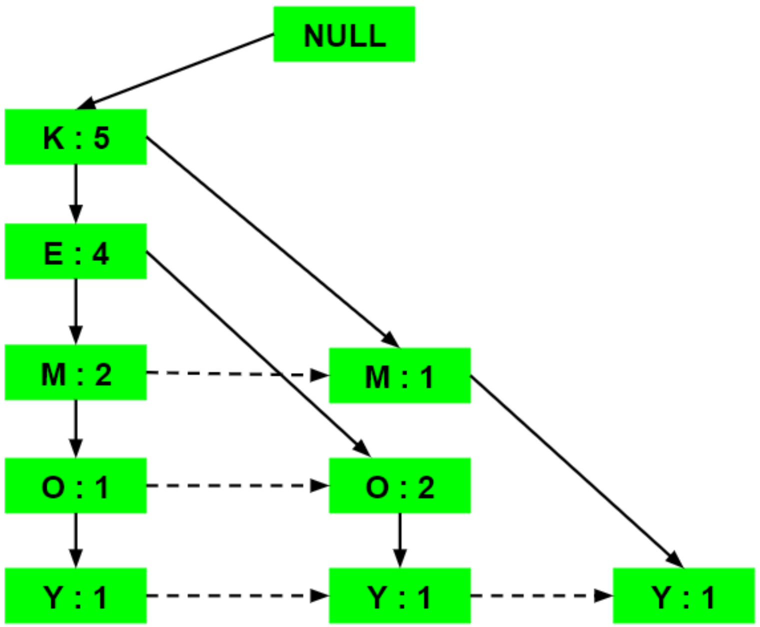
Step 3

Transaction ID	Items	Ordered-Item Set
T1	{E,K,M,N,O,Y}	{K,E,M,O,Y}
T2	{D,E,K,N,O,Y}	{K,E,O,Y}
T3	{A,E,K,M}	{K,E,M}
T4	{C,K,M,U,Y}	{K,M,Y}
T5	{C,E,I,K,O,O}	{K,E,O}

After inserting {K,E,M}



Final tree

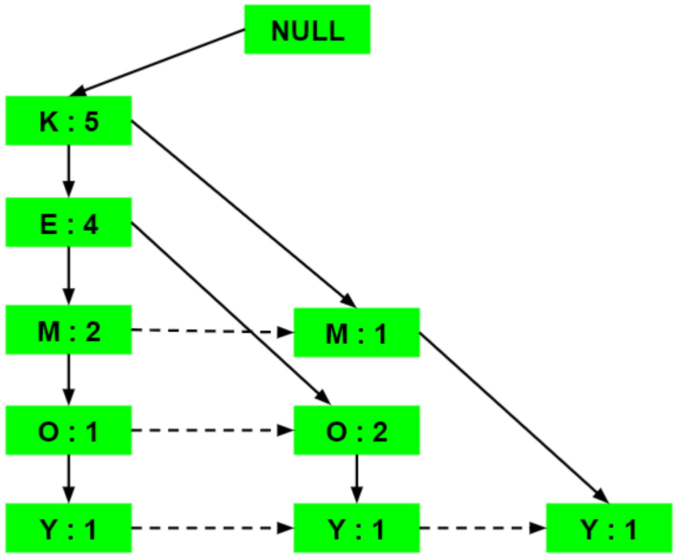


Step 4: Conditional Pattern Base

For each item, the **Conditional Pattern Base** is computed which is path labels of all the paths which lead to any node of the given item in the frequent-pattern tree.

Increasing order

Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

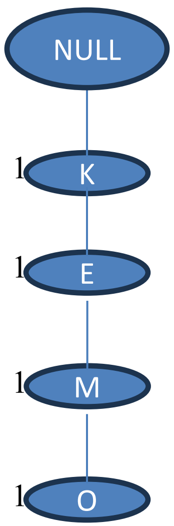


Step 5: Conditional Frequent Pattern Tree

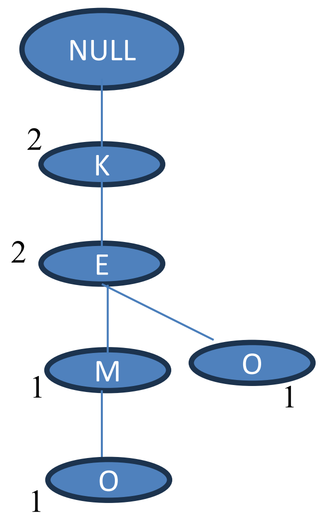
Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

For each item contruct his Conditionnal Frquent Pattern Tree

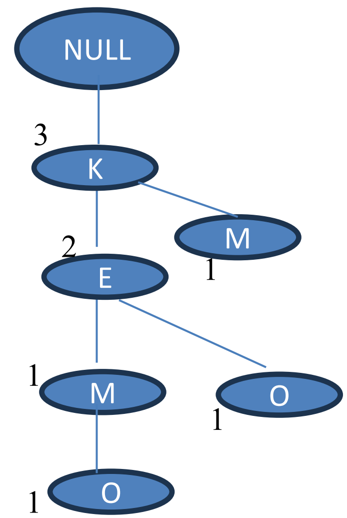
Example of constructing Conditionnal Frequent Pattern Tree for Y



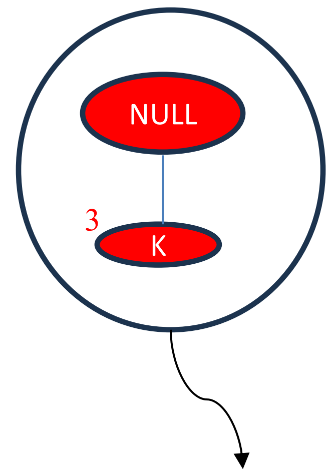
Adding {K,E,M,O :1}



Adding {K,E,O :1}



Adding {K,M :1}



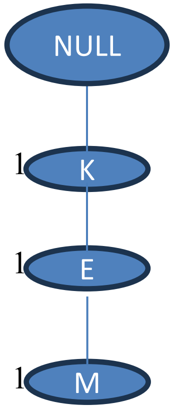
Conditionnal Frequent Pattern Tree for Y

Step 5: Conditional Frequent Pattern Tree

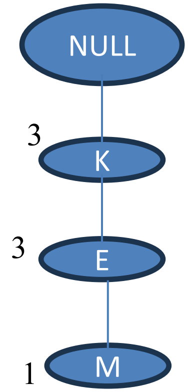
Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

For each item contruct his Conditionnal Frquent Pattern Tree

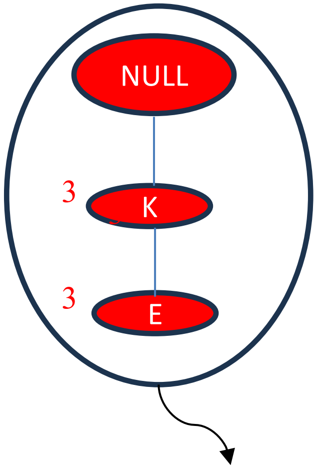
Example of contructing Conditionnal Frequent Pattern Tree for O



Adding {K,E,M :1}



Adding {K,E:2}



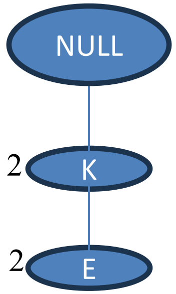
Conditionnal Frequent Pattern Tree for O

Step 5: Conditional Frequent Pattern Tree

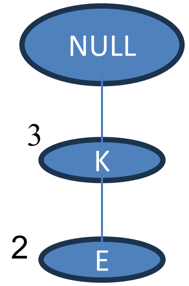
Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

For each item contruct his Conditionnal Frquent Pattern Tree

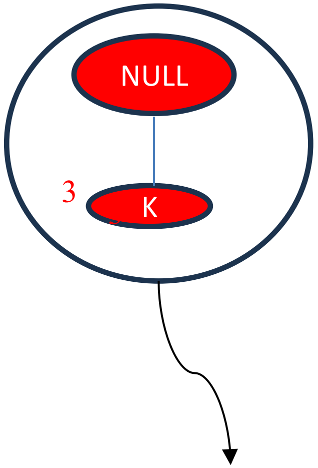
Example of contructing Conditionnal Frequent Pattern Tree for M



Adding {K,E: 2}



Adding {K:1}



Conditionnal Frequent Pattern Tree for M

Step 5: Conditional Frequent Pattern Tree

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

Step 6: Generation of Frequent Itemsets

The frequent *k*-itemsets ($k \neq 1$) are generated by pairing the items of the Conditionnal Frequent Pattern Tree set to the corrsponding item.

Remark: frequent *1*-itemsets are the items

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

Frequent *1*-itemsets = { {Y} :3, {O}:2, {M}:3, {E}:4, {K}:5 }

Frequent *2*-itemsets = { {Y,K}:3, {O,K}:3, {O,E}:3, {M,K}:3, {E,K}:4 }

Frequent *3*-itemsets = { {O,K,E}:3 }

Step 6: Generation of Strong Rules

Suppose minConf = 0.8

As example, for the frequent *2*-itemset {K,Y} there is 2 possible rules: K→Y and Y→K

conf(K→Y) = sup (K ∪ Y) / sup(k) = 3/5 = 0.6 < minConf => K→Y is not a strong rule

conf(Y→K) = sup (Y ∪ *K*) / sup(k) = 3/3 = 1 > minConf => Y→K is a strong rule

For the frequent *3*-itemset {O,K,E}, there is 3 possible rules: (O,K)→E, (O,E)→K, (E,K)→O, E→(K,O), K→(O,E) and O→(K,E)

Is the rule (O,K)→E a strong rule ?

Conf ((O,K)→E) = sup ((O,K,E)) / sup (O,K) = 3 / 3 =1 > minConf => (O,K)→E is a strong rule

Try the same example with Apriori, you should get the same result

FP-Growth vs Apriori: Key Comparison

Feature	FP-Growth	Apriori
Approach	Pattern growth using FP-tree	Candidate generation & test
Speed	Fast (no candidate generation)	Slow (many candidates)
Memory Usage	Higher (stores FP-tree)	Lower
Scalability	Excellent for large datasets	Poor for large datasets
Best For	Dense data, production systems	Sparse data, education
Key Advantage	Efficiency on large datasets	Simplicity & ease of understanding

Possible Extensions

Many possible extensions:

- **Association rules with intervals**:
  - For example: Men over 65 have 2 cars
- **Association rules when items are in a taxonomy**
  - Bread, Butter → FruitJam
  - BakedGoods, MilkProduct → PreservedGoods
- **Mining rare association rules**
- **Sequential Pattern Mining**
- **Mining negative association rules**