CSE 435:Data Mining

Chapter 4: Frequent Pattern and Association Rule Mining

Md Atikuzzaman
Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

Outline

What is Frequent Pattern Mining?

The Classic Example: Market Basket Analysis

Retailers analyze customer transaction data to find items that are frequently purchased together. **Example:** "If a customer buys bread and butter, they are likely to also buy milk."

- This is a **frequent pattern**: {Bread, Butter, Milk}
- This suggests an **association rule**: {Bread, Butter} $\rightarrow$ {Milk}

Applications

- Retail: Product placement, promotions, cross-selling.
- Web Mining: Recommendations (e.g., "Customers who bought this also bought...").
- Bioinformatics: Finding co-occurring gene mutations.

Core Definitions

- **Item:** A single object or attribute.
  - Ex: ‘Milk‘, ‘Bread‘, ‘Diapers‘.
- **Itemset (X):** A collection of one or more items.
  - Ex: $\{$Milk, Bread$\}$.
- **Transaction (T):** A single record containing an itemset.
- **Transaction Database (D):** A collection of transactions.

Key Measure 1: Support

Support

How frequently an itemset appears in the database.

- **Support Count ($\sigma(X)$):** The raw count of transactions containing itemset X.
- **Support ($s(X)$):** The fraction of transactions containing itemset X.

$$s(X) = \frac{\sigma(X)}{\text{Total \# of transactions (N)}} = P(X)$$

Frequent Itemset

An itemset is considered **frequent** if its support is greater than or equal to a user-defined **minimum support threshold (minsup)**.

Key Measure 2: Confidence & Association Rules

Association Rule

An implication of the form $X \rightarrow Y$, where X and Y are disjoint itemsets. It suggests a relationship between the itemsets.
The measure of how often items in Y appear in transactions that contain X.

- For a rule $X \rightarrow Y$, it is the conditional probability $P(Y|X)$.

$$\text{conf}(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$$

Strong Rule

A rule is **strong** if it satisfies both 'minsup' and a **minimum confidence threshold (minconf)**.

The Apriori Algorithm

Core Idea

A "bottom-up" approach where frequent itemsets of length k are used to generate candidate itemsets of length $k + 1$.

The Apriori Principle

This is the key to reducing the search space and making the problem tractable. *If an itemset is frequent, then all of its subsets must also be frequent.*

How it's used (Contrapositive)

If any subset of a candidate itemset is infrequent, then the candidate cannot be frequent and can be pruned. For example, if we find that {Beer, Diapers} is infrequent, we can immediately discard {Beer, Diapers, Nuts} without ever needing to count its support.

Apriori Algorithm: Step-by-Step

- Let C_k be the set of candidate itemsets of size k.
- Let L_k be the set of frequent itemsets of size k.

1. Scan the database to find L_1 (frequent 1-itemsets).
2. For $k = 2, 3, \ldots$
   - Use L_{k-1} to generate candidates C_k (**Join Step**).
   - Prune candidates in C_k that have an infrequent subset (**Prune Step** using Apriori Principle).
   - Scan the database to count support for the remaining candidates in C_k to find L_k.
3. Repeat until L_k is empty.

Apriori Example

Database (D)

TID	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Parameters:

- 'minsup = 2'
- 'minconf = 70

Step 1: Find L1

- Counts: A:2, B:3, C:3, D:1, E:3
- Prune {D}
- $L_1 = \{\{A\}, \{B\}, \{C\}, \{E\}\}$

Step 2: Find L2

- Join L_1: {A,C}, {B,C}, {B,E}, {C,E}, ...
- Counts: {A,C}:2, {B,C}:2, {B,E}:3, {C,E}:2
- $L_2 = \{\{A, C\}, \{B, C\}, \{B, E\}, \{C, E\}\}$

Step 3: Find L3

- Join L_2: {B,C,E}
- Prune: Subsets {B,C}, {B,E}, {C,E} are all in L_2. Keep.

Generating Association Rules

Process

For each frequent itemset l, generate all non-empty proper subsets s. For each s, form the rule $s \rightarrow (l - s)$ and check if its confidence is above 'minconf'.

$$\text{conf} = \frac{\sigma(l)}{\sigma(s)}$$

Generating Association Rules

Example using $\{B, C, E\}$ (from L_3)

We know $\sigma(\{B, C, E\}) = 2$, $\sigma(\{B, C\}) = 2$, $\sigma(\{B, E\}) = 3$. 'minconf = 70

1. **Rule:** $\{B, C\} \rightarrow \{E\}$
   - conf $= \frac{\sigma(\{B,C,E\})}{\sigma(\{B,C\})} = \frac{2}{2} = 100\%$
   - $100\% \geq 70\% \implies$ Strong Rule

2. **Rule:** $\{B, E\} \rightarrow \{C\}$
   - conf $= \frac{\sigma(\{B,C,E\})}{\sigma(\{B,E\})} = \frac{2}{3} \approx 67\%$
   - $67\% < 70\% \implies$ Weak Rule

Improving the Efficiency of Apriori

Apriori's Bottlenecks

1. Can generate a huge number of candidate sets.
2. Requires multiple scans of the entire database (one for each k-level).

Improvement Techniques

- **Hash-based Pruning:** Use a hash table in the first pass to filter candidate 2-itemsets.
- **Transaction Reduction:** After finding L_k, remove any transaction that does not contain at least one frequent k-itemset.
- **Partitioning:** Divide the database, find frequent itemsets locally, then consolidate globally with a final verification scan.
- **Sampling:** Mine on a random sample of the database.

A Pattern-Growth Approach: FP-Growth

Philosophy

Avoid the expensive "candidate generation and test" paradigm of Apriori.

Method

1. **Build FP-Tree:** Scan the database twice to build a compressed representation of the database in a tree structure (FP-Tree).
2. **Mine FP-Tree:** Recursively "mine" the tree by extracting frequent itemsets directly from conditional pattern bases.

A Pattern-Growth Approach: FP-Growth

Advantages

- Much faster than Apriori, especially on dense datasets.
- No candidate generation.
- Only two scans of the database.

Disadvantages

- Can be memory-intensive; the entire FP-Tree must fit into main memory.
- The algorithm is more complex to implement than Apriori.
- It can be expensive to rebuild the tree if the database is dynamic and updated frequently.

FP-Growth Example (1/4): Initial Scan

Let's use a sample database with a **minimum support threshold (σ) of 3**.

Step 1: Scan DB and Find Frequent 1-itemsets

Transaction Database

TID	Items
100	f, a, c, d, g, i, m, p
200	a, b, c, f, l, m, o
300	b, f, h, j, o, w
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

Item Counts (Support)

- f: 4, c: 4, a: 3, b: 3, m: 3, p: 3
- *d: 1, g: 1, i: 1, l: 2, o: 2, ... (infrequent)*

Sorted Frequent Items (L)

Items are sorted by decreasing support. For ties, we use alphabetical order.

$$L = \langle (f:4), (c:4), (a:3), (b:3), (m:3), (p:3) \rangle$$

FP-Growth Example (2/4): Construct the FP-Tree

Step 2: Construct the FP-Tree

Scan the database a second time. For each transaction, filter and sort its frequent items according to the order in $L = \langle f, c, a, b, m, p \rangle$.

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

Header Table

Item	Count
f	4
c	4
a	3
b	3
m	3
p	3

FP-Tree Construction: Initial State

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

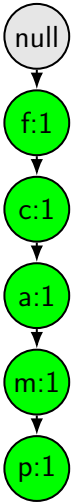
The tree starts with a single null root. We will process transactions one by one.

Processing TID 100: ⟨f, c, a, m, p⟩

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

A new path is created for the transaction. Each node has a count of 1.

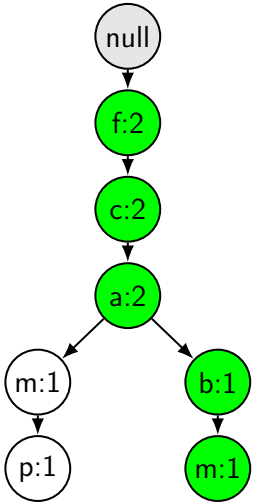


Processing TID 200: ⟨f, c, a, b, m⟩

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

The path ⟨f, c, a⟩ is shared. Their counts are incremented. A new branch ⟨b:1, m:1⟩ is added under 'a'.

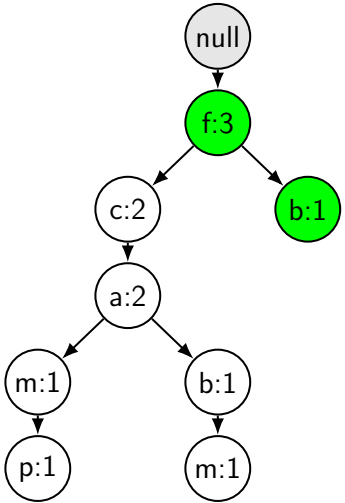


Processing TID 300: ⟨f, b⟩

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

The path starts with 'f', so its count is incremented. 'b' is not a child of 'f', so a new branch ⟨b:1⟩ is created.

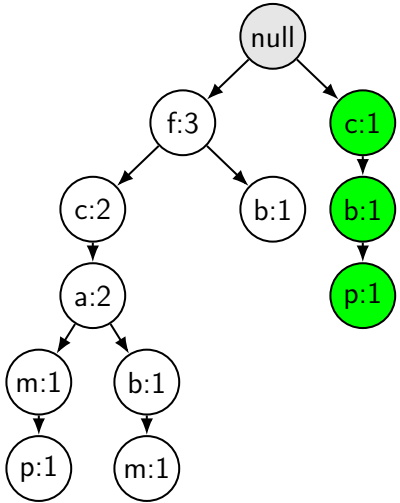


Processing TID 400: ⟨c, b, p⟩

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

This transaction does not start with 'f'. A new path ⟨c, b, p⟩ is created directly from the root.

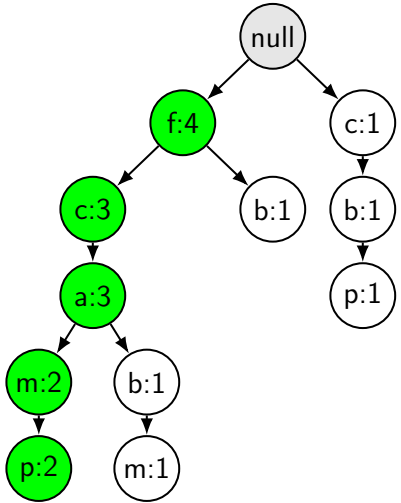


Processing TID 500: ⟨f, c, a, m, p⟩

Ordered Frequent Itemsets

TID	Ordered Items
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

This path already exists from TID 100. We traverse the path and increment the count of each node along the way.

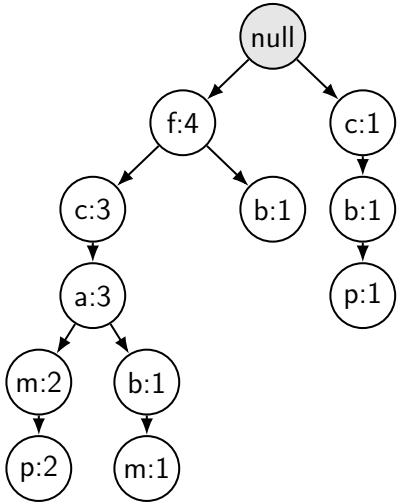


Final FP-Tree

Header Table

Item	Count
f	4
c	4
a	3
b	3
m	3
p	3

All transactions have been processed. This is the final constructed FP-Tree.



FP-Growth Example (3/4): Mining the FP-Tree

Step 3: Mine the Tree Recursively

Mine frequent patterns by creating *Conditional FP-Trees* for each item in the header table, starting from the bottom and working up.

1. **Find Conditional Pattern Base for 'p'**: Collect all prefix paths for nodes of 'p' in the FP-Tree.
   - The node `p:2` has prefix path $\langle f, c, a, m \rangle$. Set its count to 2.
   - The node `p:1` has prefix path $\langle c, b \rangle$. Set its count to 1.

   Conditional Pattern Base: $\{(\langle f, c, a, m \rangle : 2), (\langle c, b \rangle : 1)\}$
2. **Build Conditional FP-Tree for 'p'**: From the base, count item frequencies: $c : 2 + 1 = 3$, $f : 2$, $a : 2$, $m : 2$, $b : 1$. With $\sigma = 3$, only 'c' is frequent. The Conditional FP-Tree for 'p' has one node: $\langle c : 3 \rangle$.
3. **Generate Frequent Patterns**: The pattern found is {c}. Combine it with the suffix 'p'. Generated Pattern: {c, p} with support 3.

FP-Growth Example (4/4): Final Frequent Patterns

Next Steps

This process is repeated for **m, b, a, c,** and **f** to discover all frequent itemsets. For each item, a new conditional base and a new conditional tree are built.

Conclusion

The FP-Growth algorithm has generated all 12 frequent itemsets (of size ≥ 3) without any candidate generation, making it significantly more efficient than Apriori for this database.

Data Formats for Itemset Mining

Frequent itemset mining algorithms typically use one of two data layouts:

Horizontal Format

- Standard transaction database.
- Each row is a transaction ID (TID) followed by the items it contains.
- Example: `T1: {Milk, Bread, Butter}`
- Used by algorithms like Apriori.

Vertical Format

- Inverted index structure.
- Each row is an **item** followed by the set of TIDs where it appears (a "Tidset").
- Example: `Milk: {T1, T4, T5}`
- Used by algorithms like **Eclat**.

The vertical format can be much faster for finding support because it avoids repeatedly scanning the entire database.

Example: Converting to Vertical Format

Let's consider a sample transaction database with a **minimum support count of 3**.

Horizontal Data (Original)

TID	Items
T1	{A, B, C}
T2	{A, C, D}
T3	{A, B, D}
T4	{B, E}
T5	{A, B, C, D}

Vertical Data (Converted)

Item	Tidset	Support
A	{T1, T2, T3, T5}	4
B	{T1, T3, T4, T5}	4
C	{T1, T2, T5}	3
D	{T2, T3, T5}	3
E	{T4}	1

Key Idea: The support of an itemset is simply the **size of its corresponding Tidset**.

The Eclat Algorithm: Mining with Intersections

Eclat (Equivalence Class Clustering and bottom-up Lattice Traversal) finds frequent itemsets by intersecting Tidsets.

Core Principle: The Tidset of a larger itemset, like $\{A, B\}$, can be found by intersecting the Tidsets of its smaller subsets, $\{A\}$ and $\{B\}$.

$$Tidset(XY) = Tidset(X) \cap Tidset(Y)$$

$$support(XY) = |Tidset(X) \cap Tidset(Y)|$$

The Process:

1. Start with frequent 1-itemsets (those meeting min_sup). This is our set L_1.
2. Generate candidate 2-itemsets by combining members of L_1.
3. For each candidate $\{X, Y\}$, calculate its support by intersecting $Tidset(X)$ and $Tidset(Y)$. If the support is sufficient, add it to L_2.
4. Repeat for $L_2, L_3, \ldots$ until no more frequent itemsets can be found.

Eclat Example: Step 1 (Finding 1-Itemsets)

Goal: Find all frequent 1-itemsets (L_1).
Minimum Support Count: 3

We scan the vertical database to find all items that meet the minimum support.

Item	Tidset	Support	Is Frequent?
A	{T1, T2, T3, T5}	4	**Yes**
B	{T1, T3, T4, T5}	4	**Yes**
C	{T1, T2, T5}	3	**Yes**
D	{T2, T3, T5}	3	**Yes**
E	{T4}	1	No

The set of frequent 1-itemsets is $L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}\}$. We discard item E.

Eclat Example: Step 2 (Finding 2-Itemsets)

Goal: Generate frequent 2-itemsets (L_2) from L_1.
Method: Intersect the Tidsets of all pairs from L_1.

Candidate	Tidset Intersection	Support	Is Frequent?
{A, B}	Tidset(A) ∩ Tidset(B) = {T1, T3, T5}	3	**Yes**
{A, C}	Tidset(A) ∩ Tidset(C) = {T1, T2, T5}	3	**Yes**
{A, D}	Tidset(A) ∩ Tidset(D) = {T2, T3, T5}	3	**Yes**
{B, C}	Tidset(B) ∩ Tidset(C) = {T1, T5}	2	No
{B, D}	Tidset(B) ∩ Tidset(D) = {T3, T5}	2	No
{C, D}	Tidset(C) ∩ Tidset(D) = {T2, T5}	2	No

The set of frequent 2-itemsets is $L_2 = \{\{A, B\}, \{A, C\}, \{A, D\}\}$.

Eclat Example: Step 3 (Finding 3-Itemsets)

Goal: Generate frequent 3-itemsets (L_3) from L_2.
Method: Combine itemsets in L_2 that share a common prefix.

Our candidates are formed from {A, B}, {A, C}, and {A, D}. The only possible candidate is {A, B, C, D}. We can check its subsets.

- Candidate {A, B, C}: Check support by intersecting Tidsets of {A, B} and {A, C}.
- Candidate {A, B, D}: Check support by intersecting Tidsets of {A, B} and {A, D}.
- ... and so on.

Candidate	Tidset Intersection	Support	Is Frequent?
{A, B, C}	Tidset({A,B}) ∩ Tidset(C) = {T1, T5}	2	No
{A, B, D}	Tidset({A,B}) ∩ Tidset(D) = {T3, T5}	2	No
{A, C, D}	Tidset({A,C}) ∩ Tidset(D) = {T2, T5}	2	No

Since no 3-itemsets are frequent, the algorithm terminates.

Final Result & Advantages

By combining all frequent itemsets found (L_1 and L_2), we get the final result.

All Frequent Itemsets (min_sup=3):

- $\{A\}, \{B\}, \{C\}, \{D\}$
- $\{A, B\}, \{A, C\}, \{A, D\}$

Advantages of the Vertical Format / Eclat:

- **Fast Support Counting:** Support is found by intersecting Tidsets, which is much faster than scanning a large database.
- **No Repeated Scans:** The original database is only scanned once to create the initial vertical format.
- **Scalability:** Works well with a very large number of transactions, especially if the Tidsets fit in memory.

The Problem with Frequent Itemsets

Mining all frequent itemsets often produces a huge number of patterns, many of which are redundant.

Example: If the itemset `{Milk, Bread, Butter}` is frequent, then all of its subsets are also frequent by definition:

- `{Milk, Bread}`
- `{Milk, Butter}`
- `{Bread, Butter}`
- `{Milk}, {Bread}, {Butter}`

This redundancy makes the results difficult to analyze. We need more compact representations.

Solution: Mine for condensed pattern types.

1. **Closed Frequent Itemsets**: A lossless compression.
2. **Max Frequent Itemsets**: A lossy compression.

Closed Frequent Itemsets

Definition

An itemset X is a **closed frequent itemset** if it is frequent and none of its immediate supersets have the exact same support count as X.

Intuition : A closed pattern is the "fullest" or most complete itemset that appears in a specific set of transactions. Adding any other item to it will necessarily make it appear in fewer transactions.

Why is it useful?

- **Lossless Compression:** The set of closed frequent itemsets is much smaller than the set of all frequent itemsets, but it contains all the necessary information.
- We can derive all frequent itemsets and their exact support counts from the closed set.

Example: Finding Closed Patterns

Dataset (min_sup = 2)

TID	Items
T1	{A, B, C}
T2	{B, C, D}
T3	{A, B, C, D}
T4	{A, B}
T5	{D}

Frequent Itemsets & Support

- {A}: 3 → Not Closed (superset {A,B} has same sup)
- {C}: 3 → Not Closed (superset {B,C} has same sup)
- **{B}: 4** ✓ **Closed**
- **{D}: 3** ✓ **Closed**
- **{A, B}: 3** ✓ **Closed**
- {A, C}: 2 → Not Closed (superset {A,B,C} has same sup)
- **{B, C}: 3** ✓ **Closed**
- {B, D}: 2 → Not Closed (superset {B,C,D} has same sup)
- **{A, B, C}: 2** ✓ **Closed**
- **{B, C, D}: 2** ✓ **Closed**

Max Frequent Itemsets

Definition

An itemset X is a **max frequent itemset** (or maximal) if it is frequent and none of its immediate supersets are frequent.

Intuition: A max pattern represents the "peak" or boundary of frequent patterns. It's a frequent itemset that cannot be extended any further without becoming infrequent.

Why is it useful?

- **Lossy Compression:** Provides the most compact summary of frequent patterns.
- The set is much smaller than the closed set.
- **Limitation:** You know a max pattern's subsets are frequent, but you lose the information about their specific support counts.

Example: Finding Max Patterns

Dataset (min_sup = 2) - Using the same frequent itemsets as before.

TID	Items
T1	{A, B, C}
T2	{B, C, D}
T3	{A, B, C, D}
T4	{A, B}
T5	{D}

Frequent Itemsets

- {A}, {B}, {C}, {D} → Not Max
- {A, B}, {A, C} → Not Max
- {B, C}, {B, D} → Not Max
- **{A, B, C}: 2** ✓ **Max**
- **{B, C, D}: 2** ✓ **Max**

Reasoning: The itemset `{A,B}` is not max because it can be extended to the frequent itemset `{A,B,C}`. However, `{A,B,C}` is max because no superset (like `{A,B,C,D}`) is frequent.

Frequent vs. Closed vs. Max

The three types of patterns form a hierarchy of subsets.

Max Patterns $\subset$ Closed Patterns $\subset$ All Frequent Patterns

Pattern Type	Key Property	Information Content
Frequent	All itemsets with *support* $\geq$ *min_sup*.	Complete but highly redundant.
Closed	A compact set with no redundancy.	**Lossless**: All frequent patterns and their supports can be recovered.
Max	The "longest" frequent patterns that cannot be extended.	**Lossy**: You know the subsets are frequent, but you don't know their exact supports.

Choosing which to mine depends on whether you can afford to lose support count information for the sake of conciseness.

The Illusion of Strong Rules

In data mining, a "strong" rule is typically one with high **support** and high **confidence**.

- **Support:** How often the items in the rule appear together.
- **Confidence:** How reliable the rule is ($P(Y|X)$).

The Central Problem

Just because a rule is statistically strong does not mean it is useful, insightful, or valuable. Many strong rules are completely uninteresting.

Interestingness depends on factors beyond raw numbers, such as unexpectedness and actionability.

Case 1: The Trivial or "Common Sense" Rule

A rule is uninteresting if it merely confirms obvious, pre-existing knowledge. It provides no new insight.

Example: A Supermarket

- **Rule:** `{Burger Buns} -> {Burger Patties}`
- **Metrics:**
  - Support = 80% (very high)
  - Confidence = 90% (extremely high)

Analysis

This is an exceptionally strong rule. However, it is utterly uninteresting. It tells the store manager something they already know from common sense. It doesn't reveal a hidden customer behavior or suggest a new strategy.

Case 2: The Misleading High-Confidence Rule

High confidence can be an illusion if the consequent item (the "then" part) is extremely popular by itself.

Example: A Convenience Store

- Let's assume 85
- We find the following rule: `{Potato Chips} -> {Bottled Water}`
- **Metrics:**
  - $s(\text{Chips}) = 10\%$, $s(\text{Chips} \cup \text{Water}) = 8.5\%$
  - **Confidence** = $\frac{s(\text{Chips} \cup \text{Water})}{s(\text{Chips})} = \frac{0.085}{0.10} = 85\%$

Is it Interesting?

A confidence of 85

$$Lift = \frac{Conf(X \rightarrow Y)}{s(Y)} = \frac{0.85}{0.85} = 1.0$$

A Lift of 1.0 means there is **no association** between the items. Buying chips doesn't make a customer more or less likely to buy water. The high confidence was purely a result of water's overall popularity.

Case 3: The Unactionable Rule

A rule can be statistically strong and even surprising, but still be uninteresting if it doesn't lead to a clear, profitable business action.

Example: Analyzing Web Traffic

- **Rule:** `{Visited 'Contact Us' page} -> {Exited Website}`
- **Metrics:**
  - Support = 90%
  - Confidence = 98%

Analysis

This is a very strong rule. It tells us that almost everyone who visits the contact page leaves immediately after. While this might point to a web design issue, it's not "interesting" from a marketing or sales perspective.

- **What is the action?** You cannot stop people from leaving after they find the information they need.
- The rule describes a functional, terminal behavior, not a pattern you can leverage for cross-selling or promotion.

Conclusion: Strong Interesting

The strength of a rule is only the first step. To be truly interesting, a pattern must provide value.

	Strong but Uninteresting Rule	**Interesting Rule**
Example	`{Milk} -> {Bread}`	`{Diapers} -> {Beer}`
Characteristic	Confirms the obvious. Statistically significant but practically useless.	Reveals a surprising, unexpected, or counter-intuitive connection.
Key Metric	High Confidence, but **Lift** $\approx$ **1**.	High **Lift** (> 1), even if support is lower.
Outcome	No new knowledge gained.	Provides a clear opportunity for action (e.g., product placement, bundling, targeted ads).

Final Takeaway: Always look beyond support and confidence. Use measures like Lift and human expertise to find the patterns that truly matter.

Beyond Frequency: The Challenge of Interestingness

Mining algorithms can generate thousands of frequent patterns. However, most of them are not useful.

The Core Problem

A pattern being **frequent** does not automatically make it **interesting**.

Example of an Uninteresting Pattern:

- In a supermarket, the rule `{Milk} -> {Bread}` might have high support and high confidence.
- **Why is it uninteresting?** This is common knowledge. It doesn't provide new insight or a competitive advantage. It's obvious.

An interesting pattern is one that is **unexpected**, **actionable**, and provides real value.

Objective vs. Subjective Measures

We can evaluate interestingness using two different approaches:

Objective Measures

- Based on the statistical properties of the data.
- Does not require domain knowledge.
- Key measures include:
  - Support
  - Confidence
  - Lift
  - All Confidence
  - Chi-Squared (χ^2)

Subjective Measures

- Requires human expertise and domain knowledge.
- A pattern is interesting if the user finds it valuable.
- Key concepts include:
  - Unexpectedness
  - Actionability
  - Novelty

Objective Interestingness: Lift

Lift is a powerful measure that tells us how much more likely two items are purchased together than if they were statistically independent.

Formula

$$Lift(X \rightarrow Y) = \frac{Conf(X \rightarrow Y)}{s(Y)} = \frac{s(X \cup Y)}{s(X)s(Y)}$$

Interpretation:

- **Lift = 1:** X and Y are independent. The rule is not interesting.
- **Lift > 1:** X and Y have a positive correlation. The presence of X increases the likelihood of Y. **Potentially interesting!**
- **Lift < 1:** X and Y have a negative correlation (substitutes). Also potentially interesting!

Example: Consider the rule `{Diapers} -> {Beer}`. If $Lift(\text{Diapers} \rightarrow \text{Beer}) = 2.5$, it means customers who buy diapers are 2.5 times more likely to buy beer than the average customer.

Subjective: Unexpectedness Vs Actionability

Even a statistically significant pattern is useless if it's not subjectively interesting.

Unexpectedness

- A pattern is interesting if it contradicts existing knowledge or hypotheses.
- **Example:** In a hardware store, finding that `{Smoke Detectors} -> {House Plants}` is unexpected. Why would these be related? This prompts further investigation.
- Obvious patterns like `{Hammer} -> {Nails}` are expected and uninteresting.

Actionability

- A pattern is interesting if it provides a clear path to taking profitable action.
- **Example:** Discovering that `{Grill Charcoal} -> {Steak Sauce}` on weekends is highly actionable.
- **Action:** Place a display of steak sauce next to the charcoal before the weekend to boost sales.

Putting It All Together: A Case Study

Imagine analyzing data from an electronics store.

Pattern	Metrics	Conclusion
`{Laptop} -> {Charger}`	High Support, High Confidence, **Lift** $\approx$ **1**	**Uninteresting**. It's an obvious, functional relationship. No new insight.
`{Gaming Console} -> {Energy Drinks}`	Moderate Support, High Confidence, **Lift = 3.2**	**Interesting! (Objective)**. The items are purchased together much more than expected.
`{New Smartphone} -> {Screen Protector}`	High Support, High Confidence, **Lift = 1.5**	**Actionable! (Subjective)**. While somewhat expected, the store can act on this by training staff to always offer a screen protector with a phone sale.
`{Digital Camera} -> {Travel Guide Books}`	Low Support, Moderate Confidence, **Lift = 4.0**	**Very Interesting! (Subjective & Objective)**. This unexpected link suggests a customer segment (travelers) and inspires a new marketing strategy (e.g., a "Travel Tech" bundle).

Comparison of Pattern Evaluation Measures

Measure	Formula	Interpretation	Pros	Cons
Support	$s(X \cup Y)$	Frequency of the pattern.	Easy to compute; fundamental for pruning.	Not an indicator of interestingness.
Confidence	$\frac{s(X \cup Y)}{s(X)}$	Strength of the implication.	Intuitive and easy to understand.	Can be misleading if consequent (Y) is very frequent.
Lift	$\frac{s(X \cup Y)}{s(X)s(Y)}$	Degree of correlation between X and Y.	True measure of interestingness vs. a baseline.	Can be sensitive to low support values.

Comparison of Pattern Evaluation Measures (cont.)

Measure	Formula	Interpretation	Pros	Cons
All-Confidence	$\frac{s(X \cup Y)}{\max(s(X), s(Y))}$	The minimum confidence of the two rules $X \rightarrow Y$ and $Y \rightarrow X$.	Symmetric; not biased by rule direction; null-invariant.	Value is constrained by the more frequent item.
Chi-Squared (χ^2)	$\sum \frac{(O-E)^2}{E}$	Statistical test for independence between X and Y.	Provides statistical significance.	Not normalized; sensitive to sample size.
Jaccard	$\frac{s(X \cup Y)}{s(X)+s(Y)-s(X \cup Y)}$	Ratio of intersection to union of transactions.	Symmetric; intuitive similarity measure.	Penalizes patterns where one itemset is a subset of a much larger one.

Comparison of Pattern Evaluation Measures (cont.)

Measure	Formula	Interpretation	Pros	Cons
Cosine	$\frac{s(X \cup Y)}{\sqrt{s(X)s(Y)}}$	Geometric mean of the confidences of $X \to Y$ and $Y \to X$.	Symmetric; less sensitive to item frequencies than Lift.	Can be less intuitive to interpret directly; null-invariant.
Kulczynski	$\frac{1}{2}\left(\frac{s(X \cup Y)}{s(X)} + \frac{s(X \cup Y)}{s(Y)}\right)$	Arithmetic mean of the confidences of $X \to Y$ and $Y \to X$.	Symmetric; provides a balanced view of the relationship; null-invariant.	Can be high even if one confidence is very low.
Max-Conf	$\max\left(\frac{s(X \cup Y)}{s(X)}, \frac{s(X \cup Y)}{s(Y)}\right)$	The higher confidence between the two rule directions.	Good for finding strong one-way predictive rules; null-invariant.	Asymmetric; ignores the weaker relationship direction.

Summary of Key Concepts

- **Frequent Pattern Mining** discovers itemsets that co-occur frequently (*support*).
- **Association Rules** ($X \rightarrow Y$) measure the strength of an implication (*confidence*).
- The **Apriori Algorithm** is a classic "generate-and-test" approach that uses the Apriori Principle for pruning.
- **FP-Growth** is an efficient "pattern-growth" approach that avoids candidate generation.
- High support and confidence are not enough! A rule is only interesting if it reveals a true correlation.
- **Lift** and other correlation measures are crucial for finding genuinely actionable patterns.
- **Closed and Max patterns** provide a compressed, less redundant representation of frequent itemsets.

References

[1] Jiawei Han, Micheline Kamber, & Jian Pei, *Data Mining: Concepts and Techniques*, 4th Edition, Morgan Kaufmann, 2012.

[2] David J. Hand, Heikki Mannila, & Padhraic Smyth, *Principles of Data Mining*, First Edition, A Bradford Book, 2001.

[3] Richard O. Duda, Peter E. Hart, & David G. Stork, *Pattern Classification*, 2nd Edition, Wiley, 2001.