Association Analysis

CI603 - Data Mining

Lecture 3 and 4

Market Basket Analysis

- **It is not only an assortment of items.**

Image source: https://favpng.com

- What one customer purchased at one time.
- A complete list of purchases made by all customers provides much more information.

What merchandise customers are buying and when.

Much more than just the content of their basket

- How characteristics of customers affect their purchases.
- What customers do not purchase, and why.
- Key drivers of purchase.

Image source: https://favpng.com

Market Basket Analysis

- Not actually a single technique, but it refers to a set of business problems related to understanding point-of-sale transaction data.
- Applications have been expanded to many different domains. i.e. Understand the parts of a website that customers visit**.**
- **Association Analysis** is the data mining technique most closely identified with *Market Basket Analysis*.

Image source: http://www.freepik.com

Association Analysis

- A type of **undirected data mining** that **finds patterns in large datasets** where the target is not specified beforehand, and **automatically generates rules**.

- Whether the patterns make sense is left to **human interpretation**.

- **Association analysis** can be applied outside the retail industry to find relationships among other types of “baskets.”
  - Items purchased on a credit card.
  - Product/service bundling.
  - Cross-selling
  - Fraud detection
  - Medical Patient histories

Image source: OpenClipart-Vectors from Pixabay

Association Rules

"If a customer purchases **bread** and **cheese**, then that customer will also purchase **butter**."

- Association rules are easy to understand. It expresses how tangible products group together.

Image source: https://vectortoons.com/

Rules are not always useful

“Customers who purchase **paint** buy **paint brushes**.”

- **Trivial** rules.
- **Inexplicable** rules.
- **Actionable** rules might suggest a **specific course of action**.

Image source: https://icon-library.com/

An Example of Market Basket Transactions

- Each row in this table corresponds to a **transaction**, which contains a **unique identifier** labelled *TID* and **set of items bought together**.

TID	Items
T1	{Bread, Milk, Eggs}
T2	{Milk, Beer}
T3	{Milk, Nappies}
T4	{Bread, Milk, Beer}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Milk, Nappies, Eggs}
T9	{Bread, Milk, Nappies}

- The **items** are the products which are sold in a supermarket.
- The **baskets** contain sets of items that are purchased together in a single transaction.

Binary Representation

- Because the presence of an item in a transaction is often considered more important than its absence, an item is an *asymmetric binary variable*.
- Each **row** corresponds to a **transaction** and each **column** corresponds to an **item**.

TID	Items
T1	{Bread, Milk, Eggs}
T2	{Milk, Beer}
T3	{Milk, Nappies}
T4	{Bread, Milk, Beer}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Milk, Nappies, Eggs}
T9	{Bread, Milk, Nappies}

TID	Bread	Milk	Nappies	Beer	Eggs
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

Itemset

- In association analysis, a collection of zero or more items is termed an **itemset**.
  - Let $I = \{i_{1,}, i_2, \ldots, i_d\}$ be the set of **all** items in a market basket data.
  - Let $T = \{t_1, t_{30}, \ldots, t_N\}$ be the set of **all** transactions.
- Each transaction $\boldsymbol{t_i}$ contains a subset of items chosen from $\boldsymbol{I}$.
- If an itemset contains k items, it is called a ***k*-itemset**.
  - {*Beer*, *Nappies*, *Milk*} is an example of *3-itemset*.
- The null (or empty) set is an itemset that does not contain any items.

Support Count

- An important property of an **itemset** is its **support count**. This is the **number of transactions** that **contain a particular itemset**.
- The **support count** of an itemset X is the number of transactions in the database which contains X as a subset.
- The **support count** $\sigma(X)$ for an itemset X can be defined as follows:

$$\sigma(X) = |\{t \in T \mid X \subseteq t\}|,$$

the number of transactions that contain the elements in itemset X.

Support Count

- What is the support count for {Milk}?

  ({Milk}) = 7

- What is the support count for {Milk, Bread}?

  ({Milk, Bread}) = 4

- What is the support count for {Milk, Bread, Nappies}?

  ({Milk, Bread, Nappies}) = 2

TID	Items
T1	{Bread, Milk, Eggs}
T2	{Milk, Beer}
T3	{Milk, Nappies}
T4	{Bread, Milk, Beer}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Milk, Nappies, Eggs}
T9	{Bread, Milk, Nappies}

Support

- The **support** of an itemset X is the **relative frequency** of the itemset X in the data set (set of all transactions).
- It represents the **probability of the itemset** occurring in a **transaction**;
    - A higher support signifies greater popularity of an itemset.
- An itemset X is called **frequent** if s(X) is greater than some user-defined threshold, **minsup**.
- The **support** is fraction of transactions in which an itemset occurs:

$$\mathbf{s(X)} = \frac{\sigma(X)}{N} = \frac{\text{number of transactions containing X}}{\text{total number of transactions}}$$

Support

- What is the support count for {Milk}?
  s({Milk}) = 7/9=0.78 = 78%

- What is the support count for {Milk, Bread}?
  s({Milk, Bread}) =4/9= 0.44 = 44%

- What is the support count for {Milk, Bread, Nappies}?
  s({Milk, Bread, Nappies}) = 2/9=0.22 = 22%

TID	Items
T1	{Bread, Milk, Eggs}
T2	{Milk, Beer}
T3	{Milk, Nappies}
T4	{Bread, Milk, Beer}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Milk, Nappies, Eggs}
T9	{Bread, Milk, Nappies}

Association Rules

- 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**,
  - **Confidence** determines **how frequently items in Y appear in transactions that contain X**.

Support, s(X → Y) $= \dfrac{\sigma(X \cup Y)}{N}$ **Confidence, c(X → Y)** $= \dfrac{\sigma(X \cup Y)}{\sigma(X)}$

Association Rules

- Consider the rule {Bread, Milk} → {Eggs}

$\sigma(\{Bread, Milk, Eggs\})$ = **2**

  - Total number of transactions = **9**

$s(\{Bread, Milk\} \rightarrow \{Eggs\}) = \frac{2}{9} =$ **0.22**

  - Number of transaction that contains {Bread, Milk} = **4**

$c(\{Bread, Milk\} \rightarrow \{Eggs\}) = \frac{2}{4} =$ **0.5**

TID	Items
T1	{Bread, Milk, Eggs}
T2	{Milk, Beer}
T3	{Milk, Nappies}
T4	{Bread, Milk, Beer}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Milk, Nappies, Eggs}
T9	{Bread, Milk, Nappies}

Why Use Support and Confidence

- **Support** is an important measure to **filter out rules** that may simply **occur by chance**, or are **unlikely to be interesting** from a business perspective.

- **Confidence** 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.

The Association Rule Mining Problem

- Given a set of transactions T, **find all the rules** having:

  support ≥ **minsup** and **confidence** ≥ **minconf**,

- A **brute-force approach**:

  1) Generate all the possible rules from the data set.

  2) Compute the **support** and **confidence** for every possible rule, and prune all those that do not fulfil the thresholds.

A brute-force approach

1) Generate all the possible rules from the data set.

TID	Items
T1	{Bread, Milk}
T2	{Bread, Nappies, Beer, Eggs}
T3	{Milk, Nappies, Beer, Coke }
T4	{Bread, Milk, Nappies, Beer}
T5	{Bread, Milk, Nappies, Coke}

$$R = 3^d - 2^{d+1} + 1$$

(*R = total number of possible rules for a data set that contains* ***d items***)

i.e. **d = 6** ⇒ $R = 3^6 - 2^7 + 1 = 602$ rules

2) Compute the **support** and **confidence** for every possible rule, and discard all those that do not satisfy the **minsup** and **minconf** threshold.

i.e. **minsup = 20%** and **minconf = 50%** ⇒ approx. **more 80% of the rules are discarded**.

Decoupling support and confidence

- Since $s(X \rightarrow Y) = s(\sigma(X \cup Y))$ then if the itemset is **infrequent** all candidate rules can be pruned **without having to compute their confidence values**.

- Given the itemset: {Bread, Milk, Eggs} there are 6 candidate rules, $(2^k - 2)$:
  - {Bread, Milk} → {Eggs}
  - {Milk, Eggs} → {Bread}
  - {Eggs} → {Bread, Milk}
  - {Bread, Eggs} → {Milk}
  - {Bread} → {Milk, Eggs}
  - {Milk} → {Bread, Eggs}

- A common strategy is to **decompose the problem** into two subtasks:

  1. **Frequent Itemset Generation**.
  2. **Rule Generation (only strong rules)**.

Frequent Itemset Generation

- **Computational requirements for frequent itemset generation are generally more expensive** than those of *rule generation*.
- The **brute-force approach** compares each candidate itemset against every transaction.
- If the candidate is contained in a transaction, its **support count** will be incremented.

i.e. *{Bread, Milk}* is contained in *T1*, *T4* and *T5*.

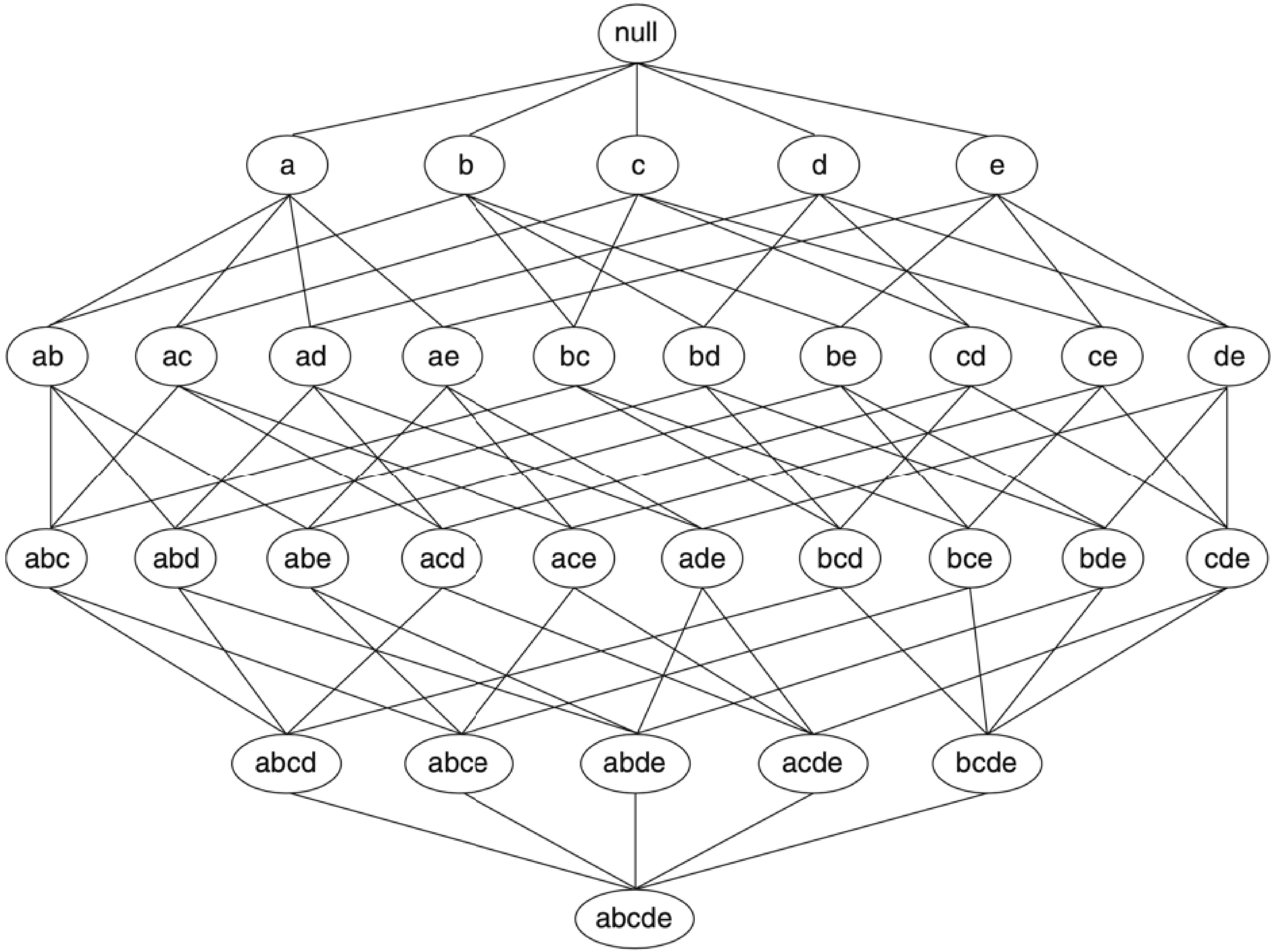
$\sigma(\{Bread, Milk\}) = 3$

TID	Items
T1	{Bread, Milk}
T2	{Bread, Nappies, Beer, Eggs}
T3	{Milk, Nappies, Beer, Coke }
T4	{Bread, Milk, Nappies, Beer}
T5	{Bread, Milk, Nappies, Coke}

Frequent Itemset Generation (Brute-force)

- The itemset lattice for I = {a, b, c, d, e} has $2^k - 1$ candidates:

  $k \rightarrow$ number of items



Frequent Itemset Generation (Brute-force)

- It is a **very expensive approach** because it requires **O(NMw)** comparisons.

  N number of transactions.

  M number of candidate itemsets, $M = 2^k - 1$

  w maximun *transaction width (number of items in a transaction)*

- Three approaches for reducing the computational complexity:

  a. Reduce the number of candidate itemsets (M).

  b. Reduce the number of comparisons.

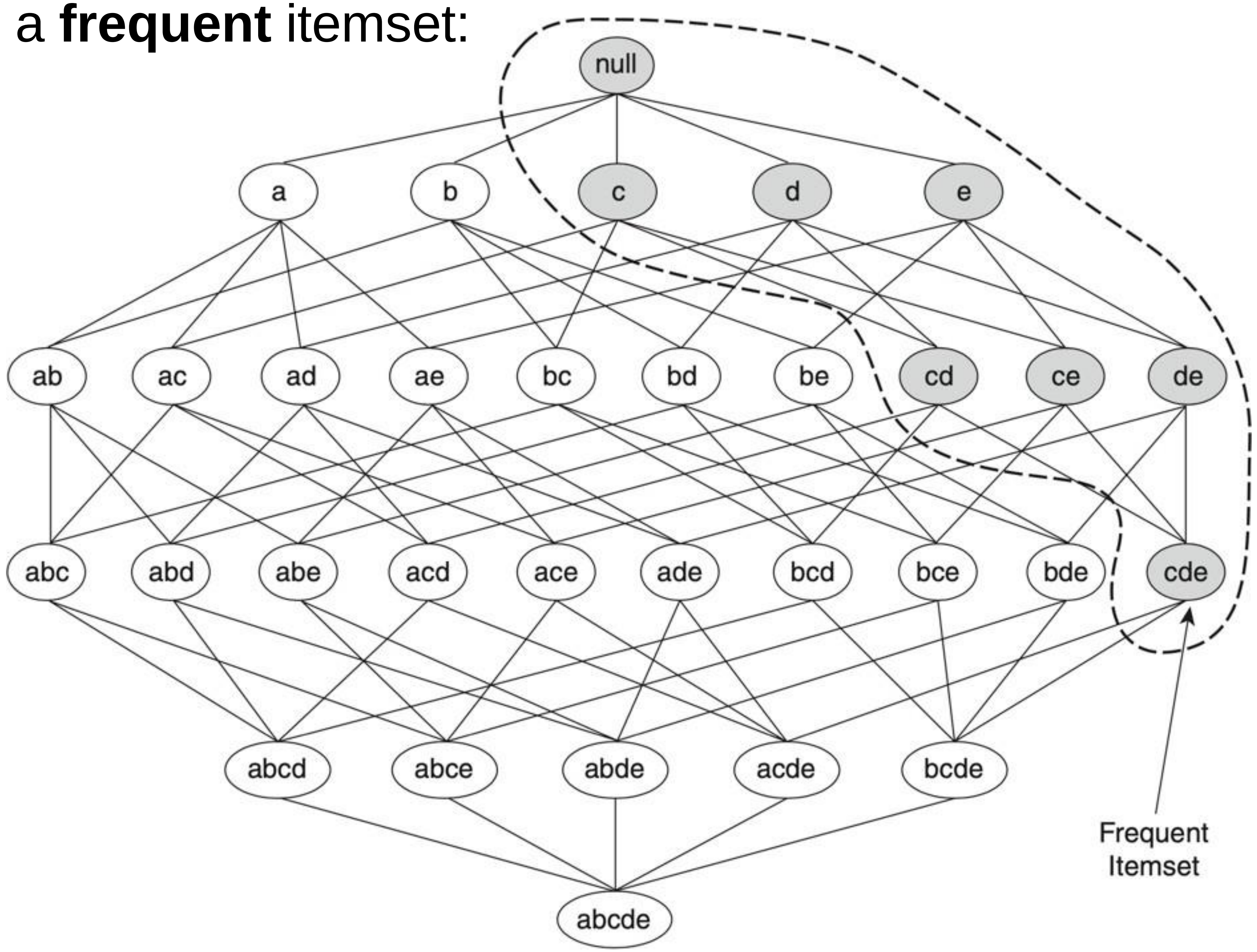
  c. Reduce the number of transactions (N).

The *Apriori* Principle

- Apriori is the first association rule mining algorithm that pioneered the use of **support-based pruning** to systematically control the exponential growth of candidate itemsets.
- It is for finding **frequent itemsets** in a dataset for **boolean association rule**.
- Name of the algorithm is *Apriori* because it uses **prior knowledge** of **frequent itemset properties**.
- **The Apriori Principle:** All subsets of a frequent itemset must be frequent (**Apriori property**).
  - If an itemset is **frequent**, then **all of its subsets** must also be **frequent**.
  - If an itemset is **infrequent**, **all its supersets** will be **infrequent**.

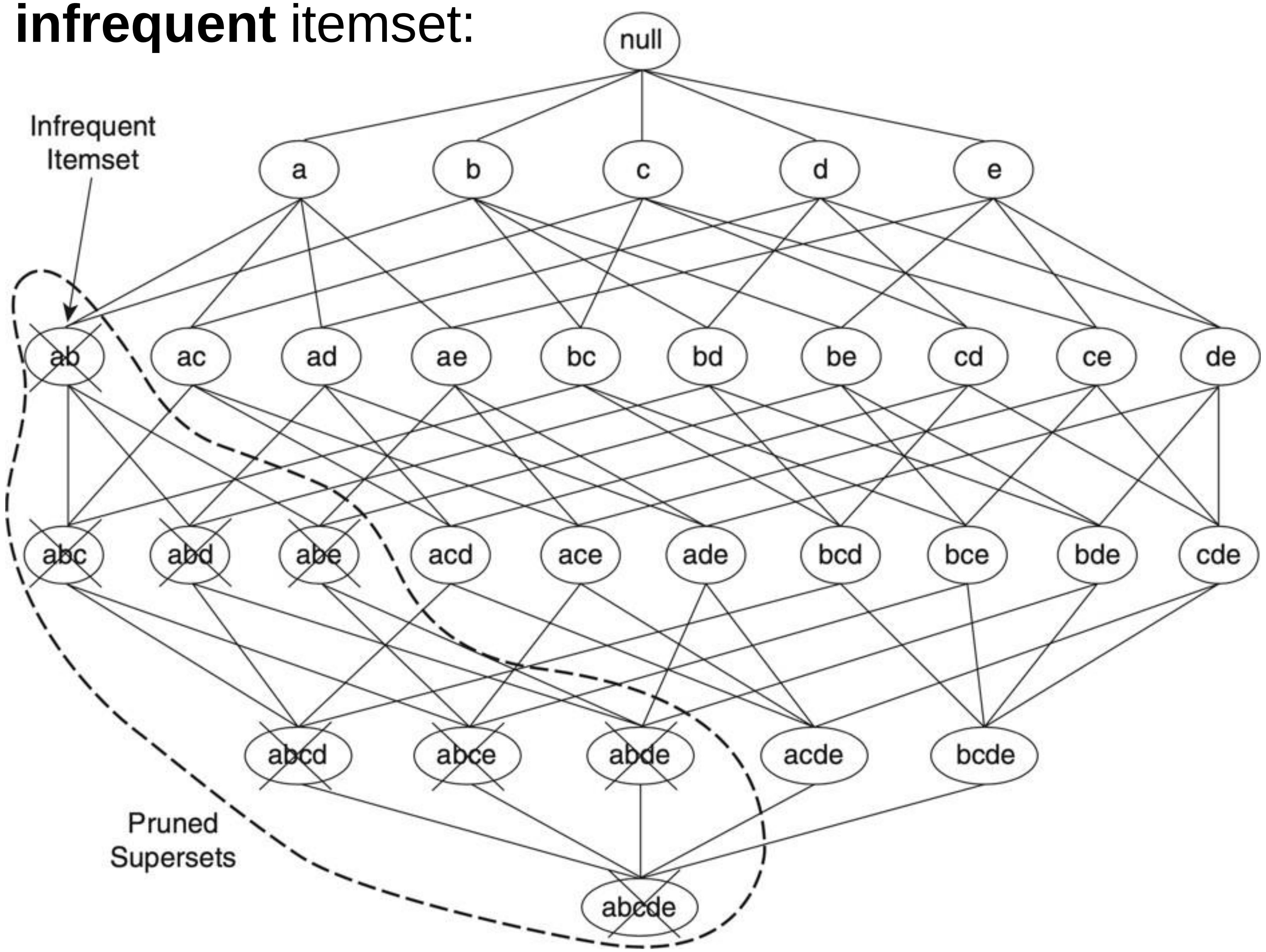
Frequent Itemset Generation in the *Apriori* Algorithm

- If an itemset is **frequent**, then **all of its subsets** must also be **frequent**.
  - i.e. {***c***, ***d***, ***e***} is a **frequent** itemset:



Frequent Itemset Generation in the *Apriori* Algorithm

- If an itemset is **infrequent**, **all its supersets** will be **infrequent**.
  - i.e. {***a, b***} is an **infrequent** itemset:



Frequent Itemset Generation in the *Apriori* Algorithm

- **Brute-force strategy**:

TID	Items
T1	{Bread, Milk}
T2	{Bread, Nappies, Beer, Eggs}
T3	{Milk, Nappies, Beer, Coke}
T4	{Bread, Milk, Nappies, Beer}
T5	{Bread, Milk, Nappies, Coke}

minsup = 3

Candidate 1-Itemset
{Bread}
{Milk}
{Nappies}
{Beer}
{Eggs}
{Coke}

Candidate 2-Itemset		
{Bread, Milk}	{Milk, Beer}	{Beer, Eggs}
{Bread, Nappies}	{Milk, Eggs}	{Beer, Coke}
{Bread, Beer}	{Milk, Coke}	{Eggs, Coke}
{Bread, Eggs}	{Nappies, Beer}	
{Bread, Coke}	{Nappies, Eggs}	
{Milk, Nappies}	{Nappies, Coke}	

Candidate 3-Itemset			
{Bread, Milk, Nappies}	{Bread, Nappies, Coke}	{Milk, Nappies, Coke}	{Nappies, Eggs, Coke}
{Bread, Milk, Beer}	{Bread, Beer, Eggs}	{Milk, Beer, Eggs}	{Beer, Eggs, Coke}
{Bread, Milk, Eggs}	{Bread, Beer, Coke}	{Milk, Beer, Coke}	
{Bread, Milk, Coke}	{Bread, Eggs, Coke}	{Milk, Eggs, Coke}	
{Bread, Nappies, Beer}	{Milk, Nappies, Beer}	{Nappies, Beer, Eggs}	
{Bread, Nappies, Eggs}	{Milk, Nappies, Eggs}	{Nappies, Beer, Coke}	

Frequent Itemset Generation in the *Apriori* Algorithm

- **Apriori pruning strategy**:

TID	Items
T1	{Bread, Milk}
T2	{Bread, Nappies, Beer, Eggs}
T3	{Milk, Nappies, Beer, Coke}
T4	{Bread, Milk, Nappies, Beer}
T5	{Bread, Milk, Nappies, Coke}

minsup = 3

Candidate 1-Itemset	σ
{Bread}	4
{Milk}	4
{Nappies}	4
{Beer}	3
{Eggs}	1
{Coke}	2

Candidate 2-Itemset					
{Bread, Milk}	3	{Milk, Beer}	2	~~{Beer, Eggs}~~	
{Bread, Nappies}	3	~~{Milk, Eggs}~~		~~{Beer, Coke}~~	
{Bread, Beer}	2	~~{Milk, Coke}~~		~~{Eggs, Coke}~~	
~~{Bread, Eggs}~~		{Nappies, Beer}	3		
~~{Bread, Coke}~~		~~{Nappies, Eggs}~~			
{Milk, Nappies}	3	~~{Nappies, Coke}~~			

Candidate 3-Itemset							
{Bread, Milk, Nappies}	2	~~{Bread, Nappies, Coke}~~		~~{Milk, Nappies, Coke}~~		~~{Nappies, Eggs, Coke}~~	
~~{Bread, Milk, Beer}~~		~~{Bread, Beer, Eggs}~~		~~{Milk, Beer, Eggs}~~		~~{Beer, Eggs, Coke}~~	
~~{Bread, Milk, Eggs}~~		~~{Bread, Beer, Coke}~~		~~{Milk, Beer, Coke}~~			
~~{Bread, Milk, Coke}~~		~~{Bread, Eggs, Coke}~~		~~{Milk, Eggs, Coke}~~			
~~{Bread, Nappies, Beer}~~		~~{Milk, Nappies, Beer}~~		~~{Nappies, Beer, Eggs}~~			
~~{Bread, Nappies, Eggs}~~		~~{Milk, Nappies, Eggs}~~		~~{Nappies, Beer, Coke}~~			

Frequent Itemset Generation in the *Apriori* Algorithm

- The **effectiveness** of the **Apriori pruning strategy** can be shown by counting the number of candidate itemsets generated.
  - A **brute-force strategy** of enumerating all itemsets (**up to size 3**) as candidates will produce:

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41 \text{ candidates}$$

With the **Apriori principle**, this number decreases to

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13 \text{ candidates}$$

Frequent Itemset Generation in the *Apriori* Algorithm

- The **frequent itemset generation** part of the **Apriori algorithm** has two important characteristics:
  - First, it is a **level-wise** algorithm: one level at a time, from *frequent 1-itemsets* to the *maximum size of frequent itemsets*.
  - Second, it employs a **generate-and-test strategy** for finding frequent itemsets.
    - At each iteration (level), **new candidate itemsets are generated** from the **frequent itemsets** found **in the previous iteration**.
    - The **support** for each candidate is then **counted and tested** against the **minsup threshold**.
    - The **total number of iterations** needed by the algorithm is $k_{max} + 1$.

      (k_{max} is the maximum size of the frequent itemsets)

.

Candidate Generation and Pruning

1. **Candidate Generation**. This operation **generates new candidate k-itemsets** based on the **frequent (k − 1)-itemsets** found in the previous iteration.

2. **Candidate Pruning**. This operation **eliminates** some of the **candidate k-itemsets** using **support-based pruning**, i.e. by removing k-itemsets whose subsets are known to be infrequent in previous iterations.

Candidate Generation

- There are many ways to generate candidate itemsets.
- An *effective candidate generation* procedure must be **complete** and **non-redundant**.
  - **Complete**. if it does not omit any frequent itemsets.
    - The set of *candidate itemsets* **must include** the set of *all frequent itemsets*.
  - **Non-redundant**. if it does not generate the same candidate itemset more than once.
    - i.e. the candidate itemset {a, b, c, d} can be generated in many ways—by merging {a,b,c} with {d}, {b,d} with {a,c}, {c} with {a,b,d}, etc
- It should **avoid generating too many unnecessary candidates**.

Candidate Generation

- **Brute-Force Method**. The brute-force method considers every k-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates whose subsets are infrequent.

- The number of candidate itemsets generated at level k:

  $\binom{d}{k}$ where d is the total number of items.

- Although **candidate generation** is rather **trivial**, **candidate pruning** becomes **extremely expensive** because a large number of itemsets must be examined.

Candidate Generation

- **F_{k-1} x F_1 Method**. An alternative method for candidate generation is to extend each frequent (k−1)-itemset with frequent items that are not part of the (k−1)-itemset.

Frequent 2-Itemset
{Beer, Nappies}
{Bread, Milk}
{Bread, Nappies}
{Milk, Nappies}

Frequent 1-Itemset
{Beer}
{Bread}
{Milk}
{Nappies}

Candidate Generation

Itemset
{Beer, Nappies, Bread}
{Beer, Nappies, Milk}
{Bread, Milk, Beer}
{Bread, Milk, Nappies}
{Bread, Nappies, Beer}
{Bread, Nappies, Milk}
{Milk, Nappies, Beer}
{Milk, Nappies, Bread}

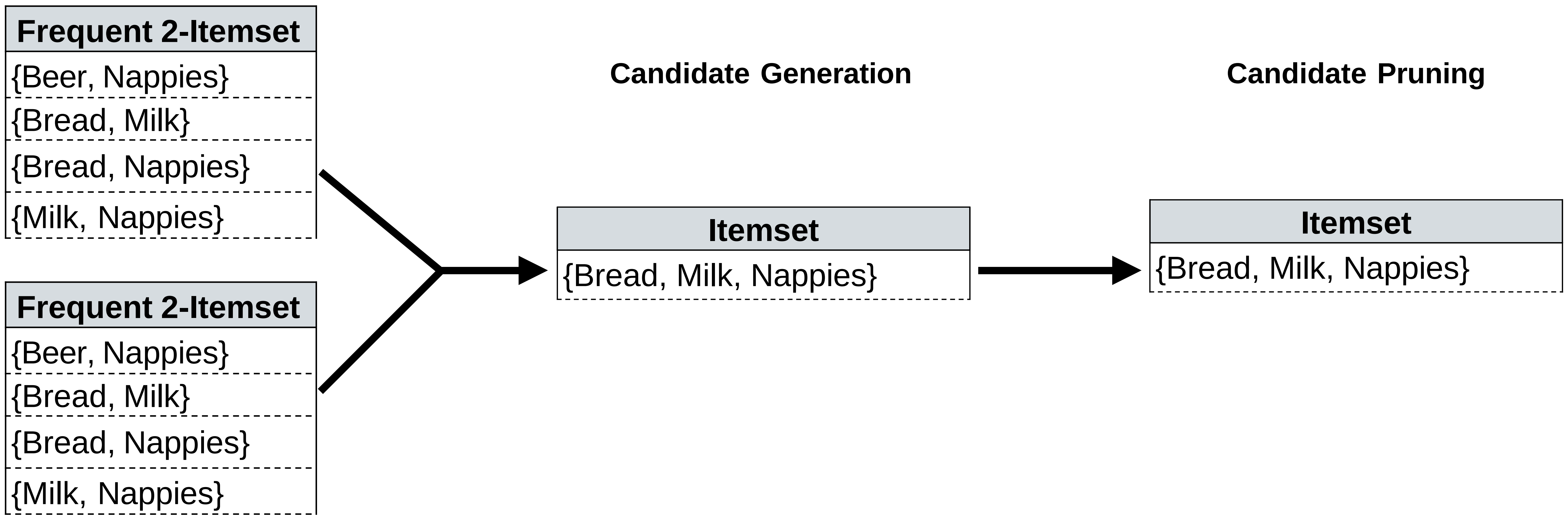
Candidate Pruning

Itemset
{Bread, Milk, Nappies}

- The procedure is **complete** because **every frequent k-itemset** is composed of a **frequent (k − 1)-itemset** and a **frequent 1-itemset**.
- However, it does **not prevent** the same candidate itemset from being **generated more than once**.
    - i.e. {*Bread, Milk, Nappies*} can be generated by merging {*Bread, Milk*} with {*Nappies*}, {*Bread, Nappies*} with {*Milk*}, or {*Milk, Nappies*} with {*Bread*}.

Candidate Generation

- **F_{k-1} x F_{k-1} Method**. This candidate generation procedure, which is used in the candidate-generation function of the **Apriori algorithm**, merges a **pair of frequent (k−1)-itemsets** only if their **first k−2 items**, arranged in lexicographic order, are **identical**:



- This candidate generation procedure is **complete**, because for every lexicographically ordered frequent k-itemset, there exists **two lexicographically ordered frequent (k−1)-itemsets** that have **identical items in the first k−2 positions**.

Candidate Pruning

- For the **brute-force candidate generation method**, candidate pruning requires checking only **k subsets of size k−1** for each candidate k-itemset.

- The $\mathbf{F_{k-1} \times F_1}$ candidate generation strategy ensures that at least **one of the (k−1)-size subsets** of every candidate k-itemset is **frequent**. Therefore, it is only needed to check for the **remaining k−1 subsets**.

- The $\mathbf{F_{k-1} \times F_{k-1}}$ strategy requires examining only **k−2 subsets** of every candidate k-itemset, since **two of its (k−1)-size subsets** are already known to be **frequent** in the candidate generation step.

Frequent itemset generation of the Apriori algorithm

Create L_1 = set of supported itemsets of cardinality one
Set k to 2
while $(L_{k-1} \neq \emptyset)$ {
    Create C_k from L_{k-1}
    Prune all the itemsets in C_k that are not
        supported, to create L_k
    Increase k by 1
}
The set of all supported itemsets with at least two members is $L_2 \cup \cdots \cup L_{k-2}$

Frequent itemset generation of the Apriori algorithm

- **minimum support count** is **2** and **minimum confidence** is **60%**.

- **Step-1: K=1**. Create a table containing support count of each item present in dataset, and remove those items in which support count < 2.

Itemset	σ
{Beer}	2
{Bread}	6
{Eggs}	2
{Milk}	7
{Nappies}	6

TID	Items
T1	{Bread, Eggs, Milk}
T2	{Beer, Milk}
T3	{Milk, Nappies}
T4	{Beer, Bread, Milk}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Eggs, Milk, Nappies}
T9	{Bread, Milk, Nappies}

Frequent itemset generation of the Apriori algorithm

- **Step-2: K=2**. Generate candidate set **C2** using **L1** (this is called join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common.

Itemset	σ
{Beer}	2
{Bread}	6
{Eggs}	2
{Milk}	7
{Nappies}	6

Itemset	σ
{Beer}	2
{Bread}	6
{Eggs}	2
{Milk}	7
{Nappies}	6

Itemset	σ
{Beer, Bread}	1
{Beer, Eggs}	0
{Beer, Milk}	1
{Beer, Nappies}	0
{Bread, Eggs}	2
{Bread, Milk}	3
{Bread, Nappies}	4
{Eggs, Milk}	2
{Eggs, Nappies}	1
{Milk, Nappies}	4

Itemset	σ
{Bread, Eggs}	2
{Bread, Milk}	3
{Bread, Nappies}	4
{Eggs, Milk}	2
{Milk, Nappies}	4

TID	Items
T1	{Bread, Eggs, Milk}
T2	{Beer, Milk}
T3	{Milk, Nappies}
T4	{Beer, Bread, Milk}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Eggs, Milk, Nappies}
T9	{Bread, Milk, Nappies}

Frequent itemset generation of the Apriori algorithm

- **Step-3: K=3**. Generate candidate set C3 using L2 (join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common.

Itemset	σ
{Bread, Eggs}	2
{Bread, Milk}	3
{Bread, Nappies}	4
{Eggs, Milk}	2
{Milk, Nappies}	4

Itemset	σ
{Bread, Eggs}	2
{Bread, Milk}	3
{Bread, Nappies}	4
{Eggs, Milk}	2
{Milk, Nappies}	4

Itemset	σ
{Bread, Eggs, Milk}	2
{Bread, Eggs, Nappies}	1
{Bread, Milk, Nappies}	2

Itemset	σ
{Bread, Eggs, Milk}	2
{Bread, Milk, Nappies}	2

TID	Items
T1	{Bread, Eggs, Milk}
T2	{Beer, Milk}
T3	{Milk, Nappies}
T4	{Beer, Bread, Milk}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Eggs, Milk, Nappies}
T9	{Bread, Milk, Nappies}

Frequent itemset generation of the Apriori algorithm

- **Step-4: K=4**. Generate candidate set C4 using L3 (join step). Condition of joining Lk-1 and Lk-1 (K=4) is that, they should have (K-2) elements in common.

Itemset	σ
{Bread, Eggs, Milk}	2
{Bread, Milk, Nappies}	2

No frequent itemsets are found further

Itemset	σ
{Bread, Eggs, Milk}	2
{Bread, Milk, Nappies}	2

TID	Items
T1	{Bread, Eggs, Milk}
T2	{Beer, Milk}
T3	{Milk, Nappies}
T4	{Beer, Bread, Milk}
T5	{Bread, Nappies}
T6	{Milk, Nappies}
T7	{Bread, Nappies}
T8	{Bread, Eggs, Milk, Nappies}
T9	{Bread, Milk, Nappies}

Support Count

- **Support count** determines the **frequency of occurrence** for **every candidate itemset** that **survives the candidate pruning step**.
- A **brute-force** approach for doing this is to **compare each transaction** against **every candidate itemset** and to **update the support counts** of candidates **contained in a transaction**.
- This approach is **computationally expensive**, especially when the **numbers of transactions** and **candidate itemsets** are **large**.

Support Count

- An alternative approach is to consider each transaction in turn. For each transaction, list the itemsets contained in **this transaction** and use this list to **update the support counts of the candidate itemsets.**

    - i.e., consider a transaction **t** that contains five items, {1, 2, 3, 5, 6}:

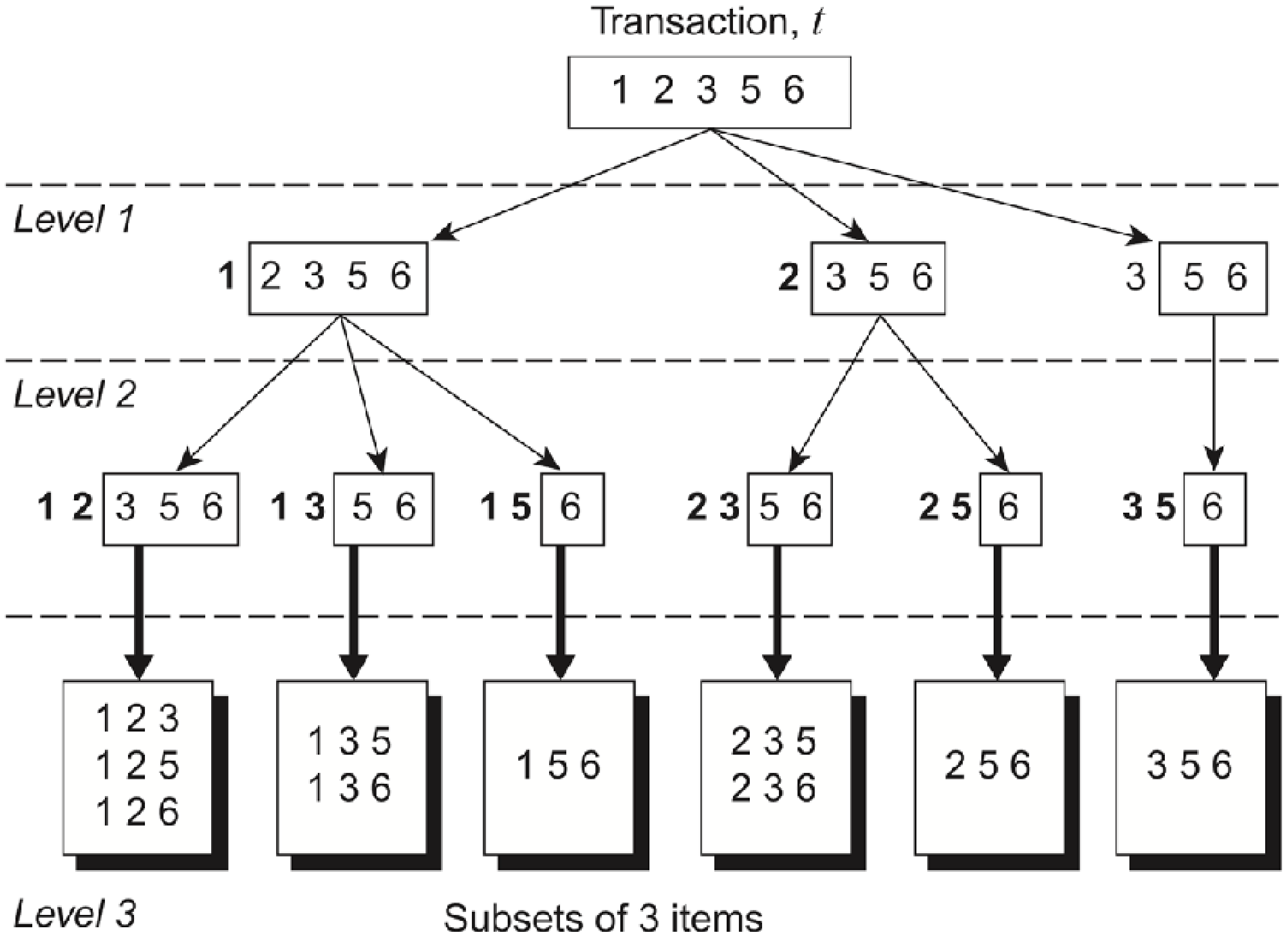
      $\binom{5}{3}$ = 10 itemsets of size 3 contained in this transaction

- Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their **support counts** are **incremented**.

- Other subsets of **t** that do **not correspond to any candidates** can be **ignored**.

Support Count

- The itemsets contained in a transaction can be systematically enumerated by specifying their items one by one, from the leftmost item to the rightmost item:



Computational Complexity

- The computational complexity, both runtime and storage, of the Apriori Algorithm can be affected by some factors such as:
  - Lowering the support threshold.
  - Number of Items (Dimensionality).
  - Number of Transactions.
  - Average Transaction Width.
  - Generation of Frequent 1-itemsets.
  - Candidate Generation.
  - Support Counting.

Formulation of the Association Rule Mining Problems

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 itemsets that satisfy the *minsup* threshold.

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.

Confidence levels

The **confidence** of a rule measures the reliability of the inference made by the rule.

For a given rule $X \longrightarrow Y$, the higher the confidence the more likely it is for Y to be present in transaction that contains X.

The confidence of a rule is defined as follows:

$$confidence(X \rightarrow Y) = \frac{support(X\ and\ Y)}{support(X)}$$

Confidence levels

$$confidence(X \rightarrow Y) = \frac{support(X\ and\ Y)}{support(X)}$$

Note:

$confidence(X \rightarrow Y)$ signifies the likelihood of item Y being purchased when item X is purchased.

$confidence(X \rightarrow Y)$ is the conditional probability $P(Y \mid X)$ the probability of Y given X.

Support(X and Y)

Let *X* and *Y* be itemsets and let *t* be a general transaction.

Recall that $support(X) = \frac{|X|}{|t|} = \frac{number\ of\ transactions\ containing\ X}{number\ of\ transactions}$.

Hence $support(X\ and\ Y) = \frac{|X \subseteq t\ and\ Y \subseteq t|}{|t|} = \frac{|(X \cup Y) \subseteq t|}{|t|} = \frac{\sigma(X \cup Y)}{|t|}$.

In terms of probability,

$$support(X) = P(X \subseteq t)$$

so,

$$support(X\ and\ Y) = P((X \cup Y) \subseteq t).$$

Example

TID	Items
1	Bread, Milk
2	Bread, Nappies, Beer, Eggs
3	Milk, Nappies, Beer, Coke
4	Bread, Milk, Nappies, Beer
5	Bread, Milk, Nappies, Coke

Rule: $X = \{Milk, Nappies\} \rightarrow \{Beer\} = Y.$

Then

$$support(X \; and \; Y) = \frac{\sigma(\{Milk, Nappies\} \cup \{Beer\})}{|t|} = \frac{\sigma(\{Milk, Nappies, Beer\})}{|t|} = \frac{2}{5} = 0.4$$

and $$confidence(X \rightarrow Y) = \frac{s(Milk, Nappies, Beer)}{s(Milk, Nappies)} = \frac{2/5}{3/5} = 0.67.$$

Note: $confidence(X \rightarrow Y) = \frac{P(\{Milk, Nappies, Beer\})}{P(\{Milk, Nappies\})} = P(\{Beer\} \mid \{Milk, Nappies\})$

Generating rules from itemsets

Examples

We can consider the rule {1,2} $\rightarrow$ {3} as being generated from the itemset X = {1 2,3} by partitioning X into two sets {1,2} and {3}.

The confidence for this rule is $\frac{s(\{1,2,3\})}{s(\{1,2\})}$.

Similarly the itemset {1,2,3,4,5} generates a rule {1,3,4} $\rightarrow$ {2,5}.

The confidence for this rule is $\frac{s(\{1,2,3,4,5\})}{s(\{1,3,4\})}$.

Rule generation

Each frequent *k*-itemset can produce up to $2^k - 2$ association rules, ignoring rules that have empty antecendants and consequents

($\emptyset \rightarrow Y$ or Y$\rightarrow \emptyset$).

An association rule can be extracted by partitioning the itemset Y into two non-empty itemsets

$$X \text{ and } Y - X$$

such that the rule

$$X \rightarrow Y - X$$

satisfies the confidence threshold.

Rule generation

Let $X = \{a, b, c\}$ be frequent itemset.
There are six candidate association rules (i.e. $2^3 - 2 = 6$) which can be generated:

$$\{a, b\} \to \{c\},$$
$$\{a, c\} \to \{b\},$$
$$\{b, c\} \to \{a\},$$
$$\{a\} \to \{b, c\},$$
$$\{b\} \to \{a, c\},$$
$$\{c\} \to \{a, b\}.$$

Example

TID	Items
1	Bread, Milk
2	Bread, Nappies, Beer, Eggs
3	Milk, Nappies, Beer, Coke
4	Bread, Milk, Nappies, Beer
5	Bread, Milk, Nappies, Coke

Consider the itemset {milk, nappies, beer}.

The six rules are:

{milk, nappies} $\rightarrow$ {beer}   {milk} $\rightarrow$ {nappies, beer}

{milk, beer} $\rightarrow$ {nappies}   {nappies} $\rightarrow$ {milk, beer}

{nappies, beer} $\rightarrow$ {milk}   {beer} $\rightarrow$ {milk, nappies}.

We calculate the confidence of each rule.

Example

TID	*Items*
1	**Bread, Milk**
2	**Bread, Nappies, Beer, Eggs**
3	**Milk, Nappies, Beer, Coke**
4	**Bread, Milk, Nappies, Beer**
5	**Bread, Milk, Nappies, Coke**

Rule	**Confidence**
{milk, nappies} $\rightarrow$ {beer}	$\frac{s(\{milk, nappies, beer\})}{s(\{milk, nappies\})} = \frac{2/5}{3/5} = 0.67$
{milk, beer} $\rightarrow$ {nappies}	$\frac{s(\{milk, nappies, beer\})}{s(\{milk, beer\})} = \frac{2/5}{2/5} = 1.0$
{nappies, beer} $\rightarrow$ {milk}	$\frac{s(\{milk, nappies, beer\})}{s(\{nappies, beer\})} = \frac{2/5}{3/5} = 0.67$

Example

TID	*Items*
1	Bread, Milk
2	Bread, Nappies, Beer, Eggs
3	Milk, Nappies, Beer, Coke
4	Bread, Milk, Nappies, Beer
5	Bread, Milk, Nappies, Coke

Rule | **Confidence**

Rule	Confidence
{milk} $\rightarrow$ {nappies, beer}	$\frac{s(\{milk, nappies, beer\})}{s(\{milk\})} = \frac{2/5}{4/5} = 0.5$
{nappies} $\rightarrow$ {milk, beer}	$\frac{s(\{milk, nappies, beer\})}{s(\{nappies\})} = \frac{2/5}{4/5} = 0.5$
{beer} $\rightarrow$ {nappies, milk}	$\frac{s(\{milk, nappies, beer\})}{s(\{beer\})} = \frac{2/5}{3/5} = 0.67$

Lift

The **lift** is another measure of a rule $X \rightarrow Y$.

It measures how often X and Y occur together rather than occurring independently.

The definition is

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)} = \frac{P(Y\,|X)}{P(Y)}$$

Lift

The **lift** of rule $X \rightarrow Y$ measures whether *X* and *Y* are:

‘positively associated’ in the sense that if *X* occurs then *Y* is **more** likely to occur (than if X had not occurred);

‘negatively associated’ in the sense that if *X* occurs then *Y* is **less** likely to occur (than if X had not occurred).

$$lift(X \rightarrow Y)\ is \begin{cases} > 1 & if\ Y\ is\ positively\ associated\ to\ X \\ = 1 & Y\ and\ X\ are\ independent \\ < 1 & if\ Y\ is\ negatively\ associated\ to\ X. \end{cases}$$

Example

TID	Items
1	Bread, Milk
2	Bread, Nappies, Beer, Eggs
3	Milk, Nappies, Beer, Coke
4	Bread, Milk, Nappies, Beer
5	Bread, Milk, Nappies, Coke

Rule **Lift**

{milk, nappies} → {beer} $\frac{conf(\{milk, nappies\} \to \{beer\})}{support(\{beer\})} = \frac{2/3}{3/5} = \frac{10}{9} = 1.1$

{milk, beer} → {nappies} $\frac{conf(\{milk, beer\} \to \{nappies\})}{support(\{nappies\})} = \frac{2/2}{4/5} = \frac{5}{4} = 1.25$

{nappies, beer} → {milk} $\frac{conf(\{nappies, beer\} \to \{milk\})}{support(\{milk\})} = \frac{2/3}{4/5} = \frac{5}{6} = 0.83$

Example

TID	Items
1	Bread, Milk
2	Bread, Nappies, Beer, Eggs
3	Milk, Nappies, Beer, Coke
4	Bread, Milk, Nappies, Beer
5	Bread, Milk, Nappies, Coke

Rule / Lift

{milk} → {nappies, beer}

$$\frac{conf(\{milk\}\rightarrow\{nappies,beer\})}{support(\{nappies,beer\})} = \frac{2/4}{3/5} = \frac{5}{6} = 0.83$$

{nappies} → {milk, beer}

$$\frac{conf(\{nappies\}\rightarrow\{milk,beer\})}{support(\{milk,beer\})} = \frac{2/4}{2/5} = \frac{5}{4} = 1.25$$

{beer} → {nappies, milk}

$$\frac{conf(\{beer\}\rightarrow\{nappies,milk\})}{support(\{nappies,milk\})} = \frac{2/3}{3/5} = \frac{10}{9} = 1.11$$

Example: summary

Rule	Confidence	Lift	
{milk, nappies} $\rightarrow$ {beer}	0.67	1.11	
{milk, beer} $\rightarrow$ {nappies}	1.0	1.25	highest
{nappies, beer} $\rightarrow$ {milk}	0.67	0.83	
{milk} $\rightarrow$ {nappies, beer}	0.5	0.83	
{nappies} $\rightarrow$ {milk, beer}	0.5	1.25	
{beer} $\rightarrow$ {nappies, milk}	0.67	1.11	

Rule Generation in Apriori Algorithm

Initially, all rules which have one item in the consequent are extracted.
These rules are then used to generate new candidate rules.

Example
If $\{a, c, d\} \to \{b\}$ and $\{a, b, d\} \to \{c\}$ are (high confidence) rules then the candidate rule $\{a, d\} \to \{b, c\}$ is generated by merging the consequents of both rules.
This is shown on the next slide.

Rule Generation in Apriori Algorithm

Merging rules

If two rules $X_1 \to Y_1$ and $X_2 \to Y_2$ are derived from the same itemset ($X_1 \cup Y_1 = X_2 \cup Y_2$) they merge to give the rule

$$X_1 \cap X_2 \to Y_1 \cup Y_2.$$

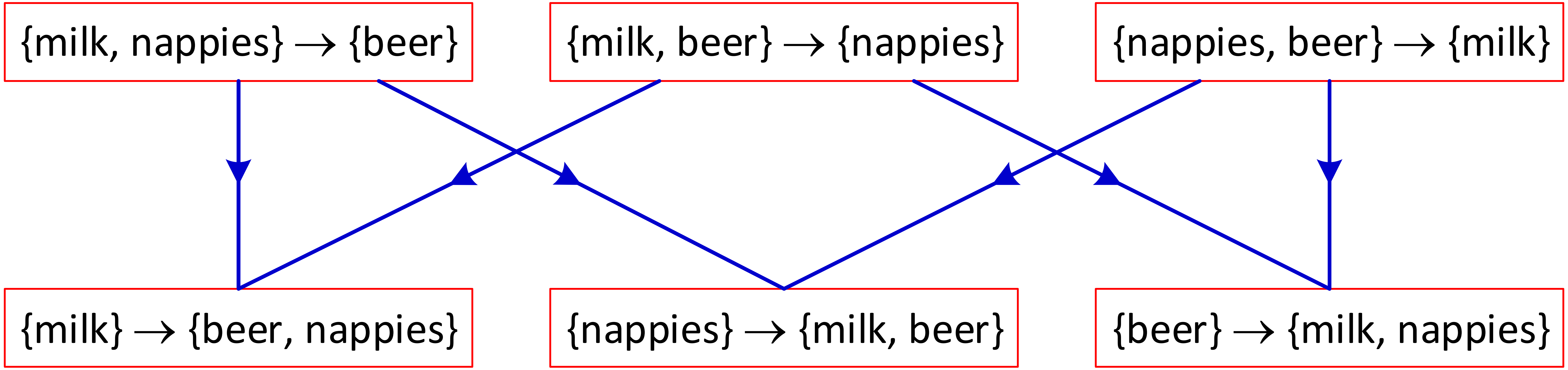
Examples

Itemset $\{a, b, c, d\}$ produces rules $\{a, b\} \to \{c, d\}$ and $\{a, c\} \to \{b, d\}$. These merge to give the rule:

$$\{a, b\} \cap \{a, c\} \to \{c, d\} \cup \{b, d\} \text{ which is } \{a\} \to \{b, c, d\}.$$

Example

From the itemset {milk, nappies, beer}, the six rules considered earlier are represented in the following diagram where the blue arrows represent merging.



Principles of Apriori Algorithm

The confidence of a rule is greater than or equal to (≥) the confidence of any rule formed by merging it with another rule.

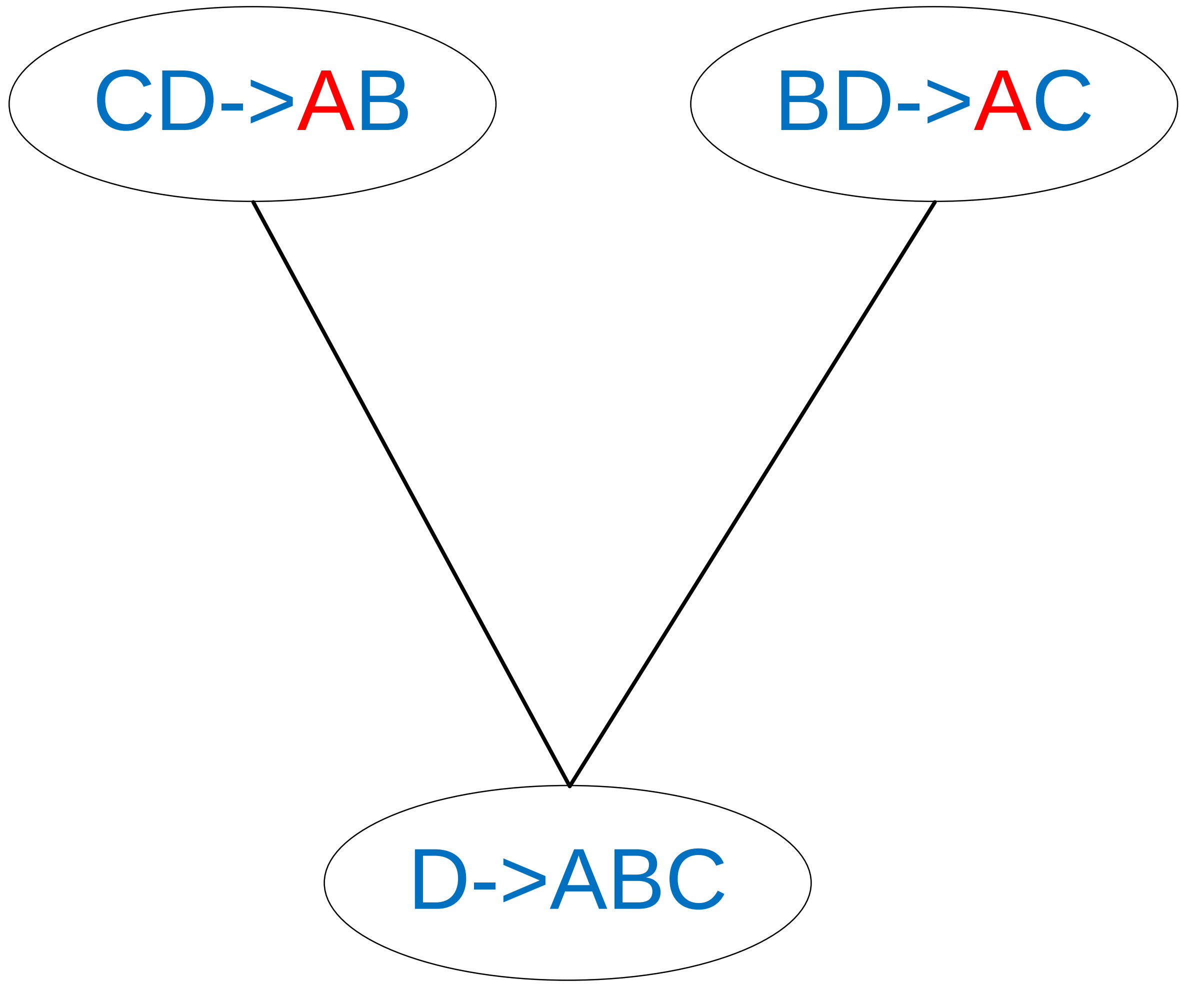
For example, confidence of $AB \rightarrow CD$ is greater than or equal to (≥) confidence of $A \rightarrow BCD$ because $AB \rightarrow CD$ can be merged with $AC \rightarrow BD$ to give $A \rightarrow BCD$

Rule Generation for Apriori Algorithm

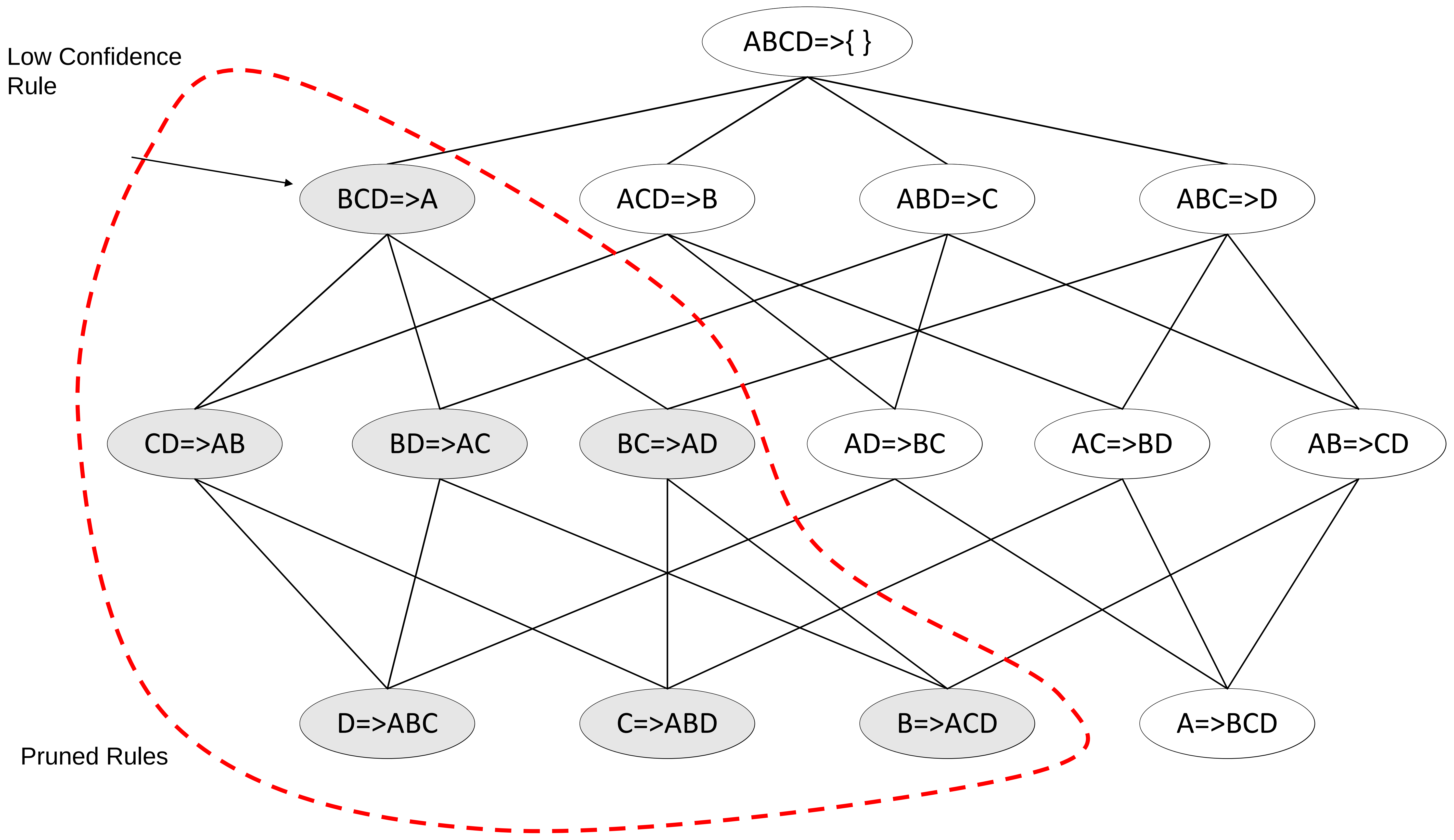
Example of merging rules from a four itemset

Merge ($CD \rightarrow AB, BD \rightarrow AC$)
would produce the candidate
rule $D \rightarrow ABC$

Prune rule $D \rightarrow ABC$ if at least one of the rules
$CD \rightarrow AB$ or $BD \rightarrow AC$ does not have
high confidence



Rule Generation for Apriori Algorithm



Python

The mlxtend library will be used for the Apriori algorithm

To install the mlxtend library use

!pip install mlxtend

To import the required libraries

```
from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent_patterns import apriori, association_rules
```

Reading Dataset

Read in a file in Pandas

```
import pandas as pd

df=pd.read_csv(‘filename.csv’)
```

OR you can create a small dataset

```
data=[['cola','eggs','bread','cheese','butter','crisps'],
      ['chocolate','coffee','milk','cheese','butter','bisuits'],
      ['potatoes','onions','carrots','chicken','gravy','cola','bread','cheese'],
      ['salad','beetroot','onions', 'lentils','coffee','avacado'],
      ['tomatoes','cheese','bread','mayonise','crisps','cola']
     ]
```

Convert to pandas DataFrame – easier to visualise

```
df=pd.DataFrame(data)
df
```

	0	1	2	3	4	5	6	7
0	cola	eggs	bread	cheese	butter	crisps	None	None
1	chocolate	coffee	milk	cheese	butter	bisuits	None	None
2	potatoes	onions	carrots	chicken	gravy	cola	bread	cheese
3	salad	beetroot	onions	lentils	coffee	avacado	None	None
4	tomatoes	cheese	bread	mayonise	crisps	cola	None	None

Converting data into transaction data

```
te=TransactionEncoder()
te_array=te.fit(data).transform(data)
```

```
array([[False, False, False,  True,  True, False,  True, False, False,
        False,  True,  True,  True, False, False, False, False, False,
        False, False, False],
       [False, False,  True, False,  True, False,  True, False,  True,
         True, False, False, False, False, False, False,  True, False,
        False, False, False],
       [False, False, False,  True, False,  True,  True,  True, False,
        False,  True, False, False,  True, False, False, False,  True,
         True, False, False],
       [ True,  True, False, False, False, False, False, False, False,
         True, False, False, False, False,  True, False, False,  True,
        False,  True, False],
       [False, False, False,  True, False, False,  True, False, False,
        False,  True,  True, False, False, False,  True, False, False,
        False, False,  True]])
```

Convert to one-hot encoded vector (array represented as 0’s and 1’s)

```
te_array.astype(int)
```

```
array([[0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]])
```

Column names of the DataFrame

```
te.columns_
```

```
['avacado',
 'beetroot',
 'bisuits',
 'bread',
 'butter',
 'carrots',
 'cheese',
 'chicken',
 'chocolate',
 'coffee',
 'cola',
 'crisps',
 'eggs',
 'gravy',
 'lentils',
 'mayonise',
 'milk',
 'onions',
 'potatoes',
 'salad',
 'tomatoes']
```

```
te_int_array=te_array.astype(int)
t=pd.DataFrame(te_int_array,columns=te.columns_)
t
```

	avacado	beetroot	bisuits	bread	butter	carrots	cheese	chicken	chocolate	coffee	...	crisps	eggs	gravy	lentils	mayonise	milk	onions	potatoes
0	0	0	0	1	1	0	1	0	0	0	...	1	1	0	0	0	0	0	0
1	0	0	1	0	1	0	1	0	1	1	...	0	0	0	0	0	1	0	0
2	0	0	0	1	0	1	1	1	0	0	...	0	0	1	0	0	0	1	1
3	1	1	0	0	0	0	0	0	0	1	...	0	0	0	1	0	0	1	0
4	0	0	0	1	0	0	1	0	0	0	...	1	0	0	0	1	0	0	0

5 rows × 21 columns

Determining the frequent itemset

Frequent itemset using a minimum support threshold

```
frequent_itemset=apriori(t,min_support=0.6, use_colnames=True)
frequent_itemset
```

	support	itemsets
0	0.6	(bread)
1	0.8	(cheese)
2	0.6	(cola)
3	0.6	(bread, cheese)
4	0.6	(bread, cola)
5	0.6	(cola, cheese)
6	0.6	(bread, cola, cheese)

Determining confidence of a rule

Setting the confidence threshold

```
frequent_rules=
    association_rules(frequent_itemset, metric='confidence’,min_thresho

frequent_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(bread)	(cheese)	0.6	0.8	0.6	1.00	1.250000	0.12	inf
1	(cheese)	(bread)	0.8	0.6	0.6	0.75	1.250000	0.12	1.6
2	(bread)	(cola)	0.6	0.6	0.6	1.00	1.666667	0.24	inf
3	(cola)	(bread)	0.6	0.6	0.6	1.00	1.666667	0.24	inf
4	(cola)	(cheese)	0.6	0.8	0.6	1.00	1.250000	0.12	inf
5	(cheese)	(cola)	0.8	0.6	0.6	0.75	1.250000	0.12	1.6
6	(bread, cola)	(cheese)	0.6	0.8	0.6	1.00	1.250000	0.12	inf
7	(bread, cheese)	(cola)	0.6	0.6	0.6	1.00	1.666667	0.24	inf
8	(cola, cheese)	(bread)	0.6	0.6	0.6	1.00	1.666667	0.24	inf
9	(bread)	(cola, cheese)	0.6	0.6	0.6	1.00	1.666667	0.24	inf
10	(cola)	(bread, cheese)	0.6	0.6	0.6	1.00	1.666667	0.24	inf
11	(cheese)	(bread, cola)	0.8	0.6	0.6	0.75	1.250000	0.12	1.6

Filtering rules

```
frequent_rules[(frequent_rules['lift']>=1) & (frequent_rules['confidence']>0.8)]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(bread)	(cheese)	0.6	0.8	0.6	1.0	1.250000	0.12	inf
2	(bread)	(cola)	0.6	0.6	0.6	1.0	1.666667	0.24	inf
3	(cola)	(bread)	0.6	0.6	0.6	1.0	1.666667	0.24	inf
4	(cola)	(cheese)	0.6	0.8	0.6	1.0	1.250000	0.12	inf
6	(bread, cola)	(cheese)	0.6	0.8	0.6	1.0	1.250000	0.12	inf
7	(bread, cheese)	(cola)	0.6	0.6	0.6	1.0	1.666667	0.24	inf
8	(cola, cheese)	(bread)	0.6	0.6	0.6	1.0	1.666667	0.24	inf
9	(bread)	(cola, cheese)	0.6	0.6	0.6	1.0	1.666667	0.24	inf
10	(cola)	(bread, cheese)	0.6	0.6	0.6	1.0	1.666667	0.24	inf