Mining Association Rules in Large Databases

Association rules

- Given a set of transactions **D**, find rules that will predict the occurrence of an item (or a set of items) based on the occurrences of other items in the transaction

Market-Basket transactions

TID	*Items*
1	Bread, Milk
2	Bread, Cookies, Butter, Eggs
3	Milk, Cookies, Butter, Coke
4	Bread, Milk, Cookies, Butter
5	Bread, Milk, Cookies, Coke

Examples of association rules

{Cookies} $\rightarrow$ {Butter},
{Milk, Bread} $\rightarrow$ {Cookies, Coke},
{Butter, Bread} $\rightarrow$ {Milk},

An even simpler concept: frequent itemsets

- Given a set of transactions **D**, find combination of items that occur frequently

Market-Basket transactions

TID	*Items*
1	**Bread, Milk**
2	**Bread, Cookies, Butter, Eggs**
3	**Milk, Cookies, Butter, Coke**
4	**Bread, Milk, Cookies, Butter**
5	**Bread, Milk, Cookies, Coke**

Examples of frequent itemsets

{Cookies, Butter},
{Milk, Bread}
{Butter, Bread, Milk},

Lecture outline

- **Task 1:** Methods for finding all frequent itemsets efficiently
- **Task 2:** Methods for finding association rules efficiently

Definition: Frequent Itemset

- **Itemset**
  - A set of one or more items
    - E.g.: {Milk, Bread, Cookies}
  - k-itemset
    - An itemset that contains k items
- **Support count (σ)**
  - Frequency of occurrence of an itemset (number of transactions it appears)
  - E.g. **σ({Milk, Bread, Cookies}) = 2**
- **Support**
  - Fraction of the transactions in which an itemset appears
  - E.g. **s({Milk, Bread, Cookies}) = 2/5**
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a ***minsup*** threshold

TID	*Items*
1	**Bread, Milk**
2	**Bread, Cookies, Butter, Eggs**
3	**Milk, Cookies, Butter, Coke**
4	**Bread, Milk, Cookies, Butter**
5	**Bread, Milk, Cookies, Coke**

Why do we want to find frequent itemsets?

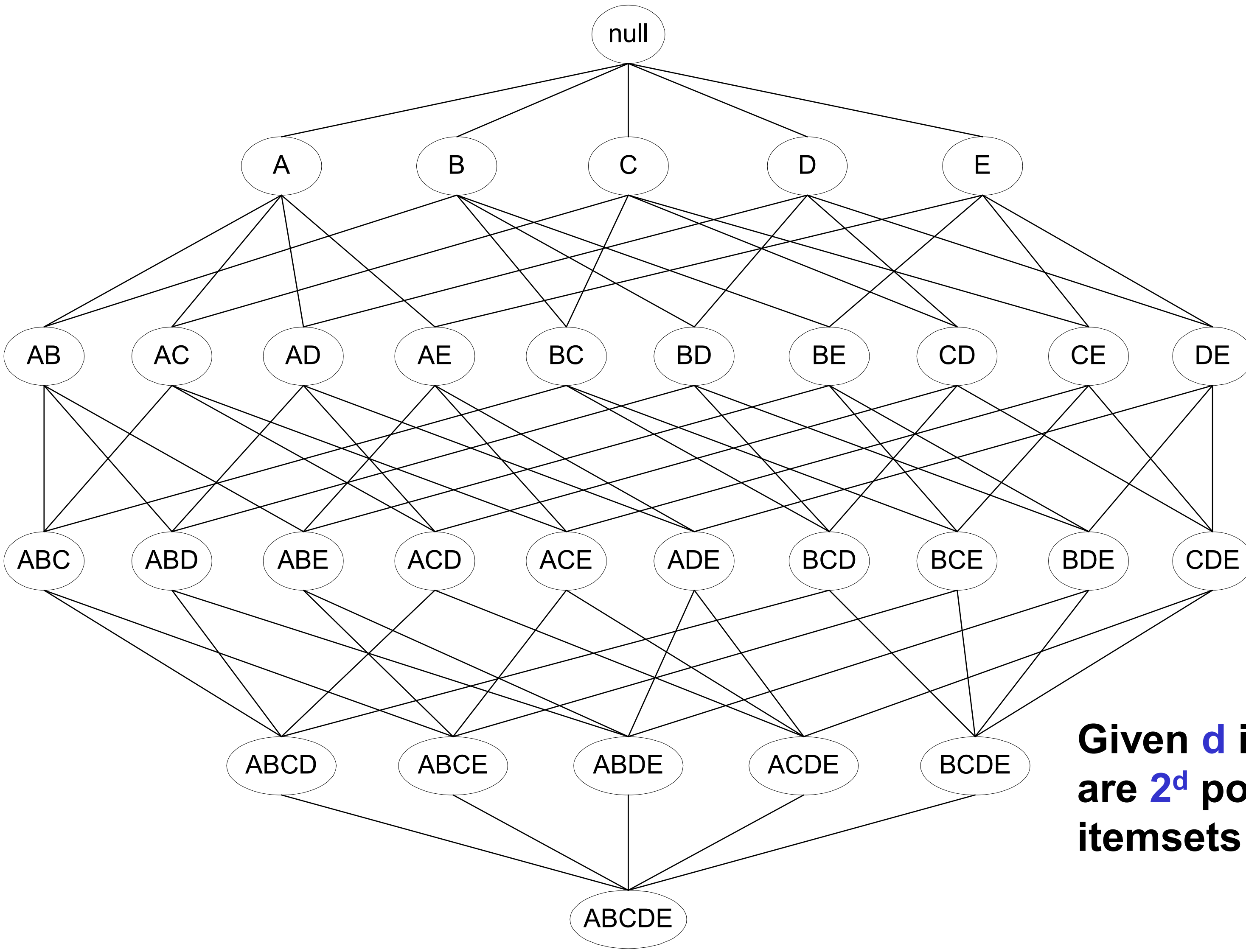
- Find all combinations of items that occur together
- They might be interesting (e.g., in placement of items in a store)
- Frequent itemsets are only positive combinations (we do not report combinations that do not occur frequently together)
- Frequent itemsets aims at providing a summary for the data

Finding frequent sets

- **Task:** Given a transaction database **D** and a **minsup** threshold find all frequent itemsets and the frequency of each set in this collection

- **Stated differently:** Count the number of times combinations of attributes occur in the data. If the count of a combination is above **minsup** report it.

How many itemsets are there?



Given d items, there are 2^d possible itemsets

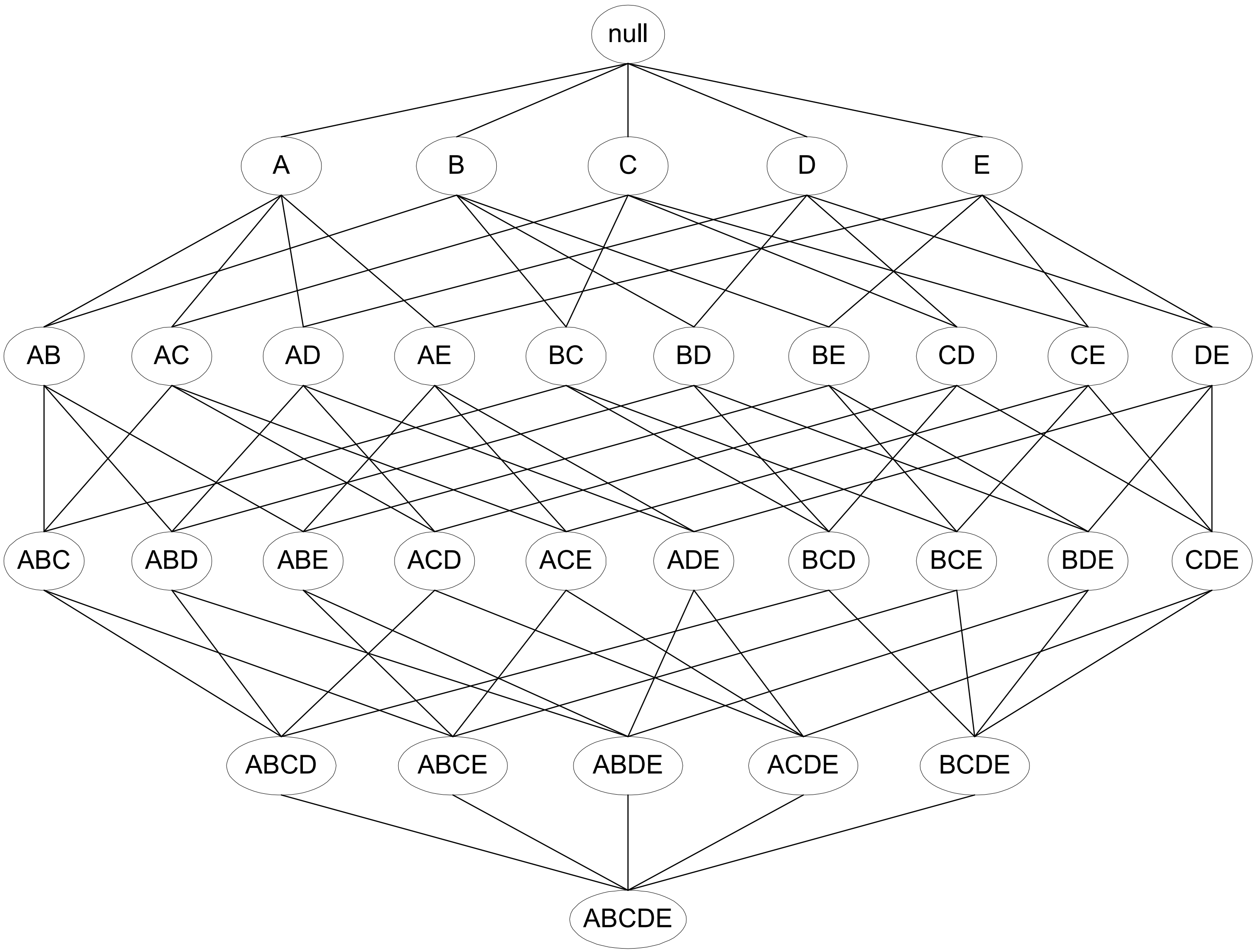
When is the task sensible and feasible?

- If **minsup = 0**, then all subsets of ***I*** will be frequent and thus the size of the collection will be very large

- This summary is very large (maybe larger than the original input) and thus not interesting

- The task of finding all frequent sets is interesting typically only for relatively large values of **minsup**

A simple algorithm for finding all frequent itemsets ??



Brute-force algorithm for finding all frequent itemsets?

- Generate all possible itemsets (lattice of itemsets)
  - Start with 1-itemsets, 2-itemsets,...,d-itemsets
- Compute the frequency of each itemset from the data
  - Count in how many transactions each itemset occurs
- If the support of an itemset is above **minsup** report it as a frequent itemset

Brute-force approach for finding all frequent itemsets

- Complexity?
  - Match every candidate against each transaction
  - For **M** candidates and **N** transactions, the complexity is~ **O(NMw)** => Expensive since $M = 2^d$!!!

Reduce the number of candidates

- Apriori principle (Main observation):
  - If an itemset is frequent, then all of its subsets must also be frequent

- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

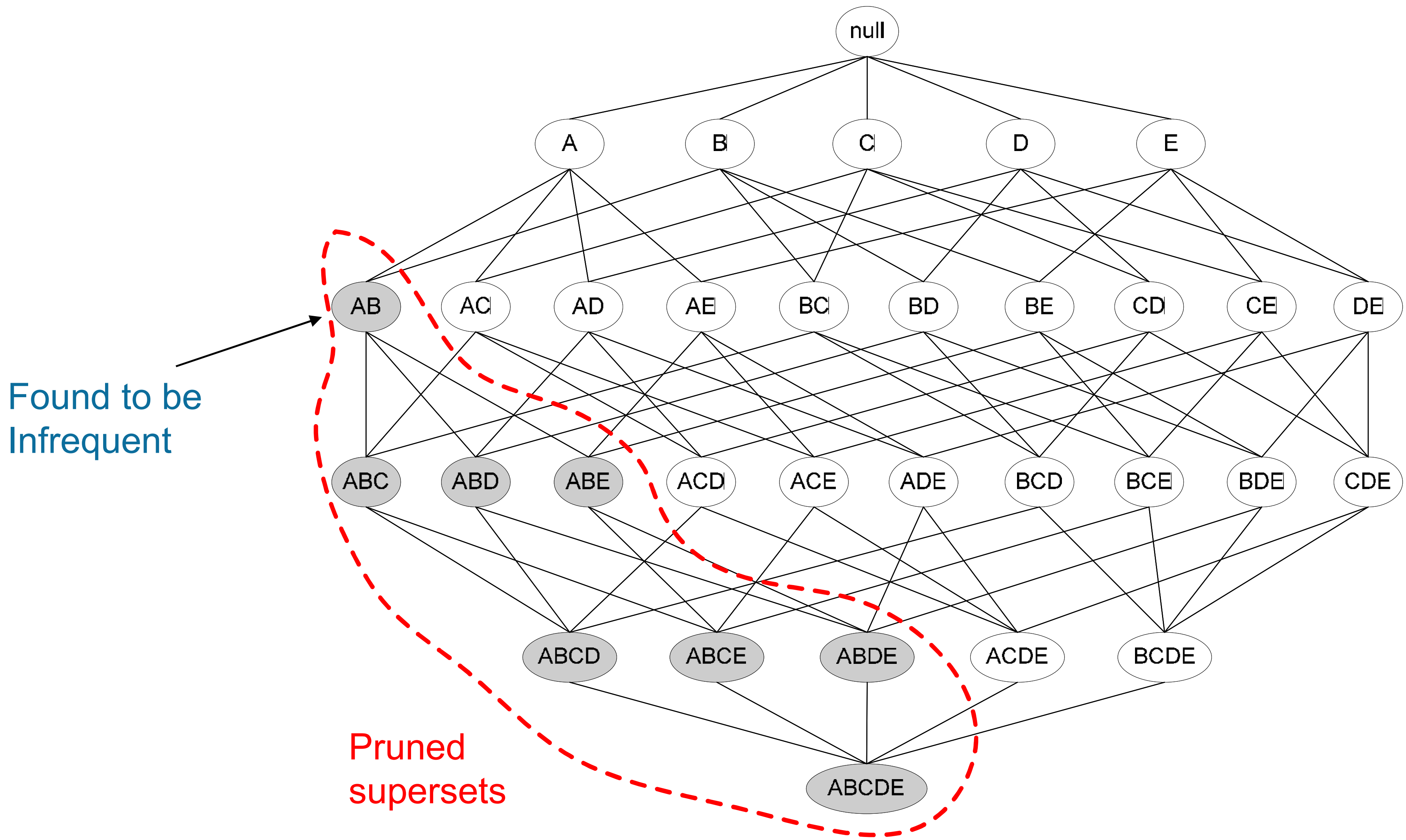
  - The support of an itemset ***never exceeds*** the support of its subsets
  - This is known as the ***anti-monotone*** property of support

Example

TID	*Items*
1	**Bread, Milk**
2	**Bread, Cookies, Butter, Eggs**
3	**Milk, Cookies, Butter, Coke**
4	**Bread, Milk, Cookies, Butter**
5	**Bread, Milk, Cookies, Coke**

s(Bread) > s(Bread, Butter)
s(Milk) > s(Bread, Milk)
s(Cookies, Butter) > s(Cookies, Butter, Coke)

Illustrating the Apriori principle



Illustrating the Apriori principle

Item	Count
Bread	4
Coke	2
Milk	4
Butter	3
Cookies	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Butter}	2
{Bread,Cookies}	3
{Milk,Butter}	2
{Milk,Cookies}	3
{Butter,Cookies}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

minsup = 3/5

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Cookies}	3

Exploiting the Apriori principle

1. Find frequent 1-items and put them to L_k (**k=1**)
2. Use L_k to generate a collection of *candidate* itemsets C_{k+1} with size (**k+1**)
3. Scan the database to find which itemsets in C_{k+1} are frequent and put them into L_{k+1}
4. If L_{k+1} is not empty
   - **k=k+1**
   - Goto step 2

R. Agrawal, R. Srikant: "Fast Algorithms for Mining Association Rules", *Proc. of the 20th Int'l Conference on Very Large Databases*, 1994.

The Apriori algorithm

C_k: Candidate itemsets of size k

L_k : frequent itemsets of size k

L_1 = {frequent 1-itemsets};

for (k = 2; L_k !=$\emptyset$; k++)

$\boxed{C_{k+1} = \text{GenerateCandidates}(L_k)}$

for each transaction ***t*** in database do

$\boxed{\text{increment count of candidates in } C_{k+1} \text{ that are contained in } \mathbf{t}}$

endfor

L_{k+1} = candidates in C_{k+1} with support ≥***min_sup***

endfor

return $\cup_k L_k$;

Discussion of the Apriori algorithm

- Much faster than the Brute-force algorithm
  - It avoids checking all elements in the lattice
- The running time is in the worst case **$O(2^d)$**
  - Pruning really prunes in practice
- It makes multiple passes over the dataset
  - One pass for every level **k**
- Multiple passes over the dataset is inefficient when we have thousands of candidates and millions of transactions

Lecture outline

- **Task 1:** Methods for finding all frequent itemsets efficiently
- **Task 2:** Methods for finding association rules efficiently

Definition: Association Rule

Let **D** be database of transactions

- e.g.:

Transaction ID	Items
2000	A, B, C
1000	A, C
4000	A, D
5000	B, E, F

- Let ***I*** be the set of items that appear in the database, e.g., ***I*={A,B,C,D,E,F}**
- A **rule** is defined by **X → Y**, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$
  - e.g.: **{B,C} → {A}** is a rule

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form **X** → **Y**, where **X** and **Y** are non-overlapping itemsets
  - Example:
    {Milk, Cookies} → {Butter}
- **Rule Evaluation Metrics**
  - **Support (s)**
    - Fraction of transactions that contain both **X** and **Y**
  - **Confidence (c)**
    - Measures how often items in **Y** appear in transactions that contain **X**

TID	*Items*
1	Bread, Milk
2	Bread, Cookies, Butter, Eggs
3	Milk, Cookies, Butter, Coke
4	Bread, Milk, Cookies, Butter
5	Bread, Milk, Cookies, Coke

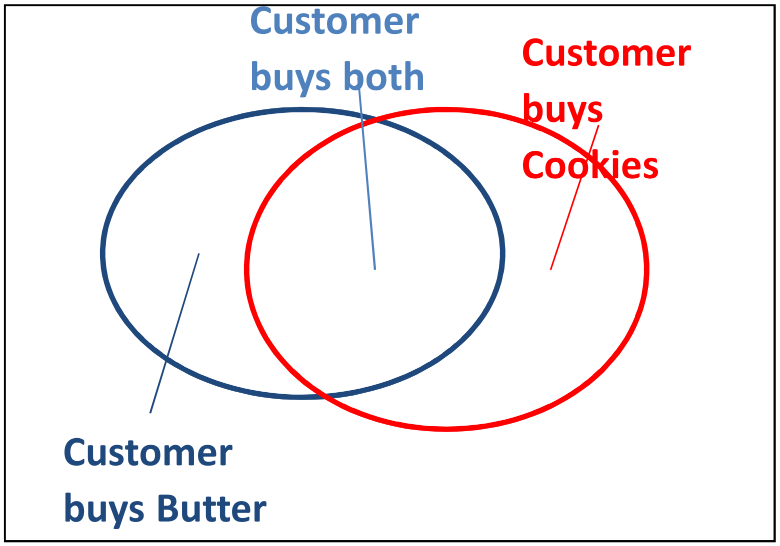
Example:

$$\{\text{Milk}, \text{Cookies}\} \rightarrow \text{Butter}$$

$$s = \frac{\sigma(\text{Milk}, \text{Cookies}, \text{Butter})}{|\text{T}|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk}, \text{Cookies}, \text{Butter})}{\sigma(\text{Milk}, \text{Cookies})} = \frac{2}{3} = 0.67$$

Rule Measures: Support and Confidence



Find all the rules ***X*** → ***Y*** with minimum confidence and support

- support, *s*, probability that a transaction contains **{X ∪ Y}**
- confidence, *c*, ***conditional probability*** that a transaction having **X** also contains **Y**

TID	Items
100	A,B,C
200	A,C
300	A,D
400	B,E,F

Let minimum support 50%, and minimum confidence 50%, we have

- *A* → *C* (50%, 66.6%)
- *C* → *A* (50%, 100%)

Example

TID	date	items bought
100	10/10/99	{F,A,D,B}
200	15/10/99	{D,A,C,E,B}
300	19/10/99	{C,A,B,E}
400	20/10/99	{B,A,D}

What is the ***support*** and ***confidence*** of the rule: {B,D} → {A}

- Support:
  - percentage of tuples that contain {A,B,D} = 75%
- Confidence:

$$\frac{\text{number of tuples that contain } \{A, B, D\}}{\text{number of tuples that contain } \{B, D\}} = 100\%$$

Association-rule mining task

- Given a set of transactions **D**, the goal of association rule mining is to find **all** rules having
  - support ≥ ***minsup*** threshold
  - confidence ≥ ***minconf*** threshold

Mining Association Rules

TID	*Items*
1	**Bread, Milk**
2	**Bread, Cookies, Butter, Eggs**
3	**Milk, Cookies, Butter, Coke**
4	**Bread, Milk, Cookies, Butter**
5	**Bread, Milk, Cookies, Coke**

Example of Rules:

{Milk,Cookies} $\rightarrow$ {Butter} (s=0.4, c=0.67)
{Milk,Butter} $\rightarrow$ {Cookies} (s=0.4, c=1.0)
{Cookies,Butter} $\rightarrow$ {Milk} (s=0.4, c=0.67)
{Butter} $\rightarrow$ {Milk,Cookies} (s=0.4, c=0.67)
{Cookies} $\rightarrow$ {Milk,Butter} (s=0.4, c=0.5)
{Milk} $\rightarrow$ {Cookies,Butter} (s=0.4, c=0.5)

Observations:

- All the above rules are binary partitions of the same itemset: {Milk, Cookies, Butter}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
  - Frequent Itemset Generation
    - Generate all itemsets whose support $\geq$ minsup
  - Rule Generation
    - Generate high confidence rules from each frequent itemset, where each rule is a binary partition of a frequent itemset

Efficient rule generation

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property

    $c(ABC \rightarrow D)$ can be larger or smaller than **$c(AB \rightarrow D)$**

  - ***But confidence of rules generated from the same itemset has an anti-monotone property***
  - Example: $X = \{A,B,C,D\}$:

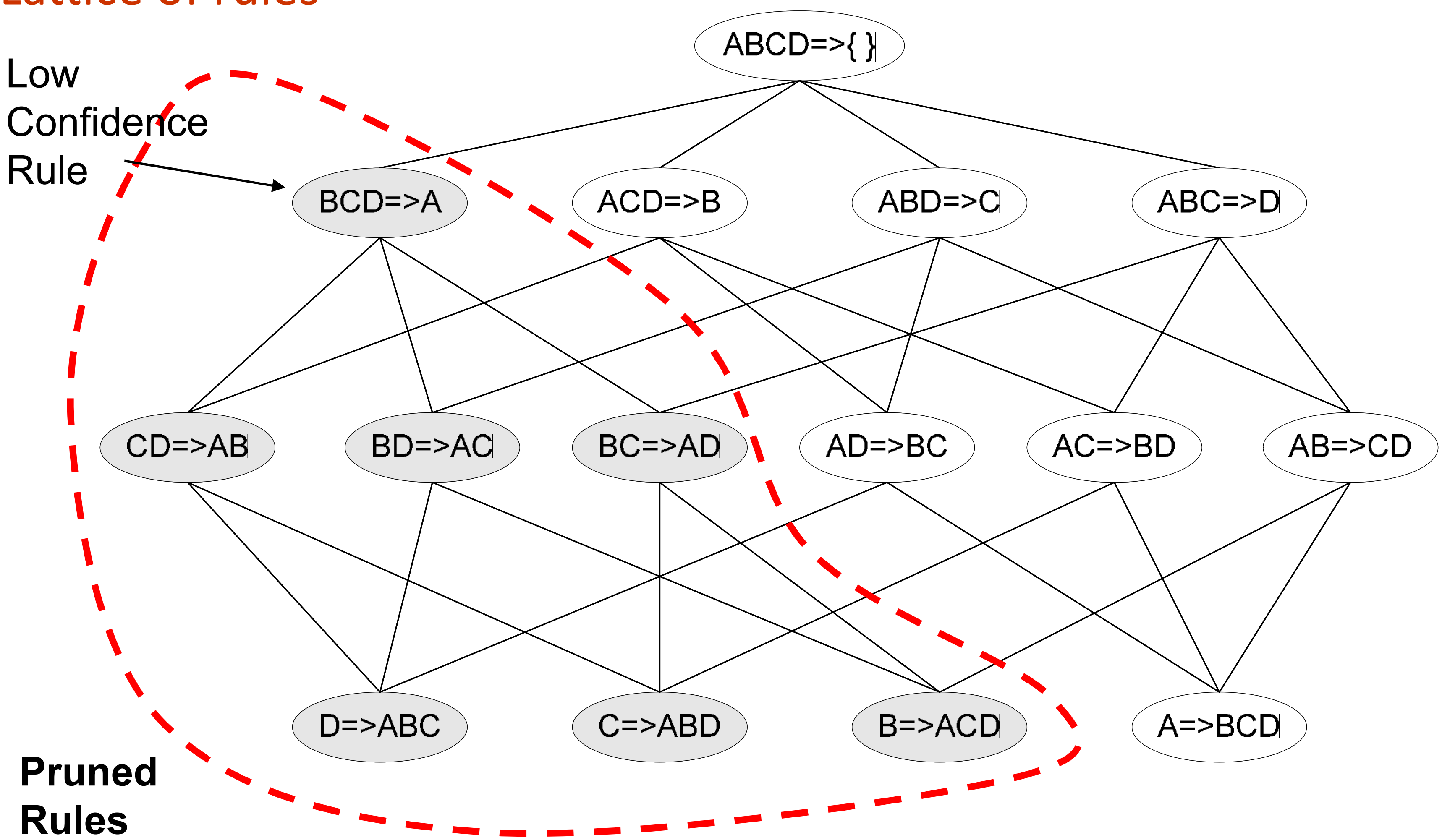
$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

  - **Why?**

Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

Rule Generation for Apriori Algorithm

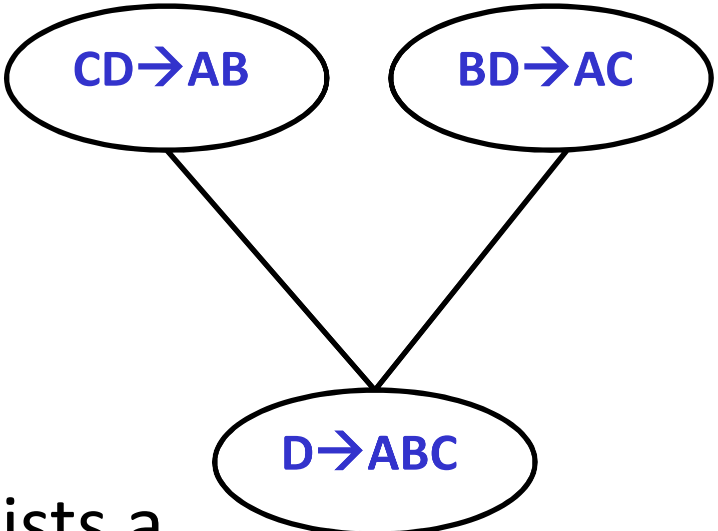
Lattice of rules



Apriori algorithm for rule generation

- Candidate rule is generated by merging two rules that share the same prefix
  in the rule consequent

- **join(CD→AB,BD—>AC)**
  would produce the candidate rule **D →ABC**



- **Prune** rule **D→ABC** if there exists a subset (e.g., **AD→BC**) that does not have high confidence