Associations Rules & Market Basket Analysis

Association rule mining

- Proposed by **Agrawal et al in 1993**.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for **Market Basket Analysis** to find how items purchased by customers are related.

Bread $\rightarrow$ Milk [sup = 5%, conf = 100%]

What Is Association Mining?

- Motivation: finding regularities in data
  - What products were often purchased together? — Beer and diapers
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?

Association rule mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

Market-Basket transactions

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

Example of Association Rules

{Diaper} $\rightarrow$ {Beer},
{Milk, Bread} $\rightarrow$ {Eggs,Coke},
{Beer, Bread} $\rightarrow$ {Milk},

Implication means co-occurrence, not causality!

Basket Data

Retail organizations, e.g., supermarkets, collect and store massive amounts sales data, called ***basket data***.

A record consist of

transaction date

items bought

Or, basket data may consist of items bought by a customer over a period.

- Items frequently purchased together:

Bread $\Rightarrow$PeanutButter

Example Association Rule

90% of transactions that purchase bread and butter also purchase milk

“IF” part = **antecedent**

“THEN” part = **consequent**

“Item set” = the items (e.g., products) comprising the antecedent or consequent

- Antecedent and consequent are *disjoint* (i.e., have no items in common)

Antecedent: bread and butter

Consequent: milk

Confidence factor: 90%

Transaction data: supermarket data

- Market basket transactions:

  t1: {bread, cheese, milk}

  t2: {apple, eggs, salt, yogurt}

  … …

  tn: {biscuit, eggs, milk}

- Concepts:
  - An *item*: an item/article in a basket
  - *I*: the set of all items sold in the store
  - A *transaction*: items purchased in a basket; it may have TID (transaction ID)
  - A *transactional dataset*: A set of transactions

Transaction data: a set of documents

- **A text document data set. Each document is treated as a "bag" of keywords**

doc1:	Student, Teach, School
doc2:	Student, School
doc3:	Teach, School, City, Game
doc4:	Baseball, Basketball
doc5:	Basketball, Player, Spectator
doc6:	Baseball, Coach, Game, Team
doc7:	Basketball, Team, City, Game

Definition: Frequent Itemset

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

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

The model: data

- $I = \{i_1, i_2, ..., i_m\}$: a set of *items*.
- **Transaction** t :
  - t a set of items, and $t \subseteq I$.
- **Transaction Database** T: a set of transactions $T = \{t_1, t_2, ..., t_n\}$.

- *I: itemset*

```
{cucumber, parsley, onion, tomato, salt, bread, olives, cheese, butter}
```

- T: set of transactions

```
1  {{cucumber, parsley, onion, tomato, salt, bread},
2   {tomato, cucumber, parsley},
3   {tomato, cucumber, olives, onion, parsley},
4   {tomato, cucumber, onion, bread},
5   {tomato, salt, onion},
6   {bread, cheese}
7   {tomato, cheese, cucumber}
8   {bread, butter}}
```

The model: Association rules

- A transaction *t* contains *X*, a set of items (itemset) in *I*, if $X \subseteq t$.
- An association rule is an implication of the form:

$$X \rightarrow Y, \text{ where } X, Y \subset I, \text{ and } X \cap Y = \emptyset$$

- An itemset is a set of items.
  - E.g., X = {milk, bread, cereal} is an itemset.
- A *k*-itemset is an itemset with *k* items.
  - E.g., {milk, bread, cereal} is a 3-itemset

Rule strength measures

- **Support:** The rule holds with **support *sup*** in *T* (the transaction data set) if sup% of transactions contain $X \cup Y$.
  - ***sup* =** probability that a transaction contains **Pr($X \cup Y$)**

  (Percentage of transactions that contain $X \cup Y$)
- **Confidence:** The rule holds in *T* with **confidence *conf*** if *conf*% of tranactions that contain *X* also contain *Y*.
  - ***conf* = conditional probability** that a transaction having *X* also contains *Y* **Pr(Y | X)**

  (Ratio of number of transactions that contain $X \cup Y$ to the number that contain X)
- An association rule is a pattern that states when *X* occurs, *Y* occurs with certain probability.

Rule strength measures

- **Lift:** any highly bought item will produce very high confidence
  - ***lift* = Pr(Y | X) / Pr(Y)**

  (transactions with X and Y / transactions with X) / fraction of transactions with Y)

  - Measures the rise (“lift”) in probability of having {Y} on the cart with the knowledge of {X} being present over the probability of having {Y} on the cart without any knowledge about presence of {X}
  - If lower than 1, buying X actually decreases the probability of buying Y.

Support and Confidence

- **Support count:** The support count of an itemset *X*, denoted by ***X.count***, in a data set *T* is the number of transactions in *T* that contain *X*. Assume *T* has *n* transactions.
- Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal: Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
  - Example:
    $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

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

- **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

Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

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

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

Is minimum support and minimum confidence can be automatically determined in mining association rules?

- For the **minimum support**, it all depends on the dataset. Usually, may start with a high value, and then decrease the values until to find a value that will generate enough paterns.
- For the **minimum confidence**, it is a little bit easier because it represents the confidence that you want in the rules. So usually, use something like 60 % . But it also depends on the data.
- In terms of performance, when ***minsup*** is higher you will find **less pattern** and the algorithm is faster. For ***minconf***, when it is set higher, there will be less pattern but it may not be faster because many algorithms don't use minconf to prune the search space. So obviously, setting these parameters also depends on how many rules you want.

An example

- Transaction data
- Assume:

  minsup = 30%

  minconf = 80%
- An example frequent *itemset*:

  {Chicken, Clothes, Milk} [sup = 3/7]
- Association rules from the itemset:

  Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

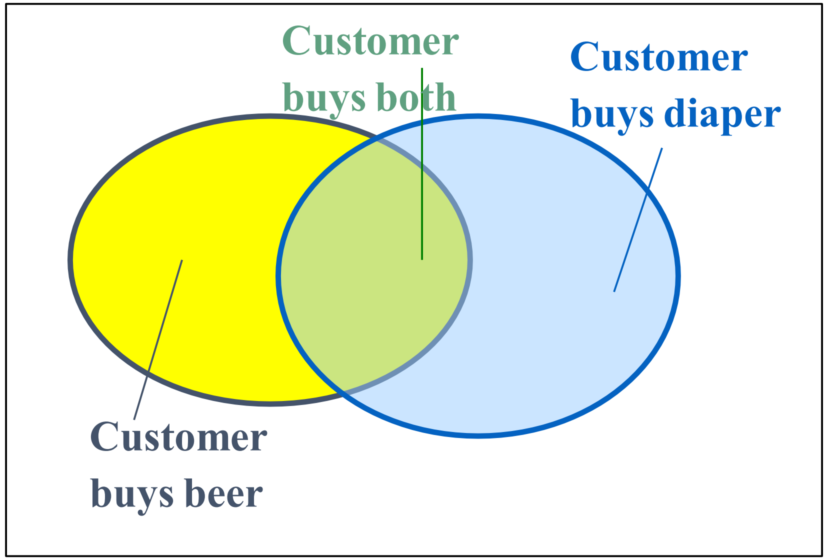
  … …

  Clothes, Chicken → Milk, [sup = 3/7, conf = 3/3]

t1:	**Bread, Chicken, Milk**
t2:	**Bread, Cheese**
t3:	**Cheese, Boots**
t4:	**Bread, Chicken, Cheese**
t5:	**Bread, Chicken, Clothes, Cheese, Milk**
t6:	**Chicken, Clothes, Milk**
t7:	**Chicken, Milk, Clothes**

Basic Concept: Association Rules

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F



- Let *min_support* = 50%, *min_conf* = 50%:
- $A \rightarrow C$ (50%, 66.7%)
- $C \rightarrow A$ (50%, 100%)

Frequent pattern	Support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

Association Rule Mining Task

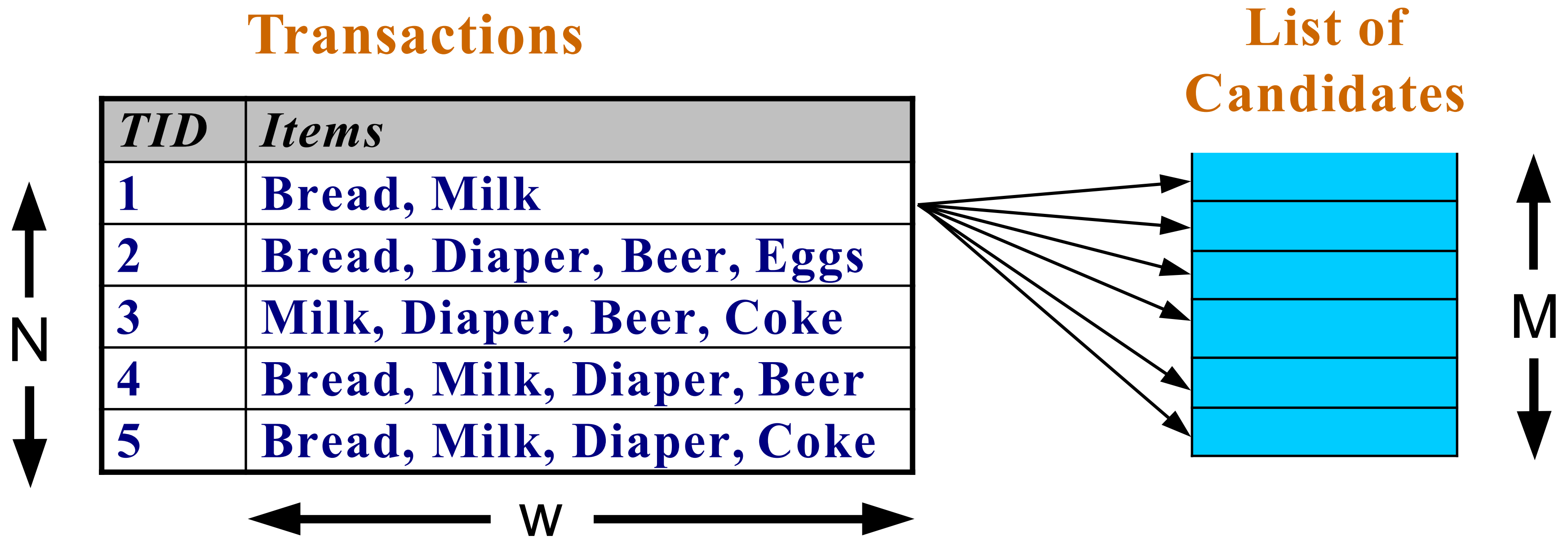
- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - **support ≥ *minsup* threshold**
  - **confidence ≥ *minconf* threshold**

- ***Brute-force approach:***
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds

  $\Rightarrow$ Computationally prohibitive!

Frequent Itemset Generation

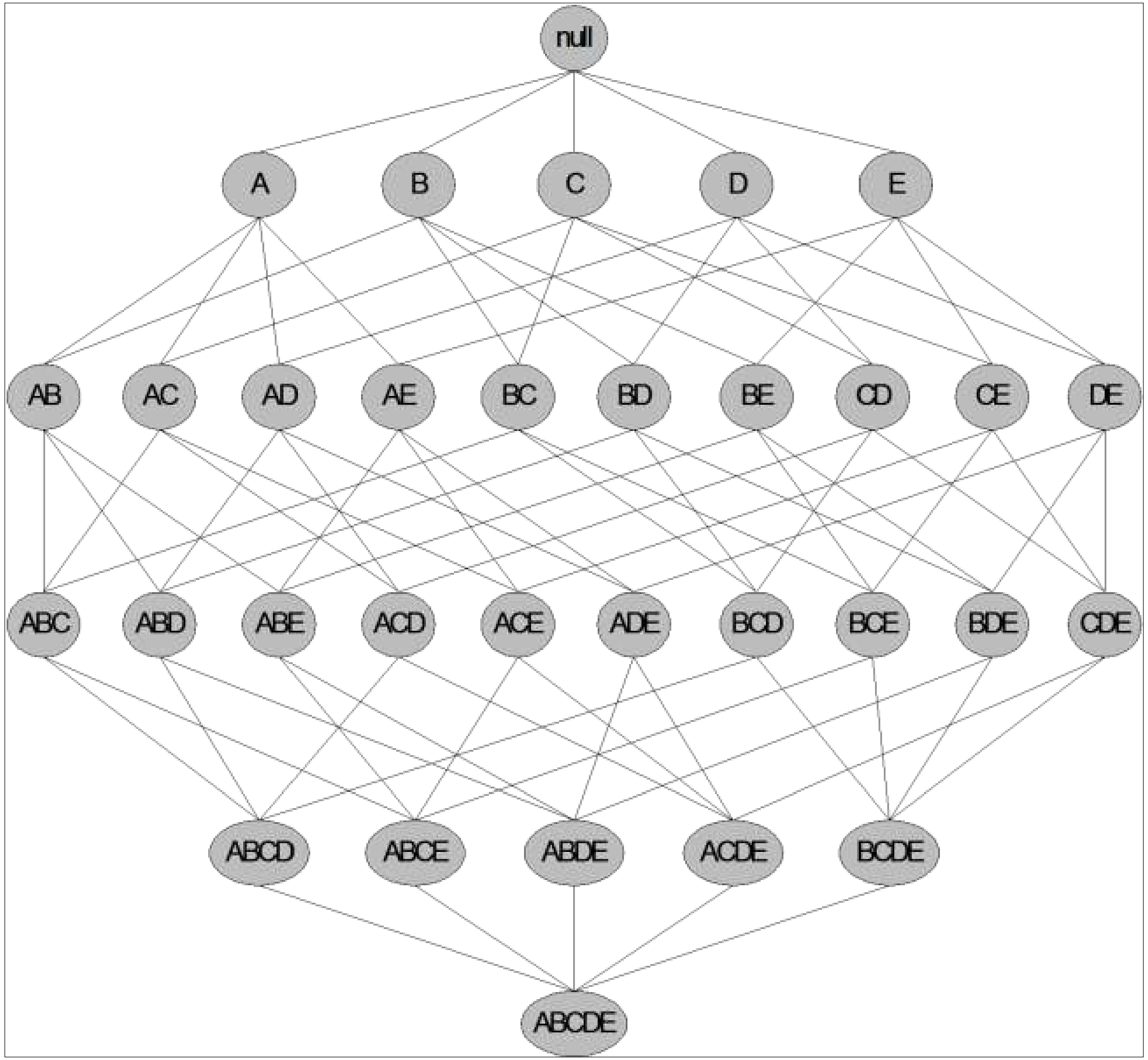
- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database



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

  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since $M = 2^d$!!!

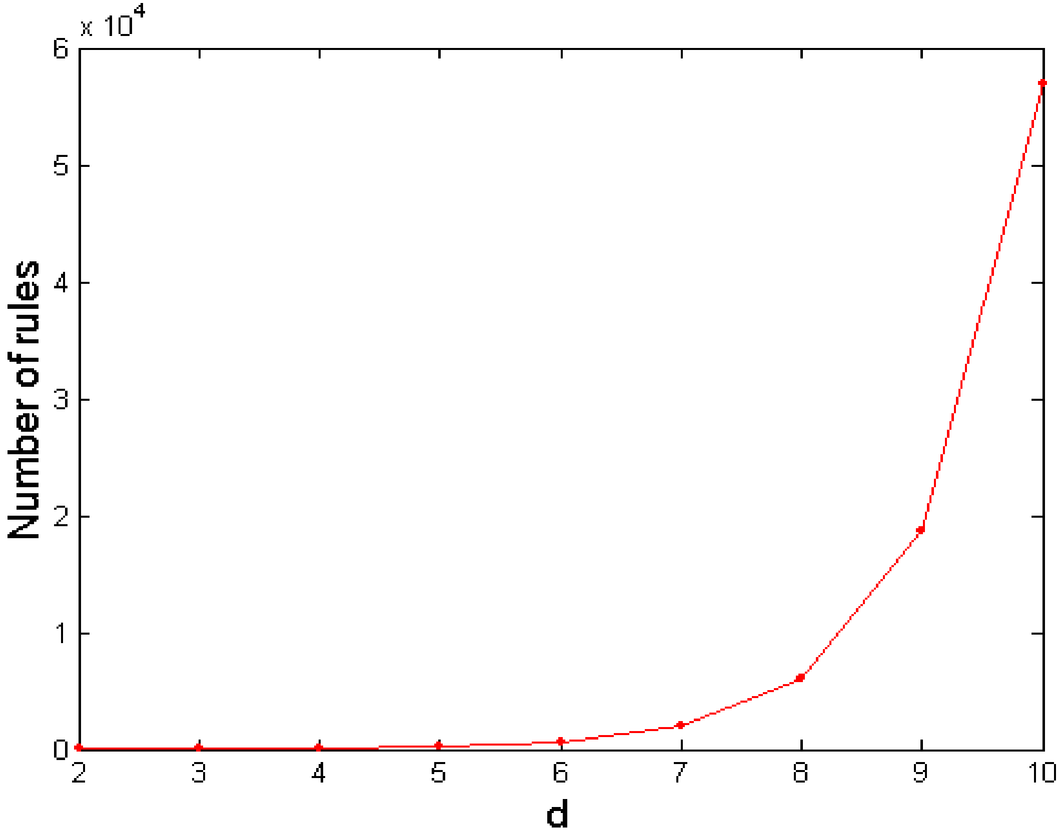
Brute-force approach:



Given d items, there are 2^d possible candidate itemsets

Computational Complexity

- Given d unique items:
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right] = 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

Mining Association Rules

- Two-step approach:

1. **Frequent Itemset Generation**
   - Generate all itemsets whose support $\geq$ minsup

   if an itemset is frequent, each of its subsets is frequent as well.

   - This property belongs to a special category of properties called ***antimonotonicity*** in the sense that if a set cannot pass a test, all of its supersets will fail the same test as well.

1. **Rule Generation**
   - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

- Frequent itemset generation is still computationally expensive

Frequent Itemset Generation

- An itemset X is ***closed*** in a data set D if there exists no proper super-itemset Y* such that Y has the same support count as X in D.

  ***(Y is a proper super-itemset of X if X is a proper sub-itemset of Y, that is, if $X \subset Y$. In other words, every item of X is contained in Y but there is at least one item of Y that is not in X.)**

- An itemset X is a ***closed frequent itemset*** in set D if X is both closed and frequent in D.

- An itemset X is a ***maximal frequent itemset (or max-itemset)*** in a data set D if X is frequent, and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in D.

Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases
  - Used by DHP (Direct Hashing & Purning) and vertical-based mining algorithms
- Reduce the number of comparisons (NM)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Many mining algorithms

- There are a large number of them!!
- They use different strategies and data structures.
- Their resulting sets of rules are all the same.
  - Given a transaction data set *T*, and a minimum support and a minimum confident, the set of association rules existing in *T* is uniquely determined.
- Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.
- We study only one: the Apriori Algorithm

The Apriori algorithm

- The algorithm uses a level-wise search, where **k-itemsets** are used to explore **(k+1)-itemsets**
- In this algorithm, frequent subsets are extended one item at a time (this step is known as ***candidate generation process***)
- Then groups of candidates are tested against the data.
- It identifies the frequent individual items in the database and extends them to larger and larger item sets as long as those itemsets appear sufficiently often in the database.
- Apriori algorithm determines frequent itemsets that can be used to determine association rules which highlight general trends in the database.

The Apriori algorithm

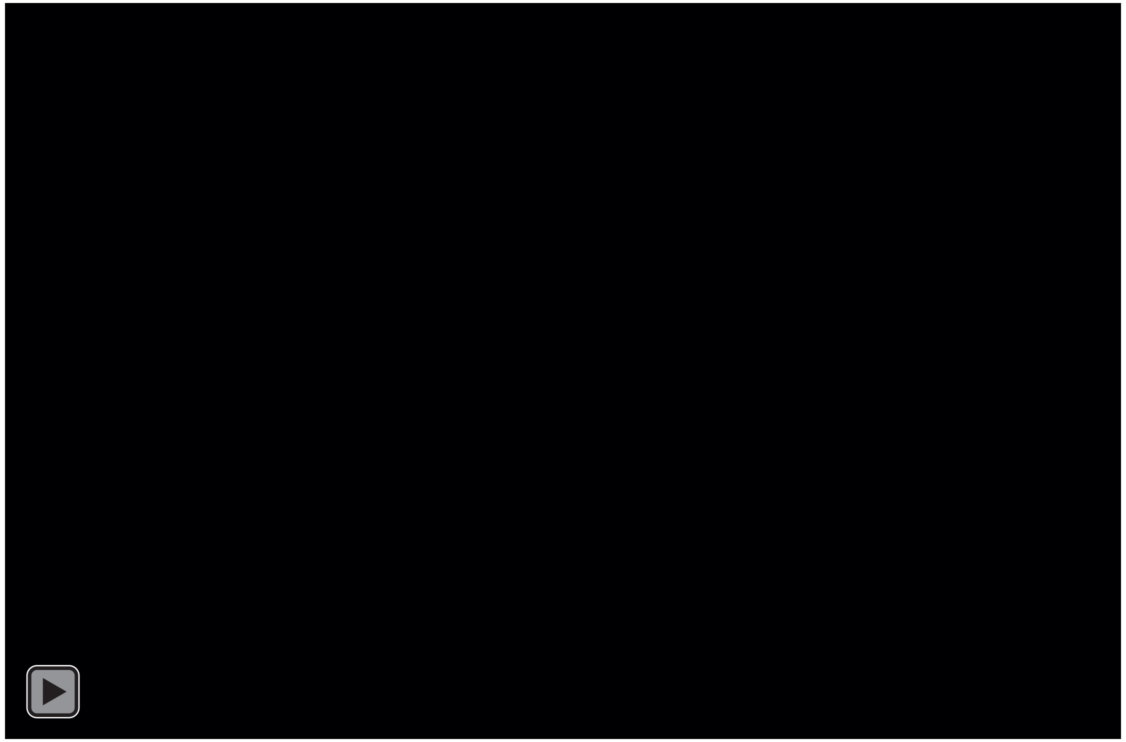
- The Apriori algorithm takes advantage of the fact that **any subset of a frequent itemset is also a frequent itemset.**
  - ***i.e., if {I1,I2} is a frequent itemset, then {I1} and {I2} should be frequent itemsets.***
- The algorithm can therefore, reduce the number of candidates being considered by only exploring the itemsets whose support count is greater than the minimum support count.
- All infrequent itemsets can be pruned if it has an infrequent subset.

How do we do that?

- So we build a ***Candidate list*** of **k-itemsets** and then extract a ***Frequent list*** of **k-itemsets** using the support count
- After that, we use the ***Frequent list*** of **k-itemsets** in determing the ***Candidate*** and ***Frequent list*** of **k+1-itemsets.**
- We use ***Pruning*** to do that
- We repeat until we have an empty ***Candidate*** or ***Frequent*** of **k-itemsets**
    - Then we return the list of ***k-1-itemsets***.

How do we do that?

How do we do that?



KEY CONCEPTS

- Frequent Itemsets: All the sets which contain the item with the minimum support (denoted by L_i for i^{th} itemset).
- Apriori Property: Any subset of frequent itemset must be frequent.
- Join Operation: To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself.

APRIORI ALGORITHM EXAMPLE

Database D

Minsup = 0.5

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
{2 3 5}	2

The Apriori Algorithm : Pseudo Code

- Join Step: C_k is generated by joining L_{k-1} with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code : C_k : Candidate itemset of size k

  L_k : frequent itemset of size k

```
L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) do begin
    C_k+1 = candidates generated from L_k;
    for each transaction t in database do
            increment the count of all candidates in C_k+1
            that are contained in t
    L_k+1  = candidates in C_k+1 with min_support
    end
return ∪_k L_k;
```

Apriori's Candidate Generation

- For k=1, C_1 = all 1-itemsets.
- For k>1, generate C_k from L_{k-1} as follows:
  - *The join step*

    C_k = k-2 way join of L_{k-1} with itself

    If both $\{a_1, \dots, a_{k-2}, a_{k-1}\}$ & $\{a_1, \dots, a_{k-2}, a_k\}$ are in L_{k-1},

    then add $\{a_1, \dots, a_{k-2}, a_{k-1}, a_k\}$ to C_k

    (We keep items **sorted**).
  - *The prune step*

    Remove $\{a_1, \dots, a_{k-2}, a_{k-1}, a_k\}$ if it contains a non-frequent (k-1) subset

Example – Finding frequent itemsets

Dataset D

TID	Items
T100	a1 a3 a4
T200	a2 a3 a5
T300	a1 a2 a3 a5
T400	a2 a5

minSup=0.5

1. scan D ➔ C_1: a1:2, a2:3, a3:3, a4:1, a5:3

➔ L_1: a1:2, a2:3, a3:3, a5:3

➔ C_2: a1a2, a1a3, a1a5, a2a3, a2a5, a3a5

2. scan D ➔ C_2: a1a2:1, a1a3:2, a1a5:1, a2a3:2, a2a5:3, a3a5:2

➔ L_2: a1a3:2, a2a3:2, a2a5:3, a3a5:2

➔ C_3: a1a2a3, a2a3a5

➔ Pruned C_3: a1a2a3

3. scan D ➔ L_3: a2a3a5:2

Order of items can make difference in process

Dataset D

TID	Items
T100	1 3 4
T200	2 3 5
T300	1 2 3 5
T400	2 5

minSup=0.5

1. scan D ➔ C_1: 1:2, 2:3, 3:3, 4:1, 5:3

➔ L_1: **1**:2, **2**:3, **3**:3, **5**:3

➔ C_2: 12, 13, 15, 23, 25, 35

2. scan D ➔ C_2: 12:1, **13:2**, 15:1, **23:2**, **25:3**, **35:2**

Suppose the order of items is: 5,4,3,2,1

➔ L_2: **31**:2, **32**:2, **52:**3, **53**:2

➔ C_3: 321, 532

➔ Pruned C_3: 532

3. scan D ➔ L_3: 532:2

Generating Association Rules From frequent itemsets

- **<u>Procedure 1:</u>**
- Let we have the list of frequent itemsets

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5
500	1 3 5

Itemset	Support
{1,3,5}	2
{2,3,5}	2

- Generate all nonempty subsets for each frequent itemset **I**
  - **For I = {1,3,5}, all nonempty subsets are {1,3},{1,5},{3,5},{1},{3},{5}**
  - **For I = {2,3,5}, all nonempty subsets are {2,3},{2,5},{3,5},{2},{3},{5}**

Generating Association Rules From frequent itemsets

- **<u>Procedure 2:</u>**
- For every nonempty subset S of I, output the rule:

  S → (I - S)

    - **If support_count(I)/support_count(s)>= min_conf**

      where min_conf is minimum confidence threshold

- **Let us assume:**
- **minimum confidence threshold is 60%**

Association Rules with confidence

- **R1 : 1,3 -> 5**
  - **Confidence = sc{1,3,5}/sc{1,3} = 2/3 = 66.66% (R1 is selected)**
- **R2 : 1,5 -> 3**
  - **Confidence = sc{1,5,3}/sc{1,5} = 2/2 = 100% (R2 is selected)**
- **R3 : 3,5 -> 1**
  - **Confidence = sc{3,5,1}/sc{3,5} = 2/3 = 66.66% (R3 is selected)**
- **R4 : 1 -> 3,5**
  - **Confidence = sc{1,3,5}/sc{1} = 2/3 = 66.66% (R4 is selected)**
- **R5 : 3 -> 1,5**
  - **Confidence = sc{3,1,5}/sc{3} = 2/4 = 50% (R5 is REJECTED)**
- **R6 : 5 -> 1,3**
  - **Confidence = sc{5,1,3}/sc{5} = 2/4 = 50% (R6 is REJECTED)**

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5
500	1 3 5

How to efficiently generate rules?

- In general, confidence does not have an anti-monotone property

  c(ABC→D) can be larger or smaller than c(AB →D)

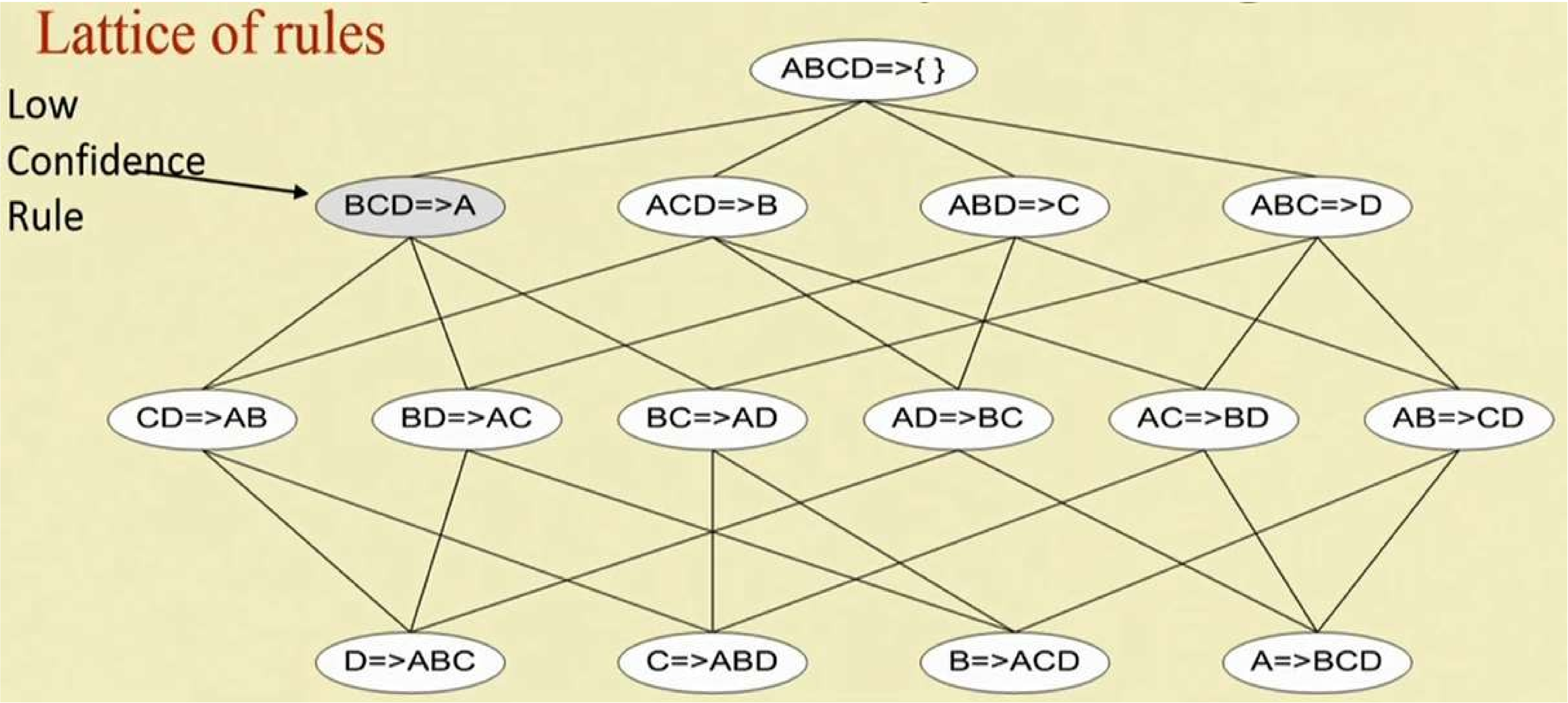
- But confidence of rules generated from the same itemset has an anti-monotone property

  - e.g., L= {A,B,C,D}

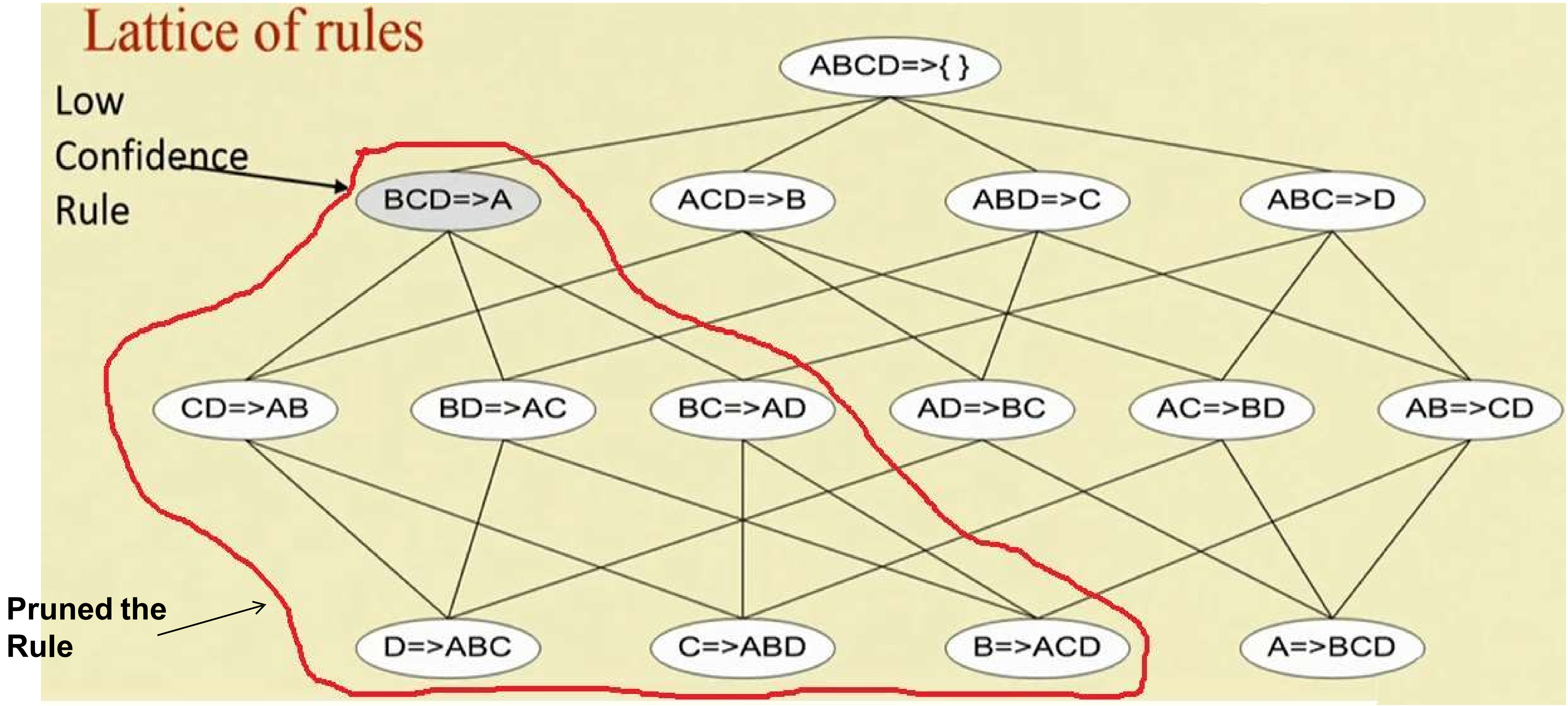
    c(ABC→D) ≥ c(AB→CD) ≥ c(A→BCD)

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

Rule generation for Apriori Algorithm



Rule generation for Apriori Algorithm



Rule generation for Apriori Algorithm

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

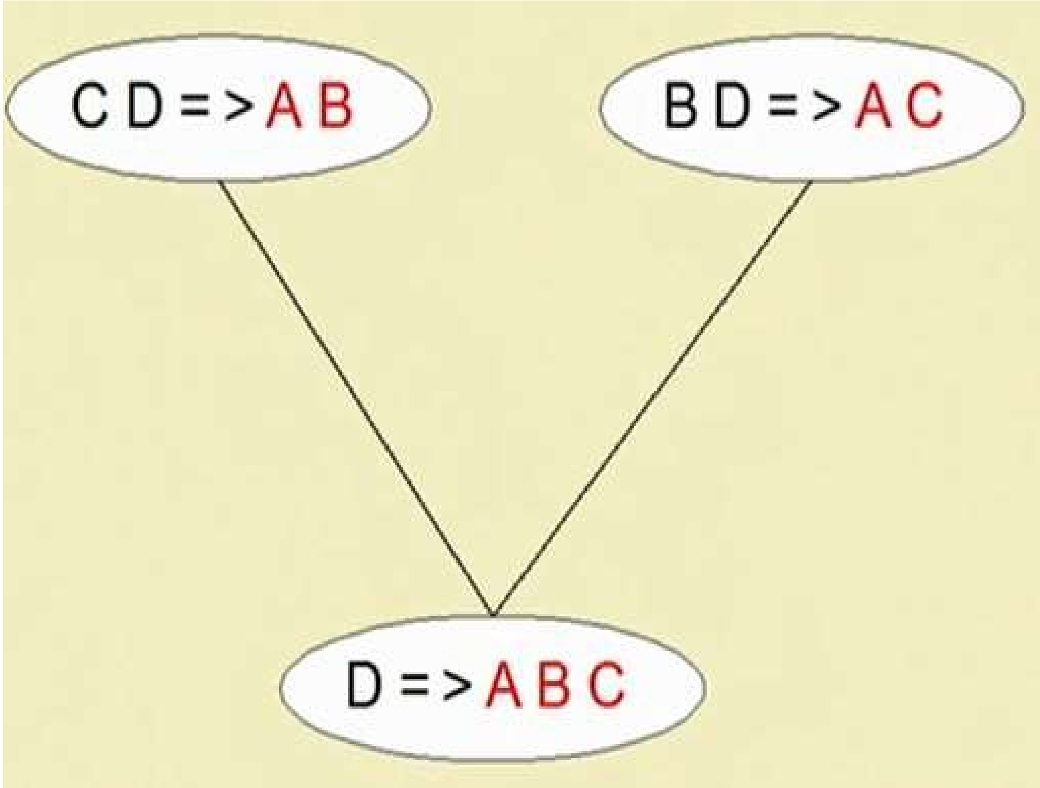
- join (CD=>AB, BD=>AC)

  would produce the candidate rule,

  D=>ABC

- Prune rule D=>ABC if its subset

  AD=>BC does not have high confidence



Rule selection for Apriori Algorithm

- Select the best rules in terms of lifts

Association rules in Python

- mlxtend package: `from mlxtend.frequent_patterns import apriori` ejemplo
- apyori package: ejemplo
- pyarmviz: visualition of association rules