Data Mining
Apriori Algorithm

- **Apriori principle**
- **Frequent itemsets generation**
- **Association rules generation**

Section 6 of course book

Association Rule Mining (*ARM*)

- ***ARM*** is not only applied to market basket data
- There are algorithm that can find any association rules
  - Criteria for selecting rules: confidence, number of tests in the left/right hand side of the rule

(Cheat = no) ∧ (Refund = yes) → (Marital Status = singel)

(Taxable Income > 100) → (Cheat = no) ∧ (Refund = yes)

- What is the difference between classification and ***ARM***?
  - ***ARM*** can be used for obtaining classification rules

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Mining Association Rules

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 Rules:

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

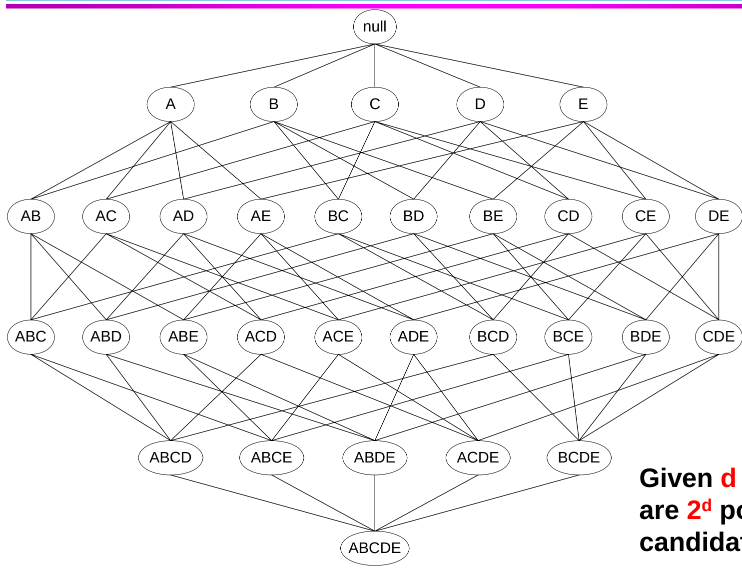
Observations:

- All the above rules are binary partitions of the same itemset: {Milk, Diaper, Beer}
- 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:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support $\geq$ *minsup*
  2. 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 computationally expensive

Frequent Itemset Generation



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

Frequent Itemset Generation

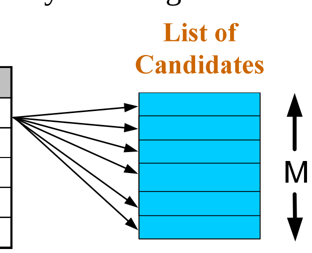
- **Brute-force approach**:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

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

N

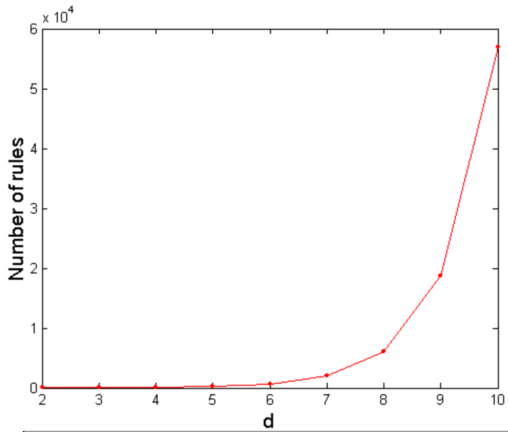
w



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

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

Frequent Itemset Generation Strategies

- Reduce the number of candidate itemsets (M)
  - Complete search: $M = 2^d$
  - Use pruning techniques to reduce M
    - Used in *Apriori algorithm*
- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases
- 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
    - Used in the Apriori algorithm

Apriori Algorithm

- Proposed by **Agrawal** R, **Imielinski** T, **Swami** AN
  - *"Mining Association Rules between Sets of Items in Large Databases."*
  - *SIGMOD*, June 1993
  - Available in Weka
- Other algorithms
  - ***D***ynamic ***H***ash and ***P***runing (***DHP***), 1995
  - FP-Growth, 2000
  - H-Mine, 2001

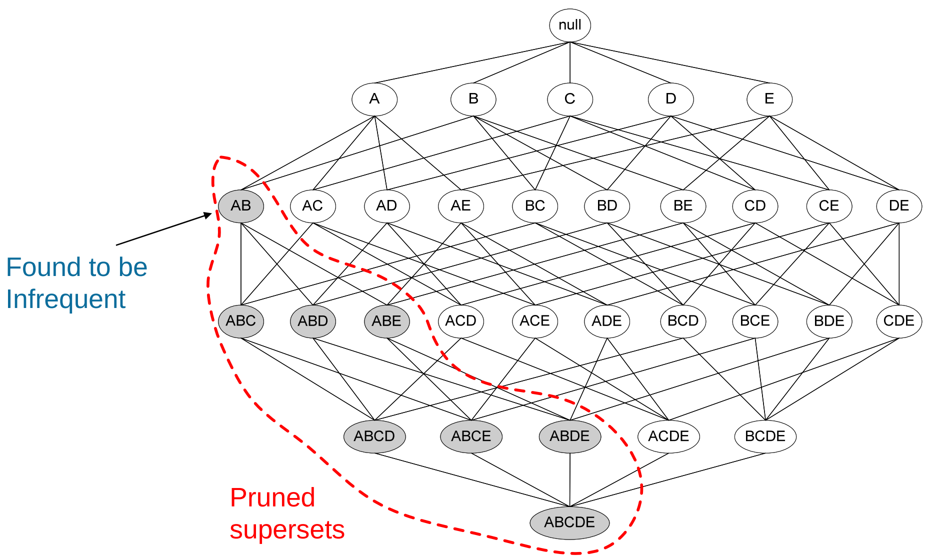
Reducing Number of Candidate Itemsets

- **Apriori principle**:
  - If an itemset is frequent, then all of its subsets must also be frequent, or
  - if an item set is infrequent then all its supersets must also be infrequent
- Apriori principle holds due to the following property of the support measure:

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

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

Illustrating Apriori Principle



Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Bread,Beer}	**2**
{Bread,Diaper}	**3**
{Milk,Beer}	**2**
{Milk,Diaper}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving **Coke** or **Eggs**)

Minimum Support = 3

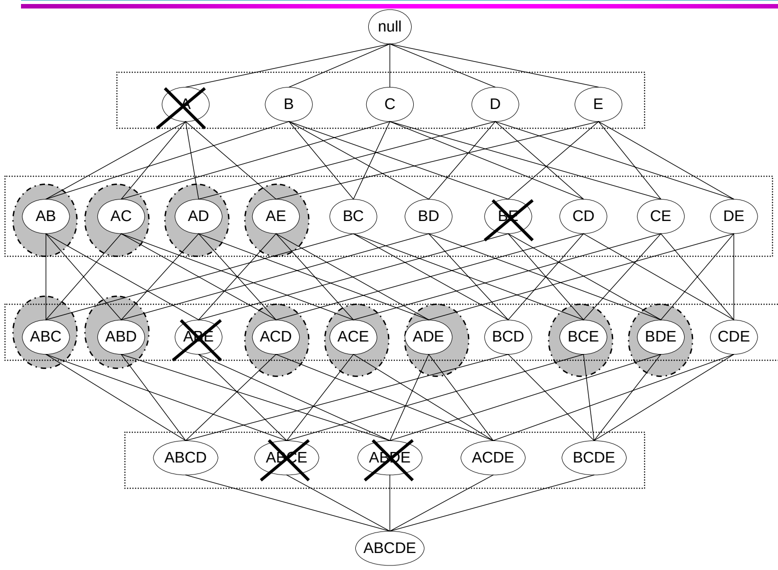
Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	**3**

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 1 = 13$

What about ***{Beer, Diaper, Milk}*** *?*
And, ***{Bread, Milk, Beer}****?*
And, ***{Bread, Diaper, Beer}****?*

Frequent Itemset Generation



Apriori Algorithm

- Level-wise algorithm:
  1. Let k = 1
  2. Generate frequent itemsets of length 1
  3. Repeat until no new frequent itemsets are identified
     1. Generate length (k+1) candidate itemsets from length k frequent itemsets
     2. Prune candidate itemsets containing subsets of length k that are infrequent
        - *How many k-itemsets contained in a (k+1)-itemset?*
     3. Count the support of each candidate by scanning the DB
     4. Eliminate candidates that are infrequent, leaving only those that are frequent

Note: steps 3.2 and 3.4 prune itemsets that are infrequent

Generating Itemsets Efficiently

- How can we efficiently generate all (frequent) item sets at each iteration?
  - Avoid generate repeated itemsets and infrequent itemsets
- Finding one-item sets easy
- **Idea:** use one-item sets to generate two-item sets, two-item sets to generate three-item sets, …
  - If (A B) is frequent item set, then (A) and (B) have to be frequent item sets as well!
  - In general: if X is a frequent *k*-item set, then all (*k*-1)-item subsets of X are also frequent
  - ⇒ Compute *k*-item set by merging two (*k*-1)-itemsets. Which ones?

E.g. Merge ***{Bread, Milk}*** with ***{Bread, Diaper}*** to get ***{Bread, Diaper, Milk}***

Example: generating frequent itemsets

- Given: five frequent 3-itemsets

(A B C) , (A B D) , (A C D) , (A C E) , (B C D)

1. Lexicographically ordered!
2. Merge ($\boxed{x_1, x_2, \ldots,}\ x_{k-1}$) with ($\boxed{y_1, y_2, \ldots,}\ y_{k-1}$),
   if $x_1 = y_1$, $x_2 = y_2$, …, $x_{k-2} = y_{k-2}$

- Candidate 4-itemsets:

(A B C D) **OK because of (A B C), (A B D), (A C D), (B C D)**

(A C D E) **Not OK because of (C D E)**

3. Final check by counting instances in dataset!

Reducing Number of Comparisons

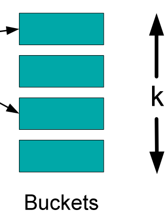
- Candidate counting:
  - Scan the database of transactions to determine the support of each generated candidate itemset
  - To reduce the number of comparisons, store the candidates in a hash structure
    - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

Transactions

N

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

Hash Structure



Rule Generation

- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \to L - f$ satisfies the minimum confidence requirement
  - If {A,B,C,D} is a frequent itemset, candidate rules:

ABC →D,	ABD →C,	ACD →B,	BCD →A,
A →BCD,	B →ACD,	C →ABD,	D →ABC
AB →CD,	AC → BD,	AD → BC,	BC →AD,
BD →AC,	CD →AB,		

- If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \to \emptyset$ and $\emptyset \to L$)

Rule Generation

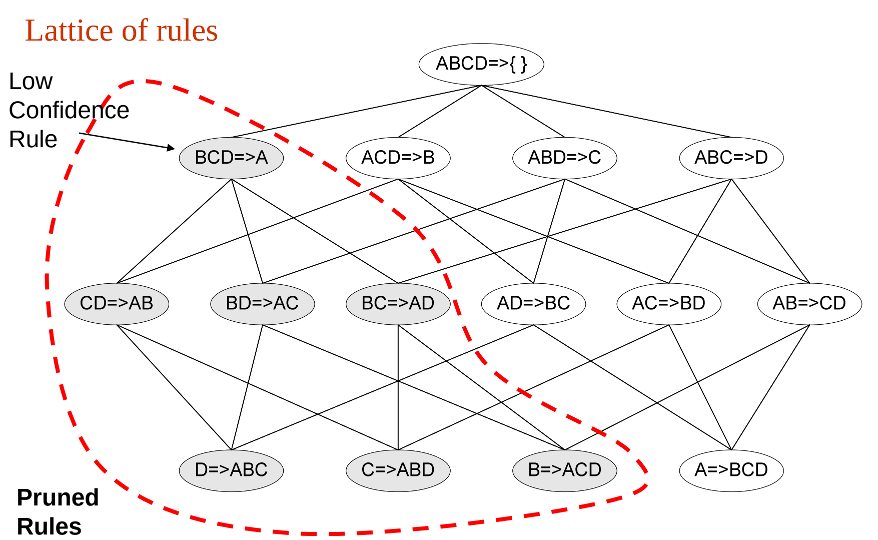
- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property

    $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$
  - But confidence of rules generated from the same itemset have an anti-monotone property
  - e.g., L = {A,B,C,D}:

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

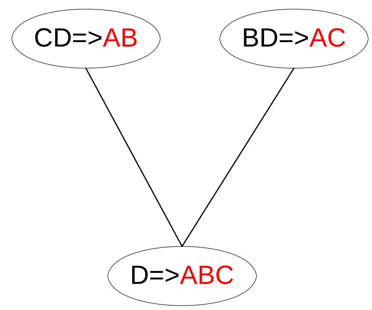
➢ Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

Rule Generation for Apriori Algorithm



Rule Generation for Apriori Algorithm

- 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
  does not have high confidence



- Support counts have been obtained during the frequent itemset generation step