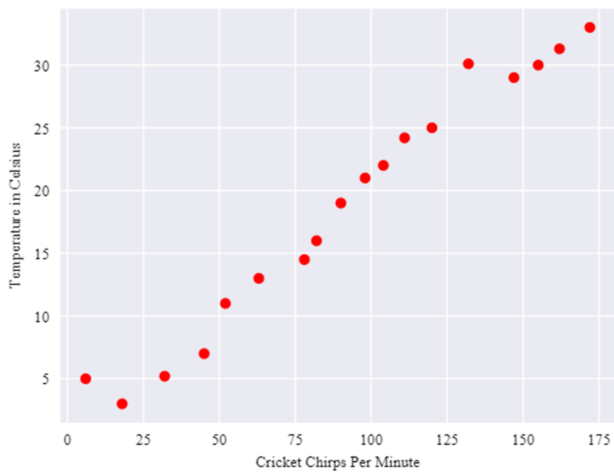
Association analysis. Basic concepts

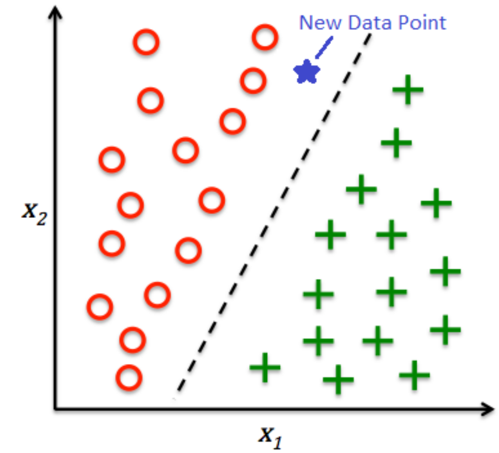
Lecture 12

Types of learning tasks

Supervised learning

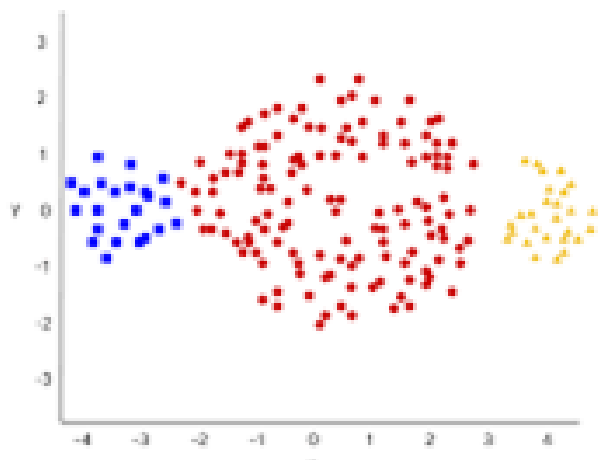


Prediction



Classification

Unsupervised learning



Clustering

TransactionId	Items
1	{A,C,D}
2	{B,C,D}
3	{A,B,C,D}
4	{B,D}
5	{A,B,C,D}

Associations

Classification rules: reminder

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	**True**	**No**
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	**True**	**No**

R1: if humidity=normal and windy=false then yes
R2: if outlook=overcast then yes
R3: if temp=hot then no
R4: if outlook=rainy and windy=true then no

- *LHS*: rule *antecedent* : in this case – combination of attribute-values
- *RHS*: rule *consequent*: in this case – class label

Association rules: no class

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	**Cool**	**Normal**	False	Yes
Rainy	**Cool**	**Normal**	True	No
Overcast	**Cool**	**Normal**	True	Yes
Sunny	Mild	High	False	No
Sunny	**Cool**	**Normal**	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

R1: if temp=cool then humidity=normal

- *LHS*: rule *antecedent* : combination of attribute-values
- *RHS*: rule *consequent*: combination of attribute-values

Association rules: **no class**

Outlook	**Temp**	**Humidity**	**Windy**	**Play**
Sunny	**Hot**	**High**	False	No
Sunny	**Hot**	**High**	True	No
Overcast	**Hot**	**High**	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	**Hot**	Normal	False	Yes
Rainy	Mild	High	True	No

R1: if temp=cool then humidity=normal
R2: if temp=hot then humidity=high

- *LHS*: rule *antecedent* : combination of attribute-values
- *RHS*: rule *consequent*: combination of attribute-values

The goal: discover relationships between attributes

Outlook	Temp	Humidity	Windy	Play
Sunny	**Hot**	**High**	False	No
Sunny	**Hot**	**High**	True	No
Overcast	**Hot**	**High**	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

R1: if temp=cool then humidity=normal
R2: if temp=hot then humidity=high

- Association rules are looking for the relationships between objects
- They discover related properties of objects by searching for the attribute-values that appear often in the same observation

Terminology: market basket

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

Market basket

Terminology: market basket

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

Item

Terminology: market basket

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

Observation = *transaction*

Terminology: itemset

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

Set of *k* items – *k*-itemset

A={Coke, Diaper} – 2-itemset

If itemset A is a subset of items in transaction t_i, we say t_i contains A or *supports* A

Terminology: support count

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

Number of transactions which contain itemset A – *support count* (σ)

support count {Coke, Diaper} =2

Terminology: support (fraction)

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

Fraction of transactions which contain itemset A – *support s*

support {Coke, Diaper} =2/5

Terminology: frequent itemset

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

An itemset whose support is greater than or equal to a *minsup* threshold – *frequent itemset*

For minsup=40% frequent itemsets are:
{Coke, Diaper}
{Bread, Milk}
…

Association rules

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

- **Rule Evaluation Metrics (*X* → *Y*)**
  - 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, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

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

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

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

Why Use Support and Confidence?

Support

- A rule that has very low support may occur simply by chance.
- Support is often used to eliminate random spurious rules.

Confidence

- Measures the reliability of the inference made by a rule.
- For a rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X.
- Confidence provides an estimate of the **conditional probability** of Y given X.

Association analysis: motivation

- Marketing and Sales Promotion:

  Let the rule discovered be

  {Bagels, … } --> {Potato Chips}

  - Potato Chips as consequent

  Can be used to determine what should be done to boost its sales.

  - Bagels in the antecedent

  Can be used to see which products would be affected if the store discontinues selling bagels

ML Task: Learning Association Rules

- Given a set of transactions **T**, the goal of association rule learning 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

  ⇒ **Computationally prohibitive**!

How many possible rules R

- Suppose there are *d* items in total. We first choose k of the items to form the left-hand side of the rule. There are $C_{d,k}$ ways for doing this.
- Now, there are $C_{d-k,i}$ ways to choose the remaining items to form the right-hand side of the rule, where $1 \le i \le d-k$.

$$
\begin{aligned}
R &= \sum_{k=1}^{d} \binom{d}{k} \sum_{i=1}^{d-k} \binom{d-k}{i} \\
&= \sum_{k=1}^{d} \binom{d}{k} \left(2^{d-k} - 1\right) \\
&= \sum_{k=1}^{d} \binom{d}{k} 2^{d-k} - \sum_{k=1}^{d} \binom{d}{k} \\
&= \sum_{k=1}^{d} \binom{d}{k} 2^{d-k} - \left(2^{d} - 1\right)
\end{aligned}
$$

We applied:

$$\sum_{i=1}^{n} \binom{n}{i} = 2^{n} - 1$$

We also have that:

$$(1+x)^{d} = \sum_{i=1}^{d} \binom{d}{i} x^{d-i} + x^{d}$$

For $x = 2$

$$3^{d} = \sum_{i=1}^{d} \binom{d}{i} 2^{d-i} + 2^{d}$$

Therefore $R = 3^{d} - 2^{d} - \left(2^{d} - 1\right) = 3^{d} - 2^{d+1} + 1$

Brute-force approach

- $R=3^d-2^{d+1}+1$
- For d=6 (small),

  $3^6-2^7+1=602$ possible rules

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

- However, 80% of the rules are discarded after applying *minsup*=20% and *minconf*=50%, thus making most of the computations wasted.

- So, it would be useful to prune the rules early without having to compute their support and confidence values.

An initial step toward improving the performance: *decouple the support and confidence requirements.*

Learning 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} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {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

If the itemset is infrequent, then all six candidate rules can be pruned immediately without us having to compute their confidence values.

Learning Association Rules

Two-step approach:

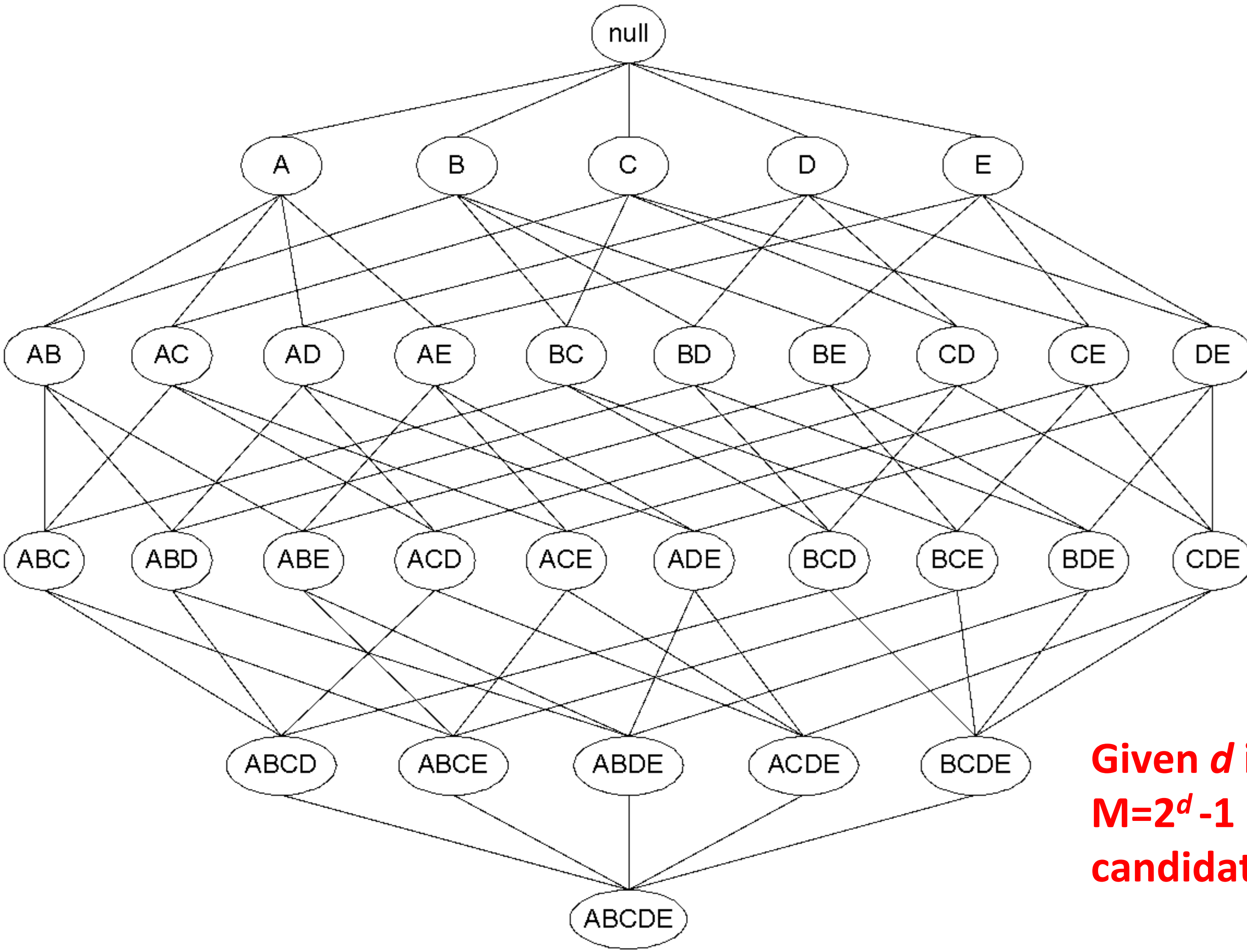
1. Frequent Itemset Generation
   – Generate all itemsets whose support ≥ minsup (these itemsets are called *frequent itemset*)
2. Rule Generation
   – Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

We focus on **frequent itemset generation** first

Step 1

FREQUENT ITEMSET GENERATION

Candidates for frequent itemsets



Given *d* items, there are $M=2^d-1$ possible candidate itemsets

Frequent Itemset Generation: brute force

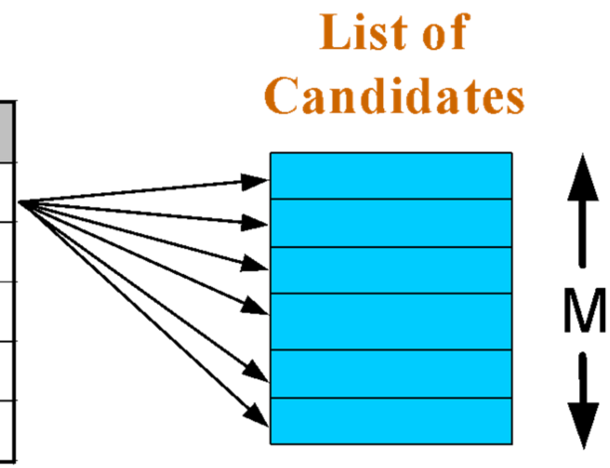
- Each itemset in the lattice is a candidate frequent itemset
- Count the support of each candidate by scanning the database
- Match each transaction against every candidate
- Complexity ~ $O(NMw)$ => Expensive since $M = 2^d$!!!
  - w is max transaction width (max number of items in one transaction).

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



Frequent itemset generation: Apriori algorithm

- The name *Apriori* is based on the fact that we use *prior* knowledge about k-itemsets in order to prune candidate k+1-itemsets

- The idea: level-wise processing
  – find frequent 1-itemsets: F_1
  – F_1 is used to find F_2
  – In general, F_k is used to find F_{k+1}

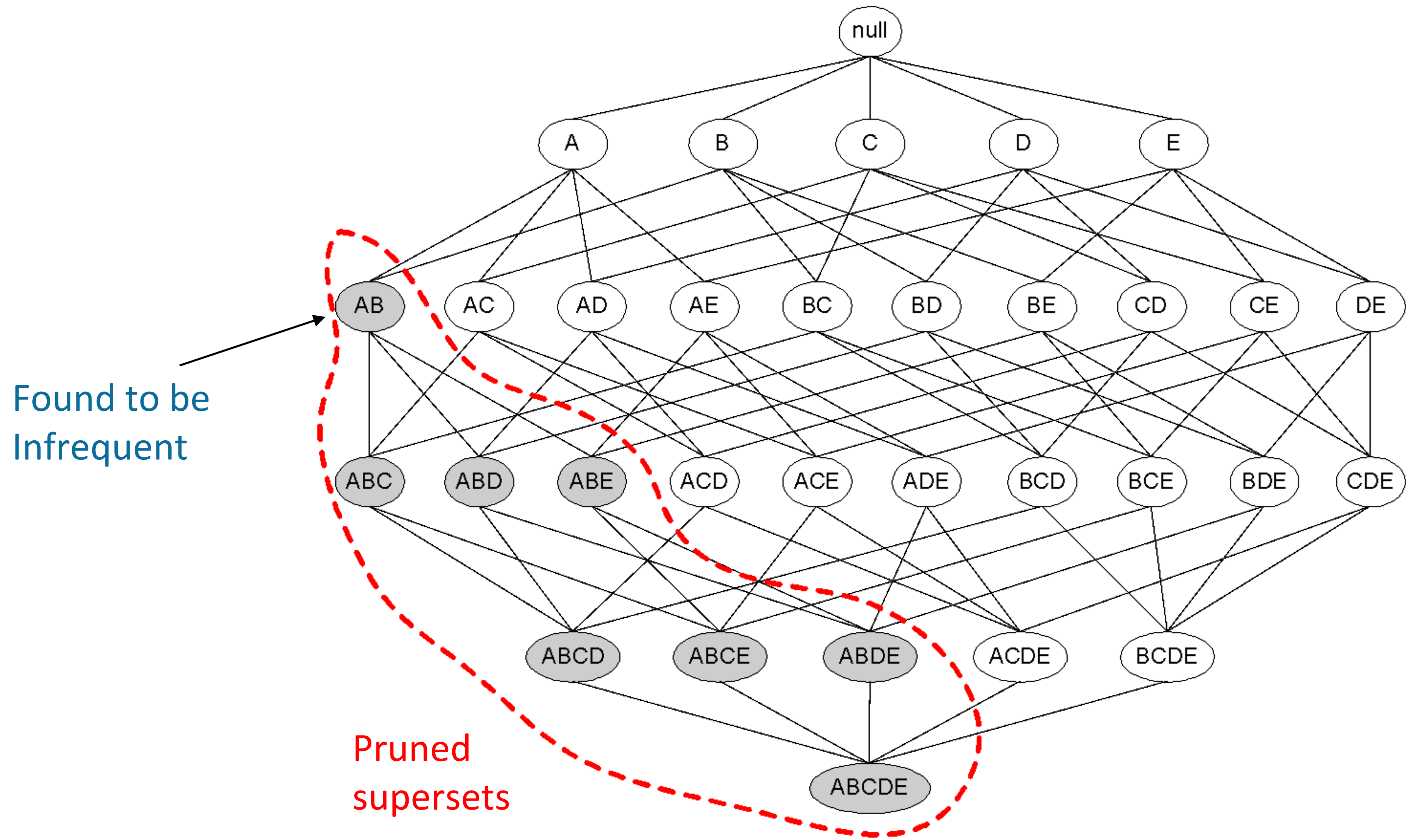
Anti-monotone property of support

- The efficiency of this approach is based on *anti-monotone* property of support: if a set cannot pass the support test, all its supersets will fail the same test:

- All subsets of a frequent itemset A must also be frequent

  If itemset A appears in less than *minsup* fraction of transactions, then itemset A with one more item added cannot occur more frequently than A. Therefore, if A is not frequent, all its supersets are not frequent as well

Illustrating Apriori Principle



Apriori Principle: example

Items (1-itemsets)

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

Minimum support count = 3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

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

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$C_{6,1} + C_{6,2} + C_{6,3} = 6 + 15 + 20 = 41$

With support-based pruning,
$6 + 6 + 1 = 13$
counting scans

With the Apriori principle we need to keep only this triplet, because it's the only one whose subsets are all frequent.

Apriori Algorithm

Let $k=1$

Generate set F_1 of frequent 1-itemsets

Repeat until F_k is empty

$k=k+1$

Generate all candidate k-itemsets C_k

from frequent k-1 - itemsets F_{k-1}

Prune candidate itemsets which contain subsets

of length k-1 that are infrequent

Count the support of each candidate in C_k by

scanning the Dataset

and eliminate candidates that are infrequent,

leaving only those that are frequent - F_k

Candidate generation and pruning

Many ways to generate candidate itemsets

An effective candidate generation procedure:

1. Should avoid generating too many unnecessary candidates.
   – A candidate itemset is unnecessary if at least one of its subsets is infrequent.
2. Must ensure that the candidate set is complete,
   – i.e., no frequent itemsets are left out by the candidate generation procedure.
3. Should not generate the same candidate itemset more than once.
   – E.g., the candidate itemset {a, b, c, d} can be generated in many ways---
      - by merging {a, b, c} with {d},
      - {c} with {a, b, d}, etc.

Generating C_{k+1} from F_k: brute force

- A brute force method considers every frequent *k*-itemset as a potential candidate and then applies the pruning step to remove any unnecessary candidates.

Frequent Items

Item
Beer
Bread
Cola
Diapers
Milk
Eggs

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Milk}
{Beer, Bread, Eggs}
{Beer, Cola, Diapers}
{Beer, Cola, Milk}
{Beer, Cola, Eggs}
{Beer, Diapers, Milk}
{Beer, Diapers, Eggs}
{Beer, Milk, Eggs}
{Bread, Cola, Diapers}
{Bread, Cola, Milk}
{Bread, Cola, Eggs}
{Bread, Diapers, Milk}
{Bread, Diapers, Eggs}
{Bread, Milk, Eggs}
{Cola, Diapers, Milk}
{Cola, Diapers, Eggs}
{Cola, Milk, Eggs}
{Diapers, Milk, Eggs}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

$F_{k-1} \times F_1$ Method

- Extend each frequent $(k - 1)$-itemset with a frequent 1-itemset.

- Is it complete?

The procedure is complete because every frequent k-itemset is composed of a frequent $(k - 1)$-itemset and a frequent 1-itemset.

- However, it doesn't prevent the same candidate itemset from being generated more than once.

E.g., {Bread, Diapers, Milk} can be generated by merging

- {Bread, Diapers} with {Milk},
- {Bread, Milk} with {Diapers}, or
- {Diapers, Milk} with {Bread}.

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 1-itemset

Item
Beer
Bread
Diapers
Milk

Lexicographic Order

- Avoid generating duplicate candidates by ensuring that the items in each frequent itemset are kept sorted in lexicographic order.
- Each frequent (k-1)-itemset X is then extended with frequent items that are lexicographically larger than the items in X.
- For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers.
- However, we don’t augment {Diapers, Milk} with {Bread} nor {Bread, Milk} with {Diapers} because they violate the lexicographic ordering condition.

Lexicographic Order - Completeness

- Is it complete?

Let $(i_1,\ldots, i_{k-1}, i_k)$ be a frequent k-itemset sorted in lexicographic order.

Since it is frequent, by the Apriori principle, $(i_1,\ldots, i_{k-1})$ and (i_k) must be frequent as well.

$(i_1,\ldots, i_{k-1}) \in F_{k-1}$ and $(i_k) \in \mathrm{F}_1$.

Since, (i_k) is lexicographically bigger than $i_1,\ldots, i_{k-1}$, we have that $(i_1,\ldots, i_{k-1})$ would be joined with (i_k) for giving $(i_1,\ldots, i_{k-1}, i_k)$ as a candidate k-itemset.

Still too many candidates…

- E.g. merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent.

- For a candidate *k*-itemset to be worthy counting,
  - every item in the candidate must be contained in at least *k*-1 of the frequent (*k*-1)-itemsets.
  - {Beer, Diapers, Milk} is a viable candidate 3-itemset only if every item in the candidate, including Beer, is contained in at least 2 frequent 2itemsets.

    Since there is only one frequent 2-itemset containing Beer, all candidate 3-itemsets involving Beer must be infrequent.
- Why?

    Because each of *k*-1-subsets containing an item must be frequent.

$F_{k\text{-}1} \times F_1$

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 1-itemset

Item
Beer
Bread
Diapers
Milk

Candidate Generation

Itemset
{Beer, Diapers, Bread}
{Beer, Diapers, Milk}
{Bread, Diapers, Milk}
{Bread, Milk, Beer}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

$F_{k-1} \times F_{k-1}$ Method

- Merge a pair of frequent (k-1)-itemsets only if their first k-2 items are identical.
  - E.g. frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}.
  - We don't merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different.
  - Indeed, if {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, Milk} instead.

- This illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic ordering to prevent duplicate candidates.

Pruning?

- Because each candidate is obtained by merging a pair of frequent (k-1) -itemsets, an additional candidate pruning step is needed to ensure that the remaining k-2 subsets of k-1 elements are frequent.

$F_{k-1} \times F_{k-1}$

Frequent
2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent
2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Candidate
Generation

Itemset
{Bread, Diapers, Milk}

Candidate
Pruning

Itemset
{Bread, Diapers, Milk}

Toy Example

Find all frequent itemsets from the following data.
Min support count = 2

Pizza toppings dataset

TID	Extra cheese	Onions	Peppers	Mushrooms	Olives	Anchovy
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		

Binary data format

2. Count 1-item frequent itemsets

TID	A	B	C	D	E	F
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		
σ	4	4	1	4	2	1

Support count

Frequent 1-itemsets: {A}, {B}, {D}, {E}

3. Generate candidate 2-itemsets

	A	B	D	E
A				
B				
D				
E				

Candidate 2-itemsets C_2
{A,B} {A,D} {A,E}
{B,D} {B,E}
{D,E}

4. Scan DB, count candidates

TID	A	B	C	D	E	F
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		

	A	B	D	E
A		3	3	2
B			2	2
D				1
E				

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}
~~{D,E}~~

2 ways of candidate generation

a) $C_k=F_k \times F_1$
b) $C_k=F_{k-1} \times F_{k-1}$

In both cases itemsets are lexicographically sorted: we may extend existing itemset only with an item which is lexicographically largest among all items in F_{k-1}

5a. Generate $C_3=F_2xF_1$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2 \backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

5a. Generate $C_3 = F_2 x F_1$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2 \backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} {A,D,E} {B,D,E}

5a. Prune C_3 before counting

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2 \backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} ~~{A,D,E} {B,D,E}~~

5b. Generate $C_3=F_2xF_2$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

The first item should be identical in order to join

$F_2\backslash F_2$	A,B	A,D	A,E	B,D	B,E
A,B					
A,D					
A,E					
B,D					
B,E					

5b. Prune C_3 before counting

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

The first item should be identical in order to join

$F_2 \backslash F_2$	A,B	A,D	A,E	B,D	B,E
A,B					
A,D					
A,E					
B,D					
B,E					

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} ~~{A,D,E} {B,D,E}~~

6. Count candidates C_3

TID	A	B	C	D	E	F
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		

$F_2 \backslash F_1$	A	B	D	E
A,B			2	2
A,D				
A,E				
B,D				
B,E				

Frequent 3-itemsets F_3
{A,B,D} {A,B,E}

7a. Generate candidates $C_4=F_3xF_1$

$F_3\backslash F_1$	A	B	D	E
A,B,D				
A,B,E				

The only candidate 4-itemset:
{A,B,D,E}
Do we need to count its support?
Can it be pruned?

Solution: all frequent *k*-itemsets, *k*>=2

- {A,B} {A,D} {A,E} {B,D} {B,E}
- {A,B,D} {A,B,E}

A Extra cheese	B Onions	C Peppers	D Mushrooms	E Olives	F Anchovy

- {Cheese, Onions} {Cheese, Mushrooms} {Cheese, Olives} {Onions, Mushrooms} {Onions, Olives}
- {Cheese, Onions, Mushrooms} {Cheese, Onions, Olives}

Larger Example

2. Count 1-item frequent itemsets

Min_sup_count = 2

TID	List of item ID’s
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

C1

Itemset
{I1}
{I2}
{I3}
{I4}
{I5}

F1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

3. Generate C2 from F1×F1

Min_sup_count = 2

TID	List of item ID's
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

F1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C2

Itemset
{I1,I2}
{I1,I3}
{I1,I4}
{I1,I5}
{I2,I3}
{I2,I4}
{I2,I5}
{I3,I4}
{I3,I5}
{I4,I5}

Itemset	Sup. C
{I1,I2}	4
{I1,I3}	4
{I1,I4}	1
{I1,I5}	2
{I2,I3}	4
{I2,I4}	2
{I2,I5}	2
{I3,I4}	0
{I3,I5}	1
{I4,I5}	0

4. Generate C3 from F2×F2

Min_sup_count = 2

TID	List of item ID’s
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

F2

Itemset	Sup. C
{I1,I2}	4
{I1,I3}	4
{I1,I5}	2
{I2,I3}	4
{I2,I4}	2
{I2,I5}	2

Itemset
{I1,I2,I3}
{I1,I2,I5}
{I1,I3,I5}
{I2,I3,I4}
{I2,I3,I5}
{I2,I4,I5}

Prune

Itemset
{I1,I2,I3}
{I1,I2,I5}
~~{I1,I3,I5}~~
~~{I2,I3,I4}~~
~~{I2,I3,I5}~~
~~{I2,I4,I5}~~

F3

Itemset	Sup. C
{I1,I2,I3}	2
{I1,I2,I5}	2

5. Generate C4 from F3×F3

Min_sup_count = 2

TID	List of item ID's
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

F3

Itemset	Sup. C
{I1,I2,I3}	2
{I1,I2,I5}	2

C4

Itemset	Sup. C
{I1,I2,I3,I5}	2

{I1,I2,I3,I5} is pruned because {I2,I3,I5} is infrequent

Apriori Algorithm. Summary

Generate F_1

Let $k=1$

Repeat until F_k is empty

$k=k+1$

Generate C_k from F_{k-1}

Prune C_k containing subsets that are not in F_{k-1}

Count support of each candidate in C_k by scanning DB

Eliminate infrequent candidates, leaving F_k

Generating and pruning this way reduces the number of candidates to be counted against the dataset

Improving performance with

Algorithms and Data Structures

Alternative to Apriori algorithm: FP-growth

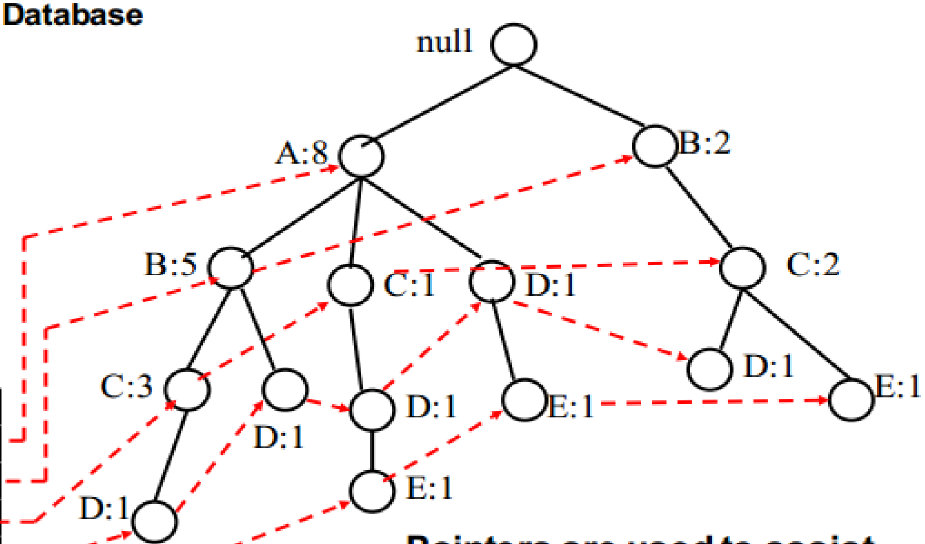
FP-Tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{A}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Header table

Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

Good explanation and code in Python: [Link]

Candidate support counting

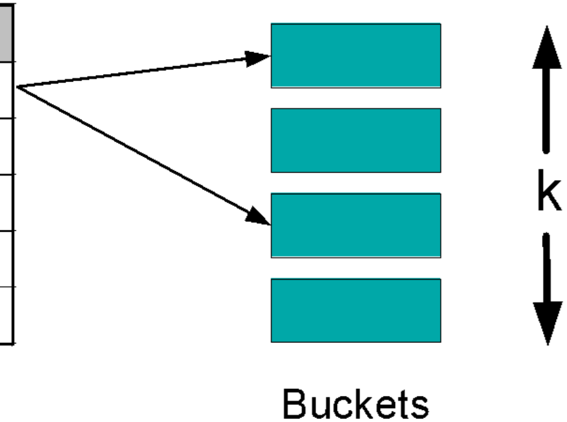
- Scan the database of transactions to determine the support of each candidate itemset
- **Brute force**: Match each transaction against every candidate.
  - Too many comparisons!
- **Better method**: Store the candidate itemsets in a hash structure
  - A transaction will be tested for match only against candidates contained in a few buckets

Transactions

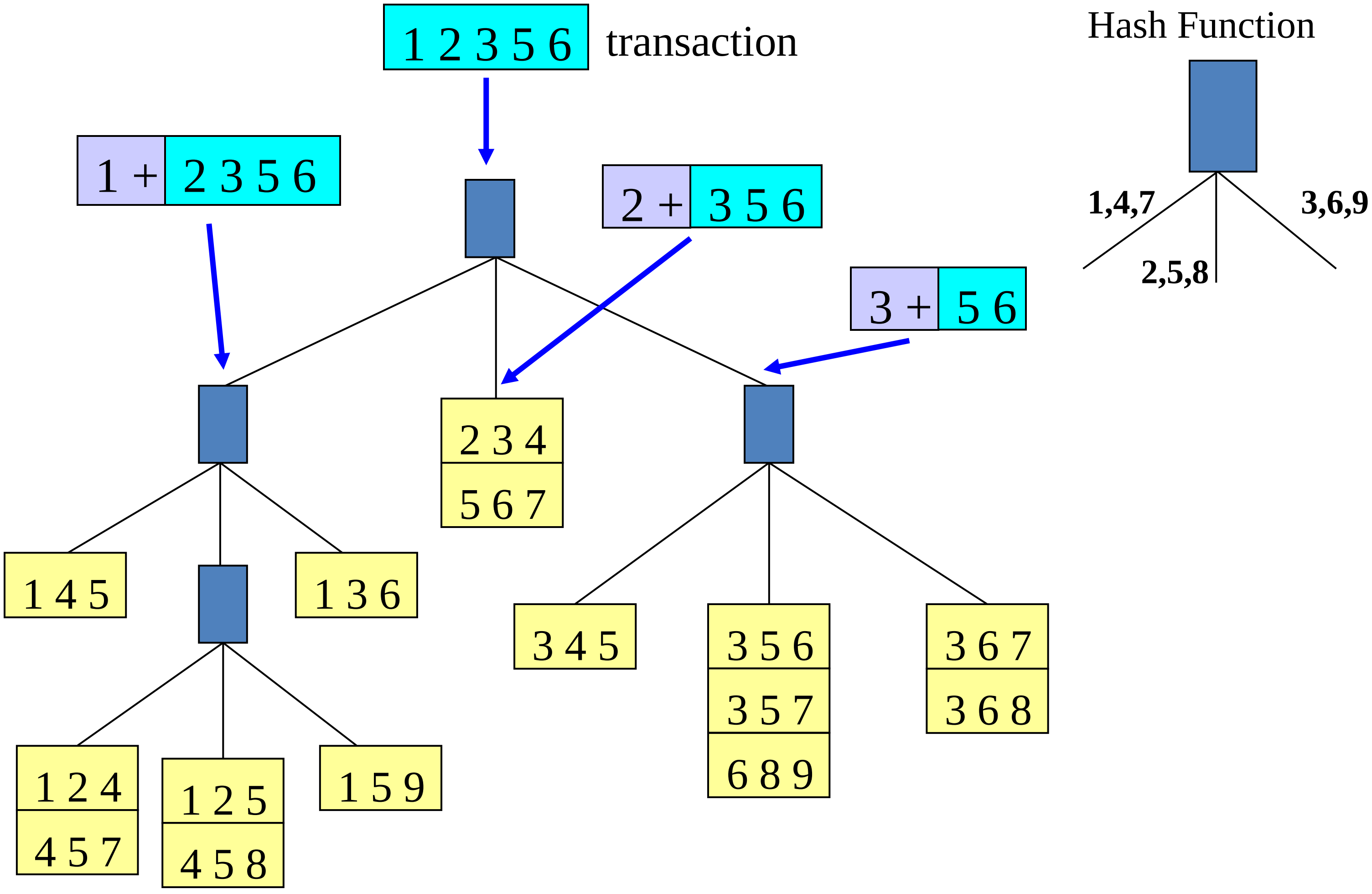
N

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



Hash tree: to make counting of candidates faster



Generate Hash Tree

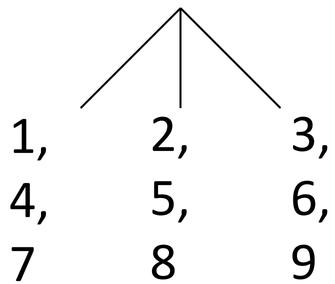
Suppose you have 15 candidate itemsets of length 3:

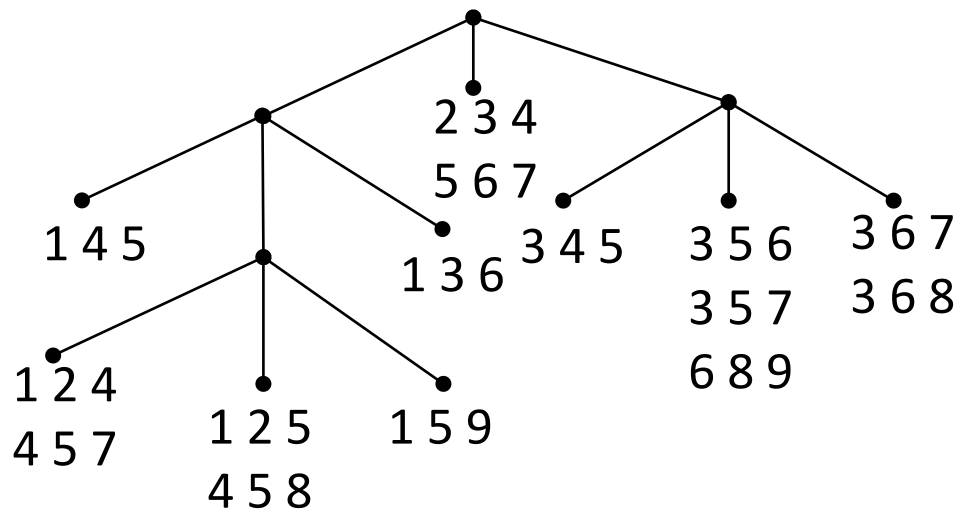
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

- A hash function (e.g. p mod 3)
- Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

Hash function



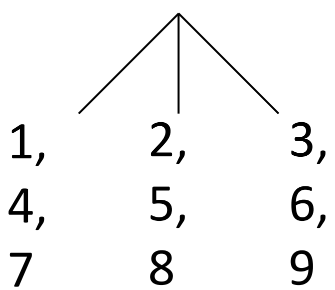


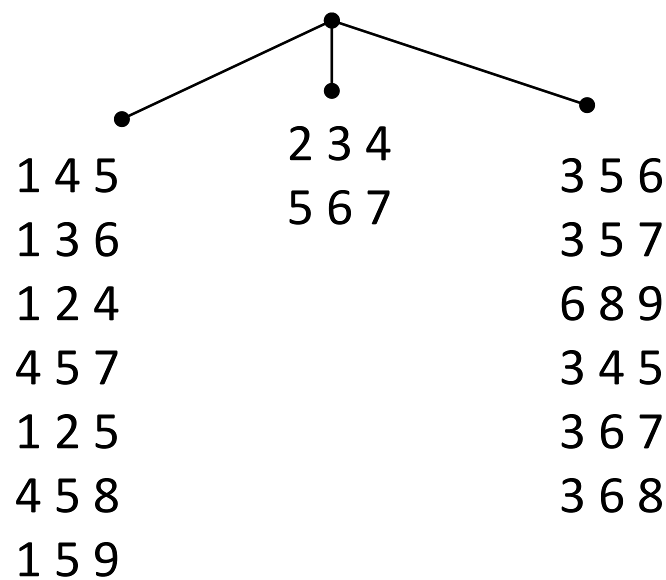
Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

Hash function



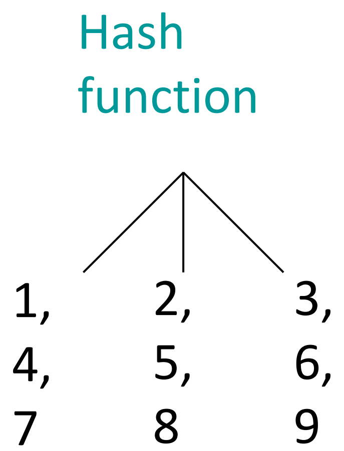


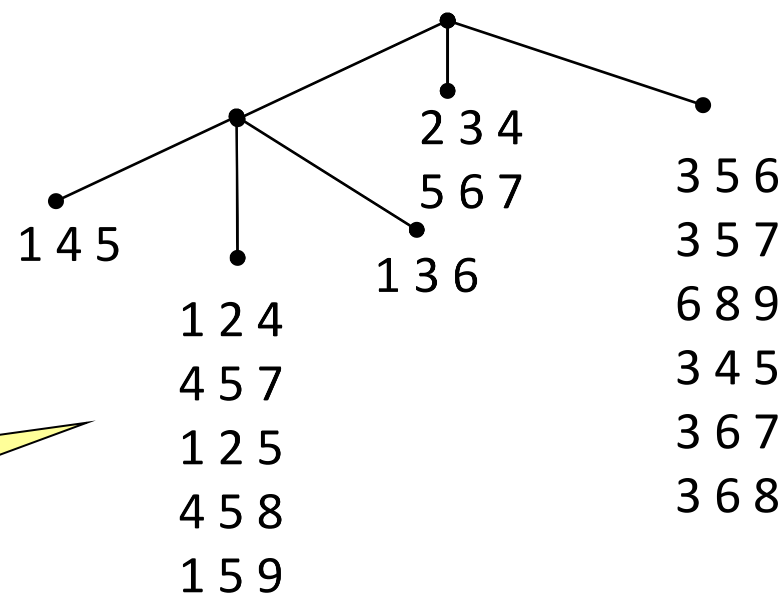
Split nodes with more than 3 candidates
using the second item and the same hash function

Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}





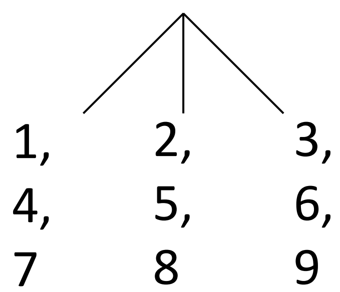
Now split nodes using the third item

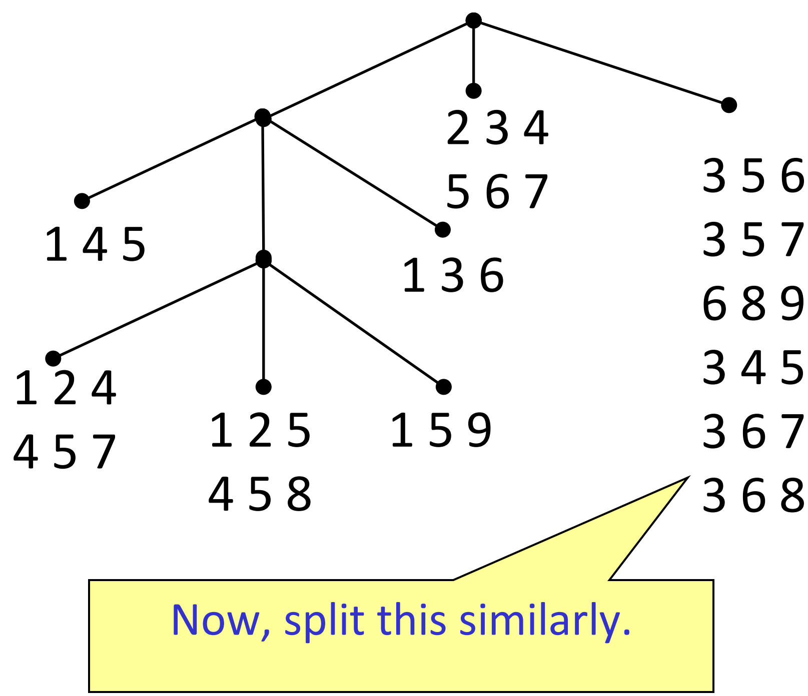
Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

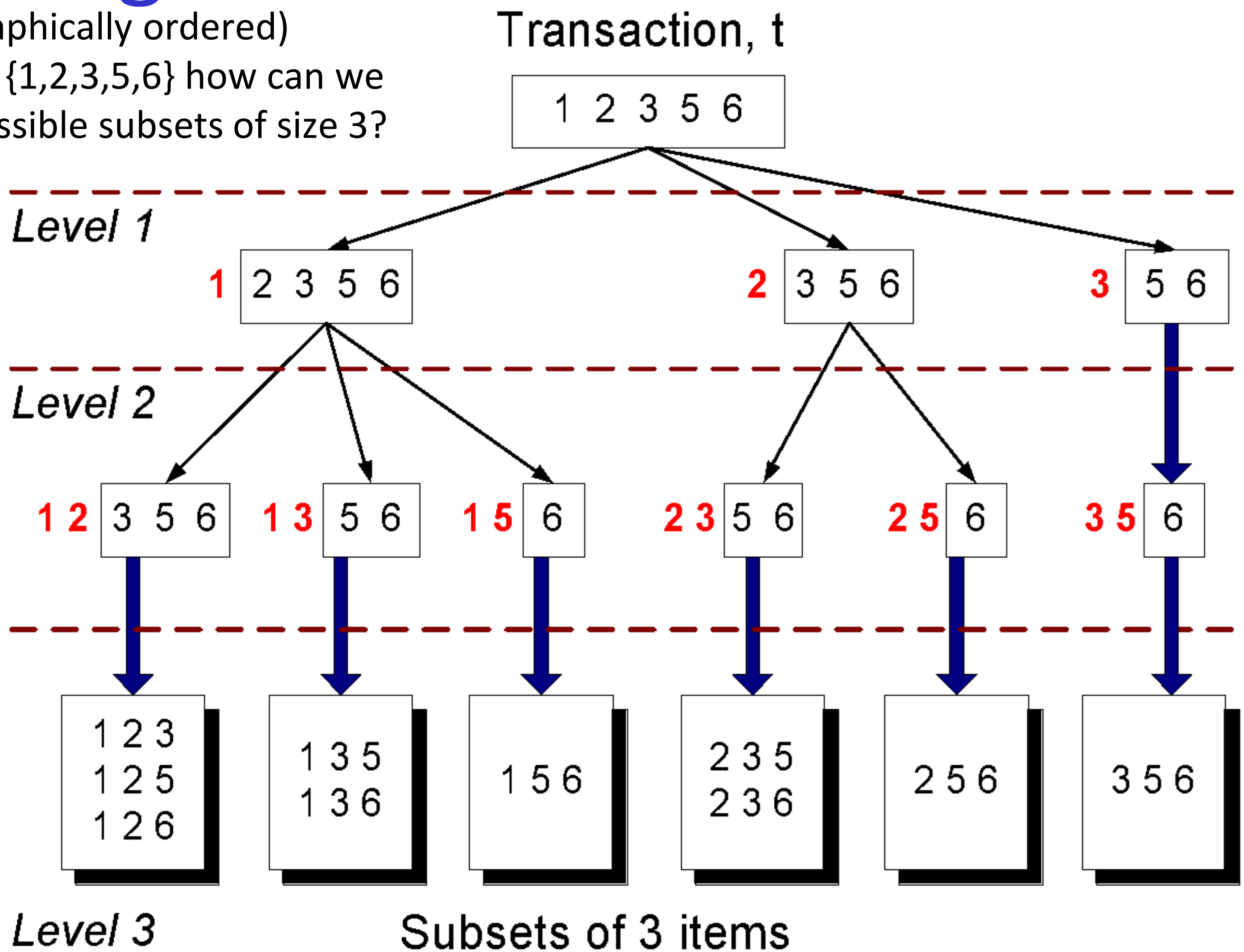
Hash function



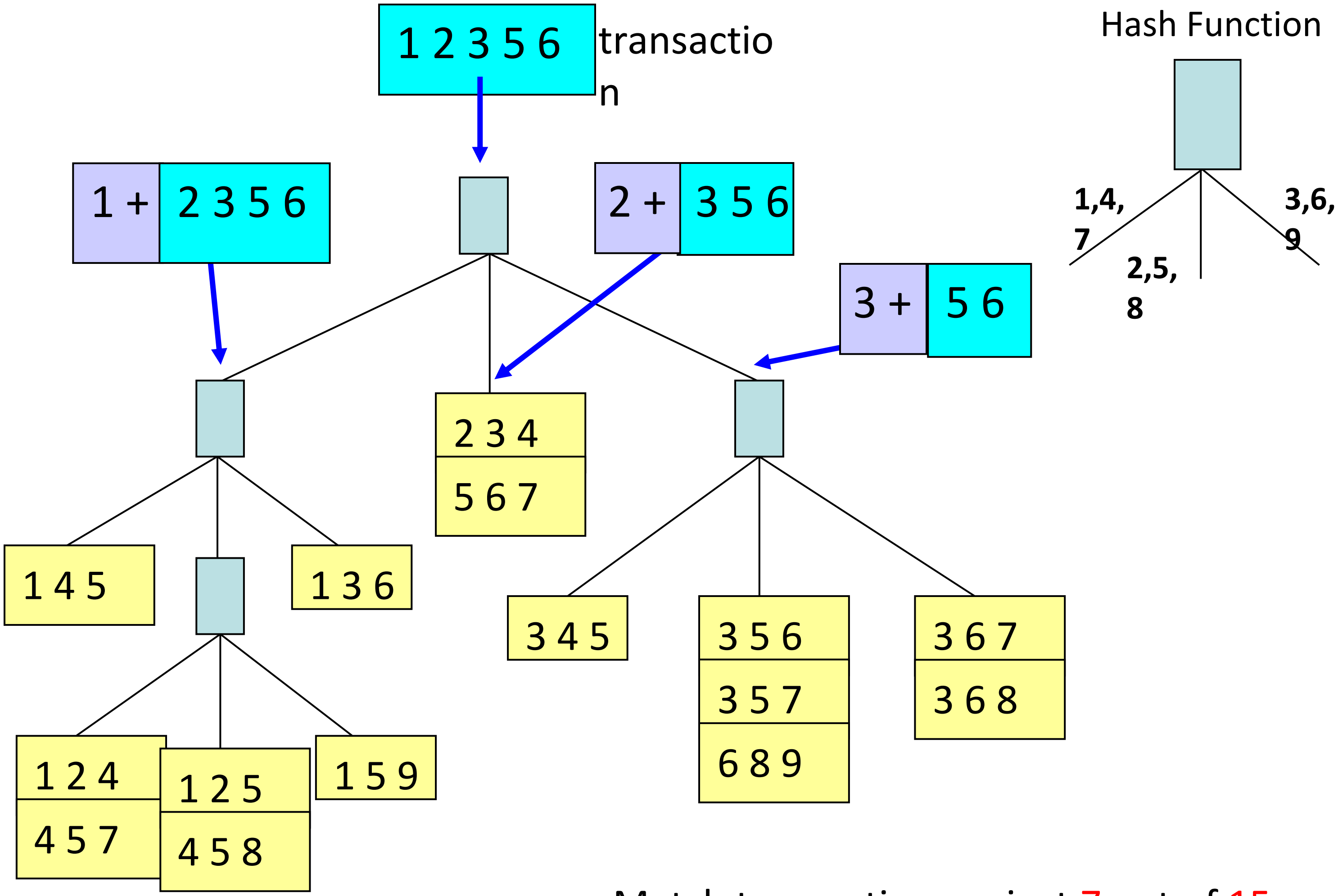


Enumerating all subsets of a given transaction

Given a (lexicographically ordered) transaction t, say {1,2,3,5,6} how can we enumerate all possible subsets of size 3?



Subset counting using Hash Tree



Match transaction against 7 out of 15 candidates

Step 2

RULE GENERATION

Rule Generation

- An association rule can be extracted by a binary partitioning of a frequent itemset Y into two nonempty subsets, X and $Y - X$, such that

$$X \rightarrow Y - X$$

  satisfies the confidence threshold.

- Each frequent k-itemset, Y, can produce up to $2^k - 2$ association rules
  - ignoring rules that have empty antecedents or consequents.

Rule Generation

Example

Let Y = {1, 2, 3} be a frequent itemset.

Six candidate association rules can be generated from Y:

{1, 2} → {3},
{1, 3} → {2},
{2, 3} → {1},
{1} → {2, 3},
{2} → {1, 3},
{3} → {1, 2}.

Computing the confidence of an association rule does not require additional scans of the database.

Consider {1, 2} → {3}.

The confidence is **σ ({1, 2, 3}) / σ ({1, 2})**

Because {1, 2, 3} is frequent, the antimonotone property of support ensures that {1, 2} must be frequent, too, and we preserve the supports of all frequent itemsets.

Confidence, unlike support is not anti-monotone:

Knowing that $c(X \rightarrow Y) < minConfidence$, we cannot tell whether $c(X' \rightarrow Y') < minConfidence$ or $c(X' \rightarrow Y') > minConfidence$, for $X' \subseteq X$ and $Y' \subseteq Y$

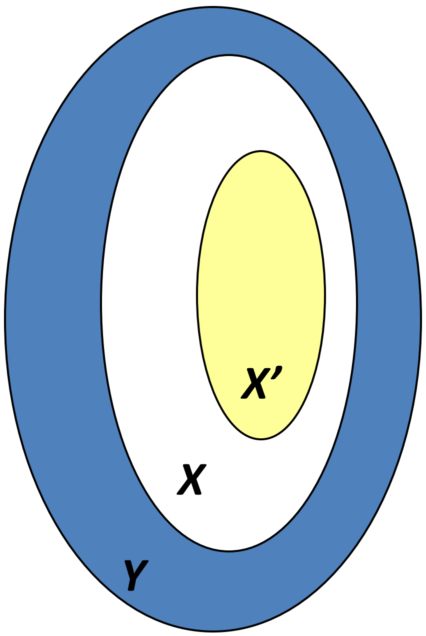
Do we need to compute confidence for all possible rules for each frequent itemset Y?

Confidence-based rule pruning

Theorem.

If a rule $X \rightarrow Y - X$ does not satisfy the confidence threshold,

then any rule $X' \rightarrow Y - X'$, where X' is a subset of X, cannot satisfy the confidence threshold as well.



Confidence-based rule pruning

Proof.

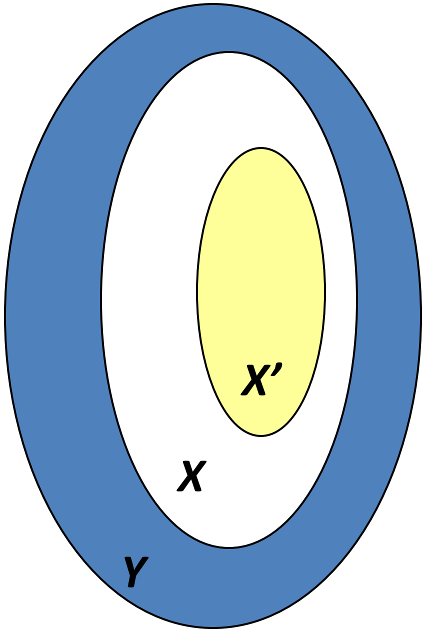
Consider the following two rules:

$X' \rightarrow Y - X'$ and $X \rightarrow Y - X$, where $X' \subseteq X$.

The confidence of the rules are $\sigma(Y) / \sigma(X')$ and $\sigma(Y) / \sigma(X)$, respectively.

Since X' is a subset of X, $\sigma(X') \geq \sigma(X)$.

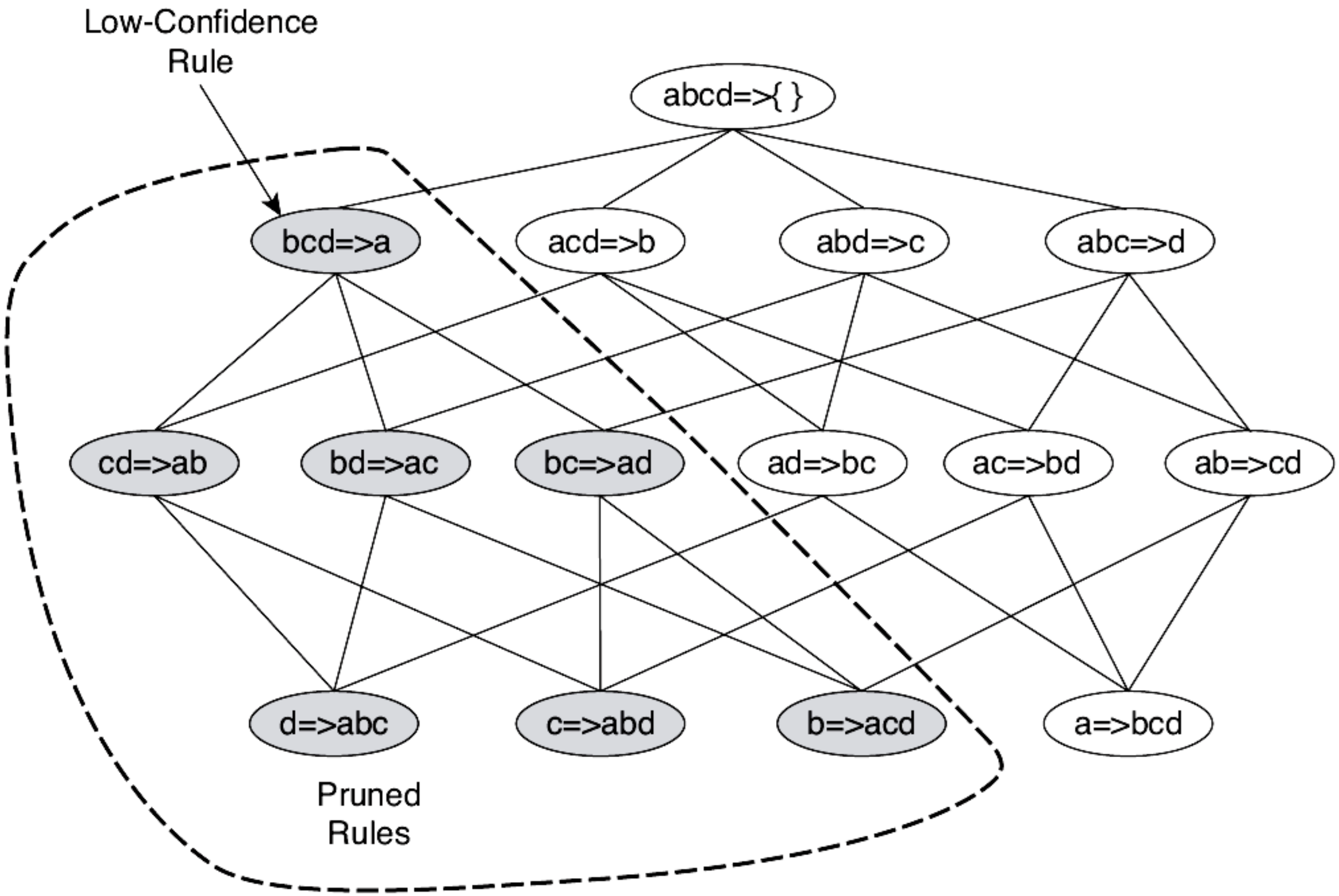
Therefore, the former rule cannot have a higher confidence than the latter rule.

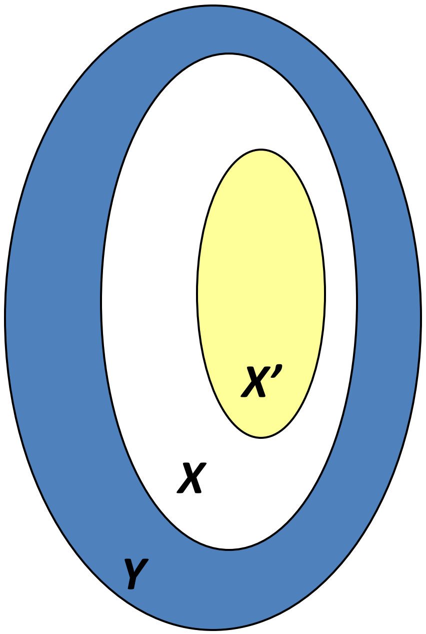


Confidence-Based Pruning

- Observe that:

  $X' \subseteq X$ implies that $Y - X' \supseteq Y - X$





Algorithm for rule generation

- Initially, all the high-confidence rules that have only one item in the rule consequent are extracted: $Y - X_1 \rightarrow Y$
- These rules are then used to generate new candidate rules.
- For example, if
  - $\{acd\} \rightarrow \{b\}$ and $\{abd\} \rightarrow \{c\}$ are high-confidence rules, then the candidate rule $\{ad\} \rightarrow \{bc\}$ is generated by merging the consequents of both rules.

Example: 1/2

Items (1-itemsets)

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

Pairs (2-itemsets)

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

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

High-confidence rules with 1 item in consequent

{Bread,Milk} →{Diaper} (confidence = 3/3) threshold=50%

{Bread,Diaper} →{Milk} (confidence = 3/3)

{Diaper,Milk} →{Bread} (confidence = 3/3)

Example: 2/2

Merge only if high-confidence:
{Bread,Milk} →{Diaper} (confidence = 3/3)
{Bread,Diaper} →{Milk} (confidence = 3/3)

{Bread} →{Diaper,Milk} (confidence = 3/4)

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

Itemset	Count
{Bread,Milk,Diaper}	3

Rule confidence:

c({Bread} →{Diaper, Milk}) = σ ({Bread, Diaper, Milk}) / σ({Bread})