

# COMP8400: Algorithms and Techniques for Data Mining

## Introduction to association mining

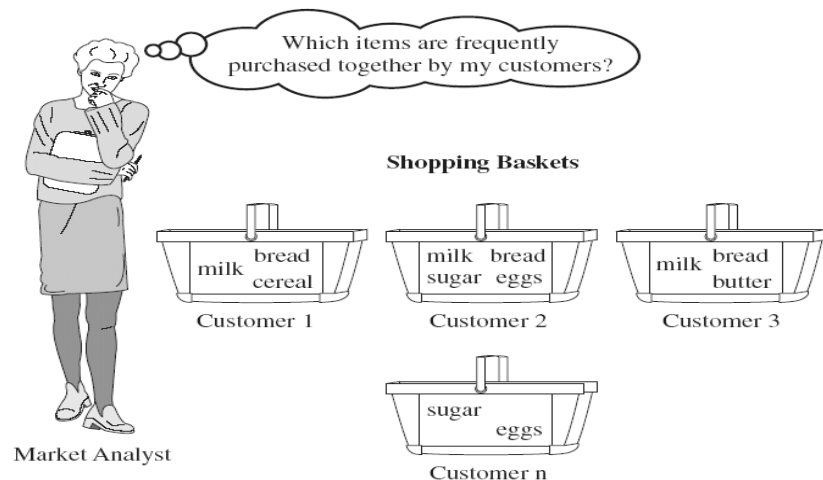
## Lecture outline

- What is association mining?
- Market basket analysis and association rules examples
- Basic concepts and formalism
- Basic rule measurements
- The *Apriori* algorithm
- Performance bottlenecks in *Apriori*
- Improve *Apriori's* efficiency

## What is association mining?

- Association mining is the task of finding frequent rules / associations / patterns / correlations / causal structures within (large) sets of items in transactional (relational) databases
- *Unsupervised* learning techniques (*descriptive* data mining, not *predictive* data mining)
- The main applications are
  - Market basket analysis (customers who buys X also buys Y)
  - Web log analysis (click-stream)
  - Cross-marketing
  - Sale campaign analysis
  - DNS sequence analysis

## Market basket analysis



## Association rules examples

- Rules form:  $body \Rightarrow head$  [support, confidence]
- Market basket:  
 $buys(X, \text{'beer'}) \Rightarrow buys(X, \text{'snacks'})$  [1%, 60%]
  - If a customer  $X$  purchased 'beer', in 60% she or he also purchased 'snacks'
  - 1% of all transactions contain the items 'beer' and 'snacks'
- Student grades:  
 $major(X, \text{'InfTech'}) \text{ and } takes(X, \text{'COMP3420'}) \Rightarrow grade(X, \text{'D'})$  [3%, 60%]
  - If a student  $X$ , who's major is 'InfTech', took the course 'COMP3420' she or he in 60% achieved a grade 'D'
  - The combination 'InfTech', 'COMP3420' and 'D' appears in 3% of all transactions (records) in the database

## Formalism

- Set of items  $X = \{x_1, x_2, \dots, x_k\}$
- Database  $D$  containing transactions
- Each transaction  $T$  is a set of items, such that  $T$  is a subset of  $X$
- Each transaction is associated with a unique identifier, called  $TID$  (for example, a unique number)
- Let  $A$  be a set of items (a subset of  $X$ )
- An association rule is an implication of the form  $A \Rightarrow B$ , where  $A$  is a subset of  $X$  and  $B$  is a subset of  $X$ , and the intersection of  $A$  and  $B$  is empty
  - No item in  $A$  can be in  $B$ , and vice versa
  - No rule of the form:  $\{\text{'beer'}, \text{'chips'}\} \Rightarrow \{\text{'chips'}, \text{'peanuts'}\}$

## Basic concepts

- Given:
  - A (large) database of transactions
  - Each transaction contains a list of one or more items (e.g. purchased by a customer in a visit)
- Find the rules that correlate the presence of one set of items with that of another set of items
- Normally one is only interested in rules that are *frequent*
  - For example, 70% of customers who buy tires and car accessories also get their car service done  
Question: How can this be improved to 80%? Possibly offer special deals like a 15% reduction of tire costs when the service is done

## Basic rule measurements

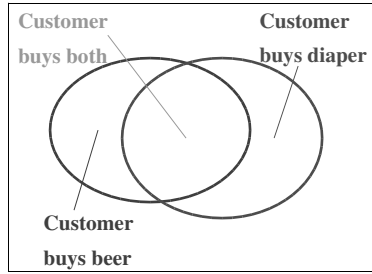
- A rule  $A \Rightarrow B$  holds in a database  $D$  with *support*  $s$ , with  $s$  being the percentage of transactions in  $D$  that contain  $A$  and  $B$
- The rule  $A \Rightarrow B$  has a *confidence*  $c$  in a database  $D$  if  $c$  is the percentage of transactions in  $D$  containing  $A$  that also contain  $B$

$$\text{support}(A \Rightarrow B) = P(A \cup B)$$

$$\text{confidence}(A \Rightarrow B) = P(B|A) = P(A \cup B) / P(A)$$

$$\text{confidence}(A \Rightarrow B) = \text{support}(A \Rightarrow B) / \text{support}(A)$$

## Rule measurements example



- Find all the rules  $\{X, Y\} \Rightarrow Z$  with minimum confidence and support
- Support,  $s$ , is the probability that a transaction contains  $\{X, Y, Z\}$
- Confidence,  $c$ , is the conditional probability that a transaction having  $\{X, Y\}$  also contains  $Z$

Transaction ID	Items Bought
2000	a, b, c
1000	a, c
4000	a, d
5000	b, e, f

Let minimum support = 50%, and minimum confidence = 50%, so we have ( $[s, c]$ ):

- $a \Rightarrow c$  [50%, 66.67%]
- $c \Rightarrow a$  [50%, 100%]

Source: Han and Kamber, DM Book, 1st Ed.

## Rule measurements example (2)

Transaction ID	Items Bought
2000	a, b, c
1000	a, c
4000	a, d
5000	b, e, f

Itemset	Support
a	75.00%
b	50.00%
c	50.00%
a, c	50.00%

- Minimum support = 50% and confidence = 50%
- Rule  $a \Rightarrow c$ 
  - support ( $a \Rightarrow c$ ): 50%
  - confidence ( $a \Rightarrow c$ ) =  $\text{support}(a \Rightarrow c) / \text{support}(a) = 50\% / 75\% = 66.67\%$

## Mining frequent item sets

- Key step: Find the *frequent sets of items* that have *minimum support* (appear in at least xx% of all transactions in a database)
- Basic principle (*Apriori* principle): A sub-set of a frequent item set must also be a frequent item set
  - For example, if  $\{a,b\}$  is frequent, both  $\{a\}$  and  $\{b\}$  have to be frequent (if 'beer' and 'chips' are purchased frequently together, then 'beer' is purchased frequently and 'chips' are also purchased frequently)
- Basic approach: Iteratively find frequent item sets with cardinality from 1 to  $k$  ( $k$ -item sets),  $k > 1$
- Use the frequent item sets to generate association rules
  - For example, frequent 3-item set  $\{a,b,c\}$  contains rules:  $a \Rightarrow c$ ,  $b \Rightarrow c$ ,  $a \Rightarrow b$ ,  $\{a,b\} \Rightarrow c$ ,  $\{a,c\} \Rightarrow b$ ,  $\{b,c\} \Rightarrow a$ , etc.
- We are normally only interested in longer rules

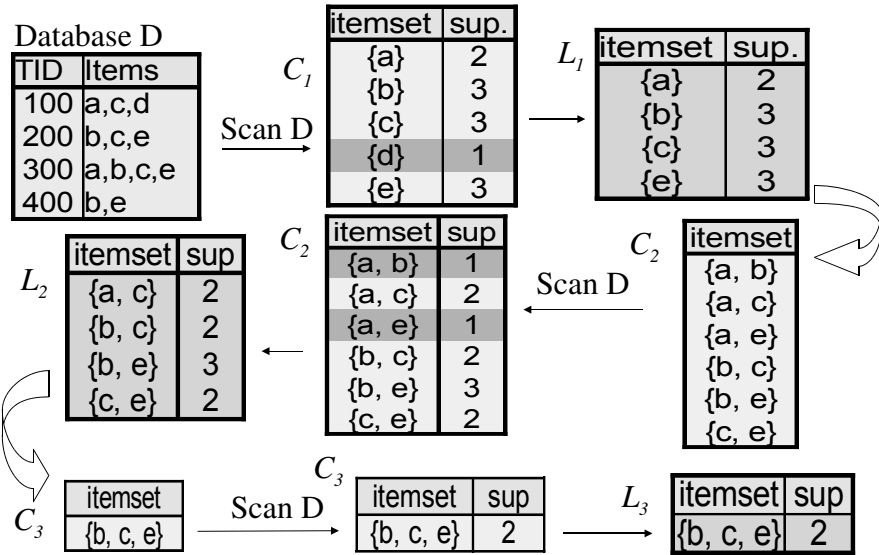
## The *Apriori* algorithm (Agrawal & Srikant, VLDB'94)

- $C_k$ : Candidate item set of size  $k$
- $L_k$ : Frequent item set of size  $k$
- Pseudo-code:

```

 $L_1 = \{\text{frequent items}\};$ 
for ( $k = 1; L_k \neq \{\}; k++$ ) do begin
     $C_{k+1} = \text{candidates generated from } L_k;$ 
    for each transaction  $t$  in database do
        increment the count of all candidates in  $C_{k+1}$ 
        that are contained in  $t$ 
     $L_{k+1} = \text{candidates in } C_{k+1} \text{ with min\_support}$ 
    end do
return  $\bigcup_k L_k;$ 
    
```

## The *Apriori* algorithm – An example (sup=50%)



## The *Apriori* algorithm – An example (2)

Database D

TID	Items
100	a,c,d
200	b,c,e
300	a,b,c,e
400	b,e

$L_3$

itemset	sup.
{b, c, e}	2

- Minimum support = 50% and minimum confidence = 50%
- Rules:
  - $b \Rightarrow c$  [50%, 66.66%]
  - $b \Rightarrow e$  [75%, 100%]
  - $c \Rightarrow e$  [50%, 66.66%]
  - $\{b, c\} \Rightarrow e$  [50%, 100%]
  - $\{b, e\} \Rightarrow c$  [50%, 66.66%]
  - $\{c, e\} \Rightarrow b$  [50%, 100%]

## Important details of the *Apriori* algorithm

- How to generate candidate sets?
  - Step 1: Self-joining  $L_k$  ( $C_k$  is generated by joining  $L_{k-1}$  with itself)
  - Step 2: Pruning (any  $(k-1)$ -item set that is not frequent cannot be a subset of a frequent  $k$ -item set)
- Example of candidate generation:
  - $L_3 = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{a,c,e\}, \{b,c,d\}\}$
  - Self-joining:  $L_3 * L_3$  ( $\{a,b,c,d\}$  from  $\{a,b,c\}$  and  $\{a,b,d\}$ , and  $\{a,c,d,e\}$  from  $\{a,c,d\}$  and  $\{a,c,e\}$ )
  - Pruning:  $\{a,c,d,e\}$  is removed because  $\{a,d,e\}$  is not in  $L_3$
  - $C_4 = \{\{a,b,c,d\}\}$
- How to count supports for candidates?

## How to generate candidate item-sets?

- Suppose the items in  $L_{k-1}$  are listed in an order (e.g.  $a < b$ )
- Step 1: Self-joining  $L_{k-1}$ 

```

insert into  $C_k$ 
select p.item1, p.item2, ..., p.itemk-1, q.itemk-1
from  $L_{k-1}$  p,  $L_{k-1}$  q
where p.item1 = q.item1, ..., p.itemk-2 = q.itemk-2, p.itemk-1 < q.itemk-1
            
```
- Step 2: Pruning
 

```

forall item sets c in  $C_k$  do
    forall (k-1)-sub-sets s of c do
        if (s is not in  $L_{k-1}$ ) then delete c from  $C_k$ 
            
```

## *Apriori* performance bottlenecks

- The core of the *Apriori* algorithm is to
  - Use frequent  $(k-1)$  item sets to generate candidate frequent  $k$  item sets
  - Use database scan and pattern matching to collect counts for candidate item sets
- Candidate generation is the main bottleneck
  - $10^4$  frequent 1-item sets (sets of length 1) will generate  $10^7$  candidate 2-item sets!
  - To discover a frequent pattern of size 100 (for example  $\{a_1, a_2, \dots, a_{100}\}$ ) one needs to generate  $2^{100} = 10^{30}$  candidates
  - Multiple scans of the database are needed ( $n+1$  scans if the longest pattern is  $n$  items long)

## Methods to improve *Apriori's* efficiency

- Reduce the number of scans of the database
  - Any item set that is potentially frequent in the database must be frequent in at least one of the partitions of the database
  - Scan 1: Partition database and find local frequent patterns
  - Scan 2: Consolidate global frequent patterns
- Shrink number of candidates
  - Select a sample of the database, mine frequent patterns within sample using *Apriori*
  - Scan database once to verify frequent item sets found in sample
  - Scan database again to find missed frequent patterns
- Facilitate support of counting candidates
  - For example, use special data structures like Frequent-Pattern tree (FP-tree)