# CS 412 Intro. to Data Mining

## Chapter 6. Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

Jiawei Han, Computer Science, Univ. Illinois at Urbana-Champaign, 2017

# Chapter 6: Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

- Basic Concepts 
- Efficient Pattern Mining Methods
- Pattern Evaluation
- Summary

# Pattern Discovery: Basic Concepts

- What Is Pattern Discovery? Why Is It Important?
- Basic Concepts: Frequent Patterns and Association Rules
- Compressed Representation: Closed Patterns and Max-Patterns

# What Is Pattern Discovery?

- What are patterns?
  - Patterns: A set of items, subsequences, or substructures that occur frequently together (or strongly correlated) in a data set
  - Patterns represent intrinsic and important properties of datasets
- Pattern discovery: Uncovering patterns from massive data sets
- Motivation examples:
  - What products were often purchased together?
  - What are the subsequent purchases after buying an iPad?
  - What code segments likely contain copy-and-paste bugs?
  - What word sequences likely form phrases in this corpus?

↳ สามารถค้นหาแล้วเจอ!

# Pattern Discovery: Why Is It Important?

- Finding inherent regularities in a data set
- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Mining sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: Discriminative pattern-based analysis
  - Cluster analysis: Pattern-based subspace clustering
- Broad applications
  - Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log analysis, biological sequence analysis

# Basic Concepts: k-Itemsets and Their Supports

- Itemset: A set of one or more items
- k-itemset: $X = \{x_1, \ldots, x_k\}$
  - Ex. {Beer, Nuts, Diaper} is a 3-itemset
- (*absolute*) *support* (*count*) of X, sup{X}:

  Frequency or the number of occurrences of an itemset X
  - Ex. sup{Beer} = 3
  - Ex. sup{Diaper} = 4
  - Ex. sup{Beer, Diaper} = 3
  - Ex. sup{Beer, Eggs} = 1

| Tid | Items bought |
|---|---|
| 10 | Beer, Nuts, Diaper |
| 20 | Beer, Coffee, Diaper |
| 30 | Beer, Diaper, Eggs |
| 40 | Nuts, Eggs, Milk |
| 50 | Nuts, Coffee, Diaper, Eggs, Milk |

- (*relative*) *support*, *s*{*X*}: The fraction of transactions that contains X (i.e., the probability that a transaction contains X)
  - Ex. s{Beer} = 3/5 = 60%
  - Ex. s{Diaper} = 4/5 = 80%
  - Ex. s{Beer, Eggs} = 1/5 = 20%

# Basic Concepts: Frequent Itemsets (Patterns)

- An itemset (or a pattern) X is *frequent* if the support of X is no less than a *minsup* threshold σ
- Let σ = *50%* (σ: *minsup* threshold)

For the given 5-transaction dataset

- All the frequent 1-itemsets:
  - Beer: 3/5 (60%); Nuts: 3/5 (60%)
  - Diaper: 4/5 (80%); Eggs: 3/5 (60%)
- All the frequent 2-itemsets:
  - {Beer, Diaper}: 3/5 (60%)
- All the frequent 3-itemsets?
  - None

Coffee : 2/5 (40%) 가능 threshold #

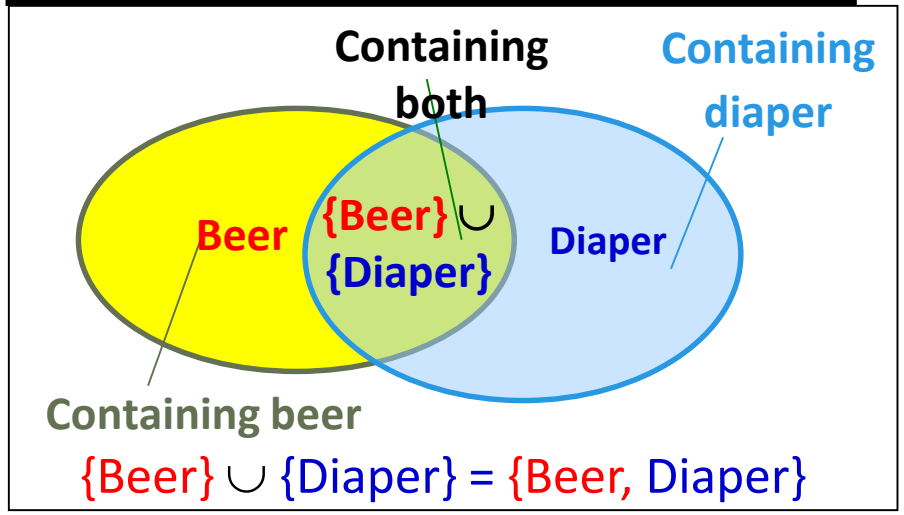
| Tid | Items bought |
|---|---|
| 10 | Beer, Nuts, Diaper |
| 20 | Beer, Coffee, Diaper |
| 30 | Beer, Diaper, Eggs |
| 40 | Nuts, Eggs, Milk |
| 50 | Nuts, Coffee, Diaper, Eggs, Milk |

- Why do these itemsets (shown on the left) form the complete set of frequent *k*-itemsets (patterns) for any *k*?
- **Observation**: We may need an efficient method to mine a complete set of frequent patterns

# From Frequent Itemsets to Association Rules

- Comparing with itemsets, rules can be more telling
  - Ex. *Diaper → Beer* ซื้อ Diaper แล้วจะไปซื้อ beer ต่อ
    - *Buying diapers may likely lead to buying beers*
- How strong is this rule? (support, confidence)
  - Measuring association rules: *X → Y* (s, c)
    - Both *X* and *Y* are itemsets
  - Support, *s*: The probability that a transaction contains X ∪ Y
    - Ex. s{Diaper, Beer} = 3/5 = 0.6 (i.e., 60%)
  - Confidence, *c: The conditional probability* that a transaction containing X also contains *Y*
    - Calculation: *c* = sup(X ∪ Y) / sup(X)
    - Ex. *c* = sup{Diaper, Beer}/sup{Diaper} = ¾ = 0.75

| Tid | Items bought |
|---|---|
| 10 | Beer, Nuts, Diaper |
| 20 | Beer, Coffee, Diaper |
| 30 | Beer, Diaper, Eggs |
| 40 | Nuts, Eggs, Milk |
| 50 | Nuts, Coffee, Diaper, Eggs, Milk |




Note: X ∪ Y: the union of two itemsets

- The set contains both X and Y

# Mining Frequent Itemsets and Association Rules

- **Association rule mining**
  - Given two thresholds: *minsup, minconf*
  - Find all of the rules, $X \rightarrow Y$ (s, c)
    - such that, s ≥ *minsup* and $c \geq minconf$

- Let *minsup = 50%*
  - Freq. 1-itemsets: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3
  - Freq. 2-itemsets: {Beer, Diaper}: 3

- Let *minconf = 50%*
  - *Beer* → *Diaper* (60%, 100%)
  - *Diaper* → *Beer* (60%, 75%)

(Q: Are these all rules?)

| Tid | Items bought |
|---|---|
| 10 | Beer, Nuts, Diaper |
| 20 | Beer, Coffee, Diaper |
| 30 | Beer, Diaper, Eggs |
| 40 | Nuts, Eggs, Milk |
| 50 | Nuts, Coffee, Diaper, Eggs, Milk |

- **Observations:**
  - Mining association rules and mining frequent patterns are very close problems
  - Scalable methods are needed for mining large datasets

# Challenge: There Are Too Many Frequent Patterns!

- A long pattern contains a combinatorial number of sub-patterns
- How many frequent itemsets does the following $TDB_1$ contain?
  - $TDB_1$: $T_1$: $\{a_1, \ldots, a_{50}\}$; $T_2$: $\{a_1, \ldots, a_{100}\}$
  - Assuming (absolute) *minsup* = 1
  - Let's have a try

1-itemsets: $\{a_1\}$: 2, $\{a_2\}$: 2, …, $\{a_{50}\}$: 2, $\{a_{51}\}$: 1, …, $\{a_{100}\}$: 1,

2-itemsets: $\{a_1, a_2\}$: 2, …, $\{a_1, a_{50}\}$: 2, $\{a_1, a_{51}\}$: 1 …, …, $\{a_{99}, a_{100}\}$: 1,

…, …, …, …

99-itemsets: $\{a_1, a_2, \ldots, a_{99}\}$: 1, …, $\{a_2, a_3, \ldots, a_{100}\}$: 1

100-itemset: $\{a_1, a_2, \ldots, a_{100}\}$: 1

- The total number of frequent itemsets:

$$\binom{100}{1} + \binom{100}{2} + \binom{100}{3} + \cdots + \binom{100}{100} = 2^{100} - 1$$

A too huge set for any one to compute or store!

# Expressing Patterns in Compressed Form: Closed Patterns

- How to handle such a challenge?
- Solution 1: **Closed patterns**: A pattern (itemset) X is closed if X is *frequent,* and there exists *no super-pattern* Y ↄ X, *with the same support* as X
  - Let Transaction DB $TDB_1$: $T_1$: $\{a_1, \ldots, a_{50}\}$; $T_2$: $\{a_1, \ldots, a_{100}\}$
  - Suppose *minsup* = 1. How many closed patterns does $TDB_1$ contain?
    - Two: $P_1$: "$\{a_1, \ldots, a_{50}\}$: 2"; $P_2$: "$\{a_1, \ldots, a_{100}\}$: 1"
- Closed pattern is a lossless compression of frequent patterns
  - Reduces the # of patterns but does not lose the support information!
  - You will still be able to say: "$\{a_2, \ldots, a_{40}\}$: 2", "$\{a_5, a_{51}\}$: 1"

# Expressing Patterns in Compressed Form: Max-Patterns

- Solution 2: **Max-patterns**: A pattern X is a max-pattern if X is frequent and there exists no frequent super-pattern Y ↄ X
- Difference from close-patterns?
  - Do not care the real support of the sub-patterns of a max-pattern
  - Let Transaction DB $TDB_1$: $T_1$: $\{a_1, \ldots, a_{50}\}$; $T_2$: $\{a_1, \ldots, a_{100}\}$
  - Suppose *minsup* = 1. How many max-patterns does $TDB_1$ contain?
    - One: P: "$\{a_1, \ldots, a_{100}\}$: 1"
- Max-pattern is a lossy compression!
  - We only know $\{a_1, \ldots, a_{40}\}$ is frequent
  - But we do not know the real support of $\{a_1, \ldots, a_{40}\}$, …, any more!
- Thus in many applications, mining close-patterns is more desirable than mining max-patterns

# Chapter 6: Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

- Basic Concepts
- Efficient Pattern Mining Methods 
- Pattern Evaluation
- Summary

# Efficient Pattern Mining Methods

- The Downward Closure Property of Frequent Patterns
- The Apriori Algorithm
- Extensions or Improvements of Apriori
- Mining Frequent Patterns by Exploring Vertical Data Format
- FPGrowth:  A Frequent Pattern-Growth Approach
- Mining Closed Patterns

# The Downward Closure Property of Frequent Patterns

- Observation: From $TDB_1$: $T_1$: $\{a_1, \ldots, a_{50}\}$; $T_2$: $\{a_1, \ldots, a_{100}\}$
  - We get a frequent itemset: $\{a_1, \ldots, a_{50}\}$
  - Also, its subsets are all frequent: $\{a_1\}$, $\{a_2\}$, …, $\{a_{50}\}$, $\{a_1, a_2\}$, …, $\{a_1, \ldots, a_{49}\}$, …
  - There must be some hidden relationships among frequent patterns!
- The downward closure (also called "Apriori") property of frequent patterns
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - Every transaction containing {beer, diaper, nuts} also contains {beer, diaper}
  - Apriori: Any subset of a frequent itemset must be frequent
- Efficient mining methodology
  - If any subset of an itemset S is infrequent, then there is no chance for S to be frequent—why do we even have to consider S!? A sharp knife for pruning!

# Apriori Pruning and Scalable Mining Methods

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not even be generated! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Scalable mining Methods: Three major approaches
  - Level-wise, join-based approach: Apriori (Agrawal & Srikant@VLDB'94)
  - Vertical data format approach: Eclat (Zaki, Parthasarathy, Ogihara, Li @KDD'97)
  - Frequent pattern projection and growth: FPgrowth (Han, Pei, Yin @SIGMOD'00)

# Apriori: A Candidate Generation & Test Approach

เขียนเป็นขั้นตอน

- Outline of Apriori (level-wise, candidate generation and test)
  - Initially, scan DB once to get frequent 1-itemset
  - Repeat
    - Generate length-(k+1) candidate itemsets from length-k frequent itemsets
      - สร้างได้เท่าไหร่ ?
    - Test the candidates against DB to find frequent (k+1)-itemsets
    - Set k := k +1
  - Until no frequent or candidate set can be generated
  - Return all the frequent itemsets derived

# The Apriori Algorithm (Pseudo-Code)

$C_k$: Candidate itemset of size k

$F_k$ : Frequent itemset of size k

K := 1;
$F_k$ := {frequent items}; // frequent 1-itemset
**While** ($F_k$ != ∅) **do {** // when $F_k$ is non-empty
  $C_{k+1}$ := candidates generated from $F_k$; // candidate generation
  Derive $F_{k+1}$ by counting candidates in $C_{k+1}$ with respect to *TDB* at minsup;
  k := k + 1
  **}**
**return** $\cup_k F_k$ // return $F_k$ generated at each level

# The Apriori Algorithm—An Example

Database TDB

| Tid | Items |
|---|---|
| 10 | A, C, D |
| 20 | B, C, E |
| 30 | A, B, C, E |
| 40 | B, E |

minsup = 2

$C_1$

1st scan

| Itemset | sup |
|---|---|
| {A} | 2 |
| {B} | 3 |
| {C} | 3 |
| {D} | 1 |
| {E} | 3 |

$F_1$ ตัดทิ้ง

| Itemset | sup |
|---|---|
| {A} | 2 |
| {B} | 3 |
| {C} | 3 |
| {E} | 3 |

2 itemset

$C_2$

| Itemset |
|---|
| {A, B} |
| {A, C} |
| {A, E} |
| {B, C} |
| {B, E} |
| {C, E} |

2nd scan

$C_2$

| Itemset | sup |
|---|---|
| {A, B} | 1 |
| {A, C} | 2 |
| {A, E} | 1 |
| {B, C} | 2 |
| {B, E} | 3 |
| {C, E} | 2 |

ตัดทิ้ง

$F_2$

| Itemset | sup |
|---|---|
| {A, C} | 2 |
| {B, C} | 2 |
| {B, E} | 3 |
| {C, E} | 2 |

$C_3$

| Itemset |
|---|
| {B, C, E} |

3rd scan

$F_3$

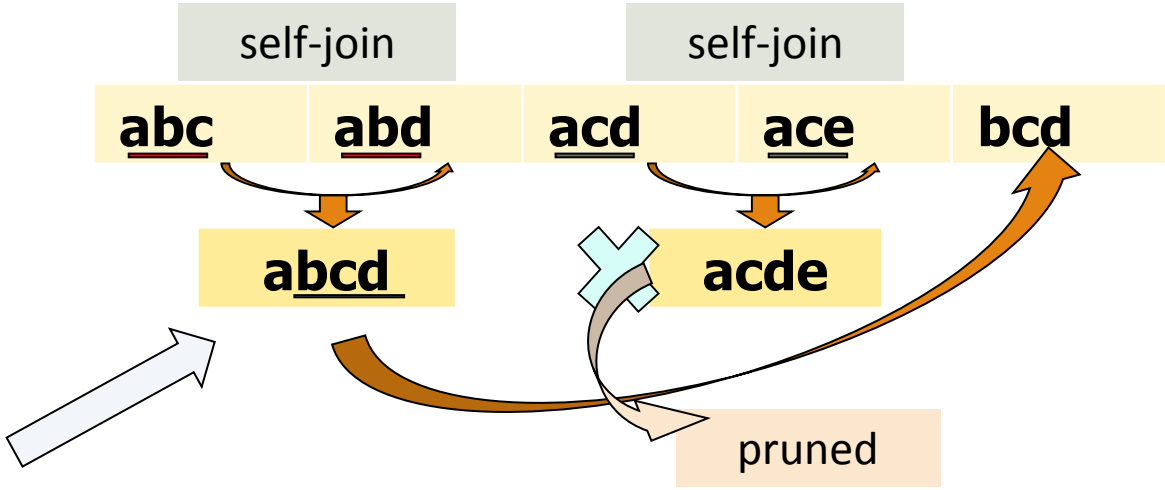
| Itemset | sup |
|---|---|
| {B, C, E} | 2 |

Stop !

Ans 1. item = 4
2. item = 4
3. item = 1

# Apriori: Implementation Tricks

- How to generate candidates?
  - Step 1: self-joining $F_k$
  - Step 2: pruning
- Example of candidate-generation
  - $F_3$ = {*abc, abd, acd, ace, bcd*}
  - Self-joining: $F_3$*$F_3$
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in $F_3$
  - $C_4$ = {*abcd*}



# Candidate Generation: An SQL Implementation

- Suppose the items in $F_{k-1}$ are listed in an order
- Step 1: self-joining $F_{k-1}$

  insert into $C_k$

  select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$

  from $F_{k-1}$ as p, $F_{k-1}$ as q

  where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
- Step 2: pruning

  for all *itemsets c in* $C_k$ do

  for all *(k-1)-subsets s of c* do

  **if** *(s is not in* $F_{k-1}$*)* **then delete** *c* **from** $C_k$