Mining Frequent Patterns, Association and Correlations

- Basic concepts
- Efficient and scalable frequent item set mining methods
- Mining various kinds of association rules
- From association mining to correlation analysis
- Constraint-based association mining
- Summary

What Is Frequent Pattern Analysis?

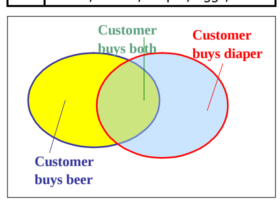
- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed in the context of frequent item sets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— 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?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis

Why Is Freq. Pattern Mining Important?

- Frequent pattern: An essential and important property of datasets
- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential and structural patterns
  - Pattern analysis in multimedia, time-series, and stream data
  - Classification: discriminative, frequent pattern analysis
  - Cluster analysis: frequent pattern-based clustering
  - Broad applications

Basic Concepts: Frequent Patterns

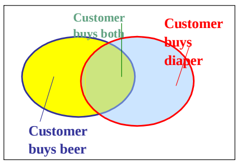
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



- itemset: A set of one or more items
- k-itemset X = $\{x_1, \ldots, x_k\}$
- *(absolute) support*, or, *support count* of X: Frequency or occurrence of an itemset X
- *(relative) support*, *s*, is the fraction of transactions that contains X (i.e., the probability that a transaction contains X)
- An itemset X is *frequent* if X's support is no less than a *minsup* threshold

Basic Concepts: Association 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



- Find all the rules *X* → *Y* with minimum support and confidence
  - support, *s*, probability that a transaction contains X ∪ Y
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

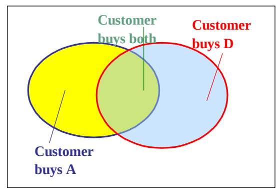
Let minsup = 50%, minconf = 50%

Freq. Pat.: Beer:3, Nuts:3, Diaper:4, Eggs:3, {Beer, Diaper}:3

- Association rules: (many more!)
  - *Beer* → *Diaper* (60%, 100%)
  - *Diaper* → *Beer* (60%, 75%)

Basic Concepts: Frequent Patterns and Association Rules

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



Let sup_{min} *= 50%,* $conf_{min}$ *= 50%*

Freq. Pat.: {A:3, B:3, D:4, E:3, AD:3}

Association rules:

A → *D* (60%, 100%)

D → *A* (60%, 75%)

Closed Patterns and Max-Patterns

- A long pattern contains a combinatorial number of sub-patterns, e.g., $\{a_1, \ldots, a_{100}\}$ contains $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100} - 1 = 1.27 \times 10^{30}$ sub-patterns!
- Solution: *Mine closed patterns and max-patterns instead*
- An itemset X is closed if X is *frequent* and there exists *no super-pattern* Y ⊃ X, *with the same support* as X
- An itemset X is a max-pattern if X is frequent and there exists no frequent super-pattern Y ⊃ X
- Closed pattern is a lossless compression of frequent patterns
  - Reducing the # of patterns and rules

Closed Patterns and Max-Patterns

- Exercise. DB = {$<a_1, \ldots, a_{100}>$, $< a_1, \ldots, a_{50}>$}
  - Min_sup = 1.
- What is the set of closed itemset?
  - $<a_1, \ldots, a_{100}>$: 1
  - $< a_1, \ldots, a_{50}>$: 2
- What is the set of max-pattern?
  - $<a_1, \ldots, a_{100}>$: 1
- What is the set of all patterns?
  - !!

Computational Complexity of Frequent Itemset Mining

- How many itemsets are potentially to be generated in the worst case?
  - The number of frequent itemsets to be generated is sensitive to the minsup threshold
  - When minsup is low, there exist potentially an exponential number of frequent itemsets
  - The worst case: M^N where M: # distinct items, and N: max length of transactions
- The worst case complexty vs. the expected probability
  - Ex. Suppose Walmart has 10^4 kinds of products
    - The chance to pick up one product 10^{-4}
    - The chance to pick up a particular set of 10 products: $\sim 10^{-40}$

Scalable Methods for Mining Frequent Patterns

- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If {beer, diaper, nuts} is frequent, so is {beer, diaper}
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Scalable mining methods: Three major approaches
  - Apriori
  - Frequent pattern growth (FPgrowth)
  - Vertical data format approach (Charm)

Apriori: A Candidate Generation-and-Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested!
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

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

1st scan → C_1

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

→ L_1

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

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

→ L_2

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

C_3

Itemset
{B, C, E}

3rd scan → L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm

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;
```


Implementation of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- Example of candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

Challenges of Frequent Pattern Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Bottleneck of Frequent-pattern Mining

- Bottlenecks of the Apriori approach
  - Breadth-first (i.e., level-wise) search
  - Candidate generation and test
    - Often generates a huge number of candidates
    - Multiple database scans are costly
- The FPGrowth Approach
  - Depth-first search
  - Avoid explicit candidate generation
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100}-1 = 1.27*10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

Mining Frequent Patterns Without Candidate Generation

- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc": DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

Construct FP-tree from a Transaction Database

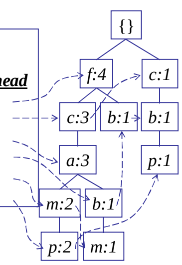
TID	*Items bought*	*(ordered) frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o, w*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

min_support = 3

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, f-list
3. Scan DB again, construct FP-tree

Header Table

Item	*frequency*	*head*
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



F-list=f-c-a-b-m-p

Benefits of the FP-tree Structure

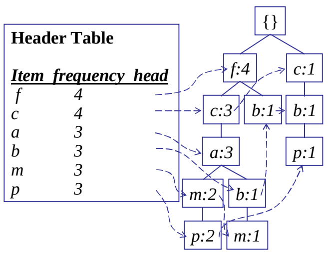
- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the *count* field)
  - For Connect-4 DB, compression ratio could be over 100

Partition Patterns and Databases

- Frequent patterns can be partitioned into subsets according to f-list
  - F-list=f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f
- Completeness and non-redundancy

Find Patterns Having P From P-conditional Database

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of *transformed prefix paths* of item *p* to form *p*'s conditional pattern base

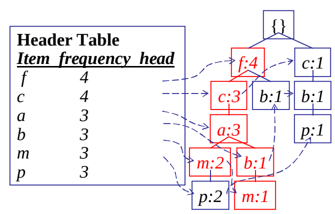


***Conditional* pattern bases**

item	cond. pattern base
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

From Conditional Pattern-bases to Conditional FP-trees

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base



***m-conditional* pattern base:**
fca:2, fcab:1

{}
|
f:3
|
c:3
|
a:3

***m-conditional* FP-tree**

All frequent patterns relate to *m*
m,
fm, cm, am,
fcm, fam, cam,
fcam

Recursion: Mining Each Conditional FP-tree

{}
|
f:3
|
c:3
|
a:3
***m-conditional* FP-tree**

Cond. pattern base of "am": (fc:3)

{}
|
f:3
|
c:3
***am-conditional* FP-tree**

Cond. pattern base of "cm": (f:3)

{}
|
f:3
***cm-conditional* FP-tree**

Cond. pattern base of "cam": (f:3)

{}
|
f:3
***cam-conditional* FP-tree**

Mining Frequent Patterns With FP-trees

- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

Advantages of the Pattern Growth Approach

- Divide-and-conquer:
  - Decompose both the mining task and DB according to the frequent patterns obtained so far
  - Lead to focused search of smaller databases
- Other factors
  - No candidate generation, no candidate test
  - Compressed database: FP-tree structure
  - No repeated scan of entire database
  - Basic ops: counting local freq items and building sub FP-tree, no pattern search and matching
- A good open-source implementation and refinement of FPGrowth
  - FPGrowth+ (Grahne and J. Zhu, FIMI'03)

CHARM: Mining by Exploring Vertical Data Format

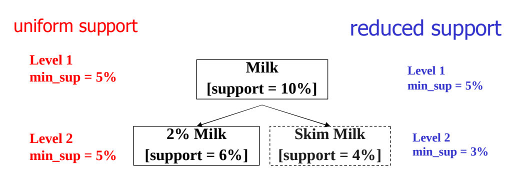
- Vertical format: $t(AB) = \{T_{11}, T_{25}, \ldots\}$
  - tid-list: list of trans.-ids containing an itemset
- Deriving closed patterns based on vertical intersections
  - $t(X) = t(Y)$: X and Y always happen together
  - $t(X) \subset t(Y)$: transaction having X always has Y
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - $t(X) = \{T_1, T_2, T_3\}$, $t(XY) = \{T_1, T_3\}$
  - Diffset $(XY, X) = \{T_2\}$

Mining Various Kinds of Association Rules

- Mining multilevel association
- Mining multidimensional association
- Mining quantitative association
- Mining interesting correlation patterns

Mining Multiple-Level Association Rules

- Items often form hierarchies
- Flexible support settings
  - Items at the lower level are expected to have lower support
- Exploration of *shared* multi-level mining



Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to "ancestor" relationships between items
- Example
  - milk ⇒ wheat bread [support = 8%, confidence = 70%]
  - 2% milk ⇒ wheat bread [support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule
- A rule is redundant if its support is close to the "expected" value, based on the rule's ancestor

Mining Multi-Dimensional Association

- Single-dimensional rules:
  - buys(X, "milk") ⇒ buys(X, "bread")
- Multi-dimensional rules: ≥ 2 dimensions or predicates
  - Inter-dimension assoc. rules (*no repeated predicates*)
    - age(X,"19-25") ∧ occupation(X,"student") ⇒ buys(X, "coke")
  - hybrid-dimension assoc. rules (*repeated predicates*)
    - age(X,"19-25") ∧ buys(X, "popcorn") ⇒ buys(X, "coke")
- Categorical Attributes: finite number of possible values, no ordering among values—data cube approach
- Quantitative Attributes: numeric, implicit ordering among values—discretization, clustering, and gradient approaches

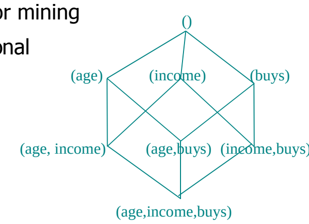
Mining Quantitative Associations

- Techniques can be categorized by how numerical attributes, such as age or salary are treated

1. Static discretization based on predefined concept hierarchies (data cube methods)
2. Dynamic discretization based on data distribution (quantitative rules)
3. Clustering: Distance-based association
   - one dimensional clustering then association
4. Deviation:
   Sex = female => Wage: mean=$7/hr (overall mean = $9)

Static Discretization of Quantitative Attributes

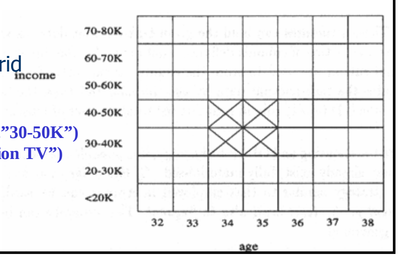
- Discretized prior to mining using concept hierarchy
- Numeric values are replaced by ranges
- In relational database, finding all frequent k-predicate sets will require *k* or *k*+1 table scans
- Data cube is well suited for mining
- The cells of an n-dimensional cuboid correspond to the predicate sets
- Mining from data cubes can be much faster



Quantitative Association Rules

- Numeric attributes are *dynamically* discretized
  - Such that the confidence or compactness of the rules mined is maximized
- 2-D quantitative association rules: $A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$
- Cluster *adjacent* association rules to form general rules using a 2-D grid
- Example

age(X,"34-35") ∧ income(X,"30-50K")
⇒ buys(X,"high resolution TV")



Interestingness Measure: Correlations (Lift)

- *play basketball* ⇒ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%
- *play basketball* ⇒ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence
- Measure of dependent/correlated events: lift

$$lift = \frac{P(A \cup B)}{P(A)P(B)}$$

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(B,C) = \frac{2000/5000}{3000/5000 * 3750/5000} = 0.89 \qquad lift(B,\neg C) = \frac{1000/5000}{3000/5000 * 1250/5000} = 1.33$$

Are *lift* and χ^2 Good Measures of Correlation?

- *"Buy walnuts ⇒ buy milk* [1%, 80%]" is misleading
  - if 85% of customers buy milk
- Support and confidence are not good to represent correlations
- So many interestingness measures?

$$lift = \frac{P(A \cup B)}{P(A)P(B)}$$

$$all\_conf = \frac{\sup(X)}{\max\_item\_\sup(X)}$$

$$coh = \frac{\sup(X)}{|universe(X)|}$$

	Milk	No Milk	Sum (row)
Coffee	m, c	~m, c	c
No Coffee	m, ~c	~m, ~c	~c
Sum(col.)	m	~m	Σ

DB	m, c	~m, c	m~c	~m~c	lift	all-conf	coh	$\chi 2$
A1	1000	100	100	10,000	9.26	0.91	0.83	9055
A2	100	1000	1000	100,000	8.44	0.09	0.05	670
A3	1000	100	10000	100,000	9.18	0.09	0.09	8172
A4	1000	1000	1000	1000	1	0.5	0.33	0

Constraints in Data Mining

- Knowledge type constraint:
  - classification, association, etc.
- Data constraint — using SQL-like queries
  - find product pairs sold together in stores in Chicago in Dec.'02
- Dimension/level constraint
  - in relevance to region, price, brand, customer category
- Rule (or pattern) constraint
  - small sales (price < $10) triggers big sales (sum > $200)
- Interestingness constraint
  - strong rules: min_support ≥ 3%, min_confidence ≥ 60%

Constraint-Based Frequent Pattern Mining

- Classification of constraints based on their constraint-pushing capabilities
  - Anti-monotonic: If constraint c is violated, its further mining can be terminated
  - Monotonic: If c is satisfied, no need to check c again
  - Data anti-monotonic: If a transaction t does not satisfy c, t can be pruned from its further mining
  - Succinct: c must be satisfied, so one can start with the data sets satisfying c
  - Convertible: c is not monotonic nor anti-monotonic, but it can be converted into it if items in the transaction can be properly ordered

Anti-Monotonicity in Constraint Pushing

- A constraint C is *antimonotone* if the super pattern satisfies C, all of its sub-patterns do so too
- In other words, *anti-monotonicity:* If an itemset S violates the constraint, so does any of its superset
- Ex. 1. *sum(S.price)* $\leq v$ is anti-monotone
- Ex. 2. range(S.profit) $\leq$ 15 is anti-monotone
  - Itemset *ab* violates C
  - So does every superset of *ab*
- Ex. 3. *sum(S.Price)* $\geq v$ is not anti-monotone
- Ex. 4. *support count* is anti-monotone: core property used in Apriori

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Item	Profit
a	40
b	0
c	-20
d	10
e	-30
f	30
g	20
h	-10

Monotonicity for Constraint Pushing

- A constraint C is *monotone* if the pattern satisfies C, we do not need to check C in subsequent mining
- Alternatively, monotonicity: *If an itemset S satisfies the constraint, so does any of its superset*
- Ex. 1. *sum(S.Price)* $\geq v$ is monotone
- Ex. 2. *min(S.Price)* $\leq v$ is monotone
- Ex. 3. C: range(S.profit) $\geq$ 15
  - Itemset *ab* satisfies C
  - So does every superset of *ab*

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Item	Profit
a	40
b	0
c	-20
d	10
e	-30
f	30
g	20
h	-10

Data Antimonotonicity: Pruning Data Space

- A constraint c is *data antimonotone* if for a pattern p cannot satisfy a transaction t under c, p's superset cannot satisfy t under c either
- The key for data antimonotone is *recursive data reduction*
- Ex. 1. *sum(S.Price)* $\geq v$ is data antimonotone
- Ex. 2. *min(S.Price)* $\leq v$ is data antimonotone
- Ex. 3. C: *range(S.profit)* $\geq$ *25* is data antimonotone
  - Itemset {b, c}'s projected DB:
    - T10': {d, f, h}, T20': {d, f, g, h}, T30': {d, f, g}
  - since C cannot satisfy T10', T10' can be pruned

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f, h
20	b, c, d, f, g, h
30	b, c, d, f, g
40	c, e, f, g

Item	Profit
a	40
b	0
c	-20
d	-15
e	-30
f	-10
g	20
h	-5

Succinctness

- Succinctness:
  - Given A_1, the set of items satisfying a succinctness constraint C, then any set S satisfying C is based on A_1 , i.e., S contains a subset belonging to A_1
  - Idea: Without looking at the transaction database, whether an itemset S satisfies constraint C can be determined based on the selection of items
  - $min(S.Price) \leq v$ is succinct
  - $sum(S.Price) \geq v$ is not succinct
- Optimization: If C is succinct, C is pre-counting pushable

Frequent-Pattern Mining: Summary

- Frequent pattern mining—an important task in data mining
- Scalable frequent pattern mining methods
  - Apriori (Candidate generation & test)
  - FPgrowth
  - Vertical format approach
- Mining a variety of rules and interesting patterns
- Constraint-based mining
- Mining sequential and structured patterns
- Extensions and applications