Data Mining
-- Association Rules

Instructor: Jen-Wei Huang

Office: 92501 in the EE building
jwhuang@mail.ncku

Association Rules

- Finding association, correlation or causal structures among sets of items or objects in transactional, relational DB
- Examples
  - bread ^ milk -> butter
  - age("25~35") ^ income("35,000~40,000) -> buyer(Lancer)

Definitions

- $I = \{i_1, i_2, i_3 \ldots i_n\}$: the set of all items
  - Itemset: a set of items
- Association rule: $A \rightarrow B$,
  - where $A \subset I$, $B \subset I$, $A \cap B = \varnothing$
- support $(A \rightarrow B) = \text{Prob.}(A \cup B)$
- confidence$(A \rightarrow B) = \text{Prob.}(A \cup B / A)$
  - Strong rule: satisfy both minimum support & confidence

Example

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

min_support = 2
min_conf = 2/3

- Strong rules
  - $\{B, E\} \rightarrow C$ (2/3)
  - $C \rightarrow A$ (2/3)
  - $A \rightarrow C$ (2/2)

References

- Slides from Prof. J.-W. Han, UIUC
- Slides from Prof. M.-S. Chen, NTU
- Slides from Prof. W.-Z. Peng, NCTU

Data Mining -- Association Rules

Instructor: Jen-Wei Huang

Office: 92501 in the EE building
jwhuang@mail.ncku

Association Rules

- Finding association, correlation or causal structures among sets of items or objects in transactional, relational DB
- Examples
  - bread ^ milk -> butter
  - age("25~35") ^ income("35,000~40,000) -> buyer(Lancer)

Example

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

min_support = 2
min_conf = 2/3

- Frequent itemsets
  - {A}, {B}, {C}, {E}, {A,C}, {B,C}, {B,E}, {C,E}, {B,C,E}
- Strong rules
  - {B, E}→C (2/3)
  - C→A (2/3)
  - A→C (2/2)

Definitions

- $I = \{i_1, i_2, i_3 \ldots i_n\}$: the set of all items
  - Itemset: a set of items
- Association rule: $A \rightarrow B$,
  - where $A \subset I$, $B \subset I$, $A \cap B = \emptyset$
- support $(A \rightarrow B)$ = Prob.$(A \cup B)$
- confidence$(A \rightarrow B)$ = Prob.$(A \cup B / A)$
  - Strong rule: satisfy both minimum support & confidence

Definitions

- $I = \{i_1, i_2, i_3 \ldots i_n\}$: the set of all items
- $T \subseteq I$: a transaction
- D: a set of T, transaction DB
- itemset: a set of items
- k-itemset: an itemset that contains k items

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

Frequent Pattern

- First proposed by Agrawal [1]
- A pattern that occurs frequently in a data set
- Finding inherent regularities in data
- Foundation for many essential data mining tasks

- In association rule mining, we want to find frequent itemsets, i.e., itemsets whose support are no less than a min_supp threshold.

Apriori Algorithm [2]

- A candidate generation and test approach
- Two steps:
  - Finding all frequent itemsets
  - Deriving valid association rules

- Downward closure property
  - Any subset of a frequent itemset must be frequent
  - E.g.) If {beer, diaper, nuts} is frequent, so is {beer, diaper}
  - If there is any itemset which is infrequent, its superset should not be frequent

Apriori Algorithm

- Scan DB once to get frequent 1-itemset
- For frequent k-itemsets, repeat followings
  - Generate length (k+1) candidate itemsets from frequent-k itemsets
  - Test the candidate itemsets against DB
  - Terminate when no frequent or candidate set can be generated
- Compute confidences from all frequent k-itemsets (k>1)

An Example

min_support = 2

Database DB

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	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

Candidate Generation

- Step 1: self-joining L_k
- Step 2: pruning
- E.g.)
  - 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*}

Pseudo-Code

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 $\cup_k L_k$;

Association Rules Computation

```
for each large itemset m do
  for each subset p of m do
      if (sup(m)/sup(m-p)>= minconf) then
            output the rule (m-p)=>p with
            conf= sup(m)/sup(m-p) and
            support=sup(m)
```

Example

- Frequent k-itemsets (k>1) generated from the previous step:
  - {A, C}, {B, C}, {B, E}, {C, E}, {B, C, E}
- Scan DB to test if the confidences of the corresponding ARs are valid.
  - A->C, C->A
  - B->C, C->B
  - B->E, E->B
  - C->E, E->C
  - B->CE, C->BE, E->BC, BC->E, BE->C, CE->B

Redundant Rules

- For the same support and confidence, if we have a rule {a,d}->{c,e,f,g}, do we need
  - {a,d}->{c,e,f}
  - {a}->{c,e,f,g}
  - {a,d,c}->{e,f,g}
  - {a}->{d,c,e,f,g} ?
- Maximal association rules

Interestingness Measure

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%.
- *play basketball* $\Rightarrow$ *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

$$lift(B,C) = \frac{2000/5000}{3000/5000 * 3750/5000} = 0.89$$

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

Improvements of Apriori

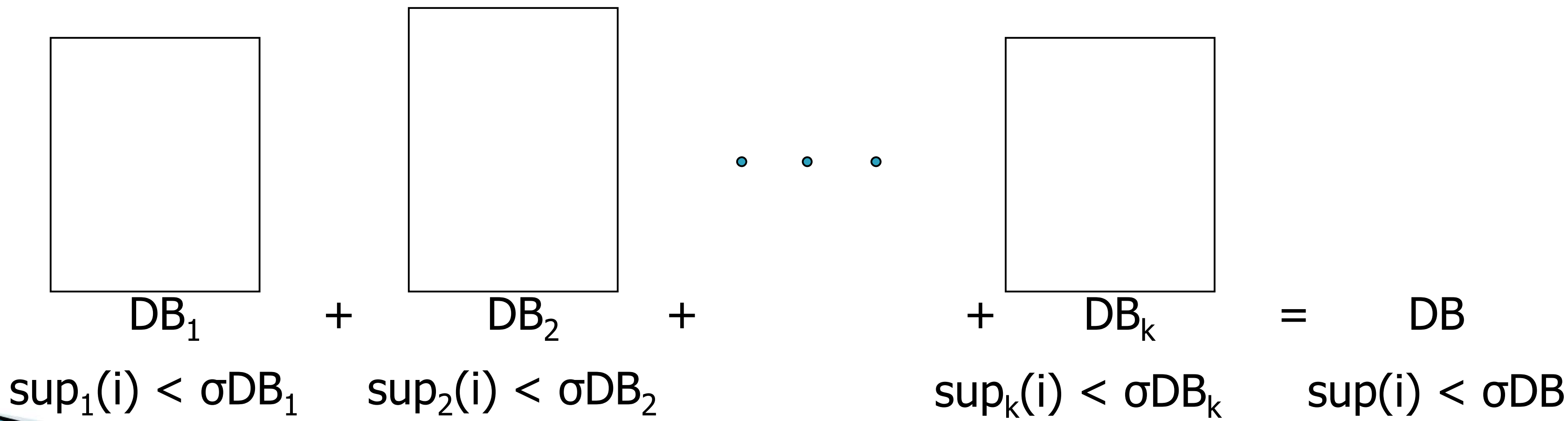
- Major computational challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting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

Scan Reduction

- Reduce Scans of database
- Compute candidate k-itemsets from candidate (k-1)-itemsets instead of frequent (k-1)-itemsets
- Two scan methods:
  - Scan DB the first time for frequent 1-itemsets
  - Compute all candidate k-frequent itemsets from frequent 1-itemsets
  - Scan DB the second time to test if candidate k-itemsets are frequent

Partition Database

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB [3]
  - Step 1: partition database and find local frequent patterns
  - Step 2: consolidate global frequent patterns



Hash-based Algorithm

- Algorithm DHP [4]: Direct Hashing and Pruning
- Hash table scheme
  - Eliminate infrequent candidate itemsets in the early phase
- Transaction items pruning
  - Eliminate infrequent items from the database

Candidate Itemsets Pruning

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B,E

L_1=<A,B,C,E> Minimum Support Frequency=2

<AC>,<AD>,<CD>
<BC>,<BE>,<CE>
<AB>,<AC>,<AE>,<BC>,<BE>,<CE>
<BE>

$h(x,y)=((ord(x)*10+ord(y)) \bmod 7$

	<CE>				<BE>		<AC>
	<CE>		<BC>		<BE>		<CD>
	<AD>	<AE>	<BC>		<BE>	<AB>	<AC>
H_2	3	1	2	0	3	1	3
	0	*1*	*2*	*3*	*4*	*5*	*6*

Hash table building

$C_2=L_1*L_1$

Itemset
<AB>
<AC>
<AE>
<BC>
<BE>
<CE>

→

C_2'

Itemset
<AC>
<BC>
<BE>
<CE>

pruning <AB>, <AE>

Candidate pruning

Transaction Items Pruning

- A transaction should contain at least k+1 k-itemsets to support (k+1)-itemsets
  - Each item should appear at least k times

C_2=<<AC>, <BC>, <BE>, <CE>>

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B,E

Before trimming

a[0]	a[1]	a[2]	a[3]
1	1	0	
2	2	2	
1	2	3	2
1	1		

Counter array

	TID	Items
Trim A,C,D	100	
	200	B,C,E
Trim A	300	B,C,E
Trim B,E	400	

After trimming

Trimming information collecting

Transaction trimming

References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD'93
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. VLDB'94
- [3] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. VLDB'95.
- [4] J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. SIGMOD'95

References

- Slides from Prof. J.-W. Han, UIUC
- Slides from Prof. M.-S. Chen, NTU
- Slides from Prof. W.-Z. Peng, NCTU

HW1

- Compute strong association rules from the following DB with
  - min_supp = 50%
  - min_conf = 66%
- DB:

100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E
500	A, C, E
600	B, C, D

Data Mining
-- Association Rules

Instructor: Jen-Wei Huang

Office: 92501 in the EE building
jwhuang@mail.ncku

FP-Growth [1]

- Mining frequent patterns without candidate generation
  - Depth-first search approach
- Grow long patterns from short ones using local frequent items only
  - "abc" is a frequent pattern
  - Get all transactions having "abc", i.e., project DB on abc: DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

Construct FP-tree

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

min_support = 3

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

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

Construct FP-tree

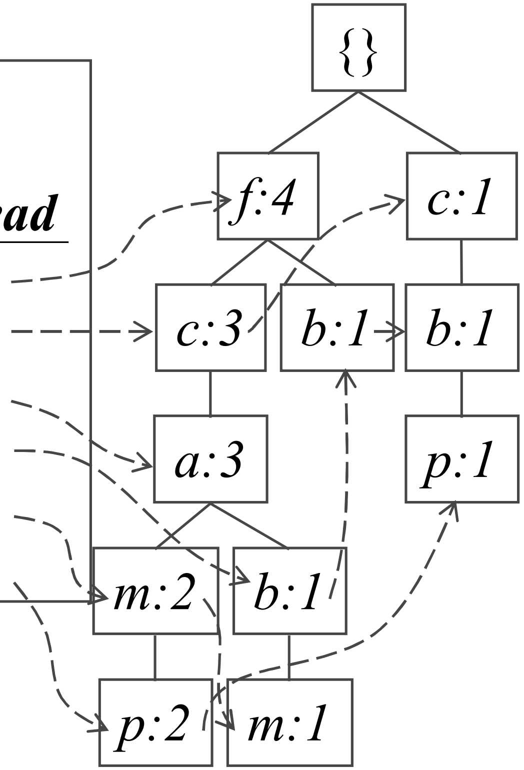
TID	*Items bought*	*(ordered) frequent items*
100	***{f, a, c, d, g, i, m, p}***	***{f, c, a, m, p}***
200	***{a, b, c, f, l, m, o}***	***{f, c, a, b, m}***
300	***{b, f, h, j, o, w}***	***{f, b}***
400	***{b, c, k, s, p}***	***{c, b, p}***
500	***{a, f, c, e, l, p, m, n}***	***{f, c, a, m, p}***

min_support = 3

3. Scan DB again, sort items in the transaction by frequency and construct FP-tree

Header Table

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



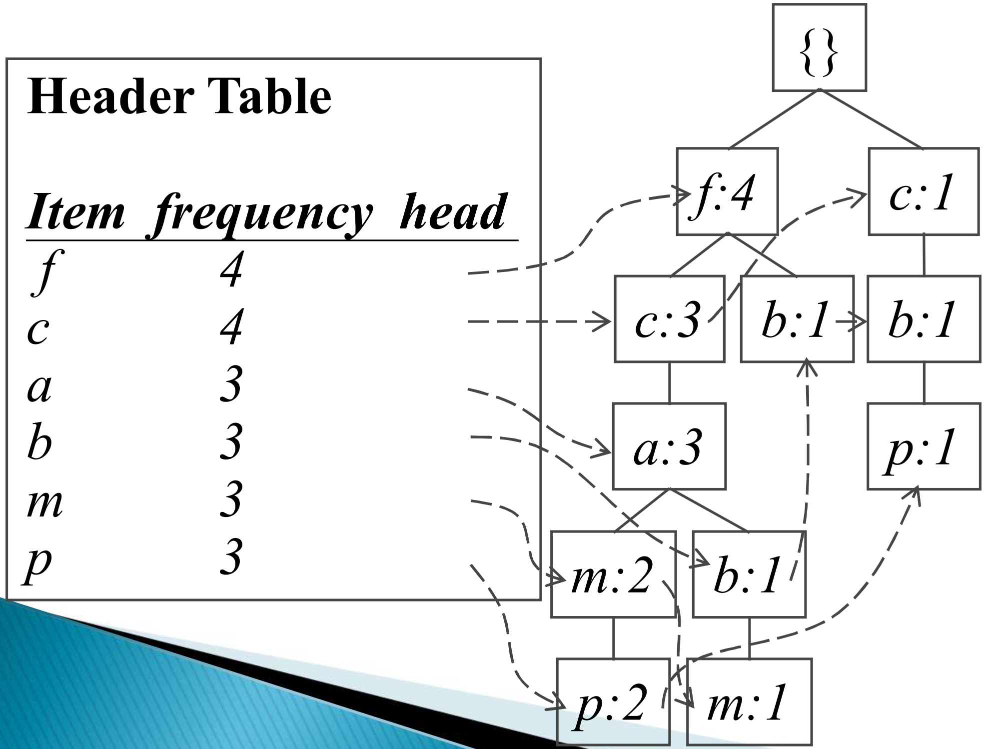
F-list = f-c-a-b-m-p

Partition Database

- 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-redundency

Conditional Pattern Bases

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



Conditional **pattern bases**

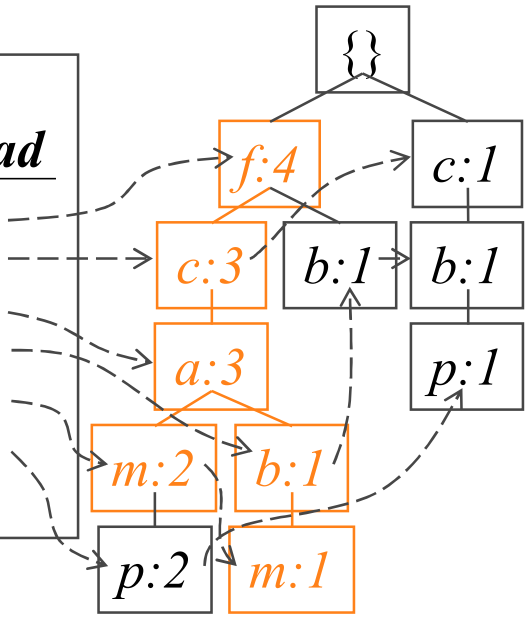
item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Conditional FP-trees

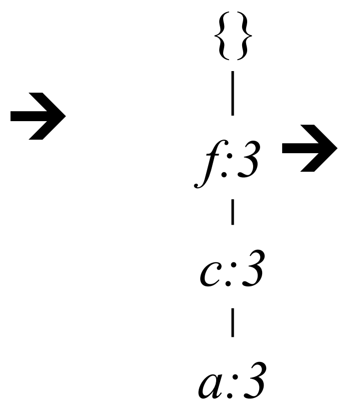
- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base

Header Table

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



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



m-conditional **FP-tree**

All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

Mining 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**

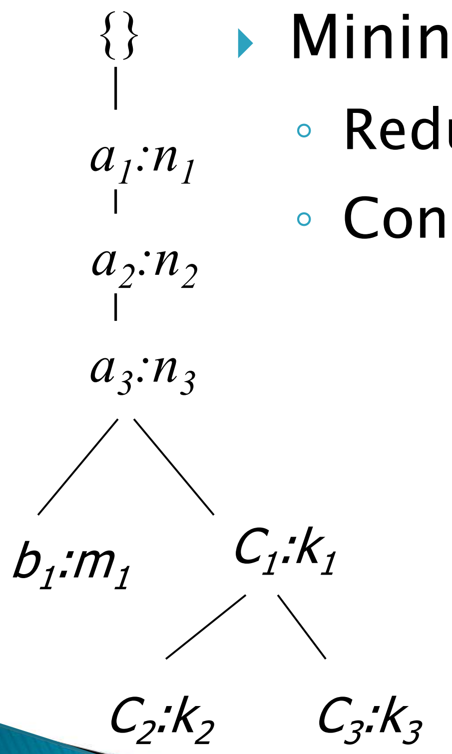
Cond. pattern base of "cam": (f:3)

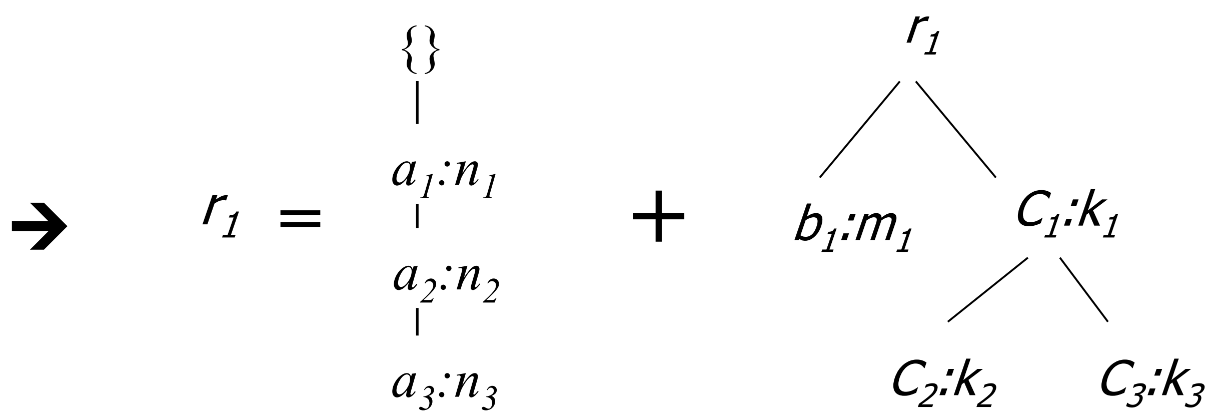
{}
|
f:3

cam-conditional **FP-tree**

Single Prefix Path

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts





Benefits of FP-tree

- 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)

FP-Growth Algorithm

- 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

Problems of FP-Growth

- What about if FP-tree cannot fit in memory?
  - DB projection

- First partition a database into a set of projected DBs
- Then construct and mine FP-tree for each projected DB

ECLAT [3]

- 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 frequent patterns based on vertical intersections
- 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 $(XY, X) = \{T_2\}$

Basic Extensions

- Max-pattern [5]
  - R. J. Bayardo. Efficiently mining long patterns from databases. SIGMOD'98.
- Closed-pattern [6]
  - N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. ICDT'99.
- Sequential pattern [7]
  - R. Agrawal and R. Srikant. Mining sequential patterns. ICDE'95

Closed Patterns and Max-Patterns

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

Examples

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

Computational Complexity

- How many itemsets are potentially to be generated in the worst case?
  - The number of frequent itemsets to be generated is senstive to the min_sup threshold
  - When min_sup 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 complexity 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: $\sim 10^{-40}$
    - What is the chance this particular set of 10 products to be frequent 10^3 times in 10^9 transactions?

References

- [1] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. SIGMOD' 00
- [2] G. Grahne and J. Zhu, Efficiently Using Prefix-Trees in Mining Frequent Itemsets, Proc. ICDM'03 Int. Workshop on Frequent Itemset Mining Implementations (FIMI'03), Melbourne, FL, Nov. 2003
- [3] M. J. Zaki, Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering, 12(3):372-390. May/June 2000
- [4] M. J. Zaki and Karam Gouda, Fast Vertical Mining Using Diffsets. In 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. August 2003.
- [5] R. J. Bayardo. Efficiently mining long patterns from databases. SIGMOD'98.
- [6] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. ICDT'99.
- [7] R. Agrawal and R. Srikant. Mining sequential patterns. ICDE'95

References

- Slides from Prof. J.-W. Han, UIUC
- Slides from Prof. M.-S. Chen, NTU
- Slides from Prof. W.-Z. Peng, NCTU