Frequent Pattern Mining

Transaction Data Analysis

- Transactions: customers’ purchases of commodities
  - (bread, milk, cheese) if they are bought together
- Frequent patterns: product combinations that are frequently purchased together by customers
- Frequent patterns: patterns (set of items, sequence, etc.) that occur frequently in a database [AIS93]

Why Frequent Patterns?

- What products were often purchased together?
- What are the frequent subsequent purchases after buying a iPod?
- What kinds of genes are sensitive to this new drug?
- What key-word combinations are frequently associated with web pages about game-evaluation?

Why Frequent Pattern Mining?

- Foundation for many data mining tasks
  - Association rules, correlation, causality, sequential patterns, spatial and multimedia patterns, associative classification, cluster analysis, iceberg cube, …
- Broad applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log (click stream) analysis, …

Frequent Itemsets

- Itemset: a set of items
  - E.g., acm = {a, c, m}
- Support of itemsets
  - Sup(acm) = 3
- Given min_sup = 3, acm is a frequent pattern
- Frequent pattern mining: finding all frequent patterns in a database

A Naive Attempt

- Generate all possible itemsets, test their supports against the database
- How to hold a large number of itemsets into main memory?
  - 100 items → $2^{100} - 1$ possible itemsets
- How to test the supports of a huge number of itemsets against a large database, say containing 100 million transactions?
  - A transaction of length 20 needs to update the support of $2^{20} - 1 = 1,048,575$ itemsets
Transactions in Real Applications

- A large department store often carries more than 100 thousand different kinds of items
  - Amazon.com carries more than 17,000 books relevant to data mining
- Walmart has more than 20 million transactions per day, AT&T produces more than 275 million calls per day
- Mining large transaction databases of many items is a real demand

How to Get an Efficient Method?

- Reducing the number of itemsets that need to be checked
- Checking the supports of selected itemsets efficiently

Candidate Generation & Test

- Any subset of a frequent itemset must also be frequent – an anti-monotonic property
  - A transaction containing \{beer, diaper, nuts\} also contains \{beer, diaper\}
  - \{beer, diaper, nuts\} is frequent \(\Rightarrow\) \{beer, diaper\} must also be frequent
- In other words, any superset of an infrequent itemset must also be infrequent
  - No superset of any infrequent itemset should be generated or tested
  - Many item combinations can be pruned!

Apriori-Based Mining

- Generate length \((k+1)\) candidate itemsets from length \(k\) frequent itemsets, and
- Test the candidates against DB

The Apriori Algorithm [AgSr94]

<table>
<thead>
<tr>
<th>Level-wise, candidate generation and test</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_k): Candidate itemset of size (k)</td>
</tr>
<tr>
<td>(L_k): frequent itemset of size (k)</td>
</tr>
<tr>
<td>(L_1) = {frequent items};</td>
</tr>
<tr>
<td>for ((k = 1; L_k \neq \emptyset; k++)) do</td>
</tr>
<tr>
<td>(C_{k+1}) = candidates generated from (L_k);</td>
</tr>
<tr>
<td>for each transaction (t) in database do</td>
</tr>
<tr>
<td>increment the count of all candidates in (C_{k+1}) that are contained in (t)</td>
</tr>
<tr>
<td>(L_{k+1}) = candidates in (C_{k+1}) with min_support</td>
</tr>
<tr>
<td>return (\bigcup_k L_k)</td>
</tr>
</tbody>
</table>

The Apriori Algorithm

- Candidate generation
- Test
Important Steps in Apriori

- How to find frequent 1- and 2-itemsets?
- How to generate candidates?
  - Step 1: self-joining \( L_k \)
  - Step 2: pruning
- How to count supports of candidates?

Finding Frequent 1- & 2-itemsets

- Finding frequent 1-itemsets (i.e., frequent items) using a one dimensional array
  - Initialize \( c[item] = 0 \) for each item
  - For each transaction \( T \), for each item in \( T \), \( c[item]++ \);
  - If \( c[item] >= \text{min_sup} \), item is frequent
- Finding frequent 2-itemsets using a 2-dimensional triangle matrix
  - For items \( i, j \) (\( i < j \)), \( c[i, j] \) is the count of itemset \( ij \)

Finding Frequent 1- & 2-itemsets

- Finding frequent 1-itemsets (i.e., frequent items) using a one dimensional array
  - Initialize \( c[item] = 0 \) for each item
  - For each transaction \( T \), for each item in \( T \), \( c[item]++ \);
  - If \( c[item] >= \text{min_sup} \), item is frequent
- Finding frequent 2-itemsets using a 2-dimensional triangle matrix
  - For items \( i, j \) (\( i < j \)), \( c[i, j] \) is the count of itemset \( ij \)

Counting Array

- A 2-dimensional triangle matrix can be implemented using a 1-dimensional array

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Example:

- \( c[3, 5] = c[(3-1)*(2n-3)/2+5-3] = c[9] \)

Example of Candidate-generation

- \( L_3 = \{abc, abd, acd, ace, bcd\} \)
- Self-joining: \( L_3 \times L_3 \)
  - \( abcd \leftarrow abc \times abd \)
  - \( acde \leftarrow acd \times ace \)
- Pruning:
  - \( acde \) is removed because \( ade \) is not in \( L_3 \)
- \( C_4 = \{abcd\} \)

How to Generate Candidates?

- Suppose the items in \( L_{k-1} \) are listed in an order
- Step 1: self-join \( L_{k-1} \)
  - \( \text{INSERT INTO} \ C_k \)
  - \( \text{SELECT p.item, p.item2, ..., p.item}_{k-1}, q.item_{k-1} \)
  - \( \text{FROM} \ L_{k-1} p, L_{k-1} q \)
  - \( \text{WHERE} p.item = q.item, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1} \)
- Step 2: pruning
  - For each itemset \( c \) in \( C_k \) do
    - For each \((k-1)\)-subset \( s \) of \( c \) do if \((s \text{ is not in } L_{k-1})\) then delete \( c \) from \( C_k \)

How to Count Supports?

- Why is counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method
  - Candidate itemsets are stored in a hash-tree
  - A leaf node of hash-tree contains a list of itemsets and counts
  - Interior node contains a hash table
  - Subset function: finds all the candidates contained in a transaction
Example: Counting Supports

- Subset function
  - 1, 4, 7
  - 2, 5, 8
  - 3, 6, 9

Transaction: 1 2 3 5 6
- 1 + 2 3 5 6
- 1 2 + 3 5 6
- 1 3 + 5 6

Apriori in SQL

- Impossible to get good performance out of pure SQL (SQL-92) based approaches alone
  - Support counting is costly
- Make use of object-relational extensions like UDFs, BLOBs, Table functions etc.
  - Get orders of magnitude improvement
- S. Sarawagi, S. Thomas, and R. Agrawal, 1998

Challenges of Freq Pat Mining

- Multiple scans of transaction database
- Huge number of candidates
- Tedious workload of support counting for candidates

Improving Apriori: Ideas

- Reducing the number of transaction database scans
- Shrinking the number of candidates
- Facilitating support counting of candidates

DIC: Reducing Number of Scans

- Once both A and D are determined frequent, the counting of AD can begin
- Once all length-2 subsets of BCD are determined frequent, the counting of BCD can begin

DHP: Reducing # of Candidates

- A hashing bucket count < min_sup → every candidate in the bucket is infrequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} ...
  - Large 1-itemset: a, b, d, e
  - The sum of counts of {ab, ad, ae} < min_sup → ab should not be a candidate 2-itemset
- J. Park, M. Chen, and P. Yu, SIGMOD’ 95
  - DHP: Direct Hashing and Pruning
A 2-Scan Method by Partitioning

- Partition the database into n partitions, such that each partition can be held into main memory
- Itemset X is frequent \( \Rightarrow \) X must be frequent in at least one partition
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- All local frequent itemsets can be held in main memory? A sometimes too strong assumption
- A. Savasere, E. Omiecinski, and S. Navathe, VLDB’95

Sampling for Frequent Patterns

- Select a sample of the original database, mine frequent patterns in the sample using Apriori
- Scan database once more to verify frequent itemsets found in the sample, only borders of closure of frequent patterns are checked
  - Example: check abcd instead of ab, ac, ..., etc.
- Scan database again to find missed frequent patterns
- H. Toivonen, VLDB’96

Eclat/MaxEclat and VIPER

- Tid-list: the list of transaction-ids containing an itemset
  - Vertical Data Format
- Major operation: intersections of tid-lists
- Compression of tid-lists
  - Itemset A: t1, t2 t3, sup(A)=3
  - Itemset B: t2, t3, t4, sup(B)=3
  - Itemset AB: t2, t3, sup(AB)=2
- M. Zaki et al., 1997
- P. Shenoy et al., 2000

Bottleneck of Freq Pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many scans and generates many candidates
  - To find frequent itemset \( i_1, i_2, \ldots, i_{100} \)
    - \# of scans: 100
    - \# of Candidates: \( \binom{100}{1} \binom{100}{2} \cdots \binom{100}{100} = 2^{100} \approx 1.2 \times 10^{30} \)
  - Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

Search Space of Freq. Pat. Mining

- Itemsets form a lattice

Set Enumeration Tree

- Use an order on items, enumerate itemsets in lexicographic order
  - a, ab, abc, abcd, ac, acd, ad, b, bc, bcd, bd, c, dc, d
- Reduce a lattice to a tree
Borders of Frequent Itemsets

- Frequent itemsets are connected
  - ∅ is trivially frequent
  - X on the border → every subset of X is frequent

Projected Databases

- To test whether Xy is frequent, we can use the X-projected database
  - The sub-database of transactions containing X
  - Check whether item y is frequent in X-projected database

Compress Database by FP-tree

- The 1st scan: find frequent items
  - Only record frequent items in FP-tree
  - F-list: f-c-a-b-m-p
- The 2nd scan: construct tree
  - Order frequent items in each transaction w.r.t. f-list
  - Explore sharing among transactions

Benefits of FP-tree

- Completeness
  - Never break a long pattern in any transaction
  - Preserve complete information for freq pattern mining
    - Not scan database anymore
- Compactness
  - Reduce irrelevant info — infrequent items are removed
  - Items in frequency descending order (f-list): the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not counting node-links and the count fields)

Partitioning Frequent Patterns

- Frequent patterns can be partitioned into subsets according to 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, or p
  - Pattern f
- Depth-first search of a set enumeration tree
  - The partitioning is complete and does not have any overlap

Find Patterns Having Item “p”

- Only transactions containing p are needed
- Form p-projected database
  - Starting at entry p of the header table
  - Follow the side-link of frequent item p
  - Accumulate all transformed prefix paths of p
Find Pat Having Item m But No p

- Form m-projected database TDB|m
  - Item p is excluded (why?)
  - Contain fca:2, fcab:1
  - Local frequent items: f, c, a
- Build FP-tree for TDB|m
  - root
  - f:4
c:
a:m
  - p:2
  - b:1

Recursive Mining

- Patterns having m but no p can be mined recursively
- Optimization: enumerate patterns from a single-branch FP-tree
  - Enumerate all combination
  - Support = that of the last item
    - m, fm, cm, am
    - fcm, fam, cam
    - fcam

Enumerate Patterns From Single Prefix of FP-tree

- A (projected) FP-tree has a single prefix
  - Reduce the single prefix into one node
  - Join the mining results of the two parts

Scaling up by DB Projection

- What if an FP-tree cannot fit into memory?
- Database projection
  - Partition a database into a set of projected databases
  - Construct and mine FP-tree once the projected database can fit into main memory
  - Heuristic: Projected database shrinks quickly in many applications

FP-growth

- Pattern-growth: recursively grow frequent patterns by pattern and database partitioning
- Algorithm
  - For each frequent item, construct its projected database, and then its projected FP-tree
  - Repeat the process on each newly created projected FP-tree
  - Until the resulted FP-tree is empty, or contains only one path – single path generates all the combinations, each of which is a frequent pattern

Parallel vs. Partition Projection

- Parallel projection: form all projected database at a time
- Partition projection: propagate projections
Why Is FP-growth Efficient?

- **Divide-and-conquer strategy**
  - Decompose both the mining task and DB
  - Lead to focused search of smaller databases
- **Other factors**
  - No candidate generation nor candidate test
  - Database compression using FP-tree
  - No repeated scan of entire database
  - Basic operations – counting local frequent items and building FP-tree, no pattern search nor pattern matching

Major Costs in FP-growth

- **Poor locality of FP-trees**
  - Low hit rate of cache
- **Building FP-trees**
  - A stack of FP-trees
- **Redundant information**
  - Transaction abcd appears in a-, ab-, abc-, ac-, ..., c- projected databases and FP-trees

Improving Locality

- Store FP-trees in pre-order depth-first traverse list

H-Mine

- **Goal**: efficient in various occasions
  - Dense vs. sparse, huge vs. memory-based data sets
- **Moderate in space requirement**
- **Highlights**
  - Effective and efficient memory-based structure and mining algorithm
  - Scalable algorithm for mining large databases by proper partitioning
  - Integration of H-mine and FP-growth

H-Structure

- Store frequent-item projections in main memory
  - Items in a transaction are sorted according to f-list
  - Each frequent item in a transaction is stored with two fields: item-id and hyper-link
  - Header table H
- **Link transactions with same first item**
- Scan database once

Find Patterns Containing Item “a”

- Only search a-projected database: transactions containing “a”
- The a-queue links all transactions in a-projected database
  - Can be traversed efficiently
Mining a-Projected Database

- Build a-header table \(H_a\)
- Traverse a-queue once, find all local frequent items within a-projected database
  - Local freq items: c, d, and e
  - Patterns: ac, ad and ae
- Link transactions having same next frequent item

Mining in Large Databases

- What if the H-struct is too big for memory?
- Find global frequent items
- Partition the database into n parts
  - The H-struct for each part can be held into memory
  - Mine local patterns in each part using H-mine(Mem)
    - Use relative minimum support threshold
- Consolidate global patterns in the third scan

Mining Dense Projected DB’s

- Challenges in dense datasets
  - Long patterns
  - Some patterns appearing in many transactions
- After projection, projected databases are denser
- Advantages of FP-tree
  - Compress dense databases many times
  - No candidate generation
  - Sub-patterns can be enumerated from long patterns
- Build FP-tree for dense projected databases
  - Empirical switching point: 1%

Why Is H-Mine(Mem) Efficient?

- No candidate generation
  - It is a pattern growth method
- Search confined in a dedicated space
  - Not physically construct memory structures for projected databases
  - H-struct is for all the mining
  - Information about projected databases are collected in header tables
- No frequent patterns stored in main memory

How to Partition in H-mine?

- Partitioning in H-mine is straightforward
  - Overhead of header tables in H-mine(Mem) is small and predictable
  - Partitioning with Apriori is not easy
    - Hard to predict the space requirement of Apriori
- Global frequent items prune many local patterns in skewed partitions

Advantages of H-Mine

- Have very small space overhead
- Absorb nice features of FP-growth
- Create no physical projected database
- Watch the density of projected databases, turn to FP-growth when necessary
- Propose space-preserving mining
  - Scalable in very large database
  - Feasible even with very small memory
  - Go beyond frequent pattern mining
Further Developments

- OP – opportunistic projection (LPWH02)
  - Opportunistically choose between array-based and tree-based representations of projected databases
- Diffsets for vertical mining (ZaGo03)
  - Only record the differences in the tids of a candidate pattern from its generating frequent patterns

Effectiveness of Freq Pat Mining

- Too many patterns!
  - A pattern $a_1a_2...a_n$ contains $2^n-1$ subpatterns
  - Understanding many patterns is difficult or even impossible for human users
- Non-focused mining
  - A manager may be only interested in patterns involving some items (s)he manages
  - A user is often interested in patterns satisfying some constraints

Borders and Max-patterns

- Max-patterns: borders of frequent patterns
  - Any subset of max-pattern is frequent
  - Any superset of max-pattern is infrequent
  - Cannot generate rules

MaxMiner: Mining Max-patterns

- 1st scan: find frequent items
  - A, B, C, D, E
- 2nd scan: find support for
  - AB, AC, AD, AE, ABCDE
  - BC, BD, BE, BCDE
  - CD, CE, CDE, DE,
  - Since BCDE is a max-pattern, no need to check BCD, BDE, CDE in later scan

Bayardo, SIGMOD ’98
Patterns and Support Counts

- ABC: 2
- ABD: 2
- ACD
- BCD

- AB: 3
- AC: 2
- AD: 3
- BC: 2
- BD: 2
- CD: 2

- A: 4
- B: 4
- C: 3
- D: 4

Frequent Closed Patterns

- For frequent itemset X, if there exists no item y not in X s.t. every transaction containing X also contains y, then X is a frequent closed pattern
  - "acdf" is a frequent closed pattern

- Concise rep. of freq pats
  - Can generate non-redundant rules
- Reduce # of patterns and rules
- N. Pasquier et al. In ICDT ’99

CLOSET for Frequent Closed Patterns

- Flist: list of all freq items in support asc. order
  - Flist: d-a-f-e-c
- Divide search space
  - Patterns having d
  - Patterns having d but no a, etc.
- Find frequent closed pattern recursively
  - Every transaction having d also has cfa → cfad is a frequent closed pattern
- PHM’00

The CHARM Method

- Use vertical data format: t(AB)={T1, T12, ...}
- Derive closed pattern based on vertical intersections
  - t(X)=t(Y): X and Y always happen together
  - t(X)⊂t(Y): transaction having X always has Y
- Use diffset to accelerate mining
  - Only keep track of difference of tids
- t(X)={T1, T2, T3}, t(Xy )={T1, T3}
  - Diffset(Xy, X)={T2}

Closed and Max-patterns

- Closed pattern mining algorithms can be adapted to mine max-patterns
  - A max-pattern must be closed
- Depth-first search methods have advantages over breadth-first search ones
  - Why?

Condensed Freq Pattern Base

- Practical observation: in many applications, a good approximation on support count could be good enough
  - Support=10000 → Support in range 10000 ± 1%
- Making frequent pattern mining more realistic
  - A small deviation has a minor effect on analysis
  - Condensed FP-base leads to more effective mining
  - Computing a condensed FP-base may lead to more efficient mining
Condensed FP-base Mining

- Compute a condensed FP-base with a guaranteed maximal error bound.
- Given: a transaction database, a user-specified support threshold, and a user-specified error bound
- Find a subset of frequent patterns & a function
  - Determine whether a pattern is frequent
  - Determine the support range
- Pei et al. ICDM ’02

An Example

Support threshold: min\_sup = 1
Error bound: k = 2

Approximation Functions

- NOT unique
  - Different condensed FP-bases have different approximation function
- Optimization on space requirement
  - The less space required, the better compression effect
  - compression ratio
    \[ \delta = \frac{\text{# of patterns in the condensed FP-base}}{\text{total # of frequent patterns}} \]

Constraint-based Data Mining

- Find all the patterns in a database autonomously?
  - The patterns could be too many but not focused!
- Data mining should be interactive
  - User directs what to be mined
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - System optimization: push constraints for efficient mining

Constraints in Data Mining

- Knowledge type constraint
  - classification, association, etc.
- Data constraint — using SQL-like queries
  - find product pairs sold together in stores in New York
- 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: support and confidence
Constrained Mining vs. Search

- Constrained mining vs. constraint-based search
  - Both aim at reducing search space
  - Finding all patterns vs. some (or one) answers satisfying constraints
  - Constraint-pushing vs. heuristic search
  - An interesting research problem on integrating both
- Constrained mining vs. DBMS query processing
  - Database query processing requires to find all
  - Constrained pattern mining shares a similar philosophy as pushing selections deeply in query processing

Optimization

- Mining frequent patterns with constraint C
  - Sound: only find patterns satisfying the constraints C
  - Complete: find all patterns satisfying the constraints C
- A naïve solution
  - Constraint test as a post-processing
- More efficient approaches
  - Analyze the properties of constraints
  - Push constraints as deeply as possible into frequent pattern mining

Anti-Monotonicity

- Anti-monotonicity
  - An itemset S violates the constraint, so does any of its superset
  - sum(S.Price) ≤ v is anti-monotone
  - sum(S.Price) ≥ v is not anti-monotone
- Example
  - C: range(S.profit) ≤ 15
  - Itemset ab violates C
  - So does every superset of ab

Monotonicity

- Monotonicity
  - An itemset S satisfies the constraint, so does any of its superset
  - sum(S.Price) ≥ v is monotone
  - min(S.Price) ≤ v is monotone
- Example
  - C: range(S.profit) ≥ 15
  - Itemset ab satisfies C
  - So does every superset of ab
Converting “Tough” Constraints

- Convert tough constraints into anti-monotone or monotone by properly ordering items.
- Examine C: \( \text{avg}(S.\text{profit}) \geq 25 \)
  - Order items in value-descending order:
    - \(<a, f, g, d, b, h, c, e>\)
  - If an itemset \( abf \) violates C:
    - So does \( abfh \), \( abf^* \)
  - It becomes anti-monotone!

Convertible Constraints

- Let \( R \) be an order of items.
- Convertible anti-monotone
  - If an itemset \( S \) violates a constraint \( C \), so does every itemset having \( S \) as a prefix w.r.t. \( R \).
- Convertible monotone
  - If an itemset \( S \) satisfies constraint \( C \), so does every itemset having \( S \) as a prefix w.r.t. \( R \).

Strongly Convertible Constraints

- \( \text{avg}(X) \geq 25 \) is convertible anti-monotone w.r.t. item value descending order \( R: <a, f, g, d, b, h, c, e> \)
  - Itemset \( af \) violates constraint \( C \), so does every itemset with \( af \) as prefix, such as \( afd \).
- \( \text{avg}(X) \geq 25 \) is convertible monotone w.r.t. item value ascending order \( R^-1: <e, c, h, b, d, g, f, a> \)
  - Itemset \( d \) satisfies constraint \( C \), so does itemsets \( df \) and \( dfa \), which having \( d \) as a prefix.
  - Thus, \( \text{avg}(X) \geq 25 \) is strongly convertible.

Can Apriori Handle Convertible Constraint?

- A convertible, not monotone nor anti-monotone nor succinct constraint cannot be pushed deep into the an Apriori mining algorithm.
  - Within the level wise framework, no direct pruning based on the constraint can be made.
  - Itemset \( df \) violates constraint \( C: \text{avg}(X)\geq25 \)
    - Since \( adf \) satisfies \( C \), Apriori needs \( df \) to assemble \( adf \), cannot be pruned.
  - But it can be pushed into frequent-pattern growth framework!

Mining With Convertible Constraints

- \( C: \text{avg}(S.\text{profit}) \geq 25 \)
- List of items in every transaction in value descending order R:
  - \(<a, f, g, d, b, h, c, e>\)
    - \( C \) is convertible anti-monotone w.r.t. \( R \)
- Scan transaction DB once
  - remove infrequent items
    - Item \( h \) in transaction 40 is dropped
  - Itemsets \( a \) and \( f \) are good

Jian Pei: Big Data Analytics -- Frequent Pattern Mining
**Leverage**

- The difference between the observed and expected joint probability of X and Y are independent and are independent.

- An "absolute" measure of the surprisingness of a rule.

- Should be used together with lift.

**Correlation and Lift**

- \( P(B|A)/P(B) \) is called the lift of rule \( A \rightarrow B \).

- The difference between the observed and expected joint probability of \( XY \) assuming \( X \) and \( Y \) are independent.

**Conviction**

- The expected error of a rule.

- Consider not only the joint distribution of \( X \) and \( Y \).

**Property of Lift**

- If \( A \) and \( B \) are independent, lift = 1.

- If \( A \) and \( B \) are positively correlated, lift > 1.

- If \( A \) and \( B \) are negatively correlated, lift < 1.

- Limitation: lift is sensitive to \( P(A) \) and \( P(B) \).

**Evaluation Criteria**

- Objective interestingness measures:
  - Examples: support, patterns formed by mutually independent items.
  - Examples: domain knowledge, templates.

- Subjective interestingness measures:
  - Examples: domain knowledge, templates.

- Not Every Pattern Is Interesting!

- Trivial patterns:
  - Pregnant \( \rightarrow \) Female 100% confidence
  - Play basketball \( \rightarrow \) eat cereal [40%, 66%]

- Misleading patterns:
  - Example shown in Table 6.8, in which many potentially interesting patterns involving independent items might be eliminated by the support threshold.

**Limitations of the Support-Confidence Framework**

- Many potentially interesting patterns involving independent items might be eliminated by the support threshold.

- The draw-

**Limitations of Interest Factor**

- Should be used together with lift.

- Misleading patterns:
  - Example shown in Table 6.8, in which many potentially interesting patterns involving independent items might be eliminated by the support threshold.

- The draw-

**Definition of Contingency Table**

- A contingency table denotes a frequency count. For example, \( A \times B \) to indicate that A and B are independent.
Odds Ratio

\[
\text{odds}(Y \mid X) = \frac{P(\bar{X}Y)}{P(\bar{X})} = \frac{P(XY)}{P(X)} = \frac{P(\bar{X})}{P(\bar{X})}
\]

\[
\text{odds}(Y \mid \bar{X}) = \frac{P(\bar{X}Y)}{P(\bar{X})} = \frac{P(X\bar{Y})}{P(X)} = \frac{P(\bar{X})}{P(\bar{X})}
\]

\[
\text{oddsratio}(X \rightarrow Y) = \frac{\text{odds}(Y \mid X)}{\text{odds}(Y \mid \bar{X})} = \frac{P(XY) \cdot P(\bar{X})}{P(X) \cdot P(\bar{X})}
\]

\[
\chi^2
\]

- Suppose attribute A has c distinct values \(a_1, \ldots, a_c\) and attribute B has r distinct values \(b_1, \ldots, b_r\).
- The \(\chi^2\) value (Pearson \(\chi^2\) statistics) is

\[
\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}
\]

- \(o_{ij}\) and \(e_{ij}\) are the observed frequency and the expected frequency, respectively, of the joint event \(a_ib_j\), respectively.

Example

\[
\chi^2 = \frac{(2000 - 2250)^2}{2250} + \frac{(1750 - 1500)^2}{1500} + \frac{(1000 - 750)^2}{750} + \frac{(250 - 500)^2}{500} = 277.8
\]

- The \(\chi^2\) value is greater than 1
- \(\text{count}(\text{basketball, cereal}) = 2000 < \text{expectation (2250)} \Rightarrow \text{play basketball and eating cereal are negatively correlated}\)

<table>
<thead>
<tr>
<th></th>
<th>Basketball</th>
<th>Not basketball</th>
<th>Sum (row)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cereal</td>
<td>2000</td>
<td>1750</td>
<td>3750</td>
</tr>
<tr>
<td>Not cereal</td>
<td>1000</td>
<td>250</td>
<td>1250</td>
</tr>
<tr>
<td>Sum (col.)</td>
<td>3000</td>
<td>2000</td>
<td>5000</td>
</tr>
</tbody>
</table>

\[
\Phi\text{-coefficient}
\]

\[
\phi = \frac{P(AB)P(\bar{AB}) - P(\bar{AB})P(AB)}{\sqrt{P(A)P(\bar{A})P(B)P(\bar{B})}}
\]

- \(-1: \text{if } A \text{ and } B \text{ perfectly negatively correlated}\)
- \(1: \text{if } A \text{ and } B \text{ perfectly positively correlated}\)
- \(0: \text{if } A \text{ and } B \text{ statistically independent}\)
- Drawback: \(\Phi\)-coefficient puts the same weight on co-occurrence and co-absence

IS Measure

\[
IS(A, B) = \sqrt{\text{conf}(A \mid B) \cdot \text{conf}(B \mid A)} = \frac{P(A|B)}{\sqrt{P(A)P(B)}}
\]

- Biased on frequent co-occurrence
- Equivalent to cosine similarity for binary variables (bit vectors)
- Geometric mean of rules between a pair of binary random variables

\[
IS(A, B) = \sqrt{\frac{P(A|B)P(\bar{A}|\bar{B})}{P(B)P(\bar{B})}} = \sqrt{\min(1 - \text{sym}(A \leftrightarrow B))}
\]

- Drawback: the value depends on \(P(A)\) and \(P(B)\)
- Similar drawbacks in lift

More Measures

- All confidence: \(\min\{P(A|B), P(B|A)\}\)
- Max confidence: \(\max\{P(A|B), P(B|A)\}\)
- The Kulczynski measure: \(\frac{1}{2} (P(A|B) + P(B|A))\)
Comparing Measures

### Properties of Measures

- Symmetry: is M(A → B) = M(B → A)
- Null-transaction dependent (null addition invariant): is ~A → B used in the measure?
- Inversion invariant: the value does not change if f_{i1} and f_{10} are exchanged with f_{00} and f_{11}
- Scaling: whether the measure remains if the number of null-transactions can outweigh the number of transactions containing any particular itemset

### Measuring More Random Variables

- Some measures, such as lift and statistical independence, can be extended
  \[ I = \frac{N^{k-1} \sum f_{i_1 \cdots i_k}}{\sum f_{i_1} + \sum f_{i_2} + \cdots + \sum f_{i_k}} \]
- \[ PS = \frac{\sum f_{i_1 \cdots i_k}}{N} - \frac{f_{i_1} + f_{i_2} + \cdots + f_{i_k}}{N^k} \]

### Simpson’s Paradox

- A trend that appears in different groups of data disappears when these groups are combined, and the reverse trend appears for the aggregate data
  - Also known as Yule-Simpson effect
  - Often encountered in social-science and medical-science statistics
  - Particularly confounding when frequency data are unduly given causal interpretations

### Imbalance Ration

- Assess the imbalance of two itemsets A and B in rule implications
  \[ IR(A, B) = \frac{|P(A) - P(B)|}{P(A) + P(B) - P(A \cup B)} \]
Kidney Stone Treatment Example

<table>
<thead>
<tr>
<th>Stone Type</th>
<th>Treatment A</th>
<th>Treatment B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>G1 81/87=96%</td>
<td>G2 234/270=87%</td>
</tr>
<tr>
<td>Large</td>
<td>G3 192/263=73%</td>
<td>G4 55/80=69%</td>
</tr>
<tr>
<td>Overall</td>
<td>273/350=78%</td>
<td>289/350=83%</td>
</tr>
</tbody>
</table>

- Which treatment, A or B, is better?

Berkeley Gender Bias Case

<table>
<thead>
<tr>
<th>Department</th>
<th>Men</th>
<th>Women</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>825</td>
<td>44%</td>
</tr>
<tr>
<td>B</td>
<td>560</td>
<td>63%</td>
</tr>
<tr>
<td>C</td>
<td>325</td>
<td>37%</td>
</tr>
<tr>
<td>D</td>
<td>417</td>
<td>33%</td>
</tr>
<tr>
<td>E</td>
<td>191</td>
<td>28%</td>
</tr>
<tr>
<td>F</td>
<td>272</td>
<td>6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applicants</th>
<th>Admitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Men</td>
<td>8442</td>
</tr>
<tr>
<td>Women</td>
<td>4321</td>
</tr>
</tbody>
</table>

Fisher Exact Test

- Directly test whether a rule \(X \rightarrow Y\) is productive by comparing its confidence with those of its generalizations \(W \rightarrow Y\), where \(W = X U Z\)
- Let \(X = W U Z\)

<table>
<thead>
<tr>
<th>(W)</th>
<th>(Y)</th>
<th>(\neg Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Z)</td>
<td>(a)</td>
<td>(b)</td>
</tr>
<tr>
<td>(\neg Z)</td>
<td>(c)</td>
<td>(d)</td>
</tr>
<tr>
<td>(a + c)</td>
<td>(b + d)</td>
<td>(Sup(W))</td>
</tr>
</tbody>
</table>

\[ a = sup(WZY) = sup(XY), b = sup(WZY) = sup(X\neg Y) \]
\[ c = sup(W\neg ZY) = sup(W\neg Y), d = sup(W\neg ZY) \]
\[ \text{oddratio} = \frac{ad}{bc} = 1 \]

Marginals

- Row marginals:
  \[ a + b = sup(WZ) = sup(X), c + d = sup(W\neg Z) \]
- Column marginals:
  \[ a + c = sup(WY), b + d = sup(W\neg Y) \]

\[ \text{oddratio} = \frac{ad}{bc} = 1 \]

Hypothesis

- \(H_0: Z\) and \(Y\) are independent given \(W\)
- \(X \rightarrow Y\) is not productive given \(W \rightarrow Y\)
- If \(Z\) and \(Y\) are independent, then
  \[ a = \frac{(a + b)(a + c)}{n}, b = \frac{(a + b)(b + d)}{n}, c = \frac{(c + d)(a + c)}{n}, d = \frac{(c + d)(b + d)}{n} \]
  \[ \text{oddratio} = \frac{ad}{bc} = 1 \]

Relation between \(a\) and \(b\), \(c\), \(d\)

- Assumption: the row and column marginals are fixed
- The value of \(a\) uniquely determines \(b\), \(c\), and \(d\)

<table>
<thead>
<tr>
<th>(W)</th>
<th>(Y)</th>
<th>(\neg Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Z)</td>
<td>(a)</td>
<td>(b)</td>
</tr>
<tr>
<td>(\neg Z)</td>
<td>(c)</td>
<td>(d)</td>
</tr>
<tr>
<td>(a + c)</td>
<td>(b + d)</td>
<td>(Sup(W))</td>
</tr>
</tbody>
</table>
Probability Mass Function of a

• The probability mass function of observing the value of a in the contingency table is given by the Hypergeometric distribution.

\[
P(a \mid S, T) = \binom{S}{a} \binom{T-S}{t-a}
\]

Calculating the p-value

• Assuming that the null hypothesis is true, the p-value is the probability of obtaining a test statistic at least as extreme as the one actually observed.

\[
p-value(a) = \sum_{i=0}^{\min(b,c)} P(a + i \mid a + c, a + b, n)
\]

• If the p-value is very small (e.g., 0.01), the null hypothesis can be rejected.

Compute p-value on Statistics

• The empirical cumulative distribution function

\[
\hat{F}(x) = \frac{1}{x} \sum_{i=1}^{x} I(\theta_i \leq x)
\]

\[
p-value(\theta) = 1 - \hat{F}(\theta)
\]

Probability Mass Function of a

• An occurrence of Z, a success.

\[
P(a \mid a + c, a + b, n) = \binom{a + b}{a} \binom{n-(a+b)}{a+c} \binom{n}{a+c} \binom{n}{a}\binom{n}{b+d} / \binom{n}{a+c}\binom{n}{b+d}
\]

Permutation (Randomization) Test

• Determine the distribution of a given test statistic by randomly modifying the observed data several times to obtain a random sample of the data sets.

• The modified data sets are used for significance testing.

• Compute the empirical probability mass function (EPMF).

• Generate the empirical cumulative distribution function.

Swap Randomization

• In permutation test, what characteristics should be preserved in permutation?

• Swap randomization keep the column and row marginals invariant.

• The support of each item does not change.

• The length of each transaction does not change.

• Swap two items in two transactions.

• Conduct a certain number of swaps to make up a new data set.
Bootstrap Sampling

- A transaction database is just a sample from a larger population
  - What is the frequency (or, range of possible frequency) of X in the underlying population?
- Given a test assessment statistic \( \theta \), how can we infer the confidence interval for the possible values of \( \theta \) at a desired confidence interval \( \alpha \)?
- Bootstrap sampling: sampling with replacement

Calculating Statistic Range

\[
\hat{F}(x) = \hat{P}(\theta \leq x) = \frac{1}{k} \sum_{i=1}^{k} I(\theta_i \leq x)
\]

Let \( v_{1-\alpha} = \hat{F}^{-1}\left(\frac{1-\alpha}{2}\right) \)
\( v_{1+\alpha} = \hat{F}^{-1}\left(\frac{1+\alpha}{2}\right) \)

\[
P(\theta \in [v_{1-\alpha}, v_{1+\alpha}]) = \hat{F}\left(\frac{1+\alpha}{2}\right) - \hat{F}\left(\frac{1-\alpha}{2}\right) = \frac{1+\alpha}{2} - \frac{1-\alpha}{2} = \alpha
\]

Thus, the \( \alpha \) confidence interval for test statistic \( \theta \) is \([v_{1-\alpha}, v_{1+\alpha}]\)

From Itemsets to Sequences

- Itemsets: combinations of items, no temporal order
- Temporal order is important in many situations
  - Time-series databases and sequence databases
  - Frequent patterns \( \rightarrow \) (frequent) sequential patterns
- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then iPod, and then digital camera, within 3 months.
  - Medical treatment, natural disasters, science and engineering processes, stocks and markets, telephone calling patterns, Web log clickthrough streams, DNA sequences and gene structures

What Is Sequential Pattern Mining?

- Given a set of sequences, find the complete set of frequent subsequences

Apriori Property of Seq Patterns

- Apriori property in sequential patterns
  - If a sequence \( S \) is infrequent, then none of the sub-sequences of \( S \) is frequent
  - E.g., \( <hb> \) is infrequent \( \rightarrow \) so do \( <hab> \) and \( <ah>b> \)

Challenges in Seq Pat Mining

- A huge number of possible sequential patterns are hidden in databases
- A mining algorithm should
  - Find the complete set of patterns satisfying the minimum support (frequency) threshold
  - Be highly efficient, scalable, involving only a small number of database scans
  - Be able to incorporate various kinds of user-specific constraints

Apriori Property of Seq Patterns

<table>
<thead>
<tr>
<th>Seq ID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;bdj&gt;abc&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;bfj&gt;bfjg&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;lah&gt;afad&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;be&gt;ycs&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;eabj&gt;ade&gt;</td>
</tr>
</tbody>
</table>

Given support threshold \( \text{min}_\text{sup} = 2 \), \( <ab>bc > \) is a sequential pattern

<table>
<thead>
<tr>
<th>Seq ID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;bdj&gt;abc&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;bfj&gt;bfjg&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;lah&gt;afad&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;be&gt;ycs&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;eabj&gt;ade&gt;</td>
</tr>
</tbody>
</table>
GSP

- GSP (Generalized Sequential Pattern) mining
- Outline of the method
  - Initially, every item in DB is a candidate of length-1
  - For each level (i.e., sequences of length-k) do
    - Scan database to collect support count for each candidate sequence
    - Generate candidate length-(k+1) sequences from length-k frequent sequences using Apriori
  - Repeat until no frequent sequence or no candidate can be found
- Major strength: Candidate pruning by Apriori

Finding Len-1 Seq Patterns

- Initial candidates
  - \(<a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>\)

- Scan database once
  - count support for candidates

<table>
<thead>
<tr>
<th>Seq-id</th>
<th>Sequence</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;bdjcb(ac)&gt;</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>&lt;bt(cedtbj)&gt;</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ayנות&gt;</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>&lt;be(ce)d&gt;</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>&lt;abijbceda&gt;</td>
<td>2</td>
</tr>
</tbody>
</table>

Generating Length-2 Candidates

51 length-2 Candidates

Without Apriori property, 8*8+8*7/2=92 candidates
Apriori prunes 44.57% candidates

Finding Len-2 Seq Patterns

- Scan database one more time, collect support count for each length-2 candidate
- There are 19 length-2 candidates which pass the minimum support threshold
  - They are length-2 sequential patterns

Generating Length-3 Candidates and Finding Length-3 Patterns

- Generate Length-3 Candidates
  - Self-join length-2 sequential patterns
    - \(<ab>, <ac> and <ba> are all length-2 sequential patterns \(\rightarrow\) <aba> is a length-3 candidate
    - \(<bd>, <bb> and <db> are all length-2 sequential patterns \(\rightarrow\) <bdj> is a length-3 candidate
  - 46 candidates are generated
- Find Length-3 Sequential Patterns
  - Scan database once more, collect support counts for candidates
  - 19 out of 46 candidates pass support threshold

The GSP Mining Process

3rd scan: 1 cand. 1 length-5 seq. pat.
4th scan: 6 cand. 6 length-4 seq. pat.
5th scan: 46 cand. 19 length-3 seq. pat.
6th scan: 20 cand. not in DB at all
7th scan: 51 cand. 10 length-2 seq. pat.
8th scan: 20 cand. not in DB at all
9th scan: 8 cand. 6 length-1 seq. pat.

<table>
<thead>
<tr>
<th>Seq-id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;bdjcb(ac)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;bt(cedtbj)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ayנות&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;be(ce)d&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;abijbceda&gt;</td>
</tr>
</tbody>
</table>
The GSP Algorithm

- Take sequences in form of \(<x>\) as length-1 candidates
- Scan database once, find \(F_1\), the set of length-1 sequential patterns
- Let \(k=1\); while \(F_k\) is not empty do
  - Form \(C_{k+1}\), the set of length-(\(k+1\)) candidates from \(F_k\);
  - If \(C_{k+1}\) is not empty, scan database once, find \(F_{k+1}\), the set of length-(\(k+1\)) sequential patterns
  - Let \(k=k+1\);

Bottlenecks of GSP

- A huge set of candidates
  - 1,000 frequent length-1 sequences generate length-2 candidates!
- Multiple scans of database in mining
- Real challenge: mining long sequential patterns
  - An exponential number of short candidates
  - A length-100 sequential pattern needs \(10^{30}\) candidate sequences!

FreeSpan: Freq Pat-projected Sequential Pattern Mining

- The itemset of a seq pat must be frequent
  - Recursively project a sequence database into a set of smaller databases based on the current set of frequent patterns
  - Mine each projected database to find its patterns

FreeSpan to PrefixSpan

- Freespan:
  - Projection-based: no candidate sequence needs to be generated
  - But, projection can be performed at any point in the sequence, and the projected sequences may not shrink much
- PrefixSpan
  - Projection-based
  - But only prefix-based projection: less projections and quickly shrinking sequences

Prefix and Suffix (Projection)

- \(<a>\), \(<aa>\), \(<ab>\) and \(<abc>\) are prefixes of sequence \(<a(abc)(ac)d(cf)>\)
- Given sequence \(<a(abc)(ac)d(cf)>\)

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Suffix (Prefix-Based Projection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a&gt;)</td>
<td>(&lt;abc&gt;(ac)d(cf)&gt;)</td>
</tr>
<tr>
<td>(&lt;aa&gt;)</td>
<td>(&lt;abc&gt;(ac)d(cf)&gt;)</td>
</tr>
<tr>
<td>(&lt;ab&gt;)</td>
<td>(&lt;bc&gt;(ac)d(cf)&gt;)</td>
</tr>
</tbody>
</table>

Mining Sequential Patterns by Prefix Projections

- Step 1: find length-1 sequential patterns
  - \(<a>\), \(<b>\), \(<c>\), \(<d>\), \(<e>\), \(<f>\)
- Step 2: divide search space. The complete set of seq. pat. can be partitioned into 6 subsets:
  - The ones having prefix \(<a>\)
  - The ones having prefix \(<b>\)
  - The ones having prefix \(<c>\)
  - The ones having prefix \(<d>\)
  - The ones having prefix \(<e>\)
  - The ones having prefix \(<f>\)
Finding Seq. Pat. with Prefix <a>

- Only need to consider projections w.r.t. <a>
  - <a>-projected database: <(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)c>, <(_f)c>
- Find all the length-2 seq. pat. having prefix <a>: <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
  - Further partition into 6 subsets
    - Having prefix <aa>:
    - Having prefix <af>

Completeness of PrefixSpan

Efficiency of PrefixSpan

- No candidate sequence needs to be generated
- Projected databases keep shrinking
- Major cost of PrefixSpan: constructing projected databases
  - Can be improved by bi-level projections

Effectiveness

- Redundancy due to anti-monotonicity
  - {<abcd>} leads to 15 sequential patterns of same support
- Closed sequential patterns and sequential generators
- Constraints on sequential patterns
  - Gap
  - Length
  - More sophisticated, application oriented constraints

Sequences and Partial Orders

Why Frequent Orders?

- Frequent orders capture more thorough information than sequential patterns
- Many important applications
  - Bioinformatics: order-preserving clustering of microarray data
  - Web mining and market basket analysis: modeling customer purchase behaviors
  - Network management and intrusion detection: frequent routing paths, signatures for intrusions
  - Preference-based services: partial orders from ranking data
Why Mining Orders Difficult?

- Use sequential patterns to assemble frequent partial orders?
  - One frequent closed partial order may summarize a few sequential patterns
  - Assembling can be costly

Sequential patterns:
- CHK → MMK → MORT → RESP
- CHK → MMK → MORT → BROK
- CHK → RRSP → MORT → RESP
- CHK → RRSP → MORT → BROK
- CHK → MORT → RESP

Model

- A sequence $s$ induces a full order $R_1$, if $R_1 \rightarrow R_2$, where $R_2$ is a partial order, then $R_1$ is said to support $R_2$
- The support of a partial order $R$ in a sequence database is the number of sequences supporting $R$ in the database
- An order $R$ is closed if there exists no any $R' \rightarrow R$ and $sup(R) = sup(R')$
- Given a minimum support threshold, order $R$ is a frequent closed partial order if it is closed and passes the support threshold

Ideas

- Depth-first search to generate frequent closed partial orders in transitive reduction
  - Transitive reduction is a succinct representation of partial orders
- Pruning infrequent items, edges and partial orders
- Pruning forbidden edges
- Extracting transitive reductions of frequent partial orders directly

Interesting Orders

- Example patterns in various datasets: