Lecture Slides - 3

Mining Frequent Patterns, Association and Correlations

Overview

- Basic concepts and a road map
- Efficient and scalable frequent itemset mining methods
- Mining various kinds of association rules
- From association mining to correlation analysis
- Summary

Frequent Pattern Analysis?

- Frequent pattern: A pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- Proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets 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 a particular drug?
- Applications
  - Basket data analysis, catalog design, sale campaign analysis, Web log analysis, and DNA sequence analysis.

Basic Concepts

- Itemset: A set of items
- k-itemset: A set of k items
- Frequency or support count of an itemset: No. of transactions containing the itemset
- Frequent itemset: An itemset that has frequency count more that the minimum support threshold
- Association Rule mining problem: (a 2-step process)
  - Finding all frequent itemsets
  - Generate strong association rules from the frequent itemsets

What is Association Rule?

- Let $X=\{X_1, X_2, \ldots, X_m\}$ be a set of item, D is dataset in which each transaction $T \subseteq X$

  let A is a set of items. A transaction T is said to contain A iff $A \subseteq T$.
- An association rule is defined as an implication of the form $A \rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \varphi$

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

- Itemset $X = \{x_1, \ldots, x_k\}$
- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - Support: Probability that a transaction contains $X \cup Y$

    $sup(X \rightarrow Y) = P(X \cup Y)$
  - Confidence: Conditional probability that a transaction having X also contains *Y*

    $conf(X \rightarrow Y) = P(Y/X)$

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

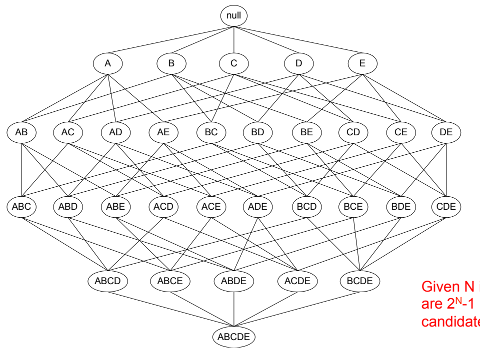
Freq. Pat.: {A:3, B:3, D:4, E:3, AD:3}

Association rules:

A → *D* (60%, 100%)

D → *A* (60%, 75%)

Frequent Itemset Generation



Given N items, there are 2^N-1 possible candidate itemsets

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 (Agrawal & Srikant, 1994)
  - Freq. pattern growth – Fpgrowth (Han, Pei & Yin, 2000)
  - Vertical data format approach – Charm (Zaki & Hsiao, 2002)

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

TDB

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

C_1 — 1^{st} scan

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}

C_2 — 2^{nd} scan

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}

3^{rd} scan — L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm

- 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 $\cup_k L_k$

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *bcd*
    - *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
  - Huge number of candidates
  - Huge data size
  - Multiple scans of transaction database
- Improving Apriori: general ideas
  - Shrink number of candidates
  - Transaction reduction
  - Reduce passes of transaction database scans

Bottlenecks with Apriori

- Uses a generate-and-test approach generates candidate itemsets and tests if they are frequent
  - Generation of candidate itemsets is expensive (in both space and time)
  - Support counting is expensive
    - Subset checking (computationally expensive)
    - Multiple Database scans (I/O)

Speeding up Apriori Algorithm

- Dynamic Hashing and Pruning
- Transaction Reduction
- Partitioning

DHP: Reduce the Number of Candidates

- Hashing itemsets into corresponding buckets
- Can be used to reduce the size of candidate k-itemsets, Ck, for k>1 – specially 2-itemsets
- While scanning DB to L1, generate C2 for each t ∈ T and hash them into different bucket of a hash table – increase hash count
- If Supp_count(itemset)<min_sup then remove it from C2

DHP: Example

TID	List of items IDs
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

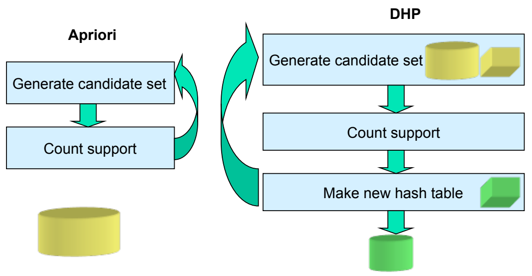
Bucket address	0	1	2	3	4	5	6
Bucket Count	2	2	4	2	2	4	4
Bucket Content	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Hash Function: h(x,y)=((order of x)×10+(order of y)) mod 7

How to Trim Candidate Itemsets

- In k-iteration, hash all "appearing" k+1 itemsets in a hash table, count all the occurrences of an itemset in the correspondent bucket.
- In k+1 iteration, examine each of the candidate itemset to see if its correspondent bucket count value is above the min. support (necessary condition)

A Quick look on Simple Apriori & DHP



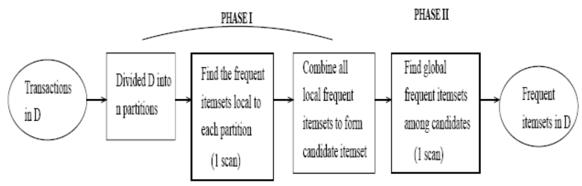
Transaction Reduction

- Reduce no. of transactions for future iterations
- A transaction, *t*, not containing any frequent k-itemset cannot contain any frequent (k+1)-itemsets
    - Delete/mark *t* from further consideration

Partitioning (DB scan only twice)

- A two-phase process
- Phase1:
    - Divide D into n disjoint partitions d_i
    - min_sup(d_i)=min_sup $\times|d_i|$
    - Generate frequent itemsets for each di – local frequent itemsets (special DS is employed to scan D only once)
    - Merge local frequent itemsets to generate global candidate itemsets
- Phase2:
    - Scan D once more to find global frequent itemsets

Partitioning (cont…)



Mining frequent itemsets by partitioning dataset

Frequent Pattern Growth (FP-Growth) Algorithm

- Allows frequent itemset discovery without candidate itemset generation.
- Two step approach:
    - Step 1: Build a compact data structure called the FP-tree
        - Built using 2 passes over the data-set.
    - Step 2: Extracts frequent itemsets directly from the FP-tree
        - Traversal through FP-Tree

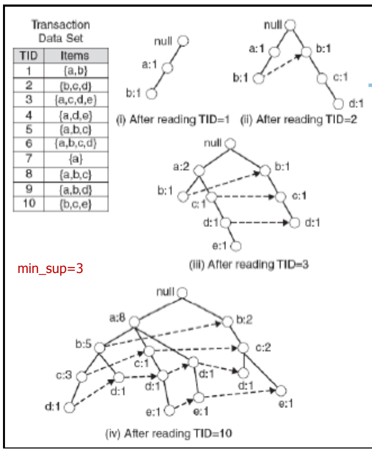
Step-1: FP-Tree Construction

- FP-Tree is constructed using 2 passes over the data-set:
- Pass 1:
    - Scan data and find support for each item.
    - Discard infrequent items.
    - Sort frequent items in decreasing order based on their support.
        - For our example: a, b, c, d, e
        - Use this order when building the FP-Tree, so common prefixes can be shared.

FP-Tree Construction (cont…)

- Pass 2: FP-Tree construction
  - Read transaction 1: {a, b}
    - Create 2 nodes a and b and the path null→a→b. Set counts of a and b to 1.
  - Read transaction 2: {b, c, d}
    - Create 3 nodes for b, c and d and the path null→b→c→d. Set counts to 1.
    - Note that although transaction 1 and 2 share b, the paths are disjoint as they don't share a common prefix. Add the link between the b's.
  - Read transaction 3: {a, c, d, e}
    - It shares common prefix item a with transaction 1 so the path for transaction 1 and 3 will overlap and the frequency count for node a will be incremented by 1. Add links between the c's and d's.
  - Continue until all transactions are mapped to a path in the FP-tree.

Example: FP-Tree Construction



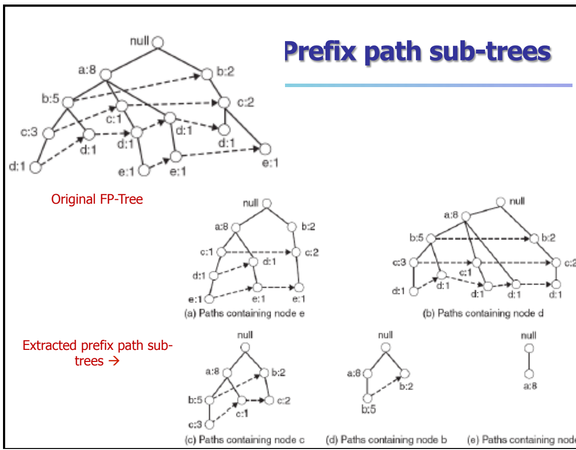
Step-2: Frequent Itemset Generation

- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Bottom-up algorithm which traverses leaves towards the root
  - Divide and conquer: first look for frequent itemsets ending in e, then de, etc. . . then d, then cd, etc. . .

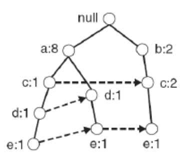
Cont…

- Method
  - Identify prefix path sub-trees ending in an item(set)
  - 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

Prefix path sub-trees



Mining Patterns Containing 'e'

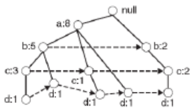

Prefix path sub-tree for 'e'

1. (e:3)
2. Conditional pattern base for e:
   {(acd:1), (ad:1), (bc:2)}

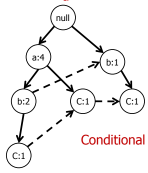
3.FP-Tree based on this conditional pattern base (called e's conditional FP-Tree) leads to only one frequent branch (c:3)

4.So, a frequent pattern (ce:3) can be derived

Mining Patterns Containing 'd'



Prefix path sub-tree for 'd'



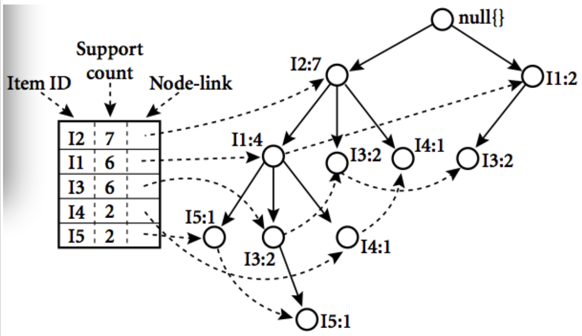
Conditional FP-Tree for d

1. (d:5)
2. Conditional pattern base for d:
   {(abc:1), (ab:1), (ac:1), (a:1), (bc:1)}

Frequent pattern (cd:3), (bd:3), (ad:4) can be derived

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

L={{I2:7}, {I1:6}, {I3:6}, {I4:2}, {I5:2}}



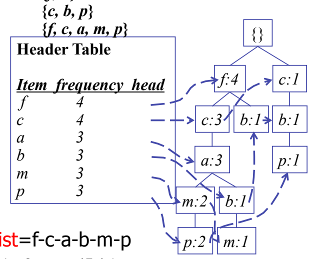
Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Example 2: FP-Growth Algorithm

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



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

Frequent-Pattern Mining: Research Problems

- Mining fault-tolerant frequent, sequential and structured patterns
  - Patterns allows limited faults (insertion, deletion, mutation)
- Mining truly interesting patterns
  - Surprising, novel, concise, …
- Application exploration
  - E.g., DNA sequence analysis and bio-pattern classification
  - “Invisible” data mining

Assignment-3

- Text Book
  - Exercises:
    - 5.3(a) [Page: 275],
    - 5.8 [Page: 276]