Association Rule Mining

FPGrowth

Huiping Cao

Issues with Apriori-like approaches

- Candidate set generation is costly, especially when there exist prolific patterns and/or long patterns.

- Jiawei Han, Jian Pei, Yiwen Yin: Mining Frequent Patterns without Candidate Generation. SIGMOD 2000:1-12.

Concepts

- Set of items: $I = \{a_1, \cdots, a_m\}$
- Transaction database: $DB = \langle T_1, \cdots, T_n \rangle$ where T_i is a transaction containing a set of items in I.
- A pattern A: a set of items
- Support (or occurrence frequency) of a pattern A: the number of transactions that contain A, denoted as $sup(A)$
- Frequent pattern: if $sup(A) \geq \xi$
- Problem: Given DB and ξ, find the complete set of frequent patterns.

Running example & basic ideas

- Given $\xi = 3$ and DB

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
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

- Observations and basic ideas
  - Only keep the frequent items in the transaction (one scan)
  - Store the set of frequent items in a compact data structure (FP-tree)

Construct a frequent pattern tree (Example)

- Scan DB once, find frequent 1-itemset (single item pattern)
  A scan of *DB* to derive a list of frequent items
  $\langle (f:4), (c:4), (a:3), (b:3), (m:3), (p:3) \rangle$

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	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

- Sort frequent items in frequency descending order, f-list = $f - c - a - b - m - p$

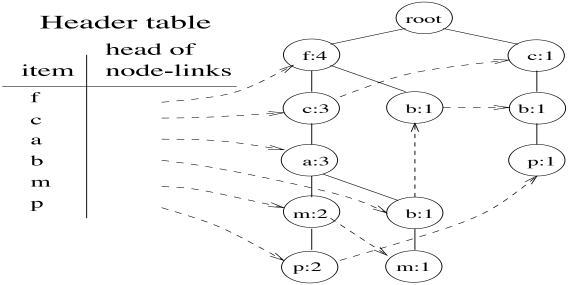
- Scan DB again, construct FP-tree

Construct a frequent pattern tree (Example)

- Create the root of a tree labeled with *null*.
- Scan the DB the second time to update the tree.
  - 1st transaction: creates a branch $\langle(f:1),(c:1),(a:1),(m:1),(p:1)\rangle$
  - 2nd transection: (f,c,a,b,m), which shares a common prefix (f,c,a) with the first transaction
    – the count of each node along the prefix is incremented by 1
    – Create a new node (b:1) as a child of (a:2)
    – Create a new node (m:1) as a child of (b:1)
  - 3rd transaction: (f,b), which share a common prefix f with the previous two transactions
    – the count for node with f is incremented by 1
    – create a new node (b:1) as a child of (f:3)
  - 4th transaction: (c,b,p), create a second branch $\langle(c:1),(b:1),(p:1)\rangle$
  - 5th transaction: is identical to the 1st transaction, increment the counts on each node.

Header table

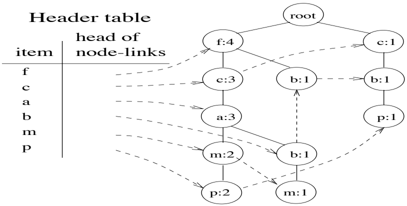
- head of node-link
- node-links



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,
  - $\cdots$
  - Patterns having c but no a nor b, m, p
  - Pattern f

Conditional pattern base



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

Algorithm – Mining frequent patterns using FP-Tree

- Node-Link property: for any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node links, starting from a_i's head in the *FP-tree* header.
- All patterns that a_i participate: start from a_i's head and follow a_i's node-links
- Start from the bottom of the header table: $p, m, \cdots$
    - 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

Algorithm – FPGrowth

Input: FP-tree, minimum support threshold ξ

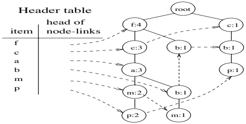
Output: the complete set of frequent patterns

Initial call: FP-Growth(FP-tree *tree*, *null*)

FP-Growth(FP-tree *tree*,α)

- If *tree* contains a single path P
  - for each node-combination β of P,
    – generate $\beta \cup \alpha$ with support $= sup(\beta)$
- else
  - for each α_i in the header of *tree*
    (1) generate pattern $\beta = \alpha_i \cup \alpha$ with support $= sup(\alpha_i)$
    (2) Calculate β's conditional pattern base
    (3) Construct β's FP-tree $tree_\beta$
    (4) if $tree_\beta \neq \emptyset$, call FP-Growth($Tree_\beta, \beta$)

FPGrowth example



Given tree $t1$ as shown in the figure.

Initial call: **FP-Growth(**$t1$**,** *null***)**

- The **else** branch of FP-Growth is executed because $t1$ contains a complex tree (not a single path p).

  The else branch needs to check every itemset in the header table. For this example, α_i can be p, m, b, a, c, and f.

  - For $\alpha_i = \{p\}$, (1) generate a pattern $\beta = \{p\}$ with support 3; (2) calculate p's conditional base, which are $fcam : 2$ and $cb : 1$; (3) create a FP tree t_p from the conditional base; (4) recursively call **FP-Growth(**t_p**,** p**)**. Details see following slides.
  - For $\alpha_i = \{m\}$, (1) generate a pattern $\beta = \{m\}$ with support 3; (2) calculate m's conditional base, which are $fca : 2$ and $fcab : 1$; (3) create a FP tree t_m from the conditional base; (4) recursively call **FP-Growth(**t_m**,** m**)**. Details see following slides.
  - For $\alpha_i = \{b\}$, $\{a\}$, $\{c\}$, and $\{f\}$ do similar.

Find Patterns Having p from p-conditional Database

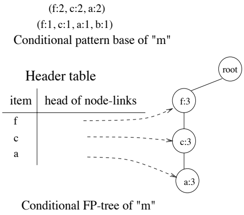
- Two paths: $\langle f : 4, c : 3, a : 3, m : 2, p : 2\rangle, \langle c : 1, b : 1, p : 1\rangle$
- Two prefix paths: $(f : 2, c : 2, a : 2, m : 2), (c : 1, b : 1)$. These paths are called p's conditional pattern base.
- Construct an FP-tree on this conditional pattern base, which consists of $(c : 3)$ as the only branch. This FP-tree is called p's conditional FP-tree. I.e., tree t_p consists of $(c : 3)$ as the only branch.
- Call FP-Growth(t_p, p).
- The if branch of FP-Growth is executed because it is a path. Thus, it reports frequent pattern $(cp : 3)$

Algorithm – Mining frequent patterns using FP-Tree

- For node m
  - Two paths: $\langle f:4, c:3, a:3, m:2\rangle, \langle f:4, c:3, a:3, b:1, m:1\rangle$
  - m's conditional pattern base: $\{(f:2, c:2, a:2), (f:1, c:1, a:1, b:1)\}$.
  - Construct an FP-tree on this conditional pattern base, m's conditional FP-tree, which only has one branch $\langle f:3, c:3, a:3\rangle$.
  - From m's conditional FP-tree t_m, $mine(\langle f:3, c:3, a:3\rangle | m)$

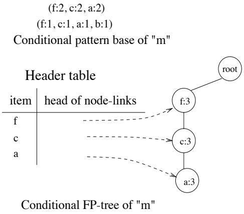
Algorithm – $mine(\langle f : 3, c : 3, a : 3 \rangle | m)$

- m's conditional FP-tree t_m is shown below.



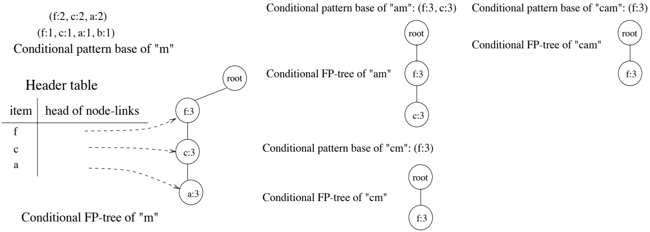
- Call **FP-Growth(t_m, m)**.
- **FP-Growth(t_m, m)** will execute the **if** brach because it contains only one path.
- All the possible combinations are f, fc, fca, c, ca, and a.
- Thus the frequent patterns are fm, fcm, $fcam$, cm, cam, and am.

Algorithm – FP-Growth(t_m, m), run else branch (1)



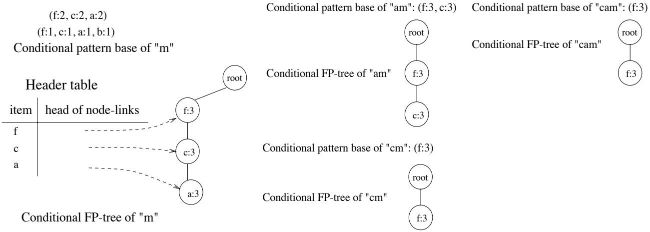
- We demonstrate the execution of the else branch using FP-Growth($t_m = \langle f:3, c:3, a:3 \rangle$, m). α_i can be $\{a\}$, $\{c\}$, and $\{f\}$
  - when $\alpha_i = \{a\}$: (1) $\beta = \{am\}$, β is frequent. OUTPUT *am*. (2) get *am*'s conditional base, which consists of $f:3, c:3$, (3) construct a FP-tree with one path $f:3, c:3$, call FP-Growth($t_{am} = \langle f:3, c:3 \rangle$, am)
  - when $\alpha_i = \{c\}$: (1) $\beta = \{cm\}$, β is frequent. OUTPUT *cm*. (2) get *cm*'s conditional base, which consists of $f:3$, (3) construct a FP-tree with one path $f:3$, call FP-Growth($t_{cm} = \langle f:3 \rangle$, cm)
  - when $\alpha_i = \{f\}$: (1) $\beta = \{fm\}$, β is frequent. OUTPUT *fm*. (2) get *fm*'s conditional base, which is $\emptyset$. No recursive call.

Algorithm – FP-Growth(t_m, m), run else branch (2)



- Run FP-Growth($t_{am} = \langle f:3, c:3 \rangle$, am). α_i can be f, c.
  - when $\alpha_i = \{c\}$: (1) $\beta = \{cam\}$, β is frequent. OUTPUT cam. (2) get cam's conditional base, which consists of $f:3$, (3) construct a FP-tree with one path $f:3$, call FP-Growth($t_{cam} = \langle f:3 \rangle$, cam)
  - when $\alpha_i = \{f\}$: (1) $\beta = \{fam\}$, β is frequent. OUTPUT fam. (2) get fm's conditional base, which is $\emptyset$. No recursive call.
- Run FP-Growth($t_{cm} = \langle f:3 \rangle$, cm). The only α_i is f: (1) $\beta = \{fcm\}$, β is frequent. OUTPUT fcm. (2) get fcm's conditional base, which is $\emptyset$. No recursive call.

Algorithm – FP-Growth(t_m, m), run else branch (3)



- Run FP-Growth($t_{cam} = \langle f : 3 \rangle$, cam). The only α_i is f: (1) $\beta = \{fcam\}$, β is frequent. OUTPUT *fcam*. (2) get *fcam*'s conditional base, which is $\emptyset$. No recursive call.
- The final results are:
  am, *cam*, *fcam*, *fam*
  cm, *fcm*,

  fm.

Algorithm – Mining frequent patterns using FP-Tree

- For node b
  - Three paths: $\langle f:4, c:3, a:3, b:1\rangle, \langle f:4, b:1\rangle, \langle c:1, b:1\rangle$
  - b's conditional pattern base:
    $\{(f:1, c:1, a:1), (f:1), (c:1)\}$.
  - This generates no frequent items. Terminates.
- For node a
  - a's conditional pattern base: $\{(f:3, c:3)\}$.
  - Frequent patterns $\{(fa:3), (ca:3), (fca:3)\}$
- For nodes c and f, do similar things

Analysis – FPGrowth

- Construct FP-tree: one scan of the data in DB, output $tree$, which is generally much smaller than DB
- The size of FP-tree shrinks in a factor of $20 \sim 100$
- Size of FP-tree is not exponential to the number of frequent patterns.
  – E.g., a frequent pattern $a_1, \cdots, a_{100}$, the complete set of frequent patterns contains 2^{100}
  – Size of the tree is still 100 (a path)

Scaling FP-growth by DB Projection

- 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
- Parallel projection vs. Partition projection techniques
- Parallel projection is space costly

References

- Chapter 5: Introduction to Data Mining (2nd Edition) by Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar
- Implementation of the FPGrowth algorithm: `https://pypi.org/project/fpgrowth-py/`