technische universität
dortmund

Chair of Data Science and
Data Engineering
Prof. Dr. Emmanuel Müller

Big Data Analytics
– Chapter 7: Frequent Itemset Mining –

Prof. Dr. Emmanuel Müller

Chair of Data Science and Data Engineering

Content Overview

- Introduction
  - Transaction databases, market basket data analysis
- Mining Frequent Itemsets
  - Apriori algorithm, hash trees, FP-tree
- Simple Association Rules
  - Basic notions, rule generation, interestingness measures
- Summerizing Frequent Itemsets
  - Maximal, closed, non-derivable itemsets
- Summary

What is Frequent Itemset Mining?

Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.

- Given:
  - A set of items $I = \{i_1, i_2, \ldots, i_m\}$
  - A database of transactions D, where a transaction $T \subseteq I$ is a set of items
- <u>Task 1:</u> find all subsets of items that occur together in many transactions.
  e.g.: 85% of transactions contain the itemset {milk, bread, butter}
- <u>Task 2:</u> find all rules that correlate the presence of one set of items with that of another set of items in the transaction database.
  e.g.: 98% of people buying tires and auto accessories also get automotive service done
- Applications:
  Basket data analysis, cross-marketing, catalog design, clustering, classification, recommendation systems, etc.

Example: Basket Data Analysis

- Transaction database

  T= { {butter, bread, milk, sugar};
  {butter, flour, milk, sugar};
  {butter, eggs, milk, salt};
  {eggs};
  {butter, flour, milk, salt, sugar} }

- Question of interest:
  - Which items are bought together frequently?
- Applications
  - Improved store layout; Cross marketing; Focused attached mailings / add-on sales

- Generalization of „association rules" beyond marketing:
  - buys(x, "diapers") → buys(x, "beers")
  - major(x, "CS") ^ takes(x, "DB") → grade(x, "A")

Basic Notions I

- *Items* $I = \{i_1, i_2, \dots, i_m\}$: a set of literals (denoting items)
- *Itemset* X*:* Set of items $X \subseteq I$

- *Database* D:
  - Set of *transactions* T, each transaction is a set of items $\mathrm{T} \subseteq I$
  - Transaction T *contains* an itemset X*:* $X \subseteq T$

The items in transactions and itemsets are sorted lexicographically:

- itemset $X = (x_1, x_2, \dots, x_k)$, where $x_1 \leq x_2 \leq \dots \leq x_k$

Length of an itemset: number of elements in the itemset

k-itemset: itemset of length *k*

Basic Notions II

- *Items* $I = \{i_1, i_2, \dots, i_m\}$: a set of literals (denoting items)
- *Itemset* X*:* Set of items $X \subseteq I$

The *support* of an itemset X is defined as: $support(X) = |\{T \in D | X \subseteq T\}|$

Frequent itemset: an itemset X is called frequent for database D iff it is contained in more than $minSup$ many transactions: $support(X) \geq minSup$

<u>Goal 1:</u>
Given a database D and a threshold $minSup$,
find all frequent itemsets $X \in Pot(I)$.

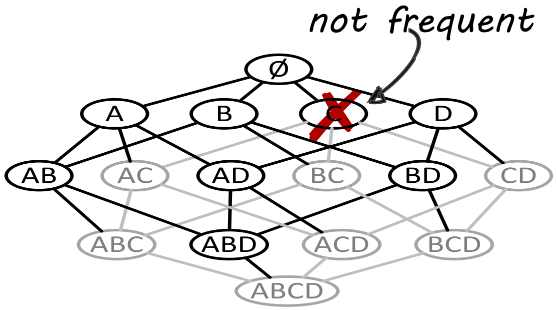
Basic Idea

Naïve Algorithm

- count the frequency of all possible subsets of I in the database

→ *too expensive* since there are 2^m such itemsets for $|I| = m$ items

cardinality of power set



Basic Idea

The *Apriori* principle (anti-monotonicity):

Any non-empty subset of a frequent itemset is frequent, too!

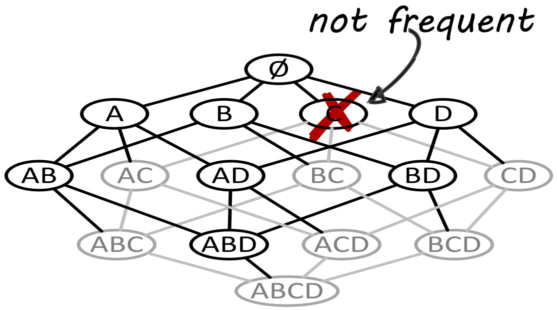
$\mathrm{A} \subseteq \mathrm{I}$ with $\mathrm{support}(\mathrm{A}) \geq \mathrm{minSup} \;\Rightarrow \forall \mathrm{A}' \subset \mathrm{A} \wedge \mathrm{A}' \neq \emptyset\colon \mathrm{support}(\mathrm{A}') \geq \mathrm{minSup}$

Any superset of a non-frequent itemset is non-frequent, too!

$\mathrm{A} \subseteq \mathrm{I}$ with $\mathrm{support}(\mathrm{A}) < \mathrm{minSup} \;\Rightarrow \forall \mathrm{A}' \supset \mathrm{A}\colon \mathrm{support}(\mathrm{A}') < \mathrm{minSup}$

Method based on the apriori principle

- First count the 1-itemsets, then the 2-itemsets, then the 3-itemsets, and so on
- When counting (k+1)-itemsets, only consider those (k+1)-itemsets where all subsets of length k have been determined as frequent in the previous step



The Apriori Algorithm

variable C_k: candidate itemsets of size k
variable L_k: frequent itemsets of size k
L_1 = {frequent items}
for (k = 1; L_k $!= \emptyset$; k++) **do begin**

produce candidates

// JOIN STEP: join L_k with itself to produce C_{k+1}
// PRUNE STEP: discard (k+1)-itemsets from C_{k+1} that contain non-frequent k-itemsets as subsets
C_{k+1} = candidates generated from L_k

for each transaction t in database do

prove candidates

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

return $\cup_k L_k$

Generating Candidates (Prune Step)

Step 2: Pruning: $L_{k+1} = \{X \in C_{k+1} \mid support(X) \geq minSup\}$

- Naïve: Check support of every itemset in C_{k+1} → inefficient for huge C_{k+1}
- Instead, apply Apriori principle first:
  Remove candidate *(k+1)* -itemsets which contain a non-frequent *k* -subset *s*, i.e., s $\notin L_k$

forall *itemsets c* ***in*** C_{k+1} **do**
    forall *k-subsets s* ***of*** *c* **do**
        if *(s is not in L_k)* **then delete** *c* **from** C_{k+1}

Example 1

- L_3 = {(ACF), (ACG), (AFG), (AFH), (CFG)}
- Candidates after the join step: {(ACFG), (AFGH)}
- In the pruning step: delete (AFGH) because (FGH) $\notin L_3$,
  i.e., (FGH) is not a frequent 3-itemset; also (AGH) $\notin L_3$

→ C_4 = {(ACFG)} → check the support to generate L_4

Generating Candidates (Join Step)

Requirements for set of all candidate $(k+1)$ -itemsets C_{k+1}

- *Completeness*: Must contain all frequent $(k+1)$ -itemsets (superset property $C_{k+1} \supseteq L_{k+1}$)
- *Selectiveness*: Significantly smaller than the set of all $(k+1)$-subsets

Step 1: Joining ($C_{k+1} = L_k \bowtie L_k$)

- Suppose the items are sorted by any order (e.g. lexicograph.)
- Consider frequent k -itemsets p and q
- p and q are joined if they share the same first *k-1* items

insert into C_{k+1}

select $p.i_1, p.i_2, \ldots, p.i_{k-1}, p.i_k, q.i_k$

from $L_k : p, L_k : q$

where $p.i_1=q.i_1, \ldots, p.i_{k-1} =q.i_{k-1}, p.i_k < q.i_k$

$p \in L_{k=3}$ (A, C, F)

(A, C, F, G) $\in C_{k+1=4}$

$q \in L_{k=3}$ (A, C, G)

Apriori Algorithm – Full Example

minSup=0.5
database D

TID	items
100	1 3 4 6
200	2 3 5
300	1 2 3 5
400	1 5 6

scan D →

C_1

itemset	count
{1}	3
{2}	2
{3}	3
{4}	1
{5}	3
{6}	2

→

L_1

itemset	count
{1}	3
{2}	2
{3}	3
{5}	3
{6}	2

$L_1 \bowtie L_1$

C_2

itemset
{1 2}
{1 3}
{1 5}
{1 6}
{2 3}
{2 5}
{2 6}
{3 5}
{3 6}
{5 6}

prune C_1 →

C_2

itemset
{1 2}
{1 3}
{1 5}
{1 6}
{2 3}
{2 5}
{2 6}
{3 5}
{3 6}
{5 6}

scan D →

C_2

itemset	count
{1 2}	1
{1 3}	2
{1 5}	2
{1 6}	2
{2 3}	2
{2 5}	2
{2 6}	0
{3 5}	2
{3 6}	1
{5 6}	1

→

L_2

itemset	count
{1 3}	2
{1 5}	2
{1 6}	2
{2 3}	2
{2 5}	2
{3 5}	2

$L_2 \bowtie L_2$

C_3

itemset
{1 3 5}
{1 3 6}
{1 5 6}
{2 3 5}

prune C_2 →

C_3

itemset
{1 3 5}
{1 3 6} ✗
{1 5 6} ✗
{2 3 5}

scan D →

C_3

itemset	count
{1 3 5}	1
{2 3 5}	2

→

L_3

itemset	count
{2 3 5}	2

$L_3 \bowtie L_3$

C_4 is empty

Is Apriori Fast Enough? – Performance Bottlenecks –

The core of the Apriori algorithm:

- Use frequent (k – 1)-itemsets to generate candidate frequent k-itemsets
- Use database scan and pattern matching to collect counts for the candidate itemsets

The bottleneck of *Apriori*: candidate generation

- Huge candidate sets:
  - 10^4 frequent 1-itemsets will generate 10^7 candidate 2-itemsets
  - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
- Multiple scans of database:
  - Needs n or n+1 scans, n is the length of the longest pattern

→ Is it possible to mine the complete set of frequent itemsets without candidate generation?

Mining Frequent Patterns Without Candidate Generation

Compress a large database into a compact,
Frequent-Pattern tree (*FP-tree*) structure

- highly condensed, but complete for frequent pattern mining
- avoid costly database scans

Develop an efficient, FP-tree-based frequent pattern mining method

- A divide-and-conquer methodology: decompose mining tasks into smaller ones
- Avoid candidate generation: sub-database test only!

Idea:

- Compress database into FP-tree, retaining the itemset association information
- Divide the compressed database into conditional databases, each associated with one frequent item and mine each such database separately.

Construct FP-tree from a Transaction DB (1)

Steps for compressing the database into a FP-tree:

1. Scan DB once, find frequent 1-itemsets (single items)
2. Order frequent items in frequency descending order

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*}

minSup=0.5

header table:

item	frequency
f	4
c	4
a	3
b	3
m	3
p	3

sort items in the order of descending support

Construct FP-tree (2)

Steps for compressing the database into a FP-tree:

1. Scan DB once, find frequent 1-itemsets (single items)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree starting with most frequent item per transaction

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*}

for each transaction only keep its frequent items sorted in descending order of their frequencies

header table:

item	frequency
f	4
c	4
a	3
b	3
m	3
p	3

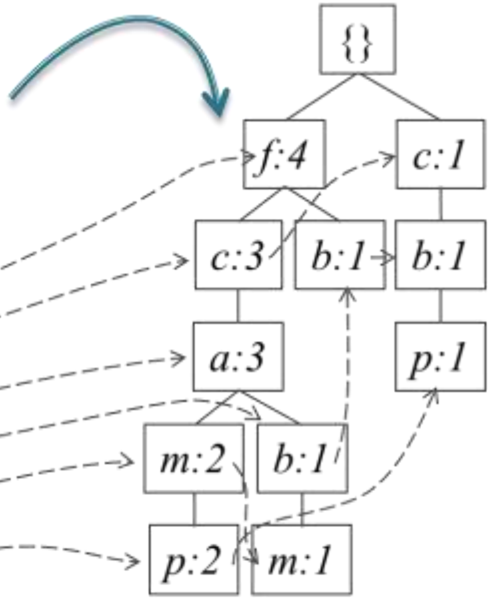
for each transaction built a path in the FP-tree:
- If a path with common prefix exists: increment frequency of nodes on this path and append suffix
- Otherwise: create a new branch

Construct FP-tree (3)

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*}

header table:

item	frequency	head
f	4	•
c	4	•
a	3	•
b	3	•
m	3	•
p	3	•



header table references the occurrences of the frequent items in the FP-tree

Benefits of the FP-tree Structure

- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining
- Compactness
  - reduce irrelevant information—infrequent items are gone
  - frequency descending ordering: more frequent items are more likely to be shared
  - never be larger than the original database
    (if not count node-links and counts)
  - Experiments demonstrate compression ratios over 100

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree

- Method
  - For each item, construct its conditional pattern-base (*prefix paths*), 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)

Major Steps to Mine FP-tree

1. Construct conditional pattern base for each node in the FP-tree

2. Construct conditional FP-tree from each conditional pattern-base

3. Recursively mine conditional FP-trees and grow frequent patterns obtained so far
   - If the conditional FP-tree contains a single path, simply enumerate all the patterns

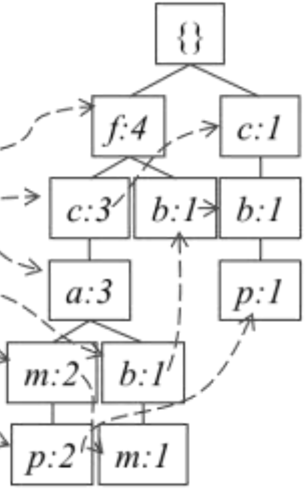
Major Steps to Mine FP-tree: Conditional Pattern Base

1. Construct conditional pattern base for each node in the FP-tree

- Starting at the frequent header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a conditional pattern base
  - For each item its prefixes are regarded as condition for it being a suffix. These prefixes form the conditional pattern base. The frequency of the prefixes can be read in the node of the item.

header table:

item	frequency	head
f	4	•
c	4	•
a	3	•
b	3	•
m	3	•
p	3	•



conditional pattern base:

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

Properties of FP-tree for Conditional Pattern Bases

- 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

- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P needs to be accumulated, and its frequency count should carry the same count as node a_i.

Major Steps to Mine FP-tree: Conditional FP-tree

2. Construct conditional FP-tree from each conditional pattern-base

- The prefix paths of a suffix represent the conditional basis.
  →They can be regarded as transactions of a database.
- Those prefix paths whose support ≥ minSup, induce a conditional FP-tree
- 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

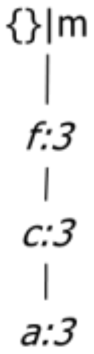
conditional pattern base:

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

e.g. item m

item	frequency
f	3
c	3
a	3
b	1 ✗

m-conditional FP-tree



Conditional FP-tree

2. Construct conditional FP-tree from each conditional pattern-base

conditional pattern base:

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

{}|f = {}

{}|c — *f:3*

{}|a — *f:3* — *c:3*

{}|b = {}

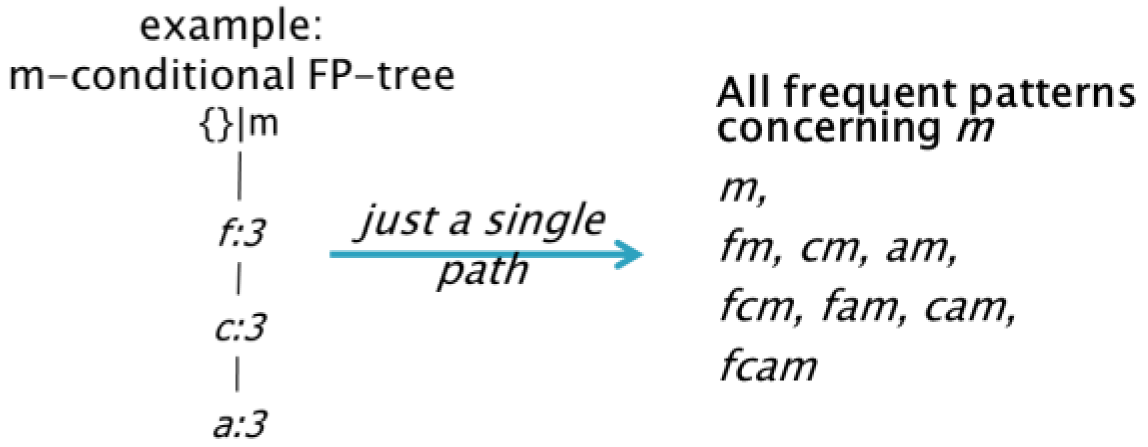
{}|m — *f:3* — *c:3* — *a:3*

{}|p — *c:3*

Major Steps to Mine FP-tree

3. Recursively mine conditional FP-trees and grow frequent patterns obtained so far

- If the conditional FP-tree contains a single path, simply enumerate all the patterns (enumerate all combinations of sub-paths)



FP-tree: Full Example

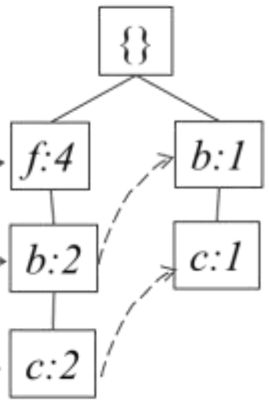
database:

TID	items bought	(ordered) frequent items
100	{*b*, *c*, *f*}	{*f*, *b*, *c*}
200	{*a*, *b*, *c*}	{*b*, *c*}
300	{*d*, *f*}	{*f*}
400	{*b*, *c*, *e*, *f*}	{*f*, *b*, *c*}
500	{*f*, *g*}	{*f*}

minSup=0.4

header table:

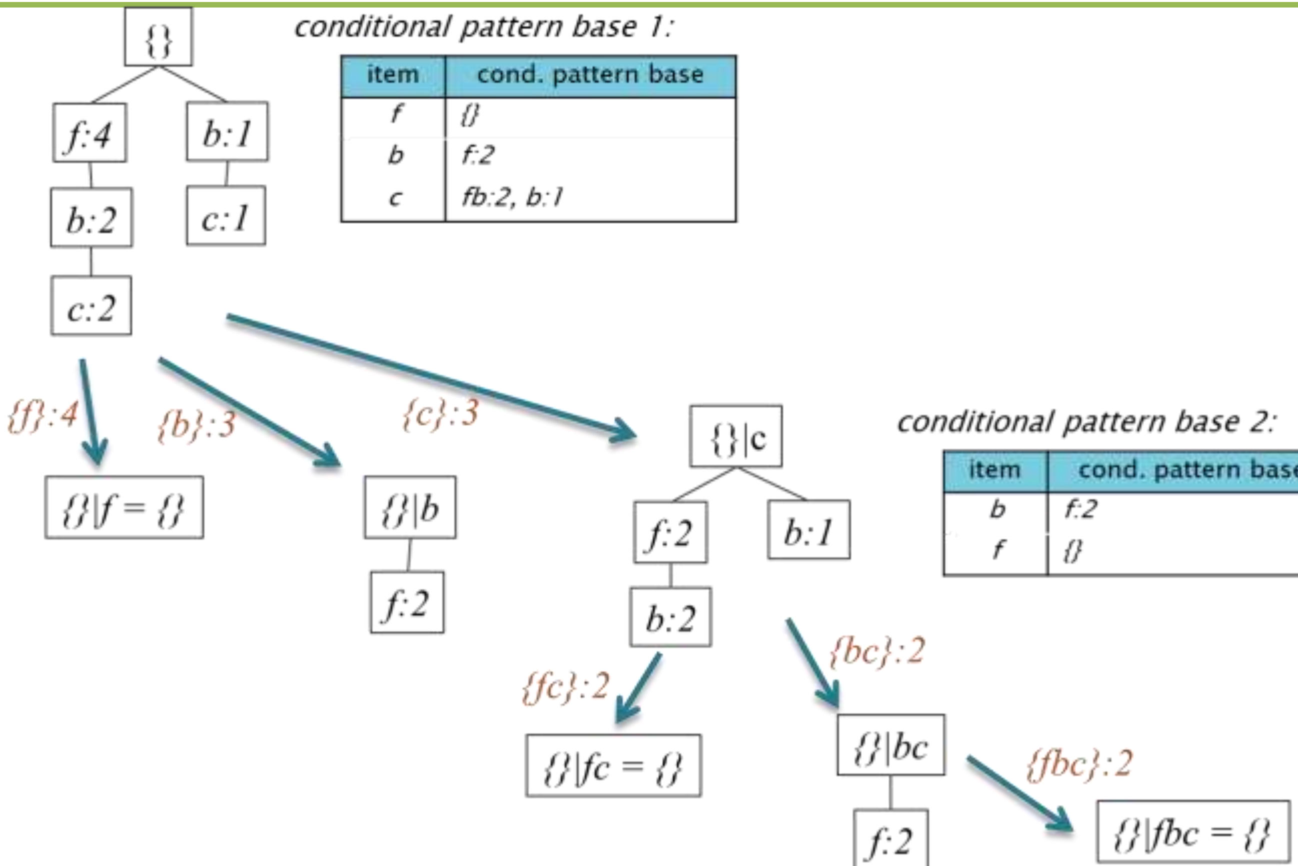
item	frequency	head
f	4	•
b	3	•
c	3	•



conditional pattern base:

item	cond. pattern base
f	{}
b	*f*:2
c	*fb*:2, *b*:1

FP-tree: Full Example



conditional pattern base 1:

item	cond. pattern base
f	*{}*
b	*f:2*
c	*fb:2, b:1*

conditional pattern base 2:

item	cond. pattern base
b	*f:2*
f	*{}*

Principles of Frequent Pattern Growth

- Pattern growth property
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B.

    Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.

- "abcdef " is a frequent pattern, <u>if and only if</u>
  - "abcde " is a frequent pattern, and
  - "f " is frequent in the set of transactions containing "abcde "

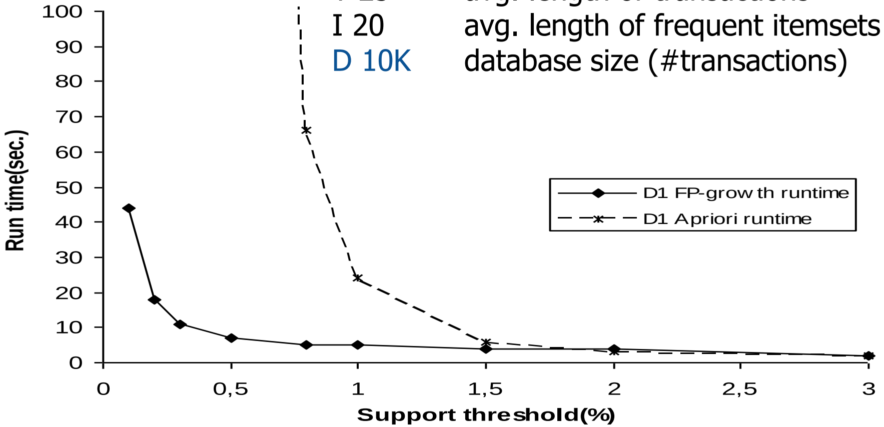
Why Is Frequent Pattern Growth Fast?

Performance study in [Han, Pei&Yin ’00] shows

- FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection

Data set T25I20D10K:

T 25	avg. length of transactions
I 20	avg. length of frequent itemsets
D 10K	database size (#transactions)



Reasoning

- No candidate generation, no candidate test
  - Apriori algorithm has to proceed breadth-first
- Use compact data structure
- Eliminate repeated database scan
- Basic operation is counting and FP-tree building

Content Overview

- Introduction
  - Transaction databases, market basket data analysis
- Mining Frequent Itemsets
  - Apriori algorithm, hash trees, FP-tree
- Summerizing Frequent Itemsets
  - Maximal, closed, non-derivable itemsets
- Simple Association Rules
  - Basic notions, rule generation, interestingness measures
- Summary

cf. Zaki‘s Book!

Maximal & Closed Frequent Itemsets

Big challenge: database contains potentially a huge number of frequent itemsets (especially if minSup is set too low).

- A frequent itemset of length 100 contains 2^{100}-1 many frequent subsets

***Closed frequent itemset*:**
An itemset X is *closed* in a data set D if there exists no proper super-itemset Y such that $support(X) = support(Y)$ in D.

- The set of closed frequent itemsets contains complete information regarding its corresponding frequent itemsets.

***Maximal frequent itemset*:**
An itemset X is *maximal* in a data set D if there exists no proper super-itemset Y such that $support(Y) \geq minSup$ in D.

- The set of maximal itemsets does not contain the complete support information
- More compact representation

Maximal Frequent Itemsets

Given a binary database $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$, over the tids $\mathcal{T}$ and items $\mathcal{I}$, let $\mathcal{F}$ denote the set of all frequent itemsets, that is,

$$\mathcal{F} = \{X \mid X \subseteq \mathcal{I} \text{ and } sup(X) \geq minsup\}$$

A frequent itemset $X \in \mathcal{F}$ is called *maximal* if it has no frequent supersets. Let $\mathcal{M}$ be the set of all maximal frequent itemsets, given as

$$\mathcal{M} = \{X \mid X \in \mathcal{F} \text{ and } \nexists Y \supset X, \text{ such that } Y \in \mathcal{F}\}$$

The set $\mathcal{M}$ is a condensed representation of the set of all frequent itemset $\mathcal{F}$, because we can determine whether any itemset X is frequent or not using $\mathcal{M}$. If there exists a maximal itemset Z such that $X \subseteq Z$, then X must be frequent; otherwise X cannot be frequent.

Example for Maximal Itemsets

Transaction database

Tid	Itemset
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Frequent itemsets (*minsup* = 3)

sup	Itemsets
6	*B*
5	*E*, *BE*
4	*A*, *C*, *D*, *AB*, *AE*, *BC*, *BD*, *ABE*
3	*AD*, *CE*, *DE*, *ABD*, *ADE*, *BCE*, *BDE*, *ABDE*

Mining Maximal Frequent Itemsets

Mining maximal itemsets requires additional steps beyond simply determining the frequent itemsets. Assuming that the set of maximal frequent itemsets is initially empty, that is, $\mathcal{M} = \emptyset$, each time we generate a new frequent itemset X, we have to perform the following maximality checks

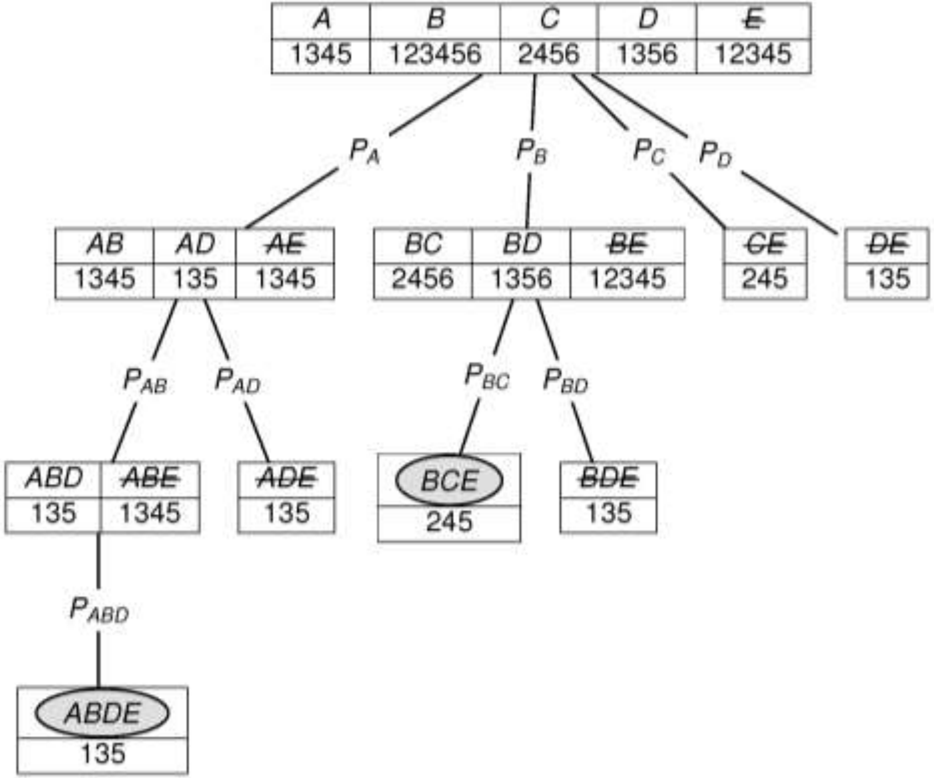
- **Subset Check:** $\nexists Y \in \mathcal{M}$, such that $X \subset Y$. If such a Y exists, then clearly X is not maximal. Otherwise, we add X to $\mathcal{M}$, as a potentially maximal itemset.
- **Superset Check:** $\nexists Y \in \mathcal{M}$, such that $Y \subset X$. If such a Y exists, then Y cannot be maximal, and we have to remove it from $\mathcal{M}$.

GenMax Algorithm

// Initial Call: $\mathcal{M} \leftarrow \emptyset$,
$P \leftarrow \{\langle i, \mathbf{t}(i)\rangle \mid i \in \mathcal{I}, sup(i) \geq minsup\}$

GenMax (P, $minsup$, $\mathcal{M}$):

$Y \leftarrow \bigcup X_i$
if $\exists Z \in \mathcal{M}$, *such that* $Y \subseteq Z$ **then**
  return // prune entire branch
foreach $\langle X_i, \mathbf{t}(X_i)\rangle \in P$ **do**
  $P_i \leftarrow \emptyset$
  foreach $\langle X_j, \mathbf{t}(X_j)\rangle \in P$, *with* $j > i$ **do**
    $X_{ij} \leftarrow X_i \cup X_j$
    $\mathbf{t}(X_{ij}) = \mathbf{t}(X_i) \cap \mathbf{t}(X_j)$
    if $sup(X_{ij}) \geq minsup$ **then** $P_i \leftarrow P_i \cup \{\langle X_{ij}, \mathbf{t}(X_{ij})\rangle\}$
  if $P_i \neq \emptyset$ **then** GenMax (P_i, $minsup$, $\mathcal{M}$)
  else if $\nexists Z \in \mathcal{M}, X_i \subseteq Z$ **then**
    $\mathcal{M} = \mathcal{M} \cup X_i$ // add X_i to maximal set


A
1345
B
123456
C
2456
D
1356
~~E~~
12345
P_A
P_B
P_C
P_D
AB
1345
AD
135
~~AE~~
1345
BC
2456
BD
1356
~~BE~~
12345
~~CE~~
245
~~DE~~
135
P_{AB}
P_{AD}
P_{BC}
P_{BD}
ABD
135
~~ABE~~
1345
~~ADE~~
135
BCE
245
~~BDE~~
135
P_{ABD}
ABDE
135

Content Overview

- Introduction
  - Transaction databases, market basket data analysis
- Mining Frequent Itemsets
  - Apriori algorithm, hash trees, FP-tree
- Summerizing Frequent Itemsets
  - Maximal, closed, non-derivable itemsets
- Simple Association Rules
  - Basic notions, rule generation, interestingness measures
- Summary

Association Rules: Basic Notions

Items $I = \{i_1, i_2, \ldots, i_m\}$: a set of literals (denoting items)

Itemset X*:* Set of items $X \subseteq I$

Database D: Set of *transactions* T, each transaction is a set of items $T \subseteq I$

Transaction T *contains* an itemset X: $X \subseteq T$

The *support* of an itemset X is defined as: $support(X) = |\{T \in D | X \subseteq T\}|$

Frequent itemset: an itemset X is called frequent iff $support(X) \geq minSup$

Association rule: An association rule is an implication in the form $A \Rightarrow B$ where $A \subset I$ and $B \subset I$ are two itemsets with $A \cap B = \emptyset$

Note: simply enumerating all possible association rules is not meaningful!
→ What are the interesting association rules w.r.t. D?

Interestingness of Association Rules

Interestingness of an association rule:

Quantify the interestingness of an association rule with respect to a transaction database D:

- Support: frequency (probability) of the entire rule with respect to D

$$support(A \Rightarrow B) = P(A \cup B) = \frac{|\{T \in D | A \cup B \subseteq T\}|}{|D|} = support(A \cup B)$$

  "probability that a transaction in D contains the itemset $A \cup B$"

- Confidence: indicates the strength of implication in the rule

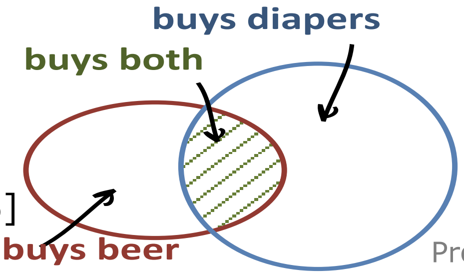
$$confidence(A \Rightarrow B) = P(B|A) = \frac{|\{T \in D | A \cup B \subseteq T\}|}{|\{T \in D | A \subseteq T\}|} = \frac{support(A \cup B)}{support(A)}$$

  "conditional probability that a transaction in D containing the itemset A also contains itemset B"

- Rule form: "$Body \Rightarrow Head\ [support, confidence]$"

Association rule examples:

- buys diapers ⇒ buys beers [0.5%, 60%]
- major in CS ∧ takes DB ⇒ avg. grade A [1%, 75%]



Mining of Association Rules

Given a database D, determine all association rules having
a $support \geq minSup$ and a $confidence \geq minConf$ (so-called *strong association rules*).

Key steps of mining association rules:

e.g. Apriori, FP-growth

1) Find *frequent itemsets*, i.e., itemsets that have at least $\text{support} = minSup$
2) Use the frequent itemsets to generate association rules
   - n frequent items yield $2^n - 2$ association rules

Example

transaction ID	items
2000	A, B, C
1000	A, C
4000	A, D
5000	B, E, F

$minSup = 50\%$ →

frequent items	support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

For $minSup = 50\%$ and $minConf = 50\%$ we obtain 2 rules:

- $A \Rightarrow C$: $support(A \Rightarrow C) = 50\%, confidence(A \Rightarrow C) = 66.67\%$
- $C \Rightarrow A$: $support(C \Rightarrow A) = 50\%, confidence(C \Rightarrow A) = 100\%$

Generating Rules

For each frequent itemset *X*

- For each nonempty subset Y of *X*, form a rule $Y \Rightarrow (X - Y)$
- Delete those rules that do not have minimum confidence

Computation of the confidence of a rule $Y \Rightarrow (X - Y)$

$$confidence(Y \Rightarrow (X - Y)) = \frac{support(X)}{support(Y)}$$

Store the frequent itemsets and their support in a hash table in main memory → no additional database access

Example: $X = \{A, B, C\}$, $minConf = 60\%$

- conf (A ⇒ B, C) = 2/2; ✓ conf (B, C ⇒ A) = ½ ✗
- conf (B ⇒ A, C) = 2/4; ✗ conf (A, C ⇒ B) = 1 ✓
- conf (C ⇒ A, B) = 2/5; ✗ conf (A, B ⇒ C) = 2/3 ✓

itemset	support
{A}	2
{B}	4
{C}	5
{A, B}	3
{A, C}	2
{B, C}	4
{A, B, C}	2

Interestingness Measurements

Objective measures

- Two popular measurements:
- support and
- confidence

Subjective measures
[Silberschatz & Tuzhilin, KDD95]

- A rule (pattern) is interesting if it is
- unexpected (surprising to the user) and/or
- actionable (the user can do something with it)

Criticism to Support and Confidence

Example 1 [Aggarwal & Yu, PODS98]

Among 5000 students

- 3000 play basketball (=60%)
- 3750 eat cereal (=75%)
- 2000 both play basket ball and eat cereal (=40%)

Rule *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading because the overall percentage of students eating cereal is 75% which is higher than 66.7%

Rule *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is far more accurate, although with lower support and confidence

Observation: *play basketball* and *eat cereal* are *negatively correlated*

Other Interestingness Measures: Correlation

Lift is a simple correlation measure between two items *A* and *B:*

$$corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)} = \frac{P(A|B)}{P(A)} = \frac{P(B|A)}{P(B)}$$

! The two rules $A \Rightarrow B$ and $B \Rightarrow A$ have the same correlation coefficient

take both P(A) and P(B) in consideration

$corr_{A,B} > 1$ the two items A and B are positively correlated

$corr_{A,B} = 1$ there is no correlation between the two items A and B

$corr_{A,B} < 1$ the two items A and B are negatively correlated

Summary

Mining Frequent Itemsets

- Apriori algorithm, FP-tree

Summerizing Frequent Itemsets

- Maximal itemsets

Simple Association Rules

- Basic notions, rule generation, interestingness measures