Data Mining for Knowledge Management

Association Rules

Themis Palpanas

University of Trento

http://disi.unitn.eu/~themis

Thanks for slides to:

- Jiawei Han
- Jeff Ullman

Roadmap

- Frequent Patterns
  - Frequent Pattern Analysis
  - Applications
  - Market-Basket Model
  - Association Rules
- A-Priori Algorithm
- Improvements to A-Priori

What Is Frequent Pattern Analysis?

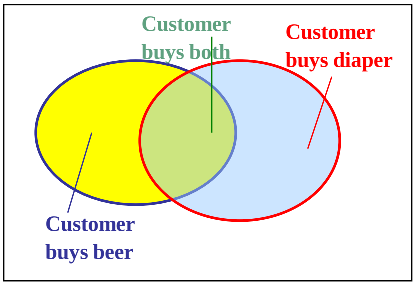
- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Why Is Freq. Pattern Mining Important?

- Discloses an intrinsic and important property of data sets
- Forms the foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: associative classification
  - Cluster analysis: frequent pattern-based clustering
  - Data warehousing: iceberg cube and cube-gradient
  - Semantic data compression: fascicles
  - Broad applications

Basic Concepts: 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, *s*, probability that a transaction contains $X \cup Y$
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

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%)

The Market-Basket Model

- A large set of *items*, e.g., things sold in a supermarket.
- A large set of *baskets*, each of which is a small set of the items, e.g., the things one customer buys on one day.

Support

- Simplest question: find sets of items that appear "frequently" in the baskets.
- *Support* for itemset I = the number of baskets containing all items in I.
- Given a support *threshold* s, sets of items that appear in $\geq s$ baskets are called *frequent itemsets*.

Example

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

  B_1 = {m, c, b} B_2 = {m, p, j}
  B_3 = {m, b} B_4 = {c, j}
  B_5 = {m, p, b} B_6 = {m, c, b, j}
  B_7 = {c, b, j} B_8 = {b, c}

- Frequent itemsets?

Example

- Items={milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.

  B_1 = {m, c, b} B_2 = {m, p, j}
  B_3 = {m, b} B_4 = {c, j}
  B_5 = {m, p, b} B_6 = {m, c, b, j}
  B_7 = {c, b, j} B_8 = {b, c}

- Frequent itemsets:
  - {m}, {c}, {b}, {j}, {m, b}, {c, b}, {j, c}.

Applications --- (1)

- Real market baskets: chain stores keep terabytes of information about what customers buy together.
  - Tells how typical customers navigate stores, lets them position tempting items.
  - Suggests tie-in "tricks," e.g., run sale on diapers and raise the price of beer.
- High support needed, or no $$'s .

Applications --- (2)

- "Baskets" = documents; "items" = words in those documents.
  - Lets us find words that appear together unusually frequently, i.e., linked concepts.
- "Baskets" = sentences, "items" = documents containing those sentences.
  - Items that appear together too often could represent plagiarism.

Applications --- (3)

- "Baskets" = Web pages; "items" = linked pages.
  - Pairs of pages with many common references may be about the same topic.
- "Baskets" = Web pages *p* ; "items" = pages that link to *p* .
  - Pages with many of the same links may be mirrors or about the same topic.

Important Point

- "Market Baskets" is an abstraction that models any many-many relationship between two concepts: "items" and "baskets."
  - Items need not be "contained" in baskets.
- The only difference is that we count co-occurrences of items related to a basket, not vice-versa.

Scale of Problem

- WalMart sells 100,000 items and can store billions of baskets.
- The Web has over 100,000,000 words and billions of pages.

Association Rules

- If-then rules about the contents of baskets.
- $\{i_1, i_2,\ldots,i_k\} \rightarrow j$ means: "if a basket contains all of $i_1,\ldots,i_k$ then it is *likely* to contain j."
- *Confidence* of this association rule is the probability of j given $i_1,\ldots,i_k$.

Example

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$
$B_3 = \{m, b\}$ $B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

- An association rule: $\{m, b\} \rightarrow c$.
  - Confidence = 2/4 = 50%.

Interest

- The *interest* of an association rule $X \rightarrow Y$ is the absolute value of the amount by which the confidence differs from the probability of Y.

Example

B_1 = {m, c, b} B_2 = {m, p, j}
B_3 = {m, b} B_4 = {c, j}
B_5 = {m, p, b} B_6 = {m, c, b, j}
B_7 = {c, b, j} B_8 = {b, c}

- For association rule {m, b} → c, item *c* appears in 5/8 of the baskets.
- Interest = | 2/4 - 5/8 | = 1/8 --- not very interesting.

Relationships Among Measures

- Rules with high support and confidence may be useful even if they are not "interesting."
  - We don't care if buying bread *causes* people to buy milk, or whether simply a lot of people buy both bread and milk.
- But high interest suggests a cause that might be worth investigating.

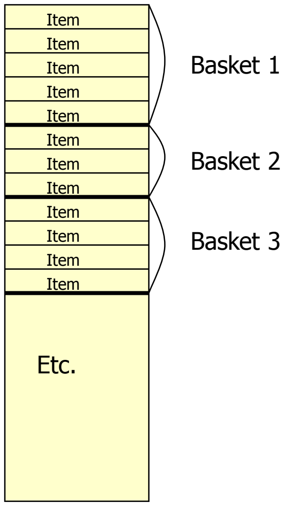
Finding Association Rules

- A typical question: "find all association rules with support $\geq s$ and confidence $\geq c$."
  - Note: "support" of an association rule is the support of the set of items it mentions.
- Hard part: finding the high-support (*frequent*) itemsets.
  - Checking the confidence of association rules involving those sets is relatively easy.

Computation Model

- Typically, data is kept in a "flat file" rather than a database system.
  - Stored on disk.
  - Stored basket-by-basket.
  - Expand baskets into pairs, triples, etc. as you read baskets.

File Organization



Computation Model --- (2)

- The true cost of mining disk-resident data is usually the number of disk I/O's.
- In practice, association-rule algorithms read the data in *passes* --- all baskets read in turn.
- Thus, we measure the cost by the number of passes an algorithm takes.

Main-Memory Bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource.
  - As we read baskets, we need to count something, e.g., occurrences of pairs.
  - The number of different things we can count is limited by main memory.
  - Swapping counts in/out is a disaster.

Finding Frequent Pairs

- The hardest problem often turns out to be finding the frequent pairs.

- We'll concentrate on how to do that, then discuss extensions to finding frequent triples, etc.

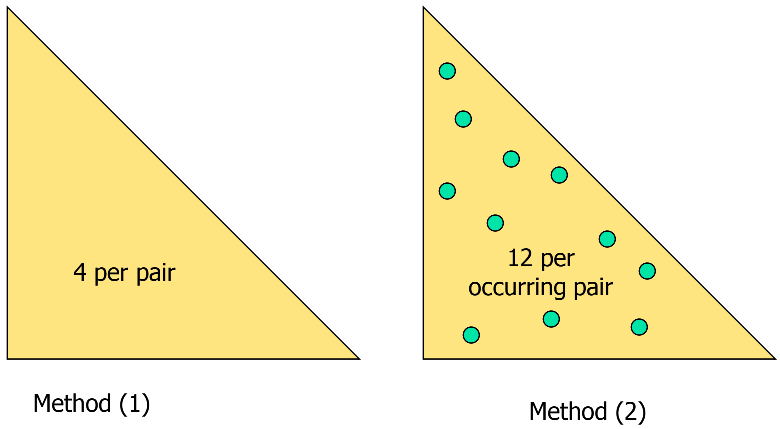
Naïve Algorithm

- Read file once, counting in main memory the occurrences of each pair.
  - Expand each basket of *n* items into its *n* (*n* -1)/2 pairs.
- Fails if (#items)^2 exceeds main memory.
  - Remember: #items can be 100K (Wal-Mart) or 10B (Web pages).

Details of Main-Memory Counting

- Two approaches:
  1. Count all item pairs, using a triangular matrix.
  2. Keep a table of triples [*i*, *j*, *c*] = the count of the pair of items {*i,j* } is *c*.
- (1) requires only (say) 4 bytes/pair.
- (2) requires 12 bytes, but only for those pairs with count > 0.

Details of Main-Memory Counting



Details of Approach #1

- Number items 1, 2,…
- Keep pairs in the order {1,2}, {1,3},…, {1,n}, {2,3}, {2,4},…,{2,n}, {3,4},…, {3,n},…{n-1,n}.
- Find pair {i, j} at the position:
  - $(i-1)(n-i/2) + j - i$
- Total number of pairs $n(n-1)/2$; total bytes about $2n^2$.

Details of Approach #2

- You need a hash table, with *i* and *j* as the key, to locate (*i, j, c*) triples efficiently.
  - Typically, the cost of the hash structure can be neglected.
- Total bytes used is about 12*p*, where *p* is the number of pairs that actually occur.
  - Beats triangular matrix if at most 1/3 of possible pairs actually occur.

Roadmap

- Frequent Patterns
- A-Priori Algorithm
  - Monotonicity Property
  - Algorithm Description
- Improvements to A-Priori

A-Priori Algorithm --- (1)

- A two-pass approach called *a-priori* limits the need for main memory.
- Key idea: *monotonicity*: if a set of items appears at least *s* times, so does every subset.
  - Contrapositive for pairs: if item *i* does not appear in *s* baskets, then no pair including *i* can appear in *s* baskets.

(Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)

A-Priori Algorithm --- (2)

- Pass 1: Read baskets and count in main memory the occurrences of each item.
  - memory requirements?

A-Priori Algorithm --- (2)

- Pass 1: Read baskets and count in main memory the occurrences of each item.
  - Requires only memory proportional to #items.

A-Priori Algorithm --- (2)

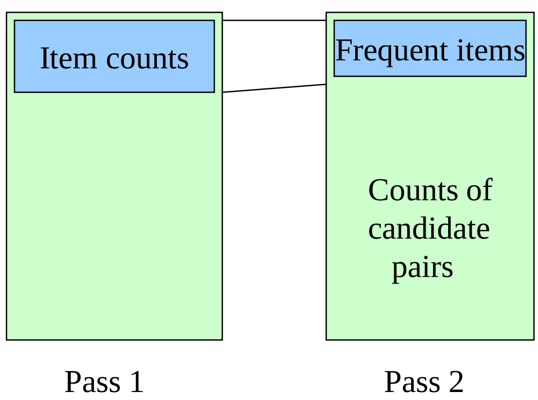
- Pass 1: Read baskets and count in main memory the occurrences of each item.
  - Requires only memory proportional to #items.

- Pass 2: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent.
  - memory requirements?

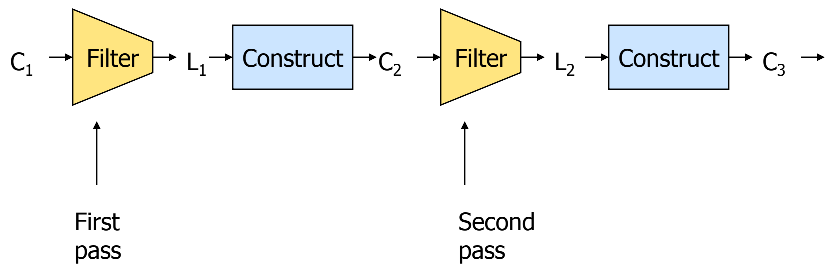
A-Priori Algorithm --- (2)

- Pass 1: Read baskets and count in main memory the occurrences of each item.
  - Requires only memory proportional to #items.
- Pass 2: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent.
  - Requires memory proportional to square of frequent items only.

Picture of A-Priori



Frequent Triples, Etc.



A-Priori for All Frequent Itemsets

- One pass for each *k*.
- Needs room in main memory to count each candidate *k* –tuple.
- For typical market-basket data and reasonable support (e.g., 1%), *k* = 2 requires the most memory.

The Apriori Algorithm—An Example

Database TDB

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

$Sup_{min} = 2$

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

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 *abd*
    - *acde* from *acd* and *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-joining L_{k-1}

  insert into $\boldsymbol{C_k}$

  select $\boldsymbol{p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}}$

  from $\boldsymbol{L_{k-1}\ p, L_{k-1}\ q}$

  where $\boldsymbol{p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}}$
- Step 2: pruning

  forall ***itemsets c in*** $\boldsymbol{C_k}$ do

      forall ***(k-1)-subsets s of c*** do

          if *(s is not in* L_{k-1}*)* **then delete** *c* **from** C_k

How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a *hash-tree*
  - *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

Where are the Association Rules?

- so far we have seen how A-priori efficiently computes all the frequent itemsets
- but how are the association rules generated from the frequent itemsets?

Association Rule Generation

- given the frequent itemsets, generate association rules as follows
  - for each frequent itemset l
    - generate all non-empty subsets of l
  - for each non-empty subset s of l
    - output association rule: s -> (l-s)

Association Rule Generation

- given the frequent itemsets, generate association rules as follows
  - for each frequent itemset l
    - generate all non-empty subsets of l
  - for each non-empty subset s of l
    - output association rule: s -> (l-s), if supp(l)/supp(s)>=c

Association Rule Generation

- given the frequent itemsets, generate association rules as follows
  - for each frequent itemset *l*
    - generate all non-empty subsets of *l*
  - for each non-empty subset s of *l*
    - output association rule: *s* -> (*l*-*s*), if supp(*l*)/supp(*s*)>=*c*
- we know supp(rule)>=s
  - generated from frequent itemsets