Association Rule Mining

Basics, Apriori

Huiping Cao

Motivation

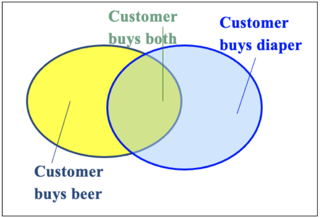
- Market basket analysis
  - Products that are often bought together by a customer
- Applications
  - Design store layouts $\rightarrow$ combined sale
  - Discount for package sale

What Is Frequent pattern?

- 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.

Basic Concepts: Frequent Itemsets 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 $X = \{x_1, \cdots, x_k\}$
- 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

Basic concepts

- Support$(X \rightarrow Y) = P(X \cup Y) = \frac{count(X \cup Y)}{total\ \ transaction\ \ count}$
  - Relative support
  - Absolute support: occurrence frequency vfill
  - $X \cup Y$: means the transaction contains all the items in X and Y; does NOT mean that it contains EITHER X or Y
- K-itemset
  - Frequent k-itemset L_k
- Confidence$(X \rightarrow Y) = P(Y|X)$
  - $= \frac{support\_count(X \cup Y)}{support\_count(X)}$

Basic Concepts: Frequent Itemsets 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

- Let supmin = 50%, confmin = 50%
  Freq. Itemset.: {A:3, B:3, D:4, E:3, AD:3}
  Association rules:
  $A \to D$ (60%, 100%)
  $D \to A$ (60%, 75%)

{A}: 3
{B}: 3
{D}: 4
{E}: 3
{AD}: 3
A-> D <==> {A} -> {D}
count(A,D): 3
total count: 5
Support= 3/5

count(A,D) = 3
count(A) = 3
conf. = 3/3=1.0

D->A
count(A,D)=3
count(D)=4
conf. = 3/4

Association rule mining

- Two-step process
  - Find all frequent itemsets (Expensive)
  - Generate strong association rules from the frequent itemsets

{A}. \emptyset->{A}
{A}->\emptyset

- Monotonicity
  - If any itemset is frequent, all its subset are frequent
- Challenge
  - A large itemset contains a combinatorial number of sub-itemsets, e.g., $\{a_1, \cdots, a_{100}\}$ contains $2^{100} - 1$ sub-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 & SrikantVLDB'94)
  - Freq. pattern growth (FPgrowthHan, Pei & Yin SIGMOD'00)
  - Vertical data format approach (CharmZaki& Hsiao SDM'02)

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! (Agrawal & Srikant VLDB'94, Mannila, et al. KDD' 94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset

    $L_1 \rightarrow L_2$ L_3 L_4
  - 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 AlgorithmAn Example

$Sup_{min} = 2$

Database TDB

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

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
- $L_1 = \{\text{frequent items}\}$
- **for** $(k = 1;\ L_k \neq \emptyset,\ k++)$ **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
  - **end for**
- **end for**
- L_{k+1} = candidates in C_{k+1} with min_support
- Return $\cup_k L_k$;

Important Details of Apriori

- How to generate candidates? $L_k \rightarrow C_{k+1}$
  - 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

$k-2$

- Step 1: self-joining L_{k-1} **insert into** C_k
  select $p.item_1, p.item_2, \cdots, p.item_{k-1}, q.item_{k-1}$ K
  from L_{k-1} p, L_{k-1} q
  where $p.item_1 = q.item_1, \cdots, p.item_{k-2} = q.item_{k-2}$,
  $p.item_{k-1} < q.item_{k-1}$

- Step 2: pruning
  for all itemsets c in C_k do
  - **for all** (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 very huge
  - One transaction may contain many candidates
- Naive Method:
  - Compare each transaction against every candidate itemset
  - Expensive!

Support counting (1)

- Enumerate itemsets
- {1,2,3,4,5} the 3-itemsets are
  - 1 [2, 3, 4, 5]
    - 12[3, 4, 5] => (1, 2, 3), (1, 2, 4), (1, 2, 5)
    - 13[4, 5] => (1, 3, 4), (1, 3, 5)
    - 14[5] => (1, 4, 5)
  - 2[3, 4, 5]
    - 23[4, 5] => (2, 3, 4), (2, 3, 5)
    - 24[5] => (2, 4, 5)
  - 3[4, 5]
    - 34[5] => (3, 4, 5)

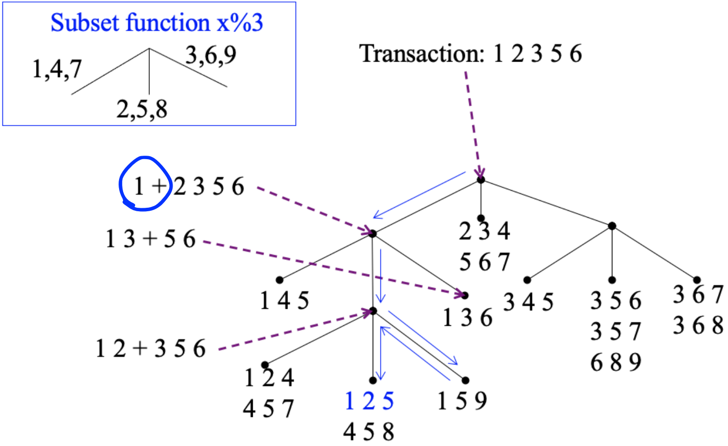
Support counting (2)

- Hash-tree based method:

  C_k

  - Candidate itemsets are stored in a hash tree
  - Leaf node of hash-tree contains a list of candidate itemsets and counts
  - Interior node contains a hash table
  - Subset function: finds all the candidates contained in a transaction

Example: Counting Supports of Candidates



Complexity

- Support Threshold
  - Maximum size of frequent itemsets
- Number of items (dimensionality)
- Number of transactions
- Average transaction width
  - More frequent itemsets

L_k 2^l

Challenges of Frequent Pattern Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

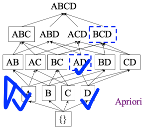
Partition: Scan Database Only Twice

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In VLDB'95

Partition based

- A partition p of the database refers to any subset of the transactions.
  - Any two different partitions are non-overlapping.
  - Local support of a partition: the fraction of transactions containing that item in a partition

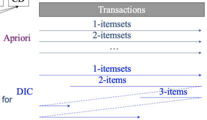
DIC: Reduce Number of Scans



Itemset lattice

S. Brin R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*

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



Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1, i_2, \cdots, i_{100}$
    \# of scans: 100 # of candidtes: $2^{100} - 1 = 1.27 * 10^{30}$
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

References

- Chapter 5: Introduction to Data Mining (2nd Edition) by Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar
- Implementation of Apriori algorithm with Python 2.7 and 3.3 - 3.5: `https://pypi.org/project/apyori/`

R