Fast Algorithms for Mining Association Rules

Rakesh Agrawal , Ramakrishnan Srikant

Presented by Wenhao Xu

Discussion led by Sophia Liang

Outline

- This is an important paper because
  - VLDB 10 Years Best Paper Award
  - Has been 1st highest cited paper of all papers in the fields of databases and data mining until 2007 in Citeseer
  - 2009 Citeseer Citations: Rank 18 in all computer science papers
  - Two authors all better jobs!!!

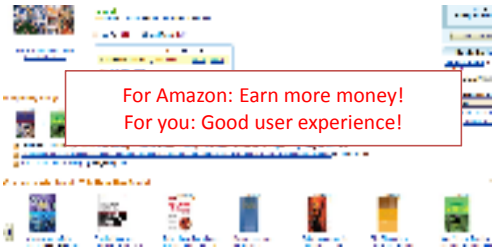
Why it is so important?

- It addresses an important problem.
- It proposes an algorithm that is better than previous algorithms
- Lots of papers afterwards are based on its basic concepts

Agenda

- What is the problem?
- Apriori Algorithm
- What are its basic concepts?
- Recent development
- Conclusion

Example of Association Rule Mining



Amazon

Example & Notions

Transaction	Items
1	{milk, diaper, beer, Coke}
2	{milk, bread}
3	{milk, bread, beer, diaper}
4	{milk, bread, diaper, coke}
5	{bread, diaper, beer, eggs}

{milk, diaper} → {beer}

{milk} → {bread}

- **Item Sets**: a set of items, like {milk, diaper}, is an item set;
- **Association rule**: implication in the form of ***X → Y; X*** and Y are both item sets. Like {milk, diaper} → {beer}
  - Implication means co-occurrence, not causation
- **Support** of the rule: the fraction of transactions that contain both ***X*** and ***Y. I.e.*** F***({X, Y})***

  S(({milk, diaper} → {beer}) = F({milk, diaper, beer}) = 2/5
- **Confidence** of the rule: the ratio of transactions that contain ***X*** contain ***Y,*** i.e. F(***X, Y***)/F(***X***)

  C({milk, diaper} → {beer}) = F({milk, diaper, beer})/F({milk, diaper}) = (2/5)/(3/5) = 2/3

Formal definition: Association Rule Mining

- Given a **large** set of transaction D, generate all association rules that have support and confidence greater than the user-specified minimum support (called ***minsup***), and minimum confidence (called ***minconf***) respectively.
- ***Minsup & Minconf*** : ensure usefulness
- **Large**:
  - A significant of data sets in data mining
  - require effective algorithms

Generic Algorithms

- Step 1: Find all itemsets that have transaction support above minimum support. These itemsets are called ***large itemsets***.
  - Focus of this paper: find large itemsets
    - AIS, SETM
    - Apriori, AprioriTid, AprioriHybrid
- Step 2: Use the large itemsets to generate the desired rules.
  - A straightforward algorithm:

```
For every large itemset L
    for every non-empty subset a of L,
        rule <- a→ (L-a)
        if(C(rule) >= minconf )
            output
    endfor
endfor
```

- Refer to <fast algorithms for mining association rules in large databases> for a fast algorithm

```
L1 = {large 1-itemsets};
for ( k = 2; Lk-1 ≠ ∅; k++ ) do begin
   Ck = apriori-gen(Lk-1);  //
   forall transactions t ∈ D do begin
      Ct = subset(Ck, t);  // C
      forall candidates c ∈ Ct do
         c.count++;
   end
   Lk = {c ∈ Ck | c.count ≥ minsup}
end
Answer = ⋃k Lk;
```

Apriori-gen(L_{k-1})

Subset(Ck, t)

- Ba
  -
  -
- As in lex
- Basic steps:
  - Generate candidate k-itemsets from large (k-1)-itemsets
  - For each candidate k-itemsets, calculate its support;
  - If its support is larger than minsup, add it to large k-itemsets
  - Continue the above three steps by adding 1 to k until no large k itemsets are found.

Apriori-gen

- Join step

Get the superset of the set of all large k-itemsets from (k-1)-itemsets

```
insert into Ck
select p.item1, p.item2, ..., p.itemk-1, q.itemk-1
from Lk-1 p, Lk-1 q
where p.item1 = q.item1, ..., p.itemk-2 = q.itemk-2,
      p.itemk-1 < q.itemk-1;
```

Prune step

Delete all itemsets that have some (k-1)-subsets which are not in (k-1) large itemsets

```
forall itemsets c ∈ Ck do
   forall (k-1)-subsets s of c do
      if (s ∉ Lk-1) then
         delete c from Ck;
```

Example

L_3 = { {1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4} }

After joining

Where is {1,4,5}?

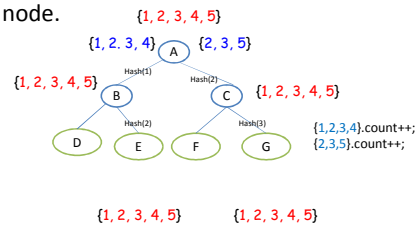
{ {1 2 3 4}, {1 3 4 5} }

After pruning

{1 2 3 4}

Subset

- Candidate itemsets are stored in a hash-tree
- Leaf-node: contains a list of itemsets
- Interior node: contains a hash table, each bucket of the hash table points to a children node.



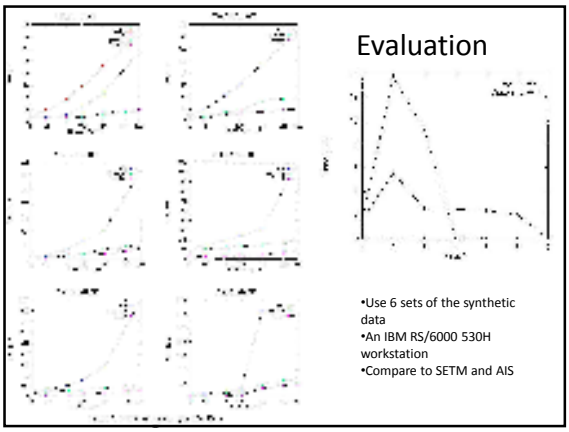
AprioriTid & AprioriHybrid

```
L1 = {large 1-itemsets};
for ( k = 2; Lk-1 ≠ ∅; k++ ) do begin
   Ck = apriori-gen(Lk-1); // New candidates
   forall transactions t ∈ D do begin
      Ct = subset(Ck, t); // Candidates contained in t
      forall candidates c ∈ Ct do
         c.count++;
   end
   Lk = {c ∈ Ck | c.count ≥ minsup}
end
Answer = ⋃k Lk;
```

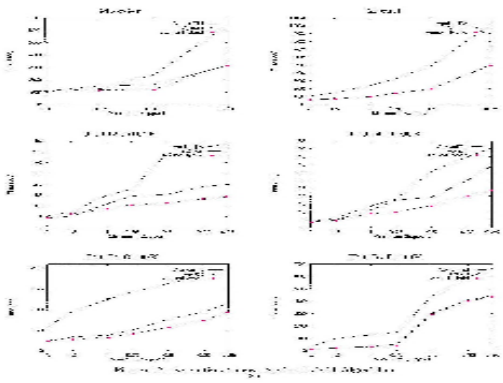
Scan the whole database every time

- Still use apriori-gen to generate candidate itemsets
- Try to reduce the times of scan the database
  - Use a candidate set (called D_k) that include TID of the corresponding transaction
  - Due to the time limit, refer to the paper by yourself.
- D_k could be smaller than the whole database when k is large and can fit into the memory
- However, may be slower than Apriori because when k is small, D_k is even larger than the original transaction.
- AprioriHybrid to combine the benefits of Apriori and AprioriTid by using a heuristic to swich from Apriori to AprioriTid on the fly.

Evaluation



- Use 6 sets of the synthetic data
- An IBM RS/6000 530H workstation
- Compare to SETM and AIS



Recent Development

```
1)  L_1 = {large 1-itemsets};
2)  for ( k = 2; L_{k-1} ≠ ∅; k++ ) do begin
3)     C_k = apriori-gen(L_{k-1}); // New candidate
4)     forall transactions t ∈ D do begin
5)        C_t = subset(C_k, t); // Candidates contained in t
6)        forall candidates c ∈ C_t do
7)           c.count++;
8)     end
9)     L_k = {c ∈ C_k | c.count ≥ minsup}
10) end
11) Answer = ∪_k L_k;
```

Inefficient when there is a large number of large itemsets and /or long large itemsets.
A large itemset with the size of 100, need to generate $2^{100}-2$ candidates in total.

- FP-tree
  - Refer to "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Jiawei Han, Jian Pei, et al.
  - About an order of magnitude faster than Aprori
- I don't find other significant improved approaches.

Conclusion

- Important problem
- Good paper as a foundation of association-rule mining
- Can be improved
  - Fp-tree

Thanks for your attention!