ASSOCIATION RULE MINING

Everything in the world is repeated.
Francis Bacon

Data Mining

Outline

2

- Basic concepts
- Frequent itemset mining
- Association Rule Mining

Motivation

3

- What are the products that often bought together?
- Is beer usually bought with diapers?
- Is the milk bought when both toasts and honey are bought together?

Basic definitions

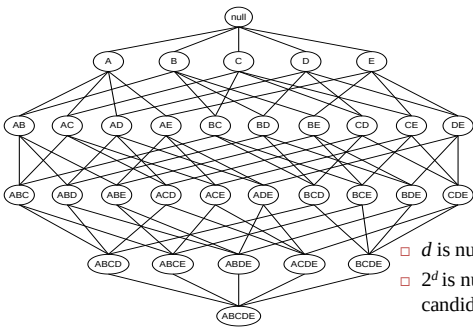
4

- *Itemset* – a collection of one or more items.
- *k-itemset* – an itemset that contains k items.
- *Support count* – frequency of occurrence of an itemset.
  - P({bread,milk,sugar})=2
- Support – fraction of transactions that contain an itemset.
  - sup({bread,milk,sugar})=0.4
- *Frequent itemset* – an itemset whose support is greater than or equal to a *minsup* threshold.

TID	Items
1	bread, milk
2	bread, coffee, eggs, sugar
3	milk, coffee, coke, sugar
4	bread, coffee, milk, sugar
5	bread, coke , milk, sugar

How many candidates?

5



- d is number of items
- 2^d is number of candidates

Basic approaches

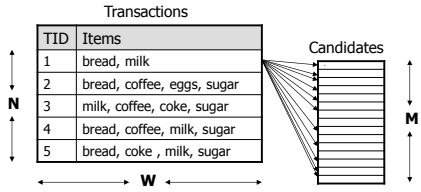
6

- Brute force
- Apriori
- Vertical data format
- FP-growth

Brute force approach

7

- Each itemset in the lattice is a *candidate* frequent itemset
- Count the support of each candidate by scanning the database
- Match each transaction against every candidate
- Complexity is O($N \cdot M \cdot W$)=O($N \cdot 2^d \cdot W$)



Frequent itemset generation strategies

8

- Reduce the number of candidates (M)
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases
  - Used by vertical-based mining algorithms
- Reduce the number of comparisons ($N \cdot M$)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Reducing number of candidates

9

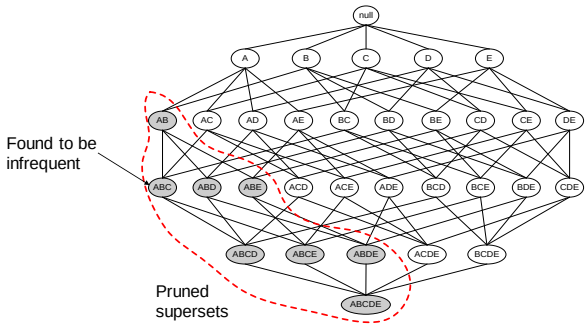
- *Anti-monotone property* of support
  - Support of an itemset never exceeds the support of its subsets (e.g. *sup*({bread, milk, chips})≤*sup*({bread, milk}))

$$\forall X, Y : (X \subseteq Y) \Rightarrow \sup(X) \geq \sup(Y)$$

- *Apriori* principle
  - If an itemset is frequent, then all of its subsets must also be frequent
  - Converse assertion: if some subset is infrequent then whole set is infrequent

Pruning using Apriori principle

10



Finding frequent itemsets using Apriori

11

Transactions

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

***minsup*=2**

Apriori algorithm

12

- Generate frequent 1-itemsets
- Let k=1
- Repeat until no new frequent itemsets are identified
  - Generate candidate $(k+1)$-itemsets using frequent k-itemsets
  - Prune candidate itemsets containing infrequent k-subsets
  - Count the support of each candidate by scanning the DB
  - Eliminate candidates that are infrequent, leaving only those that are frequent

Apriori algorithm

13

C_k: candidate itemset of size k
L_k: frequent itemset of size k

```
L1 = {frequent items};
for (k=1; Lk != ∅; k++) do begin
 Ck+1 = candidates generated from Lk;
 for each transaction t in database do begin
  increment the count of all candidates in Ck+1
   that are contained in t
  Lk+1 = candidates in Ck+1 with minsup
 end
return ∪k Lk;
```


Important details of Apriori

14

- 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

15

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

```
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
```

- Step 2: pruning

```
forall itemsets c in Ck do
  forall (k-1)-subsets s of c do
    if (s is not in Lk-1) then delete c from Ck
```

Factors affecting complexity

16

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets; this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - this may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Improving the efficiency of Apriori

17

- Reduce the number of comparisons
- Reduce the number of transactions
- Partitioning the data to find candidate itemsets
- Sampling: mine on a subset of the given data

Reducing number of comparisons

18

- To reduce the number of comparisons while scanning the transactions database to determine the support of each candidate itemset we can store the candidates in a hash table.
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

h(i1,i2) = (Ord(i1) * 10 + Ord(i2) mod 7

H2

minsup = 3

h(i1,i2)	0	1	2	3	4	5	6
Itemsets' support	**2**	**2**	**4**	**2**	**2**	**4**	**4**
Itemsets	{A,D} {C,E}	{A,E} {A,E}	{B,C} {B,C} {B,C} {B,C}	{B,D} {B,D}	{B,E} {B,E}	{A,B} {A,B} {A,B} {A,B}	{A,C} {A,C} {A,C} {A,C}

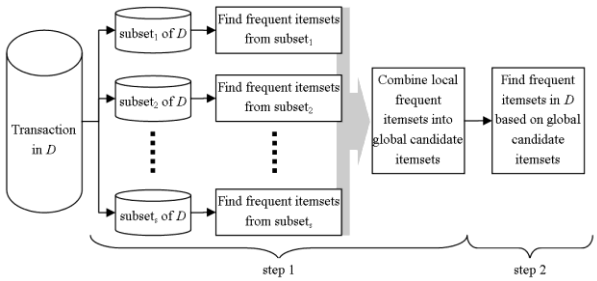
Reduce the number of transactions

19

- A transaction that doesn't contain any frequent *k*-itemset can not contain any frequent *j*-itemset, for any *j*>*k*. So such transaction can be marked or removed from subsequent scans of the database for *j*-itemsets.

Partitioning the data to find candidate itemsets

20



Sampling: mine on a subset of the given data

21

- Pick random sample *S* of the given data *D* (S should fit in the available memory)
- Search for frequent itemsets in *S* instead in *D*. Lower *minsup* to reduce the number of missed frequent itemsets.
- Find the L_s, frequent itemsets in *S*.
- The rest of the database can be used to compute the actual frequencies of the itemset in L_s.
- If L_s doesn't contain all the frequent itemsets in *D*, then a second pass will be needed.

Bottleneck of frequent-pattern mining

22

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100}-1 = 1.27*10^{30}$!
- Bottleneck is candidate-generation-and-test

Vertical data format

23

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

tid-lists

Vertical data format

24

- Determine support of any *k*-itemset by intersecting tid-lists of two of its (*k*-1) subsets.

A
1
4
5
6
7
8
9

∩

B
1
2
5
7
8
10

→

AB
1
5
7
8

- Pros: very fast support counting
- Cons: intermediate tid-lists may become too large for memory

Using RDBMS and SQL

25

- Store data in relational database, implement mining algorithms in SQL
- Pros
  - No need to export data for external mining utilities and import mining results into database
  - No memory limits
- Cons
  - RDBMS will lose the competition with mining utilities in case of data fits in memory
  - SQL has less flexibility than high-level programming languages and may produce cumbersome source code

Using RDBMS and SQL

26

TID	Transaction
1	bread, milk
2	bread, coffee, eggs, sugar
3	milk, coffee, cola, sugar

TID	Item
1	bread
1	milk
2	bread
2	coffee
2	eggs
2	sugar
3	milk
3	coffee
3	cola
3	sugar

Candidate generation in SQL

27

```
insert into Cand
 select *
 from TDB
 where item in (
  select item
  from TDB
  group by item
  having count(*) >= minsup);
```

Finding frequent itemsets in SQL

28

```
-- Frequent 2-itemsets
select A.item, B.item, count(A.tid) as
sup_count
from Cand A, Cand B
where A.tid=B.tid and A.item<B.item
group by A.item, B.item
having count(A.tid) >= minsup;

-- Frequent itemsets for k>2
-- How to implement k:=k+1 ?
```


Association rules

29

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

TID	Items
1	bread, milk
2	bread, coffee, eggs, sugar
3	milk, coffee, cola, sugar
4	bread, coffee, milk, sugar
5	bread, cola, milk, sugar

Examples of Association Rules

{sugar} → {coffee},
{milk, bread} → {eggs, cola},
{coffee, bread} → {milk},
{milk} → {coffee, bread},
…

Association rule

30

- *Association rule* is an implication expression of the form X→Y ("if X then Y"), where X and Y are itemsets
- *Rule support*
  - is a fraction of transactions that contain both X and Y
  - *sup*(X→Y)=P(X,Y)/|TID|
- *Rule confidence*
  - shows how often items in Y appear in transactions that contain X
  - *conf*(X→Y)=P(X,Y)/P(X)

Support and confidence of the association rule

31

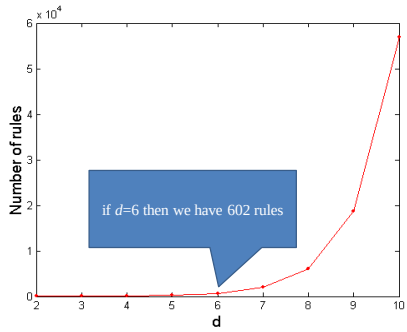
TID	Items
1	bread, milk
2	bread, coffee, eggs, sugar
3	milk, coffee, cola, sugar
4	bread, coffee, milk, sugar
5	bread, cola, milk, sugar

- {milk, sugar}→{coffee}
  - sup=P(milk, sugar, coffee)/|TID|=2/5
  - conf=P(milk, sugar, coffee)/P(milk, sugar)=2/3
- {coffee}→{milk, sugar}
  - sup=2/5
  - conf=P(coffee, milk, sugar)/P(coffee)=2/4

Computational complexity

32

- *d* items
  - itemsets: 2^d
  - rules: $3^d-2^{d+1}+1$



Association rule mining task

33

- Given a set of transactions T, the goal of association rule mining is to find all the rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold
- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds
  - Computationally prohibitive!

Mining association rules

34

TID	Items
1	bread, milk
2	bread, coffee, eggs, sugar
3	milk, coffee, cola, sugar
4	bread, coffee, milk, sugar
5	bread, cola, milk, sugar

Rule	sup	conf
{milk,sugar}→coffee	0.4	0.67
{milk,coffee}→sugar	0.4	1.0
{sugar,coffee} →milk	0.4	0.67
coffee→{milk,sugar}	0.4	0.67
sugar→{milk,coffee}	0.4	0.5
milk→{sugar,coffee}	0.4	0.5

- Observations
  - All the above rules are binary partitions of the same itemset: {milk, sugar, coffee}
  - Rules originating from the same itemset have identical support but can have different confidence

Mining association rules

35

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support grater than *minsup*
  2. Rule Generation
     - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is still computationally expensive