


Association Rule Mining

Sharma Chakravarthy
 Information Technology Laboratory (IT Lab)
 Computer Science and Engineering Department
 The University of Texas at Arlington, Arlington, TX



Email: sharma@cse.uta.edu
 URL: <http://itlab.uta.edu/sharma>



Association rules



- Capture **co-occurrence** of items / events
 - Not causality, which is to be inferred by a domain expert!
- Also called **market basket analysis** / link analysis
- Input: Transactions; each T_i is a **set** of items
- Problem: Find rules that can indicate **good** co-occurrences in the data set

➤ First paper appeared in *Sigmod* 1993 (Agarwal, Imielinski, and Swami)

© Sharma Chakravarthy  3 



Motivation

- Walmart has lots of data about point of sales
 - data on each user and the basket of items that has been bought during each visit
- Similarly, phone companies have information available about phone calls made at a particular time to a location
- Also, you have logs of what urls have been visited in each session and how much time has been spent at each url or session, what has been bought etc.
- **How can the above information be leveraged for direct marketing, better placement of items on shelves etc.**
 - In other words, for **improving a business, i.e., deriving business intelligence (BI)!**

© Sharma Chakravarthy  2 

Terminology

- $I = I_1, I_2, \dots, I_m$ set of items (sold) **Large**
- T = a database of transactions t_i **very large**
- $t[k] = 1$ if t bought item I_k , $t[k] = 0$ otherwise
- An itemset is a (proper) subset of the number of items in a $T \times t_i$
- let X be a subset of items in I
 - t satisfies X if for **all** items I_k in X , $t[k] = 1$

© Sharma Chakravarthy  4 

Association Rules

- Association rule mining was a departure from the prevalent mining problems and approaches
- Classification was being used for direct marketing, load approval etc.
- There was nothing that analyzed point of sales to figure out what items are being bought together
- Note that this varies from store to store based on demographics such as population served, their buying habits, income of the population, etc.

Confidence measure P(Y | X)

This indicates how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears. In Table, the confidence of {apple -> beer} is 3 out of 4, or 75%.

| | |
|---------------|--|
| Transaction 1 | |
| Transaction 2 | |
| Transaction 3 | |
| Transaction 4 | |
| Transaction 5 | |
| Transaction 6 | |
| Transaction 7 | |
| Transaction 8 | |

$$\text{Confidence } \{\text{apple} \rightarrow \text{beer}\} = \frac{\text{Support } \{\text{apple, beer}\}}{\text{Support } \{\text{apple}\}}$$

Support measure

This says how often an itemset occurs, as measured by the **proportion of transactions** in which an itemset appears. In Table 1, the support of {apple} is 4 out of 8, (50%)

| | |
|---------------|--|
| Transaction 1 | |
| Transaction 2 | |
| Transaction 3 | |
| Transaction 4 | |
| Transaction 5 | |
| Transaction 6 | |
| Transaction 7 | |
| Transaction 8 | |

$$\text{Support } \{\text{apple}\} = \frac{4}{8}$$

support of itemset {apple, beer, rice} is **2 out of 8**, or 25%.

Lift measure

This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is. In Table, the lift of {apple -> beer} is 1, which implies **no association** between items. A lift value greater than 1 means that item Y is *likely* to be bought if item X is bought, while a value less than 1 means that item Y is *unlikely* to be bought if item X is bought

| | |
|---------------|--|
| Transaction 1 | |
| Transaction 2 | |
| Transaction 3 | |
| Transaction 4 | |
| Transaction 5 | |
| Transaction 6 | |
| Transaction 7 | |
| Transaction 8 | |

$$\text{Lift } \{\text{apple} \rightarrow \text{beer}\} = \frac{\text{Support } \{\text{apple, beer}\}}{\text{Support } \{\text{apple}\} \times \text{Support } \{\text{beer}\}}$$

Ratio of observed Support to that Expected If X and Y were **independent**

Association rule mining

- Consider a large number of transactions each containing items associated with that transaction
 - Association rules are of the form $X \rightarrow Y$ to mean that whenever X occurs, there is a strong correlation of Y occurring!
 - Here X & Y are sets (e.g., items that have been bought in the same transaction), such that $X \cap Y = \emptyset$.

$$\text{Support } (X \rightarrow Y) = P(X \cup Y) =$$

$$\frac{\text{Count of transactions containing the items } X \cup Y = \text{frequency}(X, Y)}{\text{Total number of transactions } N}$$

$$\text{Confidence } (X \rightarrow Y) P(Y | X) = \frac{\text{Support } (X \cup Y)}{\text{Support } (X)} \text{ conditional probability!}$$

$$\text{Lift } (X \rightarrow Y) = \frac{\text{Support } (X \cup Y)}{\text{Support } (X) * \text{Support } (Y)} \quad 1 < \text{lift} > 1$$

$$P(X, Y) / P(X) * P(Y)$$

Example

| Item Tid | Bread | Cake | Eggs | Milk | Sugar |
|----------|-------|------|------|------|-------|
| 100 | X | | X | X | |
| 200 | | X | X | | X |
| 300 | | X | X | X | X |
| 400 | | X | | | X |

Problem Statement:
Find all the Associations among the items such that the **Support is 50%** and **Confidence is 75%**

Support of {Eggs, Milk} = 50.0%

Rules:

Eggs => Milk Confidence = 66.7%
Milk => Eggs Confidence = 100.0%

Rules

| Head | Symbol | Body | Support | Confidence |
|-------|--------|-------|---------|------------|
| Cake | => | Sugar | 75.0 | 100.0 |
| Sugar | => | Cake | 75.0 | 100.0 |
| Milk | => | Eggs | 50.0 | 100.0 |

Association Rules: Details

- To discover associations, we assume that we have a set of transactions, each transaction being a list of items (e.g., list of books, items bought)
- Suppose A and B appear together in only 1% of the transactions but whenever A appears there is 80% chance that B also appears
- The 1% presence of A and B together is called the **support** of the rule and 80% is called the **confidence** of the rule ($A \rightarrow B$)

Association rules

- **Beer → diapers** rule became a widely used example to illustrate that fathers watching super bowl (or sports) and taking care of babies shopped for these 2 items together!
- Other provoking examples tried to enhance the utility of this approach!

Association Rules

- Support indicates the frequency of the pattern. A minimum support (**min_sup**) is necessary if an association is going to be of some business value.
- A user might be interested in finding **all** associations which have x% support with y% confidence
- Also, all associations satisfying additional **user constraints** may be needed
- Also, associations need to be found **efficiently** from large data sets (need for real data sets and not samples)
- Confidence denotes the strength of the association. In addition, Lift can also be used



Number of itemsets

- If there are **n items**, how many itemsets are possible?
 - ${}_n C_1 + {}_n C_2 + \dots + {}_n C_n = 2^n$
 - For $n = 100$, 2^{100} is approx. $1.27 * 10^{30}$
 - Typically, **tens of thousands** (Walmart sells more items than that)
- How many transactions? (basket or point of sales)
 - **Typically in Millions**
- The problem is to count frequency of itemsets satisfying **min_sup** and generate rules satisfying **min_conf**
 - So, why is this a problem?

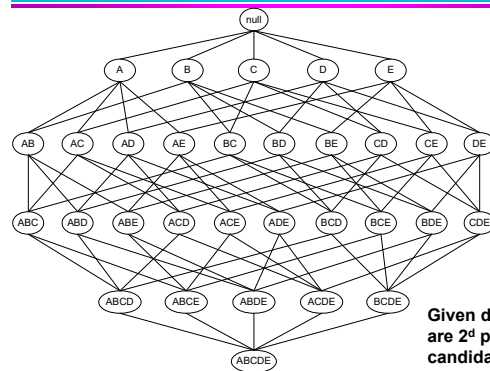


Frequent Item

- A **candidate itemset** is any **valid itemset**
- A **frequent itemset** is one that **satisfies min_sup**.
- Conceptually, finding association rules, is a simple two step approach:
 - **Step 1** - discover all **frequent items** that have support above the minimum support required
 - **Step 2** - Use the set of frequent items to generate all association rules that have high enough confidence
- Once we have frequent items (step 1) along with **count**, we can generate (enumerate) all rules to satisfy **min_conf**
- Rule generation is a separate step!
- Our focus is on step 1



Frequent Itemset Generation



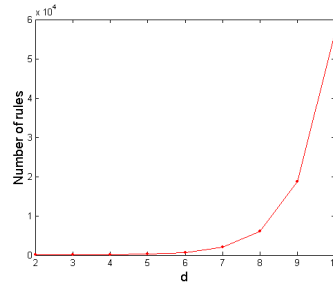
Given **d** items, there are **2^d** possible candidate itemsets

Rule generation (1)

- Large/frequent itemsets are used to generate the desired rules
- Need to generate all rules and compute its confidence to output a rule
- For **each** frequent itemset s of size k (i.e., k items), how many rules can be generated?
 1. Meaningful subsets of s that can generate rules? (how many?)
 $2^k - (k-1)$
 But how many such itemsets are there? 2^k
 2. For **every such subset** "a", output a rule of the form $x \rightarrow y$ (where $a = x \cup y$) if the ratio of $\text{support}(x \cup y)$ to $\text{support}(x)$ is at least equal to the minimum confidence. (How many?)
 $i_1 C_1 + i_2 C_2 + \dots + i_k C_{k-1}$
 Hopefully,
 1. the number of frequent itemsets may not be large! (why?)
 2. The size of the largest frequent itemset may not be large! (why?)

Computational Complexity

- Given d unique items:
 - Total number of itemsets = 2^d
 - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right]$$

$$= 3^d - 2^{d+1} + 1$$

If $d=6$, $R = 602$ rules

Rule generation (1)

- | | | |
|---|--|---|
| (a, b) [2] ${}_2C_1$ | (a, b, c) [6] ${}_3C_1 + {}_3C_2$ | (a, b, c, d) [14] ${}_4C_1 + {}_4C_2 + {}_4C_3$ |
| $a \rightarrow b$ $b \rightarrow a$ | $a \rightarrow b, c$ $b \rightarrow a, c$ $c \rightarrow a, b$ | $a \rightarrow b, c, d$ $b \rightarrow$ $c \rightarrow$ $d \rightarrow$ |
| | $a, b \rightarrow c$ $a, c \rightarrow b$ $b, c \rightarrow a$ | $a, b \rightarrow c, d$ $a, c \rightarrow b, d$ $a, d \rightarrow b, c$ $b, c \rightarrow$ $b, d \rightarrow$ |
| general formula: | | $c, d \rightarrow$ $a, b, c \rightarrow d$ $a, b, d \rightarrow$ $a, c, d \rightarrow$ $b, c, d \rightarrow$ |
| $i_1 C_1 + i_2 C_2 + \dots + i_k C_{k-1}$ | | |
| This is exponential! | | |

Frequent Itemset Generation Strategies

- 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 DHP (Direct Hashing and Pruning) and vertical-based mining algorithms
- Reduce the **number of comparisons** (NM)
 - Use efficient data structures to store the candidates or transactions
 - No need to match every candidate against every transaction

Association Rules

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

Algorithms

- AIS
- SETM
- Apriori
- AprioriTid
- AprioriHybrid
- FP-Tree

Storage issues

- It is possible to generate itemsets systematically although it is very large
 - **Main-memory approaches may be not possible!**
- Counting is the more expensive part
 - As memory may not be sufficient to hold data
 - Making one (or multiple) passes on **data stored on secondary storage** for counting is going to be expensive
- Rule generation can be expensive if number of frequent itemsets are very large and the size of frequent itemsets are large!

AIS (1993)

1. First algorithm for association rules
2. Candidate itemsets are generated and counted on-the-fly as the database is scanned
3. For each Transaction, it is determined which of the frequent itemsets of the previous pass (**frontiers**) are contained in this transaction
4. New candidate itemsets are generated by extending these frequent itemsets (frontiers) with other items in this transaction

For the first time you come across database in mining

1. Started by DBMS researchers
2. **Size of input is very large (not samples)**

AIS Algorithm

- The algorithm makes **multiple passes** over the database
- The **frontier** set for a **pass** consists of those items that are extended during the pass
- Candidate itemsets for the **current pass** are generated from the tuples in the database and the itemsets in the frontier set generated in the **previous pass**.
- Each itemset has a counter to keep track of the number of transactions in which it appears (for **min_sup** checking)
- At the end of a pass, the support for a candidate itemset is compared with **min_sup** to determine if it is frequent
- It is also determined whether it should be added to frontier set
- Algorithm terminates when frontier set is empty.

Spring 2019



CSE 6331



AIS Frontier set

- For example, let $I = \{A, B, C, D, E, F\}$ and assume that the items are ordered in **alphabetic order**.
- Further assume that the **frontier set** contains only one itemset, **AB**.
- For the database tuple $t = ABCDF$, the following candidate itemsets are generated and predicted whether it is small or large (frequent):
 - ABC expected large: continue extending
 - ABCD expected small: do not extend any further
 - ABCF expected large: cannot be extended further
 - ABD expected small: do not extend any further
 - ABF expected large: cannot be extended further
- **Statistical independence** was used to estimate support for an itemset
- **Product of prior probability and the database size is used for prediction**
 - Naïve Bayes déjà vu
- Please look up the details in the paper (as part of my lectures)

Spring 2019



CSE 6331



AIS Algorithm

- All algorithms are iterative
- In the k th pass only those itemsets that contain exactly k items are computed/generated (**candidate itemsets**). And **frequent itemsets** of size k are identified.
- Having identified some itemsets in the k th pass, we need to identify in $(k + 1)$ th pass only those itemsets that are 1-extensions (an itemset extended by exactly one item) of large itemsets found in the k th pass.
- If an itemset is small, its 1-extension is also going to be small (or not large or frequent). Thus, the frontier set for the next pass is set to candidate itemsets determined large (frequent) in the current pass, and only 1-extensions of a frontier itemset are generated and measured during a pass.
- Notion of **expected to be large** is introduced
- **Remember, apriori property was not yet established!**

Spring 2019



CSE 6331



AIS Disadvantages

- In AIS candidate sets were generated on the fly. New candidate item sets were generated by **extending** the large item sets that were generated in the **previous pass** with other items in the transaction.
- Frontier set generates more candidate itemsets than needed! There is pruning but it is **approximate!**
- Larger number of candidate itemsets were generated as compared to the frequent itemsets

This algorithm uses some heuristics (prediction) so results may differ from ground truth

But, No false positives are generated!

Some true positives may not be generated

© Sharma Chakraborty



28



SETM Algorithm (1994)

1. Used RDBMS to store and compute association rules. Schema was tid, item (2 attributes)
2. Generate 1 item frequent itemsets using SQL GROUP BY and HAVING COUNT(*) >= min_sup
3. Itemsets have items in lexicographic order

```
SELECT r1.trans-id, r1.item, r2.item
FROM SALES r1, SALES r2
WHERE r1.trans-id = r2.trans-id AND r1.item < r2.item
```

Please understand how/why it works and why lexicographic order is needed for items

4. Min_sup counting can also done by using GROUP BY and HAVING clauses



Example of SETM

Figure 1: Customer transactions, corresponding relation, and relation C1.

Figure 2: Relations R1, C1, and R2.

Figure 3: Relation R1, C1, and R2.



SETM Algorithm

- Uses RDBMS to store and compute association rules. Schema is tid, item (2 attributes)
- Uses lexicographic order for correctness
- Uses sort-merge join (for efficiency, not correctness)

```
k := 1;
sort R1 on item;
C1 := generate counts from R1;
repeat
  k := k + 1;
  sort Rk-1 on trans-id, item1, ..., itemk-1;
  Rk' := merge-scan Rk-1, Rk; // sort-merge join was used
  sort Rk' on item1, ..., itemk;
  ck := generate counts from Rk';
  Rk := filter Rk' to retain supported patterns;
until Rk = {}
```



SETM discussion

- Candidate items are generated on-the-fly as the database is scanned (using sort-merge join), but counted at the end of the pass.
- New candidate items are generated in the same way as in AIS algorithm (which does not use a DBMS), but TID of the generating Tx is saved with candidate itemset as part of the relation
- At the end of the pass, the support count is computed using GROUP BY and non-frequent itemsets are filtered



SETM

- Showed that association rule mining can be done using SQL and RDBMS
- Sort-merge was used for speed up
- Specialized black-boxes were avoided

- Main memory is not a limitation any more
- Buffer management is handled by DBMS
- Query optimization is handled by DBMS

- Has the same limitations of AIS – generates too many candidate itemsets – most of which does not become frequent!

Frequent Itemset Generation Strategies

- 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 DHP (direct hashing and pruning) and vertical-based mining algorithms
- Reduce the number of comparisons (NM)
 - Use efficient data structures to store the candidates or transactions
 - No need to match every candidate against every transaction

Apriori class of Algorithms

- Vldb94 paper presents 2 new algorithms: Apriori and AprioriTid
- These two outperform earlier algorithms (AIS and SETM)
- The performance gap is shown to increase with the size, and ranges from a factor of 3 for small problems to more for larger problems.
- Also proposes Apriori Hybrid, which is a hybrid of the above 2 new algorithms and is found to have excellent scale up properties.

Comparison

- All the previous algorithms (AIS and SETM) that were used to determine all association rules were slow (relatively)
- This is due to the generation of a large number of sets which eventually turned out to be small (did not satisfy minimum support)
- In other words, they had lower support than the user defined minimum support.

Earlier Algorithms

- In case of the AIS and SETM, candidate sets were generated on the fly. New candidate item sets were generated by extending the large item sets that were generated in the previous pass with **ALL items in the transaction**.

The Apriori Algorithm

- The Apriori and AprioriTid algorithms generate the candidate itemsets to be counted in a pass by using **only the itemsets found frequent in the previous pass** – **without** considering ALL the transactions in the database.
- The **basic intuition** is that **all** itemsets of a large/frequent itemset must be large/frequent
- **Therefore, the candidate itemsets having k items can be generated by joining large itemsets having k-1 items, and deleting those that contain any subset that is not large.**
- **Apriori principle: if an itemset is frequent, then all its subsets must be frequent!**

Earlier Algorithms

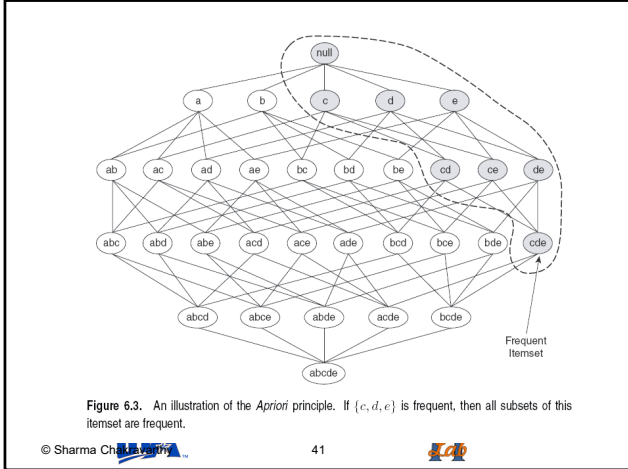
- Disadvantages
 - Algorithm makes passes over the data until the frontier set is empty; Frontier set is based on the number of itemsets that were expected to be small but turn out to be large.
 - The number of combinations that are generated that turn out to be not large is considerably greater in these 2 algorithms.

Reducing Number of Candidates

- **Apriori principle:**
 - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

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

- Support of an itemset never exceeds the support of its subsets
- This is known as the **anti-monotone** property of support



Frequent Itemset Generation

- Brute-force approach:
 - Each itemset in the lattice is a **candidate** frequent itemset
 - Count the support of each candidate by scanning the database

| TID | Items |
|-----|---------------------------|
| 1 | Bread, Milk |
| 2 | Bread, Diaper, Beer, Eggs |
| 3 | Milk, Diaper, Beer, Coke |
| 4 | Bread, Milk, Diaper, Beer |
| 5 | Bread, Milk, Diaper, Coke |

Transactions

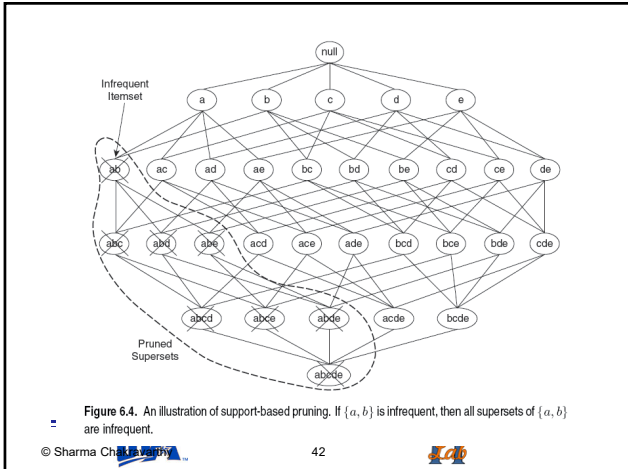
← w →

List of Candidates

↑ M ↓

- Match each transaction against every candidate
- Complexity $\sim O(NMw) \Rightarrow$ **Expensive since $M = 2^d$!!!**

© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #7



Applying Apriori Principle

| Item | Count |
|--------|-------|
| Bread | 4 |
| Coke | 2 |
| Milk | 4 |
| Beer | 3 |
| Diaper | 4 |
| Eggs | 1 |

Items (1-itemsets)

| Itemset | Count |
|-----------------|-------|
| {Bread, Milk} | 3 |
| {Bread, Beer} | 2 |
| {Bread, Diaper} | 3 |
| {Milk, Beer} | 2 |
| {Milk, Diaper} | 3 |
| {Beer, Diaper} | 3 |

Pairs (2-itemsets)

No need to generate candidates involving Coke or Eggs (why?)

Minimum Support = 3

| Itemset | Count |
|-----------------------|-------|
| {Bread, Milk, Diaper} | 3 |

Triplets (3-itemsets)

If every subset is considered, ${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$

With support-based pruning, ${}^6C_1 + {}^4C_2 + 1 = 6 + 6 + 1 = 13$

© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #8

Frequent set discovery

- Level-wise search
- Closure property of frequent sets
- Apriori algorithm
- Hashtrees to hold candidate itemsets
- Techniques to reduce I/O and computation
- Sampling (guess and correct)
- Dynamic itemset counting



Example Input

| TID | bread | hotdog | milk | coffee | mustard |
|-----|-------|--------|------|--------|---------|
| 100 | 1 | | 1 | 1 | |
| 200 | | 1 | 1 | | 1 |
| 300 | 1 | 1 | 1 | | 1 |
| 400 | | 1 | | 1 | |



Example Input

| TID | Item1 | Item2 | Item3 | Item4 | Item5 |
|-----|-------|--------|-------|--------|---------|
| 100 | Bread | | milk | coffee | |
| 200 | | hotdog | milk | | mustard |
| 300 | bread | hotdog | milk | | mustard |
| 400 | | hotdog | | coffee | |



Example of Input Table

| TID | ITEM |
|-----|------|
| 100 | 1 |
| 100 | 3 |
| 100 | 4 |
| 200 | 2 |
| 200 | 3 |
| 200 | 5 |
| 300 | 1 |
| 300 | 2 |
| 300 | 3 |
| 300 | 5 |
| 400 | 2 |
| 400 | 2 |
| 400 | 5 |

This is the input used by most algorithms. All items are numbered for efficiency of computation and for ordering them. Converted back later



The University of Texas ARLINGTON

Example of Output Rules

| Rule Head | ⇒ | Rule Body | Confidence | Support |
|-----------|---|-----------|------------|---------|
| 1 | ⇒ | 3 | 100% | 2 |
| 2 | ⇒ | 3 | 66.7% | 2 |
| 2 | ⇒ | 5 | 100% | 3 |
| 2 | ⇒ | 3, 5 | 66.7% | 2 |
| 3 | ⇒ | 1 | 66.7% | 2 |
| 3 | ⇒ | 2 | 66.7% | 2 |
| 3 | ⇒ | 5 | 66.7% | 2 |
| 3 | ⇒ | 2, 5 | 66.7% | 2 |
| 5 | ⇒ | 2 | 100% | 3 |
| 5 | ⇒ | 3 | 66.7% | 2 |
| 5 | ⇒ | 2, 3 | 100% | 3 |
| 2, 3 | ⇒ | 5 | 100% | 2 |
| 2, 5 | ⇒ | 3 | 66.7% | 2 |
| 3, 5 | ⇒ | 2 | 100% | 2 |

© Sharma Chakraborty

The Apriori steps

- Scan all transactions and find all 1-items that have support above `min_sup`. Let these be F1. (pass 1)
- Build item pairs from F1. This is the candidate set C2. Scan all transactions and find all frequent pairs in C2. Let this be F2 (support count) (pass 2)
- build sets of k items from Fk-1. This is set Ck.
- Prune Ck using the apriori principle!
 - Note this is not done in passes 1 and 2
- Scan all transactions and find all frequent sets in Ck. Let this be Fk. (pass k)
- Stop when Fk is empty for some k
- Generate all rules to satisfy confidence.
- How many # of passes in total?
- K (does not include candidate set generation, pruning)

© Sharma Chakraborty

The University of Texas ARLINGTON

Association Rules generated

| Rule Head | 'ImPLY' Symbol | Rule Body | Confidence | Support |
|------------------|----------------|---------------------|------------|---------|
| Bread | ⇒ | Milk | 90% | 10% |
| Car | ⇒ | Insurance, Register | 85% | 7% |
| Bike, Air Pumper | ⇒ | U-lock | 80% | 5% |
| Bike, Air Pumper | ⇒ | U-lock, Helmet | 60% | 3% |

For example:

© Sharma Chakraborty

Pass 2 characteristics

- Scan all transactions and find all items that have transaction support above `min_sup`. Let these be F1.
- Build item pairs from F1. This is the candidate set C2.
 - no need for pruning! (why?)
 - C2 does NOT have any subsets that are not frequent!
- C2 is larger than C in any other pass! (why?)
- n_{C_2} is larger than any other combination! Confirm this!
- Scan all transactions and find all frequent pairs in C2. Let this be F2.
- General rule: build sets of k items from Fk-1. This is set Ck. Prune Ck. Scan all transactions and find all frequent sets (support count) in Ck. This be Fk.

© Sharma Chakraborty

Apriori Algorithm

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

Example

➤ Consider the following:

| TID | Items bought |
|-----|--------------|
| 1 | {b, m, t, y} |
| 2 | {a, b, m, } |
| 3 | {m, p, s, t} |
| 4 | {a, b, c, d} |
| 5 | {a, b} |
| 6 | {e, t, y} |
| 7 | {a, b, m} |

Apriori algorithm (without pruning)

```

1  F1 = {frequent 1-itemsets};
2  For ( k=2; Fk-1 <> empty, k++ ) do
3    // generate new candidate sets
4    Ck = apriori-generation(Fk-1);
5
6  For each transaction t ∈ D do //support counting (pass)
7    Ct = subset(Ck, t); // find all candidate sets contained in t
8    For each candidate c ∈ Ct do
9      c.count++;
10   end
11 end
12 Fk = {c ∈ Ck | c.count ≥ minsup};
13 Answer = ∪k Fk;
  
```

Computing F1

Support: 30% (at least 2), confidence: 60%

Frequent itemset of size 1 (F1)

| Itemset | frequency (count) |
|---------|-------------------|
| {a} | 4 |
| {b} | 5 |
| {m} | 4 |
| {t} | 3 |

➤ These above items form the following pairs
 {{a, b}, {a, m}, {a, t}, {b, m}, {b, t}, {m, t}}.

This set is C2. Now find the support of these itemsets.

Computing F2

| Item | count | Item | count |
|-------|-------|-------|-------|
| {a,b} | 4 | {b,m} | 3 |
| {a,m} | 2 | {b,t} | 1 |
| {a,t} | 0 | {m,t} | 2 |

- {a, b}, {a, m}, {b, m}, and {m, t} have 30% support
- So, F2 is {a, b}, {a, m}, {b, m}, and {m, t}

Generating rules from F

- All frequent itemsets are:
 F1: {a}, {b}, {m}, {t} Union
 F2: {a, b}, {a, m}, {b, m}, and {m, t} Union
 F3: {a, b, m}
- No rules from F1 (why?)
- Rules from F2
 $a \Rightarrow b$ has confidence of $3/3 = 100\%$
 $b \Rightarrow a$ has confidence of $3/5 = 60\%$
 $b \Rightarrow m$ has confidence of $3/5 = 60\%$
 $m \Rightarrow b$ has confidence of $3/3 = 100\%$
- Rules from F3 (compute confidence for these)
 $a \rightarrow \{b, m\}$, $b \rightarrow \{a, m\}$, $m \rightarrow \{a, b\}$
 $\{a, b\} \rightarrow \{m\}$, $\{a, m\} \rightarrow \{b\}$, $\{b, m\} \rightarrow \{a\}$

Computing F3

- F2 is {a, b}, {a, m}, {b, m}, and {m, t}
- C3 is {a, b, m}
- Support of {a, b, m} is 2. hence F3 is {a, b, m}
- C4 is { {} }, however, empty! (why?)
- Need at least 2 F3 items to generate a C4 item!
- The algorithm stops here. Note that we did not apply the pruning step. We will come to that later.

Apriori candidate generation

- The *Apriori-generation* function takes as argument $F(k-1)$, the set of all frequent $(k-1)$ -item sets. it returns a **superset of the set of all frequent k -item sets**. The function works as follows: First, in the join step, we join $F(k-1)$ with $F(k-1)$:

```

insert into C(k)
select p.item(1), p.item(2),... p.item(k-1), q.item(k-1)
from F(k-1) as p, F(k-1) as q
where p.item(1) = q.item(1),...,p.item(k-2) = q.item(k-2),
      p.item(k-1) < q.item(k-1) //pay attention to this condition

```
- Assumes lexicographic ordering of items

Pruning step: example

- Let F_3 be $\{\{1,2,3\},\{1,2,4\},\{1,3,4\}, \{1,3,5\}, \{2,3,4\}\}$
- After the join step, C_4 will be $\{\{1,2,3,4\}, \{1,3,4,5\}\}$
- In the prune step, all itemsets $c \in C_k$ where some $(k-1)$ -subset of c is not in F_{k-1} , are deleted.
- We delete $\{1,3,4,5\}$ because the subset $\{1,4,5\}$ is not in F_3
- Hence, F_4 is $\{1,2,3,4\}$ **how does this help?**
- **Reduces the size** of F_4 from which to generate C_5
- **Still need to compute the support of F_4 itemsets!**

Comparison with AIS and SETM

- Consider a transaction $t \{1,2,3,4,5\}$.
- consider $F_3 \{\{1,2,3\},\{1,2,4\},\{1,3,4\}, \{1,3,5\}, \{2,3,4\}\}$
- AIS (and SETM) algorithms,
 - In the 4th pass will generate $\{1, 2, 3, 4\}$ and $\{1, 2, 3, 5\}$ using the frequent itemset $\{1, 2, 3\}$ and transaction t
 - Also, **an additional 3 candidate itemsets** $\{1, 2, 4, 5\}$ $\{1, 3, 4, 5\}$ $\{2, 3, 4, 5\}$ using the other large itemsets in F_3 and t which are not generated by the Apriori algorithm on account of pruning.
- Apriori is using the **monotonicity property** "all subsets of a frequent itemset are also frequent" for pruning. However, **not all** supersets of a frequent itemset are frequent.

The prune step of the algorithm

- We delete all the item sets c in $C(k)$ such that some $(k-1)$ -subset of c is not in $F(k-1)$:

```

for each item sets  $c$  in  $C(k)$  do //pass on  $C(k)$ 
  for each  $(k-1)$ -subsets  $s$  of  $c$  do //generate all subsets
    if ( $s$  not in  $F(k-1)$ ) then // check
      delete  $c$  from  $C(k)$ ;
  end for
end for
    
```

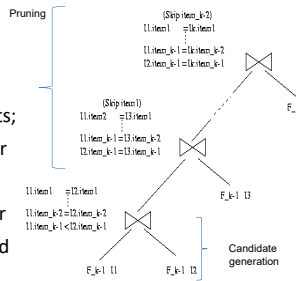
- Any subset of a frequent item set must be frequent
- Use of integers and (lexicographic) order of items is assumed!

K-way Join

- The process of support counting in K_{wj} is as follows: In any pass k :
 1. 2 Frequent itemsets of length $k-1$ are used to generate candidate itemsets of length k (C_k).
 2. Prune some of the candidate itemsets generated based on the apriori property
 3. For support counting of these candidate itemsets, k copies of input relation is joined with the C_k .
- Turns out that all of the above can be done is SQL
- As well as rule generation!

Candidate Generation and Pruning

- Prune step: additional joins with **(k-2) more copies** of F_{k-1}
- Join predicates enumerated by skipping an item at a time
- k-items have k (k-1)-item subsets; Out of that 2 have been used for generating the K item. No need to check them. Hence, the other (k-2) subsets need to be checked by doing (k-2) joins



Candidate Set Ck Generation

```

Insert into Ck
Select I1.item1, I1.item2, ..., I1.itemk-1, I2.itemk-1
From Fk-1 I1, Fk-1 I2
Where I1.item1 = I2.item1 AND
      I1.item2 = I2.item2 AND
      ...
      I1.itemk-2 = I2.itemk-2 AND
      I1.itemk-1 < I2.itemk-1
    
```

Example: $F_3: \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}\}$
 $\Rightarrow C_4: \{1, 2, 3, 4\}, \text{ and } \{1, 3, 4, 5\}.$

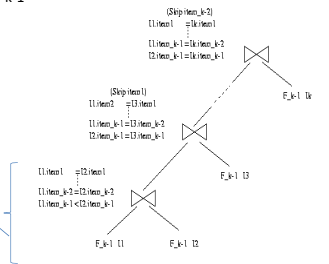


Candidate Generation and pruning in SQL

- Join step: join 2 copies of F_{k-1}

```

insert into Ck
select I1.item1, ..., I1.itemk-1, I2.itemk-1
from Fk-1 I1, Fk-1 I2
where I1.item1 = I2.item1 and ... and
      I1.itemk-2 = I2.itemk-2 and
      I1.itemk-1 < I2.itemk-1
    
```



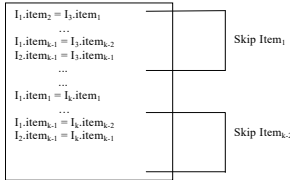
Pruning explanation

- Consider $C_k \{1\ 3\ 4\ 5\}$
- The subsets are
 - $\{1\ 3\ 4\}$ generated by skipping item at position 4
 - $\{1\ 3\ 5\}$ generated by skipping item at position 3
 - $\{1\ 4\ 5\}$ generated by skipping item at position 2
 - $\{3\ 4\ 5\}$ generated by skipping item at position 1
- First 2 have been used in the generation of $\{1\ 3\ 4\ 5\}$
- Hence, skip positions 1 and 2 or 1 through k-2 (here k is 4) to check for subsets!



SQL conditions for Pruning

Prune step: in the k-itemset of C_k , if there is any (k-1)-subset of C_k that is not in F_{k-1} , we need to delete that k-itemset from C_k .



In the above example, one of the 4-itemset in C_4 is {1, 3, 4, 5}. This 4-itemset needs to be deleted because one of the 3-item subsets {3, 4, 5} is not in F_3 .

12/13/2020



© Sharma Chakravarthy



69

SQL-92 support counting - Kway

- Join k copies of input table (T) with C_k and do a group by on the itemsets
 - insert into F_k

```

select item1, ..., itemk, count(*)
from Ck, T t1, ..., T tk
where t1.item = Ck.item1 and
...
tk.item = Ck.itemk and
t1.tid = t2.tid and t1.item < t2.item and
...
tk-1.tid = tk.tid and tk-1.item < tk.item
group by item1, item2, ..., itemk
having count(*) ≥ minsup
        
```
- requires K joins for the k th pass

12/13/2020



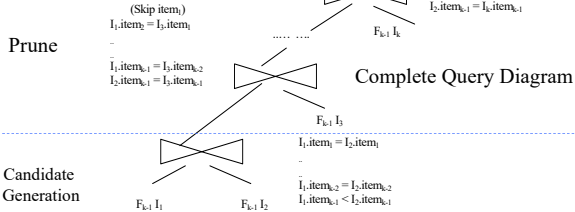
© Sharma Chakravarthy



71

Candidate Set C_k Generation

Example: $F_3: \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}\}$
 $\Rightarrow C_4: \{1, 2, 3, 4\}$, and $\{1, 3, 4, 5\}$.



12/13/2020



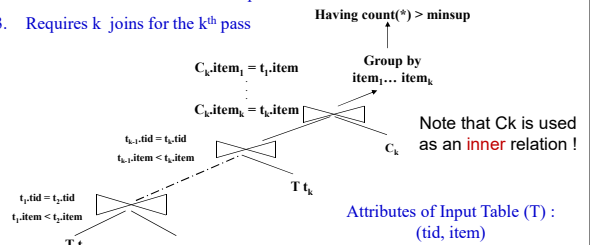
© Sharma Chakravarthy



70

Support Counting for K_{wj} in pass k

- Join C_k with k copies of T
- Follow up the join with a group by on the items and filter on minsup
- Requires k joins for the k th pass



12/13/2020



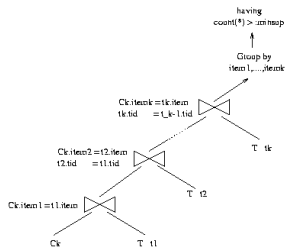
© Sharma Chakravarthy



72

K-way join plan (C_k outer)

- Series of joins of C_k with k copies of T
- Final join result is grouped on the k items



12/13/2020



© Sharma Chakravarthy



73

Writing Apriori Algorithm using SQL

- Convert given data into Transact (tid, item) relation
- Write a script (or a generator) that generates SQL for each iteration
- Create intermediate tables for C_k and F_k
- Use the stopping condition
- Once final F is generated, construct another SQL for creating rules with support and confidence
- We did performance evaluation of various Apriori algorithms for Oracle, DB2, and Sybase.

12/13/2020



© Sharma Chakravarthy



75

Difference between inner and outer C_k

- C_k needs to be materialized if inner
 - Requires additional I/O
- No materialization needed if outer
 - Can write a single (large) query for candidate generation, pruning, and support counting
 - May involve 10's of joins!
 - Current optimizers were not designed for that many join optimizations!

12/13/2020



© Sharma Chakravarthy



74

Without using SQL

- The prune step requires testing all $(k-1)$ subsets of a newly generated k -candidate itemset are present in L_{k-1}
- To make this membership test fast, large/frequent itemsets are stored in a hash table
- Candidate itemsets are stored in a **hash tree**
- Hash tree exploits the **ordering assumption**. All subsets are found by hashing the itemset

© Sharma Chakravarthy



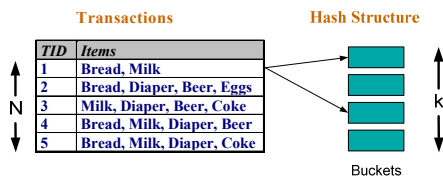
76



Reducing Number of Comparisons

- Candidate counting:

- Scan the database of transactions to determine the support of each candidate itemset
- To reduce the number of comparisons, store the candidates in a hash structure
 - ◆ Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets



Generate Hash Tree

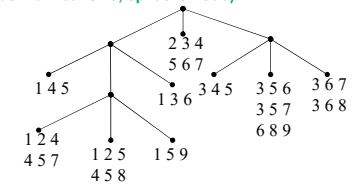
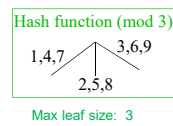
Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need: Hash function (buckets = size of itemset);

Level determines which item to hash!

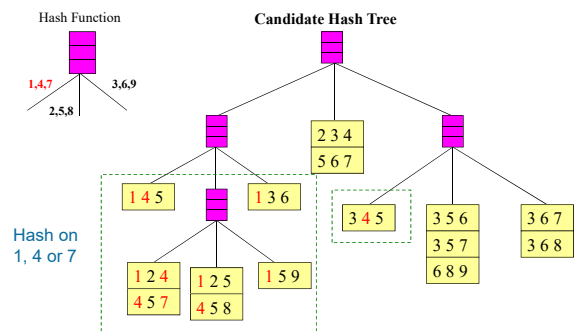
- Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)



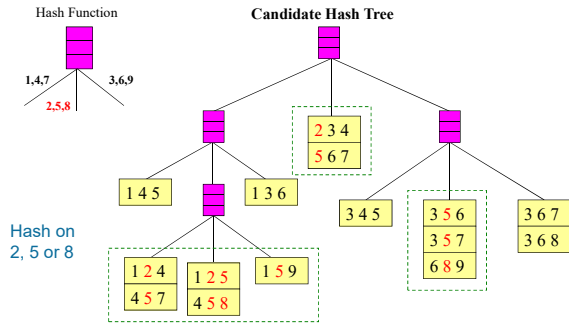
Hash Tree

- A hash tree is used that contains itemsets at leaf nodes.
- Intermediate nodes contain pointers to other leaf/intermediate nodes
- A transaction is hashed on its items to test for subset membership.
- At depth d , if the i^{th} item was used, the rest of the items after i is used for hashing.

Association Rule Discovery: Hash tree



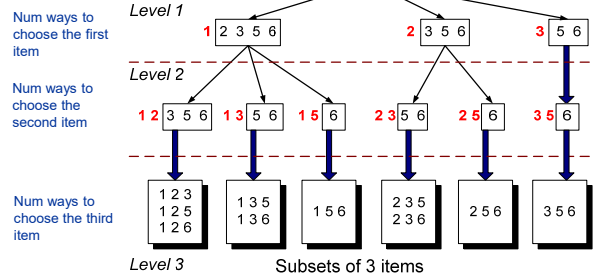
Association Rule Discovery: Hash tree



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #8

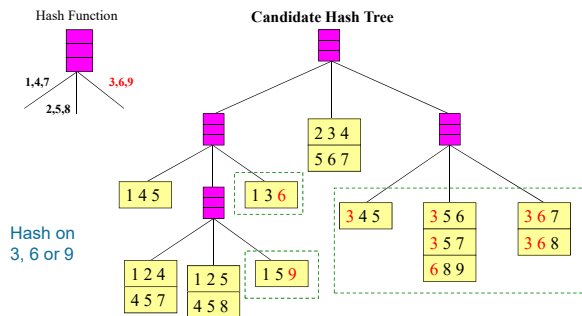
Subset Operation (support counting)

Given a transaction t , what are the possible subsets of size 3? You can only start with Items 1, 2, and 3



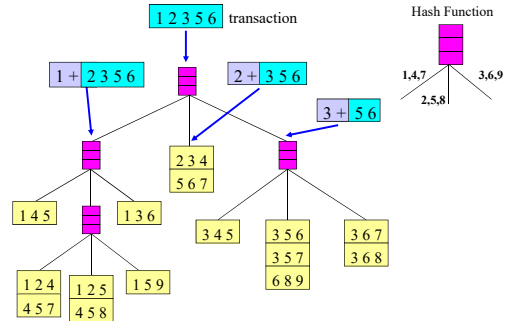
© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #9

Association Rule Discovery: Hash tree



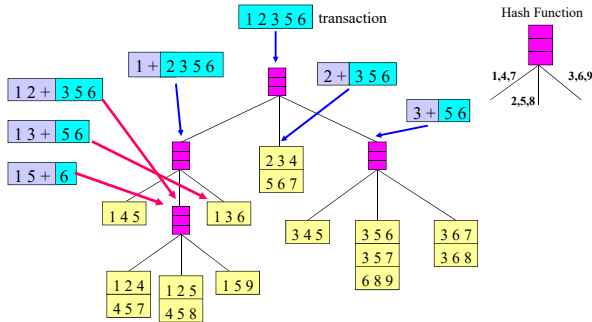
© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #10

Subset Operation Using Hash Tree



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #11

Subset Operation Using Hash Tree



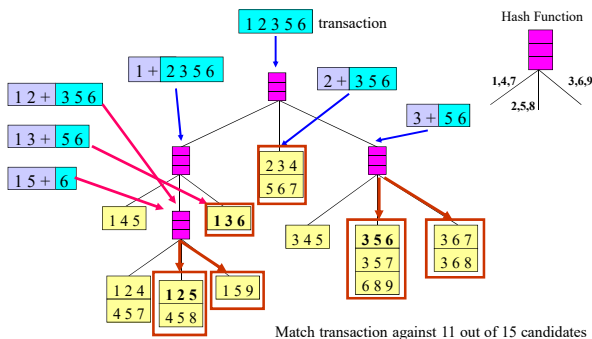
© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #7

Factors Affecting Complexity

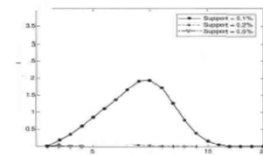
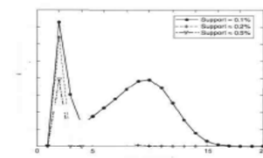
- Choice of minimum support threshold
 - lowering support threshold results in more frequent itemsets
 - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
 - more space is needed to store support count of each item
 - if number 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 i) increase max length of frequent itemsets and ii) traversals of hash tree (number of subsets in a transaction increases with its width)

© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #7

Subset Operation Using Hash Tree



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #7



Correctness

- C_k is a superset of F_k
- C_k is a superset of F_k by the way C_k is generated
- Subset pruning is based on the monotonicity property and every item pruned is guaranteed not be large
- Hence, C_k is always a superset of F_k

Impact of memory: Buffer management

- F_{k-1} fits in memory and C_k does not: generate as many C_k as possible, scan database and count support and write F_k to disk. Delete small itemsets. Repeat until all of F_k is generated for that pass.
 - # of passes on database (or transactions) is equal to the #of partitions of C_k due to memory limitation
 - Increases passes on the database!
- F_{k-1} does not fit in memory: externally sort F_{k-1} . Bring into memory F_{k-1} items in which the first $k-2$ items are the same. Generate Candidate itemsets. Scan data and generate F_k .
 - Unfortunately, pruning cannot be done (why?)

Buffer management

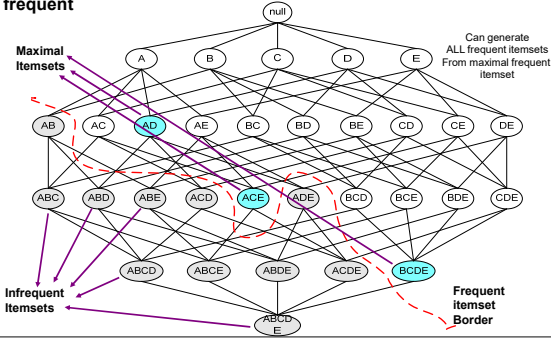
- In the candidate generation of pass k , we need storage for F_{k-1} and the candidate itemsets C_k
- In the counting phase of pass k , we need storage for C_k and at least one page to buffer the database transactions (C_t is a subset of C_k)
- Transactions are assumed be stored on the disk (whether in a database or not does not matter)

Effect of memory on pruning

- Let F_3 be $\{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{1,3,5\},\{2,3,4\}\}$ sorted on first 2 itemsets
- Suppose I can only load $\{\{1,2,3\},\{1,2,4\},\{1,3,4\}\}$ into memory
- C_4 will be $\{\{1,2,3,4\}\}$
- To test all its subsets in F_3 , I need to check $\{2,3,4\}$ is in F_3 in addition to $\{1,2,3\}$ and $\{1,2,4\}$
- However, $\{2,3,4\}$ is NOT in memory. Hence cannot be checked!
- Only checking for $\{1,2,3\}$ and $\{1,2,4\}$ is not enough!

Maximal Frequent Itemset

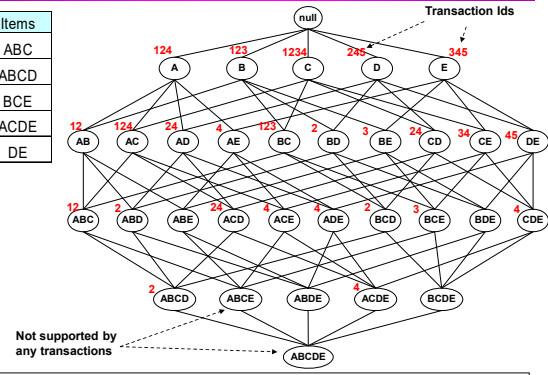
An itemset is **maximal frequent** if none of its **immediate supersets** is frequent



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #1

Maximal vs Closed Itemsets

| TID | Items |
|-----|-------|
| 1 | ABC |
| 2 | ABCD |
| 3 | BCE |
| 4 | ACDE |
| 5 | DE |



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #1

Closed Itemset

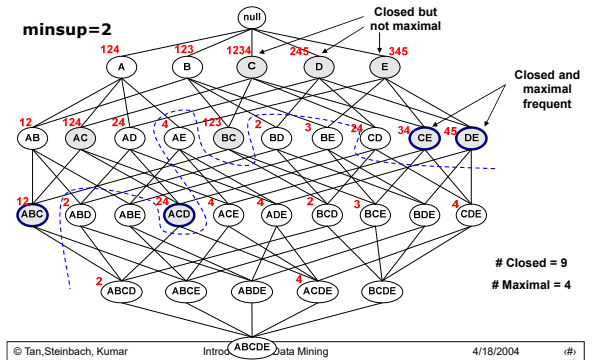
An itemset is **closed** if none of its immediate supersets has the **same support** (not min_sup!) as the itemset

| TID | Items | Itemset | Support | Itemset | Support | Itemset | Support |
|-----|-----------|---------|---------|---------|---------|-----------|---------|
| 1 | {A,B} | {A} | 4 | {A,B} | 4 | {A,B,C} | 2 |
| 2 | {B,C,D} | {B} | 5 | {A,C} | 2 | {A,B,D} | 3 |
| 3 | {A,B,C,D} | {C} | 3 | {A,D} | 3 | {A,C,D} | 2 |
| 4 | {A,B,D} | {D} | 4 | {B,C} | 3 | {B,C,D} | 3 |
| 5 | {A,B,C,D} | {A,C} | 2 | {B,D} | 4 | {A,B,C,D} | 2 |
| | | {A,D} | 3 | {C,D} | 3 | | |
| | | {B,C} | 3 | | | | |
| | | {B,D} | 4 | | | | |
| | | {C,D} | 3 | | | | |

© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #1

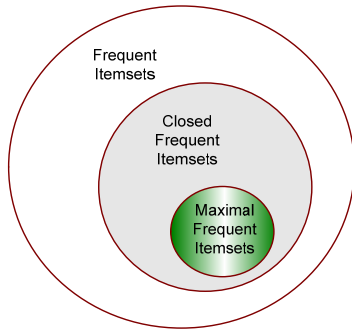
Maximal vs Closed Frequent Itemsets

minsup=2



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #1

Maximal vs Closed Itemsets



© Tan, Steinbach, Kumar Introduction to Data Mining 4/18/2004 #8

Example (generate, prune, count)

DATABASE

| TID | ITEMS |
|-----|---------|
| 100 | 1 3 4 |
| 200 | 2 3 5 |
| 300 | 1 2 3 5 |
| 400 | 2 5 |

C₁

| TID | ITEMS |
|-----|----------------------|
| 100 | {{1}, {3}, {4}} |
| 200 | {{2}, {3}, {5}} |
| 300 | {{1}, {2}, {3}, {5}} |
| 400 | {{2}, {5}} |

L₁

| ITEMSET | SUPPORT |
|---------|---------|
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {5} | 3 |

Given:
User defined minimum support is 2.

© Sharma Chakraborty

99



AprioriTid

- It is similar to the Apriori Algorithm and uses Apriori-gen function to determine the candidate sets initially.
- But the basic difference is that for determining the support, the database **is not used** after the first pass.
- Rather a set C^k is used for this purpose
- Each member of C^k is of the form <TID, {X_k} > where X_k is potentially large/frequent k itemset present in the transaction with the identifier TID
- C¹ corresponds to database D.

© Sharma Chakraborty

98



AprioriTid

- If a transaction does not contain a candidate itemset, then C^k will not have any entry for this transaction
- Hence, for large values of k the number of entries in C^k may be much smaller than the number of transactions in the database (why?)
- Number of transactions in which this itemset exists decreases as the number of items in an itemset (k) increases!

© Sharma Chakraborty

100



AprioriTid Algorithm

- Also, for **large values of k**, each entry may be smaller than the corresponding transaction because very few candidates may be contained in the transaction
- However for **small values for k** each entry may be larger than the corresponding transaction because an entry in C^k includes all candidate k itemsets contained in the transaction

Algorithm

7. For all entries t in C^{k-1} do begin
 - //determine candidate itemsets in C_k
 - //contained in the transaction with
 - //identifies t.TID
 - $C_t = \{c \text{ in } C_k \mid (c-c[k]) \text{ in } t.\text{set-of-items} \text{ and } (c-c[k-1]) \text{ in } t.\text{set-of-items}\}$
8. for all candidates c in C_t do
9. c.count++
10. If $(C_t \neq \emptyset)$ the $C^k += \langle t.TID, ct \rangle$
11. end

Algorithm AprioriTid

1. $L_1 = \{\text{large 1 itemsets}\}$;
2. $C_1 = \text{database } D$; //as sets
3. For($k = 2$; $L_{k-1} \neq \text{empty}$; $k++$) Terminology:
Large instead of frequent
4. $C_k = \text{apriori-gen}(L_{k-1})$;
5. $C^k = \text{empty}$;
6. forall transactions t belonging to C^{k-1}
- 7-11 //Determine candidate item sets in C_k contained in the transaction with identifier t.Tid
12. $L_k = \{c \text{ in } C_k \mid c.\text{count} \geq \text{minsup}\}$
13. end
14. Answer = $\cup_k L_k$

Example (generate, prune, count)

| DATABASE | | C_1 | |
|----------|---------|-------|------------------------------|
| TID | ITEMS | TID | ITEMS |
| 100 | 1 3 4 | 100 | $\{\{1\}, \{3\}, \{4\}\}$ |
| 200 | 2 3 5 | 200 | $\{\{2\}, \{3\}, \{5\}\}$ |
| 300 | 1 2 3 5 | 300 | $\{1\}, \{2\}, \{3\}, \{5\}$ |
| 400 | 2 5 | 400 | $\{\{2\}, \{5\}\}$ |

| L_1 | |
|---------|---------|
| ITEMSET | SUPPORT |
| $\{1\}$ | 2 |
| $\{2\}$ | 3 |
| $\{3\}$ | 3 |
| $\{5\}$ | 3 |

Given:
User defined minimum support is 2.

| C ₂ | | C' ₂ | |
|----------------|---|-----------------|-------------------------|
| ITEMSET | | TID | ITEMS |
| {1, 2} | 1 | 100 | {{1,3}} |
| {1, 3} | 2 | 200 | {2,3}, {2,5}, {3,5}} |
| {1, 5} | 1 | 300 | {{1,2}, {1,3}, {1,5}} |
| {2, 3} | 2 | 400 | {2, 3}, {2, 5}, {3, 5}} |
| {2, 5} | 3 | | |
| {3, 5} | 2 | | |

| C ₁ | | L ₂ | |
|----------------|----------------------|----------------|---------|
| TID | ITEMS | ITEMSET | SUPPORT |
| 100 | {{1}, {3}, {4}} | {1, 3} | 2 |
| 200 | {{2}, {3}, {5}} | {2, 3} | 2 |
| 300 | {{1}, {2}, {3}, {5}} | {2, 5} | 3 |
| 400 | {{2}, {5}} | {3, 5} | 2 |

© Sharma Chakraborty 105

Example

- The first entry C'₁ is { {1} {3} {4} } corresponding to transaction 100
- The C_t in step 7 corresponding to this entry t is { {1 3} }, because {1 3} is a member of C₂ and both ({1 3} - {1}) and ({1 3} - {3}) are members of t-itemsets.
- Then apriorigen gives L₃

© Sharma Chakraborty 107

(contd)

| C ₃ | | C' ₃ | |
|----------------|---|-----------------|-------------|
| ITEMSETS | | TID | ITEMS |
| {2, 3, 5} | 2 | 200 | {{2, 3, 5}} |
| | | 300 | {{2, 3, 5}} |

| C ₂ | | L ₃ | |
|----------------|-------------------------|----------------|---------|
| TID | ITEMS | ITEMSET | SUPPORT |
| 100 | {{1,3}} | | |
| 200 | {2,3}, {2,5}, {3,5}} | | |
| 300 | {{1,2}, {1,3}, {1,5}} | | |
| | {2, 3}, {2, 5}, {3, 5}} | | |
| 400 | {{2,5}} | | |

© Sharma Chakraborty 106

AprioriTid

- For k > 1, C'_k is generated by the algorithm (step 10). The members of C'_k corresponding to transaction t is <t.TID, {c in C_k | c contained in t}>
- If a transaction does not contain any candidate itemset, then C'_k will not have any entry for this transaction.
- Number of entries in C'_k may be less than the # of transactions especially for large values of k

© Sharma Chakraborty 108

Data Structures

- Each candidate itemset is given a unique ID
- C_k is kept in an array indexed by ID
- A member of C^k is now of the form $\langle TID, \{ID\} \rangle$
- Each C^k is stored in a sequential structure
- The above is used for apriori-generation
- These are also used for step 7 for efficiently generating C^k

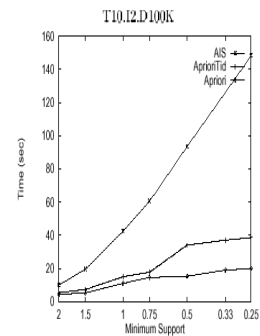
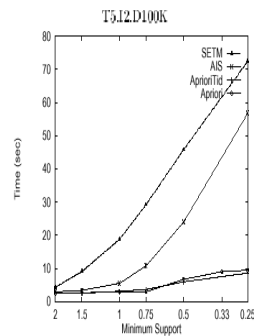
Performance

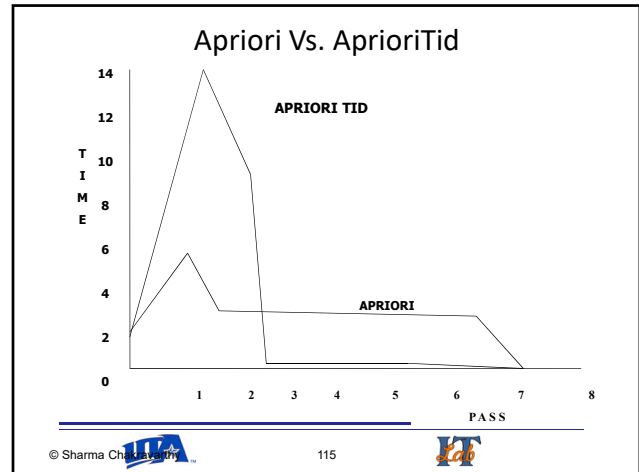
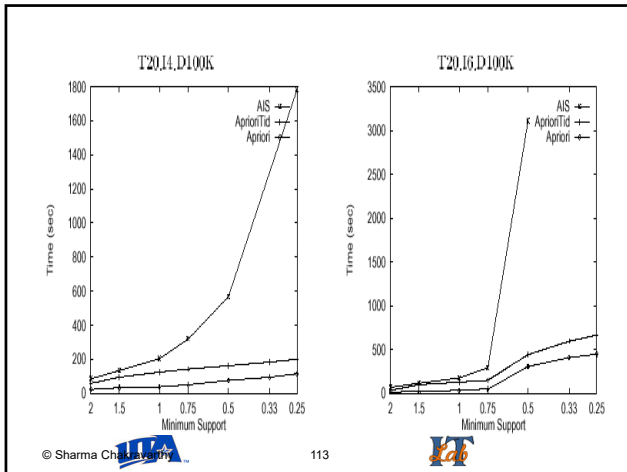
- Parameters

| | |
|-------|--|
| $ D $ | Number of transactions |
| $ T $ | Average size of the transactions |
| $ I $ | Average size of the maximal potentially large itemsets |
| $ L $ | Number of maximal potentially large itemsets |
| N | Number of items |

Buffer Management

- In the k^{th} pass, AprioriTid needs memory for L_{k-1} and C_k during candidate generation.
- During the counting phase, it needs memory for C_{k-1} , C_k , and a page each for C'_{k-1} and C'_k . entries in C'_{k-1} are needed sequentially, but C'_k can be written out as generated.





Disadvantages

- Apriori Algorithm
 - For determining the support of the candidate sets the algorithm always looks into every transaction in the database. Hence it takes a longer time (more passes on data)
- AprioriTid Algorithm
 - During initial passes the size of C^k is very large and is almost equivalent to the size of the database. Hence the time taken will be equal to that of Apriori. And also it might incur an additional cost if it cannot completely fit into the memory.

Algorithm Apriori Hybrid

- Idea: Not necessary to use the same algorithm in all passes over data.
- During the Initial Passes : Apriori Algorithm is used.
- During the Later Passes : AprioriTid Algorithm is used.
- Apriori Hybrid uses the Apriori in the initial passes and switches to AprioriTid when it expects that the set C^k at the end of the pass will fit into the memory.

Disadvantages of Apriori Hybrid

- An extra cost is incurred for switching from Apriori to AprioriTid algorithm.
- Suppose at the end of K th pass we decide to switch from Apriori to AprioriTid. Then in the $(k+1)$ pass, after having generated the candidate sets we also have to add the Tids to C^{k+1}
- If C^k remains large till the end then we do not get much benefits of using Apriori Hybrid Algorithm

Summary

- Data Mining is a tool box consisting of different tools designed for different purposes
- Choosing the appropriate tool is one of the difficult aspects of data mining
- Understanding the domain and matching the DM techniques for what one wants to do is another challenge
- Interpreting the results of the mining output is the third challenge
- **Buying a DM system is much easier than using it effectively and improving business!**

Conclusions

- The performance gap increased with the problem size and ranged from a factor of three for small problems to more than an order of magnitude for large problems.
- The algorithms presented in the paper have been implemented on several data repositories and were found to give consistent results.

Thank You !!!



For more information visit:

<http://itlab.uta.edu>

