ASSOCIATION RULE MINING

LECTURE 23

Dr. Vani V

What Is Pattern Discovery?

What are patterns?

- *Patterns: A set of items, subsequences, or substructures that occur frequently together (or strongly correlated) in a data set*
- *Patterns represent intrinsic and important properties of datasets*

Pattern discovery: Uncovering patterns from massive data sets

Motivation examples:

- *What products were often purchased together?*
- *What are the subsequent purchases after buying an iPad?*
- *What code segments likely contain copy-and-paste bugs?*
- *What word sequences likely form phrases in this corpus?*

Pattern Discovery: Why Is It Important?

Finding **inherent regularities** in a data set

Foundation for many essential data mining tasks

- *Association, correlation, and causality analysis*
- *Mining sequential, structural (e.g., sub-graph) patterns*
- *Pattern analysis in spatiotemporal, multimedia, time-series, and stream data*
- *Classification: Discriminative pattern-based analysis*
- *Cluster analysis: Pattern-based subspace clustering*

Broad applications

- ***Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log analysis, biological sequence analysis***

Association Rule Mining (ARM)

- Association rules mining (or market basket analysis) searches for interesting customer habits by looking at associations.
- The classical example is the one where a store in USA was reported to have discovered that people buying nappies tend also to buy beer. Not sure if this is actually true.
- Applications in marketing, store layout, customer segmentation, medicine, finance, and many more.

Question: Suppose you are able to find that two products x and y (say bread and milk or crackers and cheese) are frequently bought together. How can you use that information?

A Simple Example

Consider the following ten transactions of a **shop selling only nine items.**
We will return to it after explaining some terminology.

Transaction ID	Items
10	Bread, Cheese, Newspaper
20	Bread, Cheese, Juice
30	Bread, Milk
40	Cheese, Juice. Milk, Coffee
50	Sugar, Tea, Coffee, Biscuits, Newspaper
60	Sugar, Tea, Coffee, Biscuits, Milk, Juice, Newspaper
70	Bread, Cheese
80	Bread, Cheese, Juice, Coffee
90	Bread, Milk
100	Sugar, Tea, Coffee, Bread, Milk, Juice, Newspaper

Terminology…

- Let the number of different *items* sold be n
- Let the number of *transactions* be N
- Let the set of items be $\{i_1, i_2, \ldots, i_n\}$. The number of items may be large, perhaps several thousands.
- Let the set of transactions be $\{t_1, t_2, \ldots, t_N\}$. Each transaction t_i contains a subset of items from the *itemset* $\{i_1, i_2, \ldots, i_n\}$. These are the things a customer buys when they visit the supermarket. N is assumed to be large, perhaps in millions.
- Not considering the quantities of items bought.

Terminology…

- Want to find a group of items that tend to occur together frequently.
- The association rules are often written as X→Y meaning that whenever X appears Y also tends to appear. X and Y may be single items or sets of items, but the same item does not appear in both.

Terminology…

- Suppose X and Y appear together in only 1% of the transactions but whenever X appears there is 80% chance that Y also appears.
- The 1% presence of X and Y *together* is called the ***support (or prevalence)*** of the rule and 80% is called the ***confidence (or predictability)*** of the rule.
- These are measures of interestingness of the rule.

Terminology…

- **Confidence** denotes the **strength of the association between X and Y**. **Support indicates the frequency of the pattern**. A minimum support is necessary if an association is going to be of some business value.
- Let the chance of finding an item X in the N transactions is x% then we can say probability of X is P(X) = x/100 since probability values are always between 0.0 and 1.0.

Question: Why is minimum support necessary? Why is minimum confidence necessary?

Terminology…

- Suppose we know P(X U Y) then what is the probability of Y appearing if we know that X already exists. It is written as $P(Y \mid X)$.
- The **support for X→Y is the probability of both X and Y appearing** together, that is **P(X U Y).**
- The **confidence of X→Y is the conditional probability** of Y appearing given that X exists. It is written as **$P(Y \mid X)$** and read as P of Y given X.

Terminology

- Sometime the term *lift* is also used. Lift is defined as Support(X U Y)/P(X)P(Y). P(X)P(Y) is the probability of X and Y appearing together if both X and Y appear randomly.
- As an example, if support of X and Y is 1%, and X appears 4% in the transactions while Y appears in 2%, then lift = 0.01/0.04 x 0.02 = 1.25. What does it tell us about X and Y? What if lift was 1?

The task

Want to find all associations which have at least p% support with at least q% confidence such that

- *all rules satisfying any user constraints are found*
- *the rules are found efficiently from large databases*
- *the rules found are actionable*

Applications

Although we are only considering basket analysis, the technique has many other applications as noted earlier.

For example we may have many patients coming to a hospital with various diseases and from various backgrounds. We therefore have a number of attributes (items) and we may be interested in knowing which attributes appear together and may be which attributes are frequently associated to some particular disease.

Question: Can you think of some applications of ARM in an educational institution?

Association Rule Mining

- 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, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

- **Itemset**
  - *A collection of one or more items*
    - Example: {Milk, Bread, Diaper}
  - *k-itemset*
    - An itemset that contains k items
- **Support count (σ)**
  - *Frequency of occurrence of an itemset*
  - *E.g.* σ*({Milk, Bread,Diaper}) = 2*
- **Support**
  - *Fraction of transactions that contain an itemset*
  - *E.g. s({Milk, Bread, Diaper}) = 2/5*
- **Frequent Itemset**
  - *An itemset whose support is greater than or equal to a minsup threshold*

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

Definition: Association Rule

- Association Rule
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
  - Example:
    {Milk, Diaper} $\rightarrow$ {Beer}

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

- Rule Evaluation Metrics
  - Support (s)
    - Fraction of transactions that contain both X and Y
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X

Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|\text{T}|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all 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

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

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

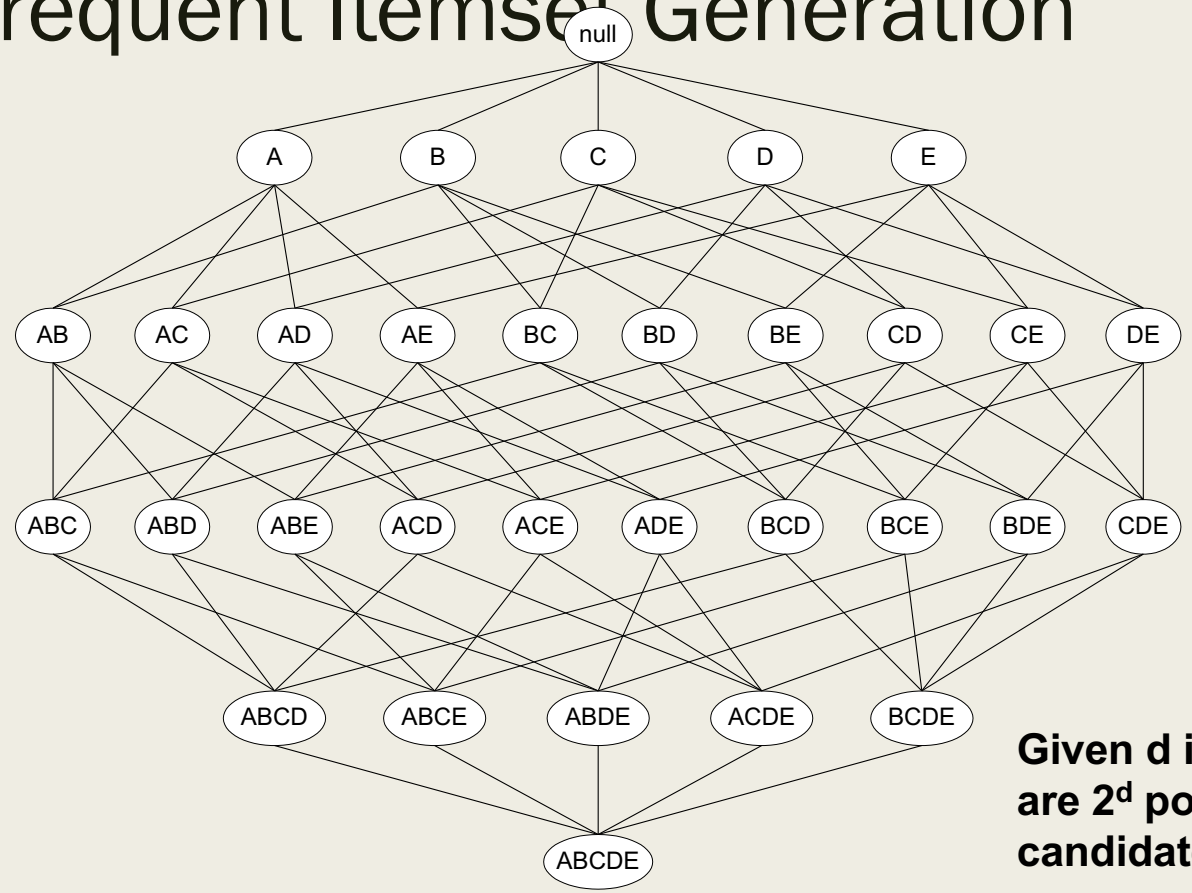
Observations:

- All the above rules are binary partitions of the same itemset:
  {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
  1. *Frequent Itemset Generation*
     - Generate all itemsets whose support $\geq$ 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

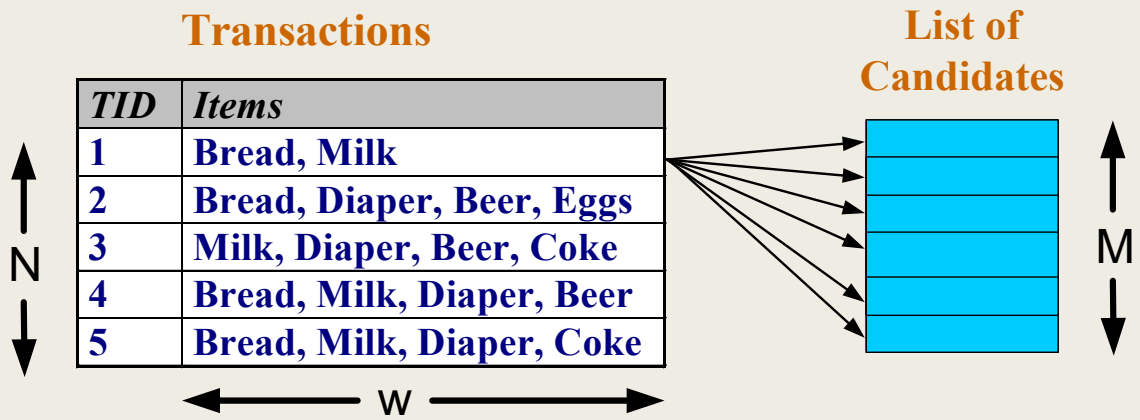
Frequent Itemset Generation



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

Frequent Itemset Generation

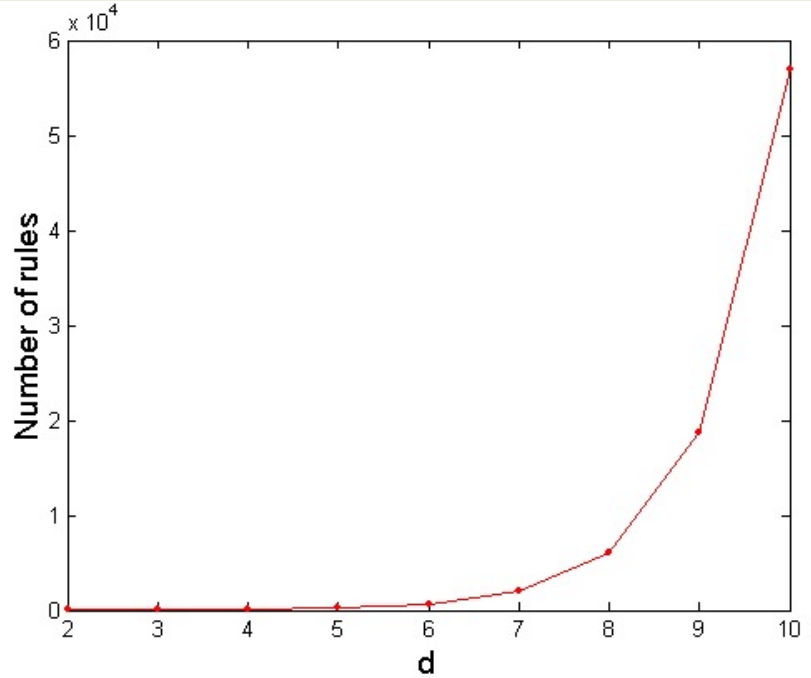
- Brute-force approach:
  - *Each itemset in the lattice is a candidate frequent itemset*
  - *Count the support of each candidate by scanning the database*



  - *Match each transaction against every candidate*
  - *Complexity ~ O(NMw) => Expensive since M = 2^d !!!*

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

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 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*

LECTURE 24

Dr. Vani V

Example

We have 9 items and 10 transactions. Find support and confidence for each pair. How many pairs are there?

Transaction ID	Items
10	Bread, Cheese, Newspaper
20	Bread, Cheese, Juice
30	Bread, Milk
40	Cheese, Juice. Milk, Coffee
50	Sugar, Tea, Coffee, Biscuits, Newspaper
60	Sugar, Tea, Coffee, Biscuits, Milk, Juice, Newspaper
70	Bread, Cheese
80	Bread, Cheese, Juice, Coffee
90	Bread, Milk
100	Sugar, Tea, Coffee, Bread, Milk, Juice, Newspaper

Example

There are 9 x 8 or 72 pairs, half of them duplicates. 36 is too many to analyse in a class, so we use an even **simpler example of n = 4 (Bread, Cheese, Juice, Milk) and N = 4. We want to find rules with minimum support of 50% and minimum confidence of 75%.**

Transaction ID	Items
100	Bread, Cheese
200	Bread, Cheese, Juice
300	Bread, Milk
400	Cheese, Juice. Milk

Example

N = 4 results in the following frequencies. All items have the 50% support we require. Only two pairs have the 50% support and no 3-itemset has the support.

We will look at the two pairs that have the minimum support to find if they have the required confidence.

Itemsets	Frequency
Bread	3
Cheese	3
Juice	2
Milk	2
(Bread, Cheese)	2
(Bread, Juice)	1
(Bread, Milk)	1
(Cheese, Juice)	2
(Cheese, Milk)	1
(Juice, Milk)	1
(Bread, Cheese, Juice)	1
(Bread, Cheese, Milk)	0
(Bread, Juice, Milk)	0
(Cheese, Juice, Milk)	1
(Bread, Cheese, Juice, Milk)	0

Terminology

Items or itemsets that have the minimum support are called *frequent*. In our example, all the four items and two pairs are frequent.

We will now determine if the two pairs {Bread, Cheese} and {Cheese, Juice} lead to association rules with 75% confidence.

Every pair {A, B} can lead to two rules A ⟶ B and B ⟶ A if both satisfy the minimum confidence. Confidence of A ⟶ B is given by the support for A and B together divided by the support of A.

Rules

We have four possible rules and their confidence is given as follows:

Bread $\rightarrow$ *Cheese with confidence of 2/3 = 67%*

Cheese $\rightarrow$ *Bread with confidence of 2/3 = 67%*

Cheese $\rightarrow$ *Juice with confidence of 2/3 = 67%*

Juice $\rightarrow$ *Cheese with confidence of 100%*

Therefore only the last rule Juice $\rightarrow$ Cheese has confidence above the minimum 75% and qualifies. Rules that have more than user-specified minimum confidence are called *confident*.

Problems with Brute Force

This simple algorithm works well with four items since there were only a total of 16 combinations that we needed to look at but if the number of items is say 100, the number of combinations is much larger, in billions.

The number of combinations becomes about a million with 20 items since the number of combinations is 2^n with n items (why?). The naïve algorithm can be improved to deal more effectively with larger data sets.

Problems with Brute Force

Question: Can you think of an improvement over the brute force method in which we looked at every itemset combination possible?

Naïve algorithms even with improvements don’t work efficiently enough to deal with large number of items and transactions. We now define a better algorithm.

Improved Brute Force

Rather than counting all possible item combinations we can look at each transaction and count only the combinations that occur as shown below (that is, we don't count itemsets with zero frequency).

Transaction ID	Items	Combinations
100	Bread, Cheese	{Bread, Cheese}
200	Bread, Cheese, Juice	{Bread, Cheese}, {Bread, Juice}, {Cheese, Juice}, {Bread, Cheese, Juice}
300	Bread, Milk	{Bread, Milk}
400	Cheese, Juice, Milk	{Cheese, Juice}, {Cheese, Milk}, {Juice, Milk}, {Cheese, Juice, Milk}

Improved Brute Force

Removing zero frequencies leads to the smaller table below. We then proceed as before, but the improvement is not large and we need better techniques.

Itemsets	Frequency
Bread	3
Cheese	3
Juice	2
Milk	2
(Bread, Cheese)	2
(Bread, Juice)	1
(Bread, Milk)	1
(Cheese, Juice)	2
(Juice, Milk)	1
(Bread, Cheese, Juice)	1
(Cheese, Juice, Milk)	1

The Apriori Algorithm

To find associations, this classical Apriori algorithm may be simply described by a two-step approach:

Step 1 — discover all frequent (single) items that have support above the minimum support required

Step 2 – use the set of frequent items to generate the association rules that have high enough confidence level

Apriori Algorithm Terminology

- A k-itemset is a set of k items.
- The set C_k is a set of candidate k-itemsets that are potentially frequent.
- The set L_k is a subset of C_k and is the set of k-itemsets that are frequent.

Step-by-step Apriori Algorithm

- Step 1 – Computing L_1

  Scan all transactions. Find all frequent items that have support above the required p%. Let these frequent items be labeled L_1.

- Step 2 – Apriori-gen Function

  Use the frequent items L_1 to build all possible item pairs like {Bread, Cheese} if Bread and Cheese are in L_1. The set of these item pairs is called C_2, the *candidate set*.

Question: Do all members of C_2 have the support? Why?

The Apriori Algorithm

- Step 3 – Pruning

  Scan all transactions and find all pairs in the candidate pair set C_2 that are frequent. Let these frequent pairs be L_2.

- Step 4 – General rule

  A generalization of Step 2. Build candidate set of k items C_k by combining frequent itemsets in the set L_{k-1}.

The Apriori Algorithm

- Step 5 – Pruning

  Step 5, generalization of Step 3. Scan all transactions and find all item sets in C_k that are frequent. Let these frequent itemsets be L_k.

- Step 6 – Continue

  Continue with Step 4 unless L_k is empty.

- Step 7 – Stop

  Stop when L_k is empty.

Reducing Number of Candidates

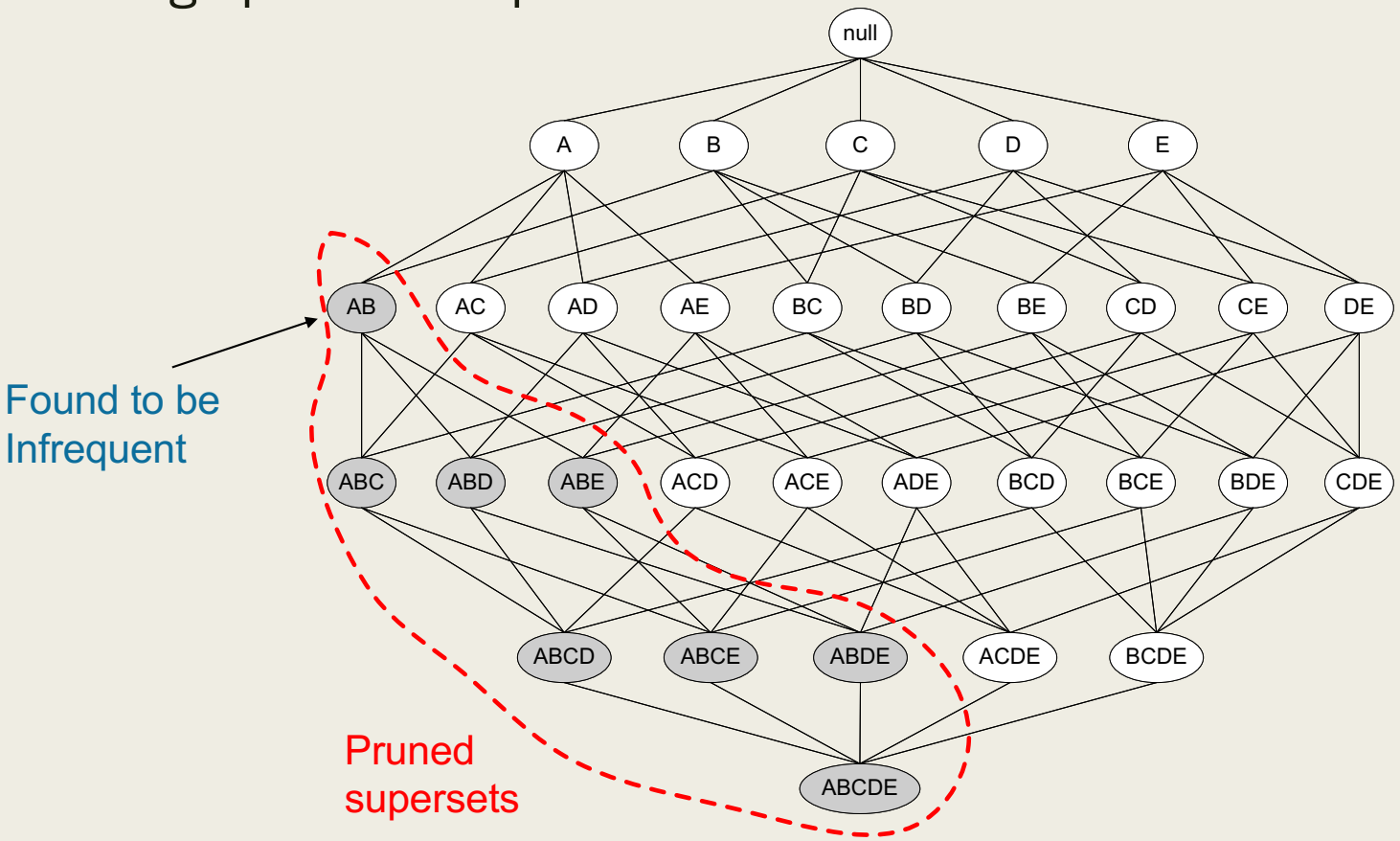
- Apriori principle:
  - *If an itemset is frequent, then all of its subsets must also be frequent*

- 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*

Illustrating Apriori Principle



Illustrating Apriori Principle

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

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)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 1 = 13$

Illustration of frequent itemset generation using the Apriori algorithm.

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

An Example

This example is like the last example, but we have added two more items and another transaction making five transactions and six items. We want to find association rules with 50% support and 75% confidence.

Transaction ID	Items
100	Bread, Cheese, Eggs, Juice
200	Bread, Cheese, Juice
300	Bread, Milk, Yogurt
400	Bread, Juice, Milk
500	Cheese, Juice, Milk

Example

First find L_1. 50% support requires that each frequent item appear in at least three transactions. Therefore L_1 is given by:

Item	Frequnecy
Bread	4
Cheese	3
Juice	4
Milk	3

Example

The candidate 2-itemsets or C_2 therefore has six pairs (why?). These pairs and their frequencies are:

Item Pairs	Frequency
(Bread, Cheese)	2
(Bread, Juice)	3
(Bread, Milk)	2
(Cheese, Juice)	3
(Cheese, Milk)	1
(Juice, Milk)	2

Question: Why did we not select (Egg, Milk)?

Deriving Rules

L_2 has only two frequent item pairs {Bread, Juice} and {Cheese, Juice}. After these two frequent pairs, there are no candidate 3-itemsets (why?) since we do not have two 2-itemsets that have the same first item (why?).

The two frequent pairs lead to the following possible rules:

Bread → *Juice*

Juice → *Bread*

Cheese → *Juice*

Juice → *Cheese*

Deriving Rules

The confidence of these rules is obtained by dividing the support for both items in the rule by the support of the item on the left hand side of the rule.

The confidence of the four rules therefore are

3/4 = 75% (Bread → Juice)

3/4 = 75% (Juice → Bread)

3/3 = 100% (Cheese → Juice)

3/4 = 75% (Juice → Cheese)

Since all of them have a minimum 75% confidence, they all qualify. We are done since there are no 3–itemsets.

LECTURE 25

Dr. Vani V

Recap

- Item set
- Support Count
- Support
- Frequent Itemset
- ARM – Brute Force approach
- Apriori Principle – Illustration
- Apriori Algorithm - Example

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup \}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k = \text{apriori-gen}(F_{k-1})$. {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = \text{subset}(C_k, t)$. {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup \}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

Frequent itemset Generation

- The pseudocode for the frequent itemset generation part of the Apriori algorithm is shown in Algorithm 6.1.
- Let C_k denote the set of candidate k-itemsets and F_k denote the set of frequent k-itemsets: The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1, will be known (steps 1 and 2).
- Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent $(k - 1)$-itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriorigen
- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t.

Candidate Generation and Pruning

The **apriori-gen function** shown in Step 5 of Algorithm 6.1 **generates candidate itemsets by performing the following two operations**

1. **Candidate Generation.** This operation generates new candidate k itemsets based on the frequent (k − 1)-itemsets found in the previous iteration.

2. **Candidate Pruning.** This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy.

Candidate Pruning

To illustrate the **candidate pruning operation**,

1. Consider a candidate k-itemset, $X = \{i_1, i_2, \ldots, i_k\}$.
2. The algorithm must determine whether all its proper subsets, $X - \{i_j\}$ ($\forall_j = 1, 2, \ldots, k$), are frequent.
3. If one of them is infrequent, then X is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting.
4. The complexity of this operation is **O(k)** for each candidate k-itemset.
5. **Note :** we do not have to examine all k subsets of a given candidate itemset. If m of the k subsets were used to generate a candidate, we only need to check the remaining k −m subsets during candidate pruning

Candidate Generation…

In principle, there are many ways to generate candidate itemsets. The following is a list of requirements for an effective candidate generation procedure:

1. It **should avoid generating too many unnecessary candidates.**
2. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the antimonotone property of support.
3. It must ensure that the candidate set is complete, i.e., **no frequent itemsets are left out by the candidate generation procedure.**

Candidate Generation…

1. **To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall_k : F_k \subseteq C_k$.**
2. It should not generate the same candidate itemset more than once. For example, the candidate itemset {a, b, c, d} can be generated in many ways—by merging {a, b, c} with {d}, {b, d} with {a, c}, {c} with {a, b, d}, etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Candidate Generation Procedure 1…

Brute-Force Method

The brute-force method considers every k-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates.

The number of candidate itemsets generated at level k is equal to dC_k, where d is the total number of items.

Although candidate generation is rather trivial, candidate pruning becomes extremely expensive because many itemsets must be examined.

Given that the number of computations needed for each candidate is O(k),the overall complexity of this method is

$$O\left(\sum_{k=1}^{d} k \times \binom{d}{k}\right) = O\left(d \cdot 2^{d-1}\right).$$

Brute-Force Method

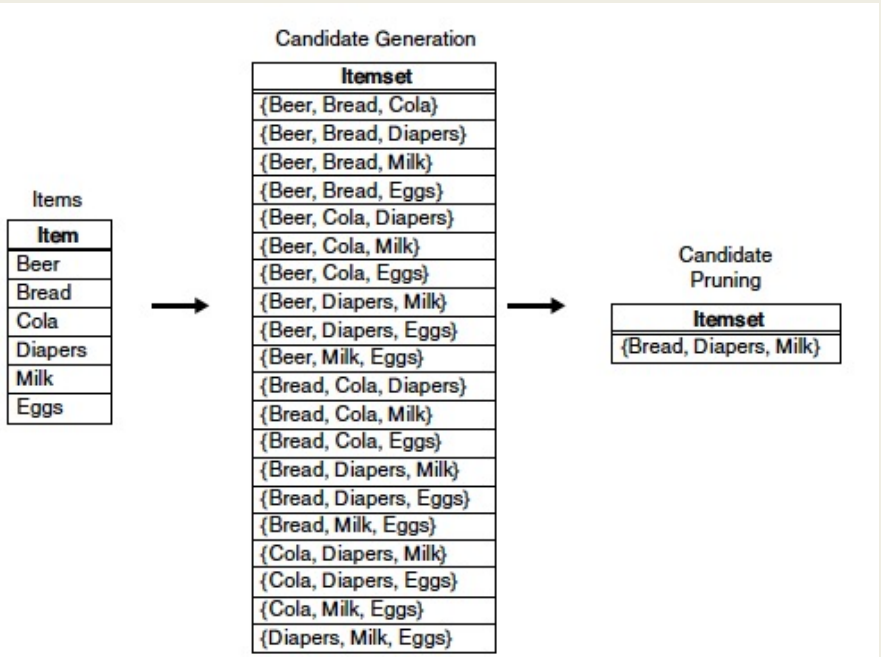


Figure 6.6. A brute-force method for generating candidate 3-itemsets.

Candidate Generation Procedure 2…

- An alternative method for candidate generation is to extend each frequent (k − 1)-itemset with other frequent items.

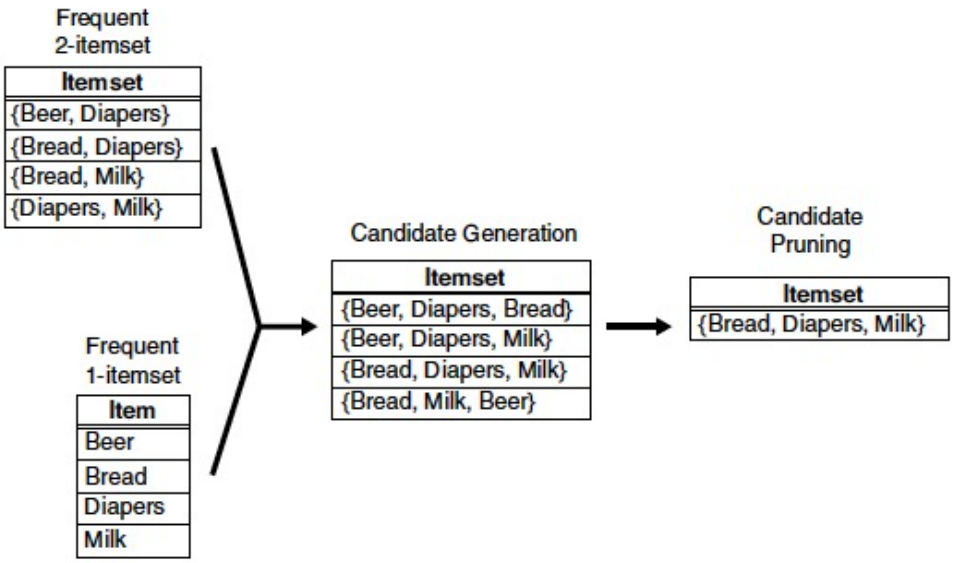


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

F_{k-1} x F_1 Method…

Figure illustrates how a frequent 2-itemset such as {Beer, Diapers} can be augmented with a frequent item such as Bread to produce a candidate 3-itemset {Beer, Diapers, Bread}. This method will produce $O(|F_{k-1}| x |F_1|)$ candidate k-itemsets, where $|F_j|$ is the number of frequent j-itemsets.

The overall complexity of this step is $O(\sum_k k|F_{k-1}||F_1|)$.

The procedure is complete because every frequent k-itemset is composed of a frequent (k − 1)-itemset and a frequent 1-itemset. Therefore, all frequent k-itemsets are part of the candidate k-itemsets generated by this procedure.

F_{k-1} x F_1 Method…

- This approach, however, does not prevent the same candidate itemset from being generated more than once.
- For instance, {Bread, Diapers, Milk} can be generated by merging {Bread, Diapers} with {Milk}, {Bread, Milk} with {Diapers}, or {Diapers, Milk} with {Bread}.
- One way to avoid generating duplicate candidates is by ensuring that the items in **each frequent itemset are kept sorted in their lexicographic order.**

F_{k-1} x F_1 Method…

- Each frequent (k−1)-itemset X is then extended with frequent items that are lexicographically larger than the items in X. For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers.

- However, we should not augment {Diapers, Milk} with {Bread} nor {Bread, Milk} with {Diapers} because they violate the lexicographic ordering condition.

- While **this procedure is a substantial improvement over the brute-force method, it can still produce a large number of unnecessary candidates.**

F_{k-1} x F_1 Method

For example, the candidate itemset obtained by merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent.

There are several heuristics available to reduce the number of unnecessary candidates. For example, note that, for every candidate k-itemset that survives the pruning step, every item in the candidate must be contained in at least $k-1$ of the frequent $(k-1)$-itemsets.

Otherwise, the candidate is guaranteed to be infrequent. For example, {Beer, Diapers, Milk} is a viable candidate 3-itemset only if every item in the candidate, including Beer, is contained in at least two frequent 2-itemsets. Since there is only one frequent 2-itemset containing Beer, all candidate itemsets involving Beer must be infrequent.

Candidate Generation Procedure 3…

- F_{k-1} x F_{k-1} Method

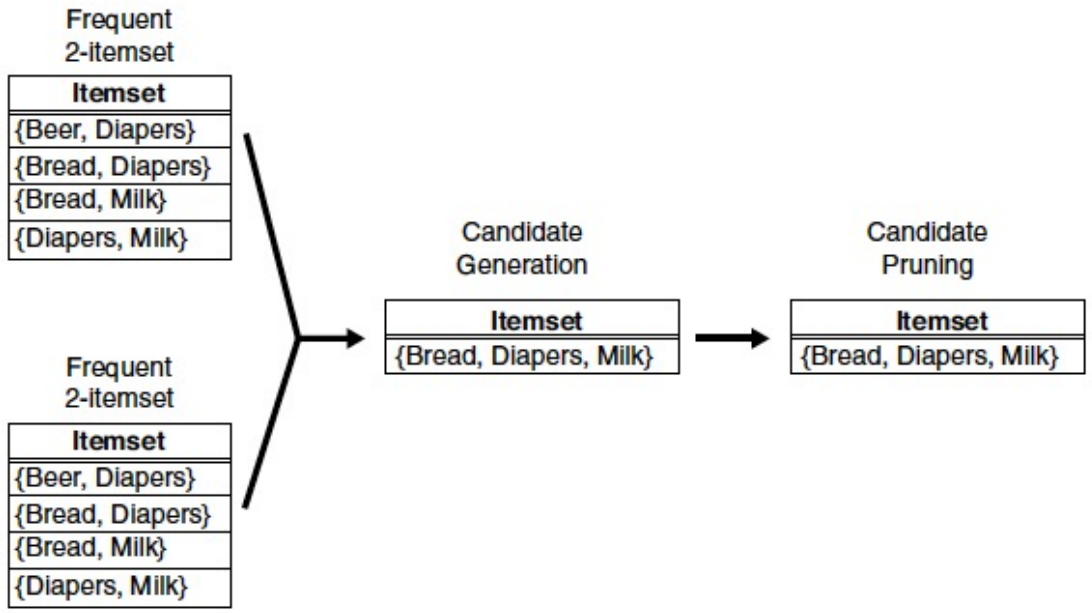


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

F_{k-1} x F_{k-1} Method...

- The candidate generation procedure in the apriori-gen function merges a pair of frequent (k–1)-itemsets only if their first k–2 items are identical.
- Let A = $\{a_1, a_2, \ldots, a_{k-1}\}$ and B = $\{b_1, b_2, \ldots, b_{k-1}\}$ be a pair of frequent (k – 1)-itemsets. A and B are merged if they satisfy the following conditions:

$$a_i = b_i \text{ (for } i = 1, 2, \ldots, k-2) \text{ and } a_{k-1} \neq b_{k-1}.$$

F_{k-1} x F_{k-1} Method…

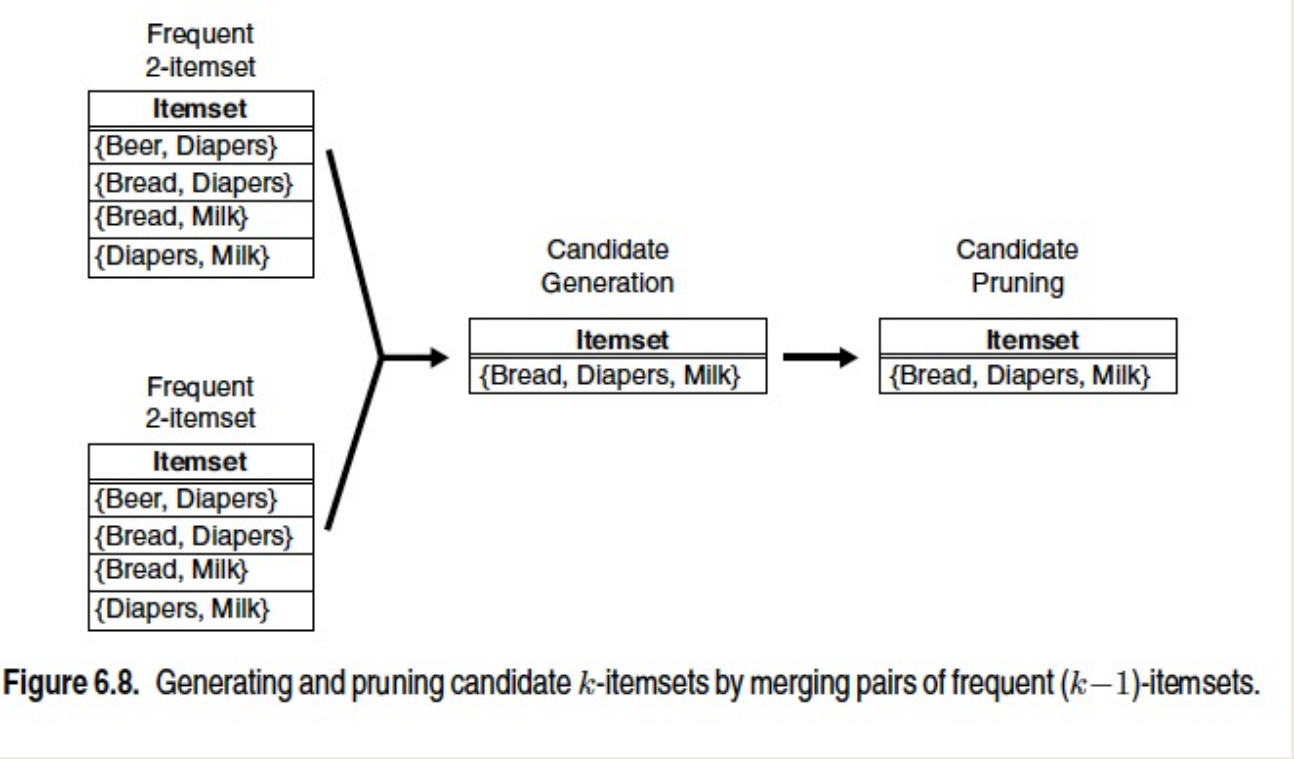


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

F_{k-1} x F_{k-1} Method

- In Figure 6.8, the frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}.
- The algorithm does not have to merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different.
- If {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, Milk} instead.
- This example **illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic** ordering to prevent duplicate candidates.
- Because each candidate is obtained by merging a pair of frequent (k−1)-itemsets, an additional candidate pruning step is needed to ensure that the remaining k − 2 subsets of the candidate are frequent.

References

- TEXTBOOKS :

1. Pang-Ning Tan, Vipin Kumar, Michael Steinbach: **Chapter 6, Introduction to Data Mining**, Pearson, 2012.
2. Jiawei Han and Micheline Kamber: **Chapter 6, Data Mining - Concepts and Techniques**, 3rd Edition, MorganKaufmann Publisher, 2014

LECTURE 26

Dr. Vani V

Python Tutorial Demos

- Demo on Data Exploration, Data Preprocessing & Classification
  - *Tutorial 3,4 & 6 (From python tutorials supplied with the textbook)*
- Decision tree Classification
  - *-DT.ipynb*

LECTURE 27

Dr. Vani V

Recap

- Frequent Itemset Generation
- Candidate Generation and Pruning
- Candidate Generation Procedures
  - *Brute-Force Method*
  - *F_{k-1} x F_1 Method*
  - *F_{k-1} x F_{k-1} Method*

Questions

1. How do we compute C_3 from F_2 or more generally C_i from F_{i-1}?
2. If there were several members of the set C_3, how do we derive F_3 by pruning C_3?
3. Why did we not consider (Bread, Cheese, Juice) in the last example?
4. Assume that the 3-itemset {A, B, M} has the minimum support. What do we do next? How do we derive association rules?

Answer for Q1, Q2

How do we compute C_3 from F_2 or more generally C_i from F_{i-1}?

To compute C_3 from F_2, or more generally C_i from F_{i-1}, we join members of F_{i-1} to other members in F_{i-1} by first sorting the itemsets in their lexicographic order and then joining those itemsets in which the first (i-2) items are common.

If there were several members of the set C_3, how do we derive F_3 by pruning C_3?

Observe that if an itemset in C_3 is (a, b, c) then F_2 must have had itemsets (a, b), (b, c) and (a, c) since all subsets of C_3 must be frequent. Why?

To repeat, itemsets in a candidate set C_i or a frequent set F_i will be frequent only if every subset is frequent.

Answer for Q3.

Why did we not consider (Bread, Cheese, Juice) in the last example?

For example, {A, B, M} will be frequent only if {A, B}, {A,M} and {B, M} are frequent, which in turn requires each of A, B and M also to be frequent.

We did not consider (Bread, Cheese, Juice) because the pairs (Bread, Cheese) and (Bread, Juice) should both have been in F_2 but (Bread, Cheese) was not .So the three-itemset could not be frequent.

Answer for Q4...

Assume that the 3-itemset {A, B, M} has the minimum support. What do we do next? How do we derive association rules?

If the set {A, B, M} has the minimum support, we can find association rules that satisfy the conditions of support and confidence by first generating all nonempty subsets of {A, B, M} and using each of it on the LHS and remaining symbols on the RHS.

Subsets of {A, B, M} are A, B, M, AB, AM, BM. Therefore, possible rules involving all three items are A → BM, B → AM, M → AB, AB → M, AM → B and BM → A.

Now we need to test their confidence.

Answer for Q4.

To test the confidence of the possible rules, we proceed as we have done before. We know that

$$\text{confidence}(A \rightarrow B) = P(B|A) = P(A \cup B)/ P(A)$$

This confidence is the likelihood of finding B if A already has been found in a transaction. It is the ratio of the support for A and B together and support for A by itself.

The confidence of all these rules can thus be computed.

Efficiency

Consider an example of a supermarket database which might have several thousand items including 1000 frequent items and several million transactions.

Which part of the apriori algorithm will be the most expensive to compute? Why?

Efficiency

The algorithm to construct the candidate set C_i to find the frequent set F_i is crucial to the performance of the Apriori algorithm. The larger the candidate set, higher the processing cost of discovering the frequent itemsets since the transactions must be scanned for each. Given that the numbers of early candidate itemsets are very large, the initial iterations dominate the cost.

In a supermarket database with about 1000 frequent items, there will be almost half a million candidate pairs C_2 that need to be searched for to find F_2.

Efficiency

Generally, the number of frequent pairs out of half a million pairs will be small and therefore (why?) the number of 3-itemsets should be small.

Therefore, **it is the generation of the frequent set F_2 that is the key** to improving the performance of the Apriori algorithm.

Comment

In class examples we usually require high support, for example, 25%, 30% or even 50%. These support values are very high if the number of items and number of transactions is large. For example, 25% support in a supermarket transaction database means searching for items that have been purchased by one in four customers! Not many items would probably qualify.

Practical applications therefore deal with much smaller support, sometimes even down to 1% or lower.

Transactions Storage

Consider only eight transactions with transaction IDs {10, 20, 30, 40, 50, 60, 70, 80}. This set of eight transactions with six items can be represented in at least three different ways as follows:

(1)

Transaction ID	Items
10	A, B, D
20	D, E, F
30	A, F
40	B, C, D
50	E, F
60	D, E, F
70	C, D, F
80	A, C, D, F

Horizontal Representation

(2)

TID	A	B	C	D	E	F
10	1	1	0	1	0	0
20	0	0	0	1	1	1
30	1	0	0	0	0	1
40	0	1	1	1	0	0
50	0	0	0	0	1	1
60	0	0	0	1	1	1
70	0	0	1	1	0	1
80	1	0	1	1	0	1

Vertical Representation

(3)

Items	10	20	30	40	50	60	70	80
A	1	0	1	0	0	0	0	1
B	1	0	0	1	0	0	0	0
C	0	0	0	1	0	0	1	1
D	1	1	0	1	0	1	1	1
E	0	1	0	0	1	1	0	0
F	0	1	1	0	1	1	1	1

LECTURE 28

Dr. Vani V

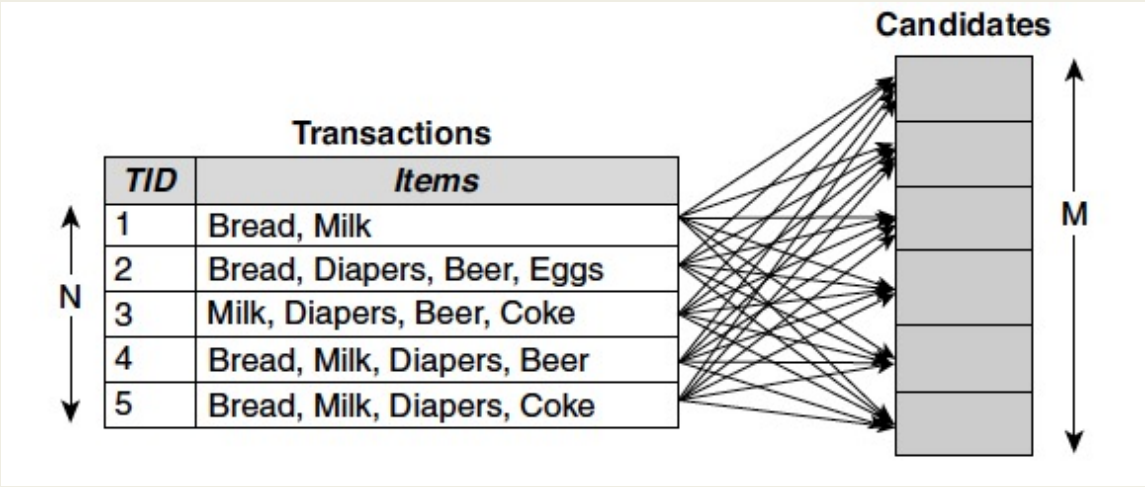
Support Counting…

- Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step of the apriori-gen function. Support counting is implemented in steps 6 through 11 of Algorithm 6.1

```
6:  for each transaction t ∈ T do
7:    C_t = subset(C_k, t).   {Identify all candidates that belong to t}
8:    for each candidate itemset c ∈ C_t do
9:      σ(c) = σ(c) + 1.   {Increment support count}
10:   end for
11: end for
```

Support Counting…

Approach 1: Compare each transaction against every candidate itemset and to update the support counts of candidates contained in the transaction. This approach is computationally expensive, especially when the numbers of transactions and candidate itemsets are large.

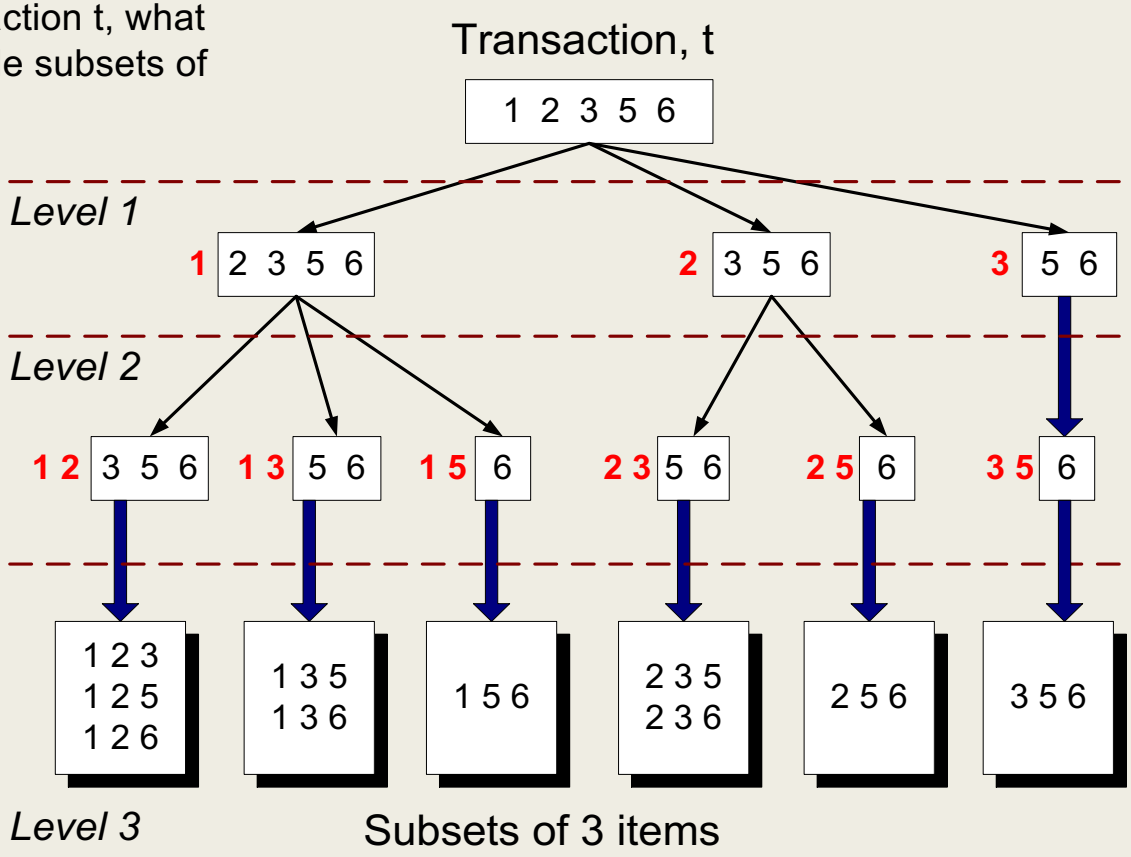


Support Counting…

- **Approach 2:** Enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemsets.
- To illustrate, consider a transaction *t* that contains five items, {1, 2, 3, 5, 6}. There are $5C_3 = 10$ itemsets of size 3 contained in this transaction.
- Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their support counts are incremented. Other subsets of *t* that do not correspond to any candidates can be ignored

Subset Operation

Given a transaction t, what are the possible subsets of size 3?



Support Counting…

- Figure shows a systematic way for enumerating the 3-itemsets contained in t.
- If each itemset keeps its items in increasing lexicographic order, an itemset can be enumerated by specifying the smallest item first, followed by the larger items.
- For instance, given t = {1, 2, 3, 5, 6}, all the 3-itemsets contained in t must begin with item 1, 2, or 3.
- It is not possible to construct a 3-itemset that begins with items 5 or 6 because there are only two items in t whose labels are greater than or equal to 5.

Support Counting…

- Once the enumeration is done, determine whether each enumerated 3-itemset corresponds to an existing candidate itemset. If it matches one of the candidates, then the support count of the corresponding candidate is incremented.
- The **matching operation** can be performed efficiently using a **hash tree structure.**

Support Counting : Matching Operation…

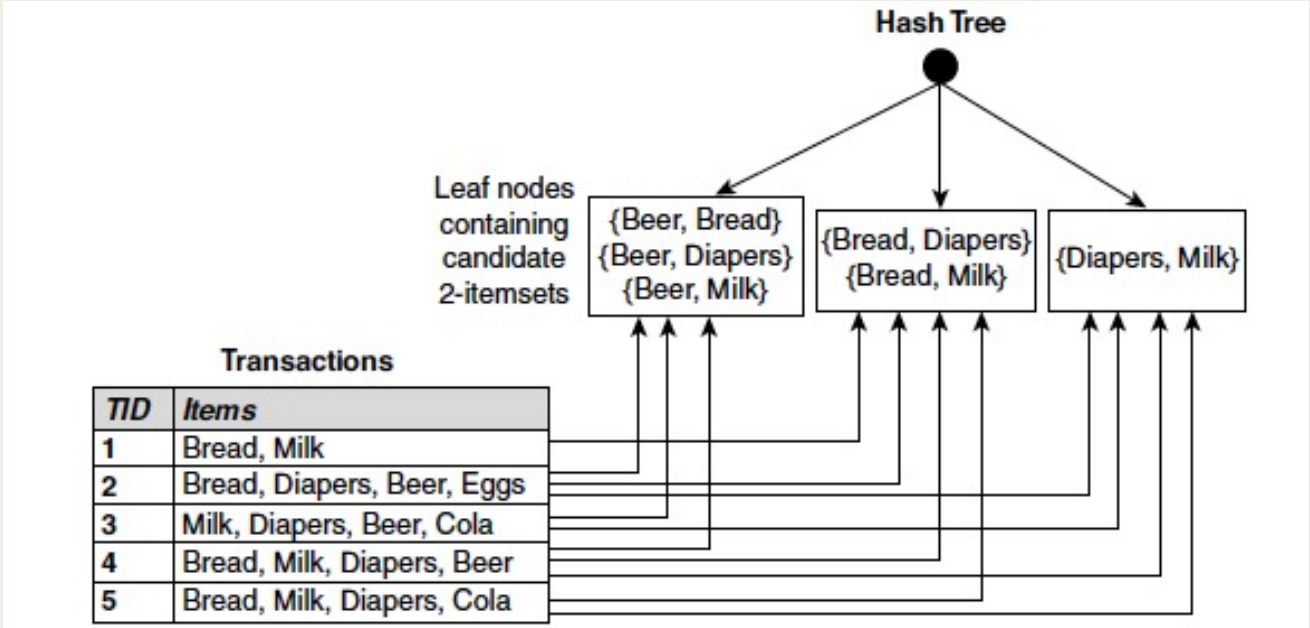
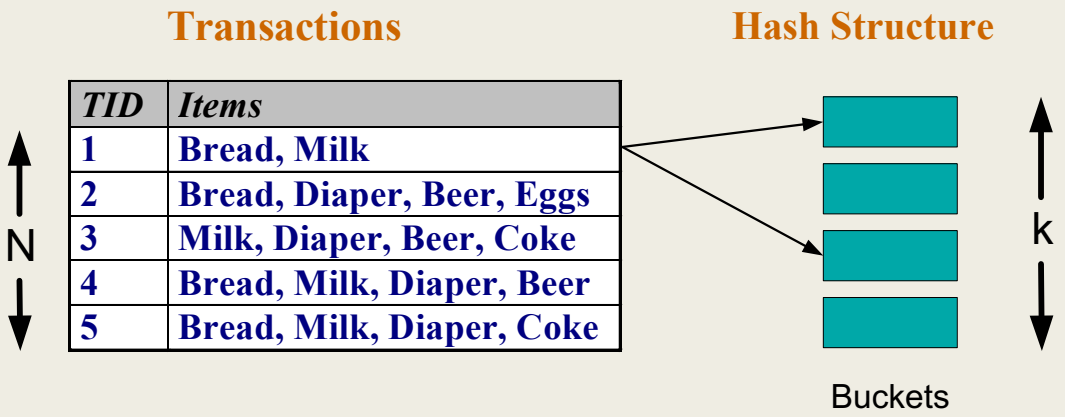


Figure 6.10. Counting the support of itemsets using hash structure.

Support Counting : 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



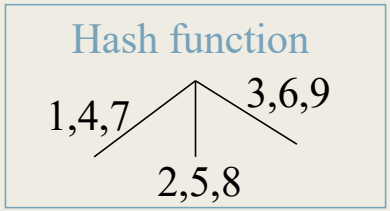
Support Counting : Generate Hash Tree …

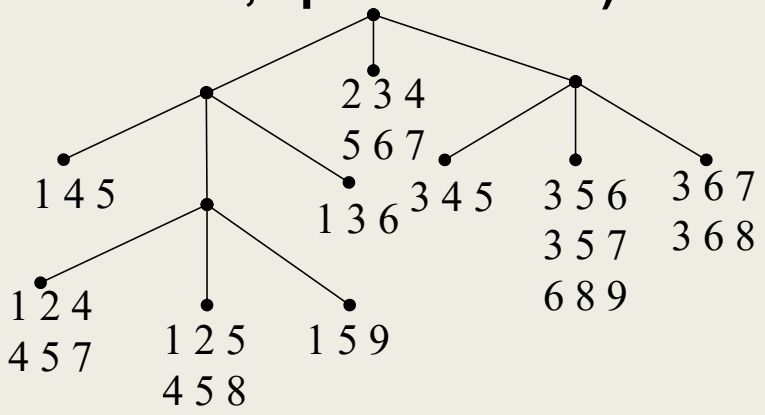
Suppose you have 15 candidate itemsets of length 3:

{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}

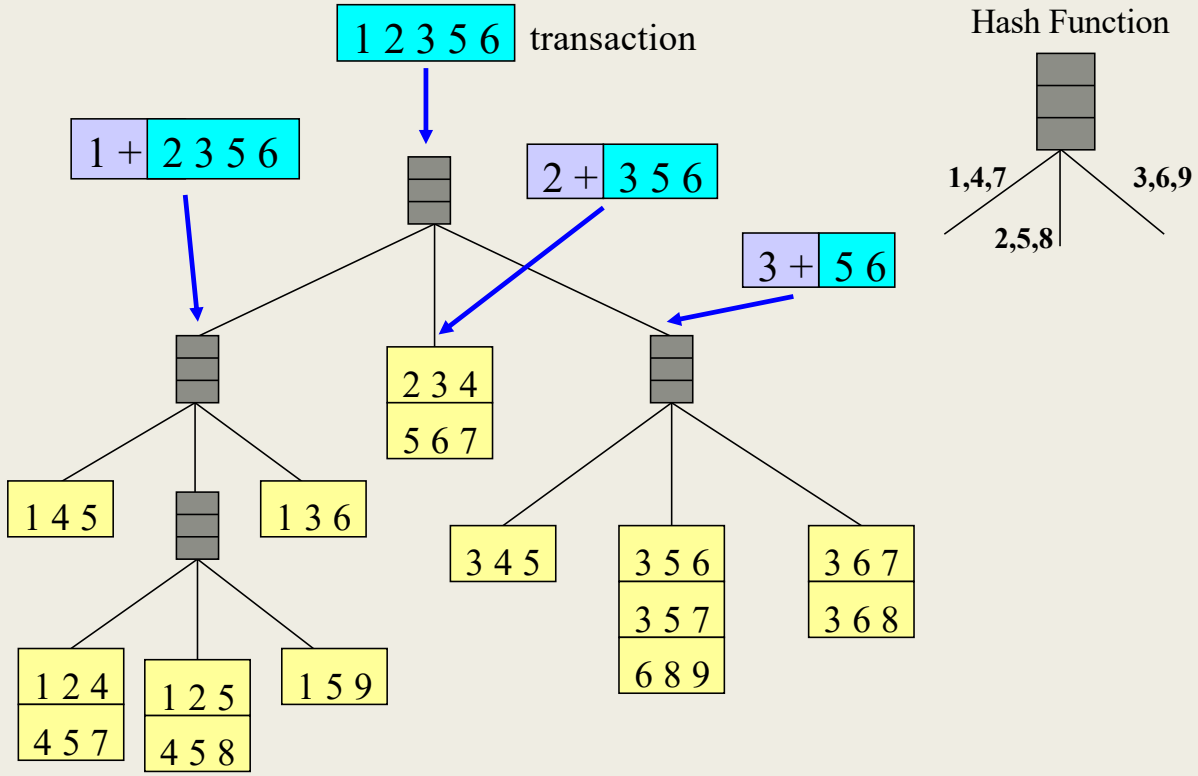
You need:

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





Subset Operation Using Hash Tree …



Support Counting : Generate Hash Tree ...

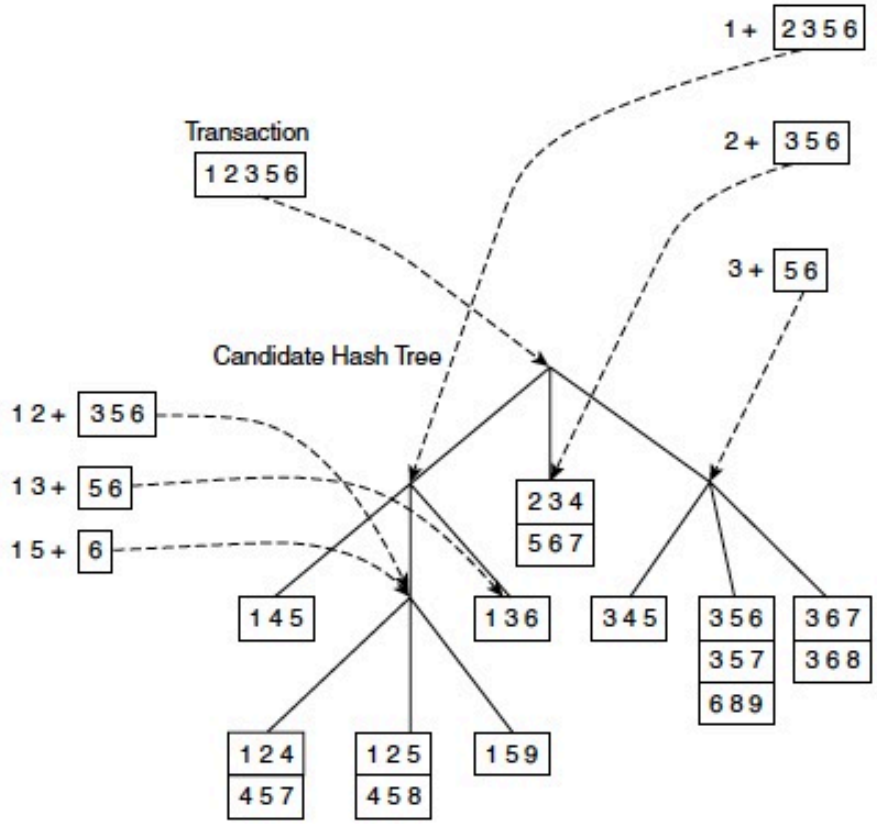
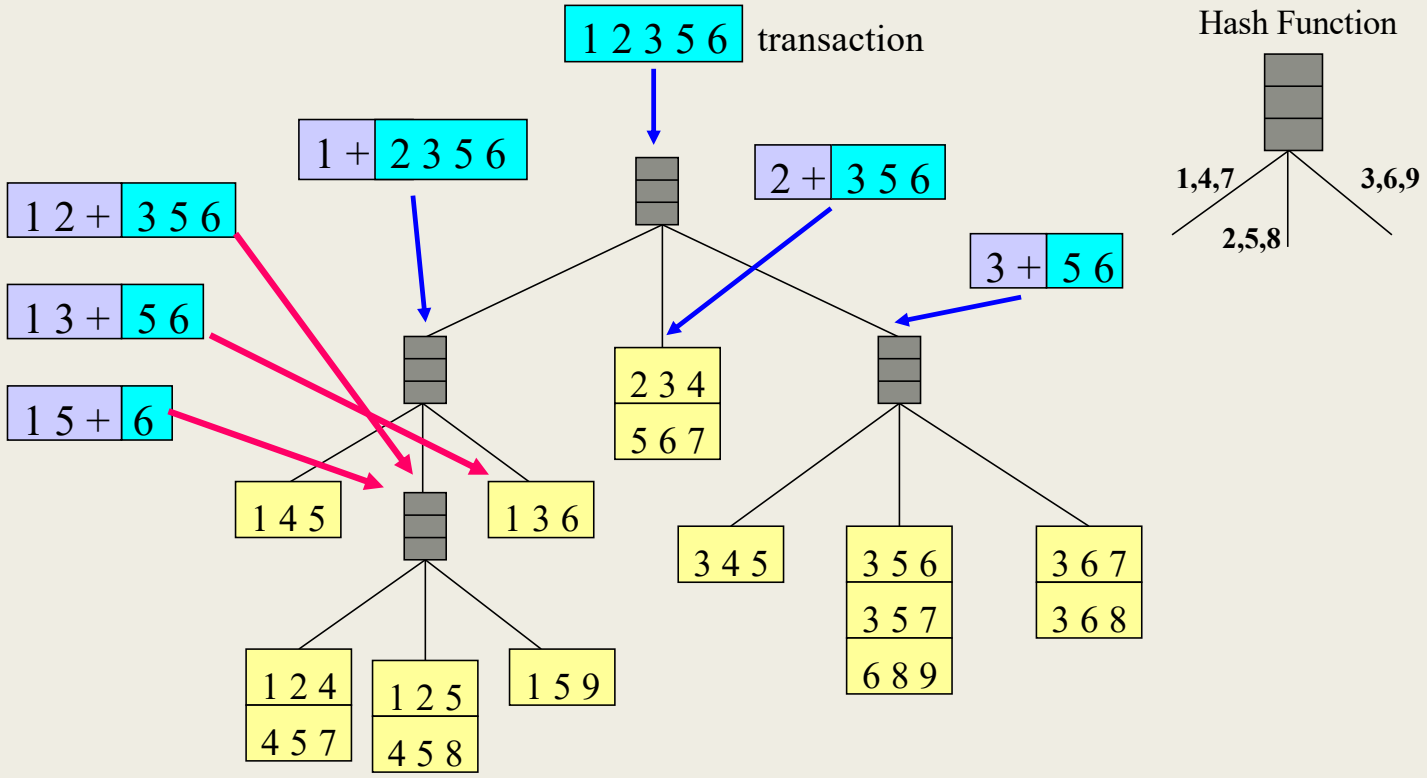
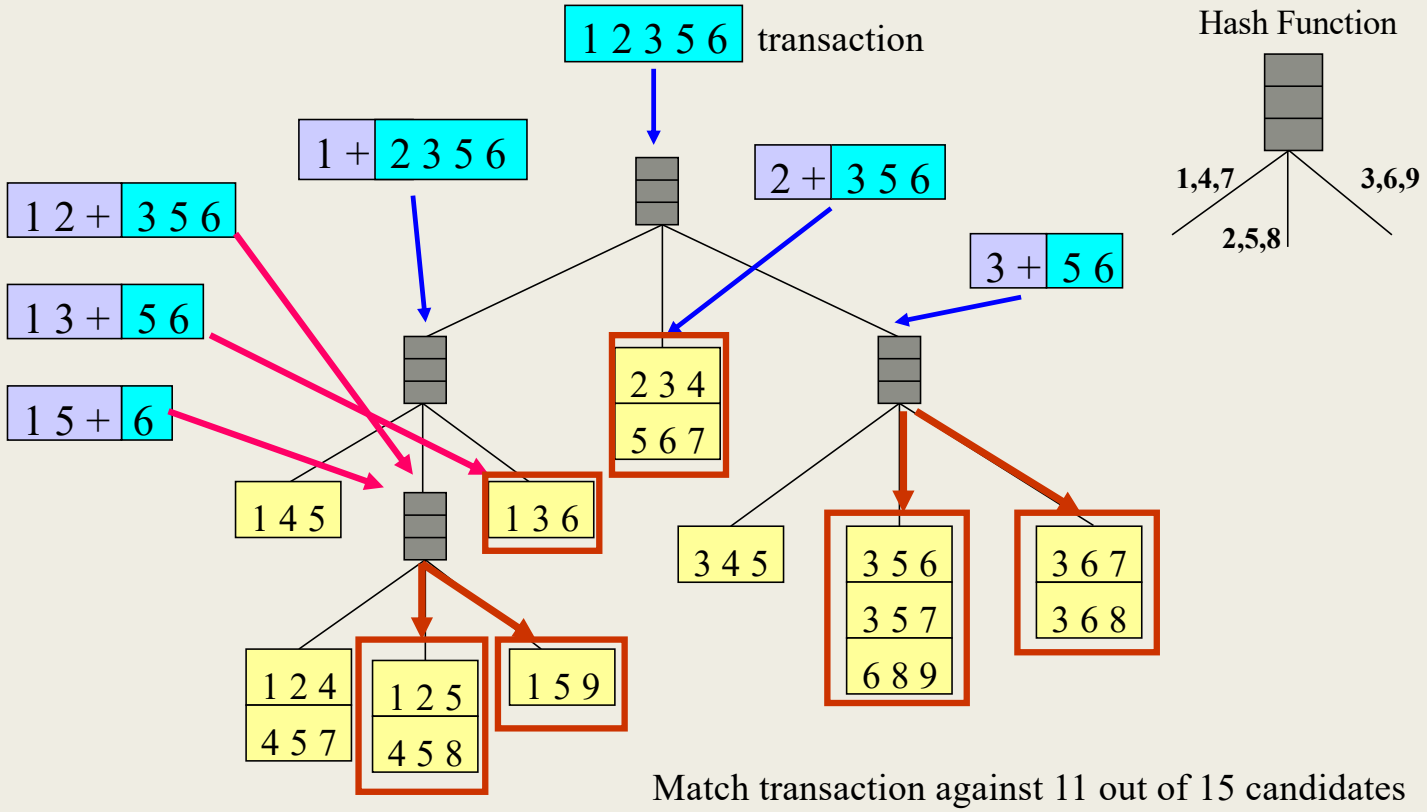


Figure 6.12. Subset operation on the leftmost subtree of the root of a candidate hash tree.

Subset Operation Using Hash Tree ...

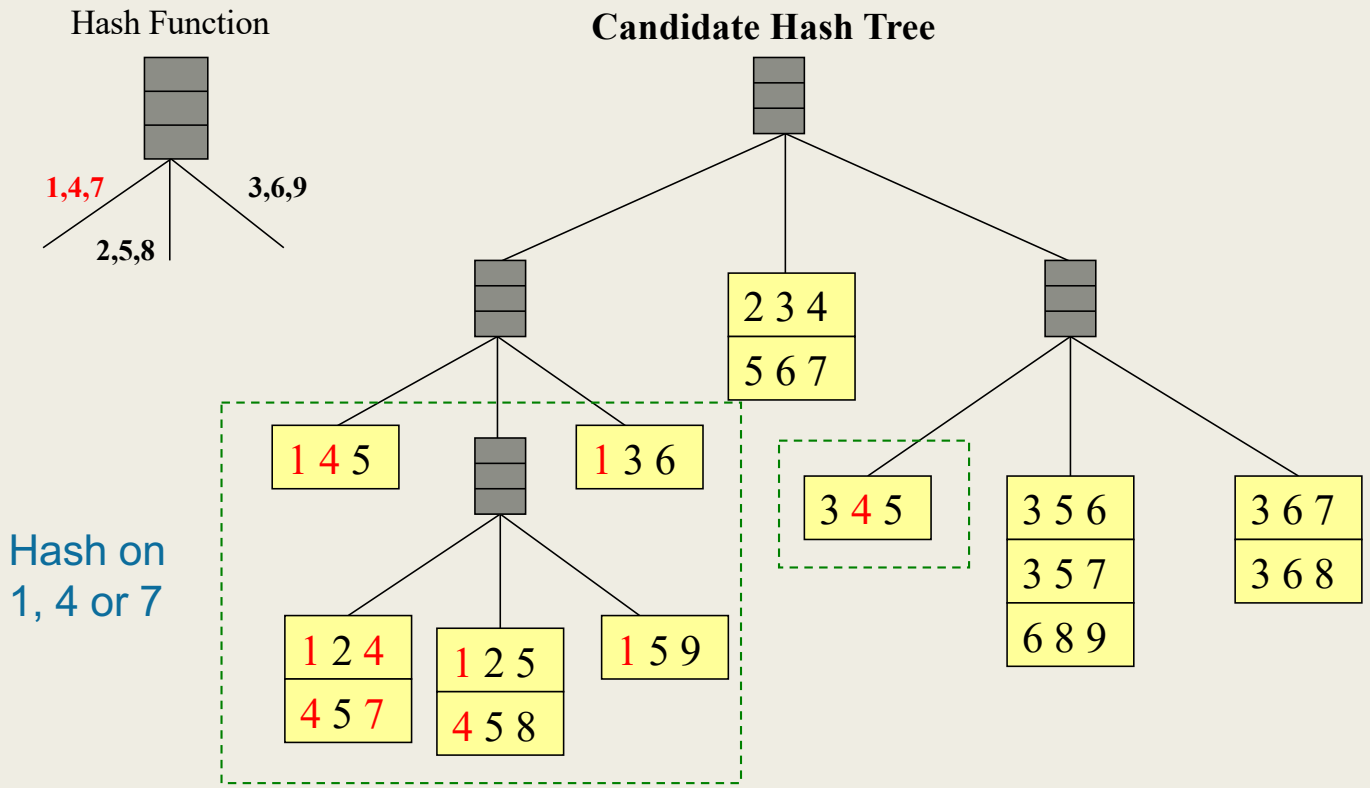


Subset Operation Using Hash Tree ...

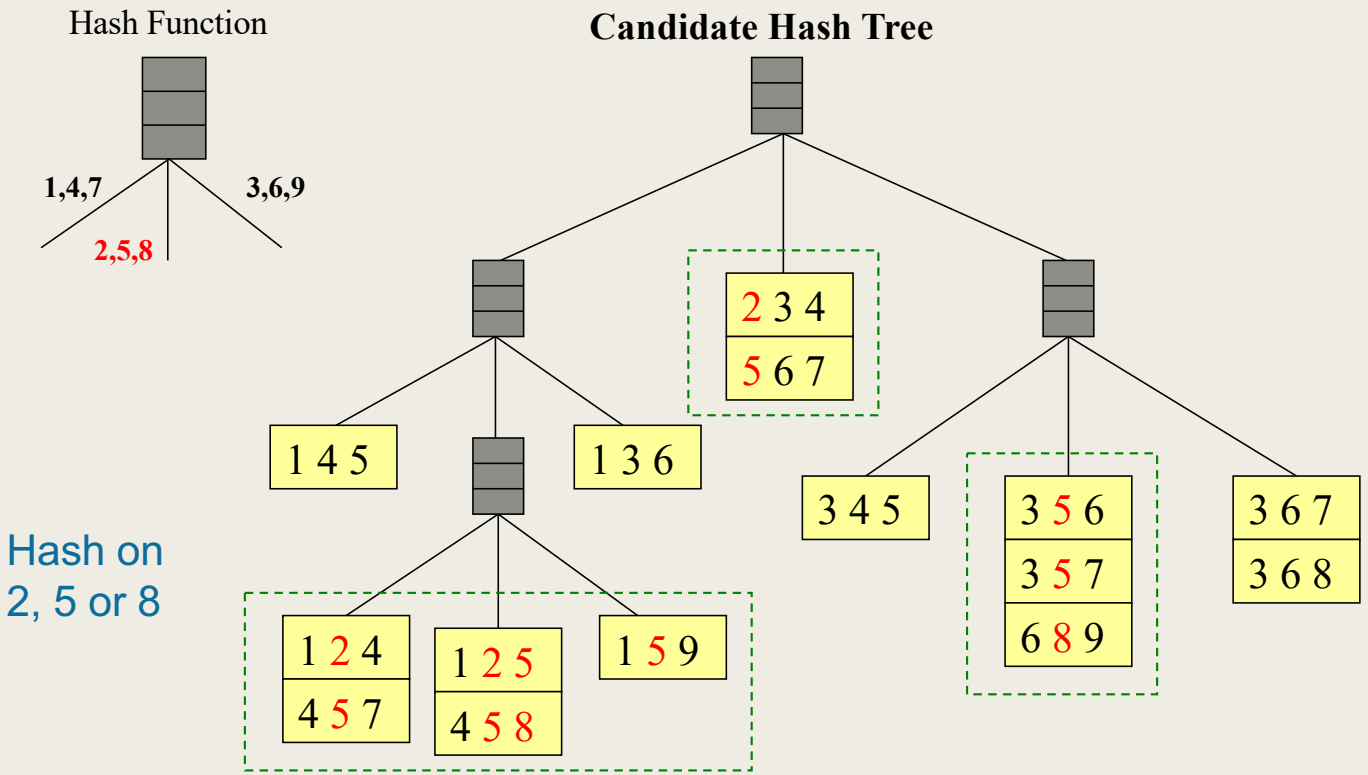


Match transaction against 11 out of 15 candidates

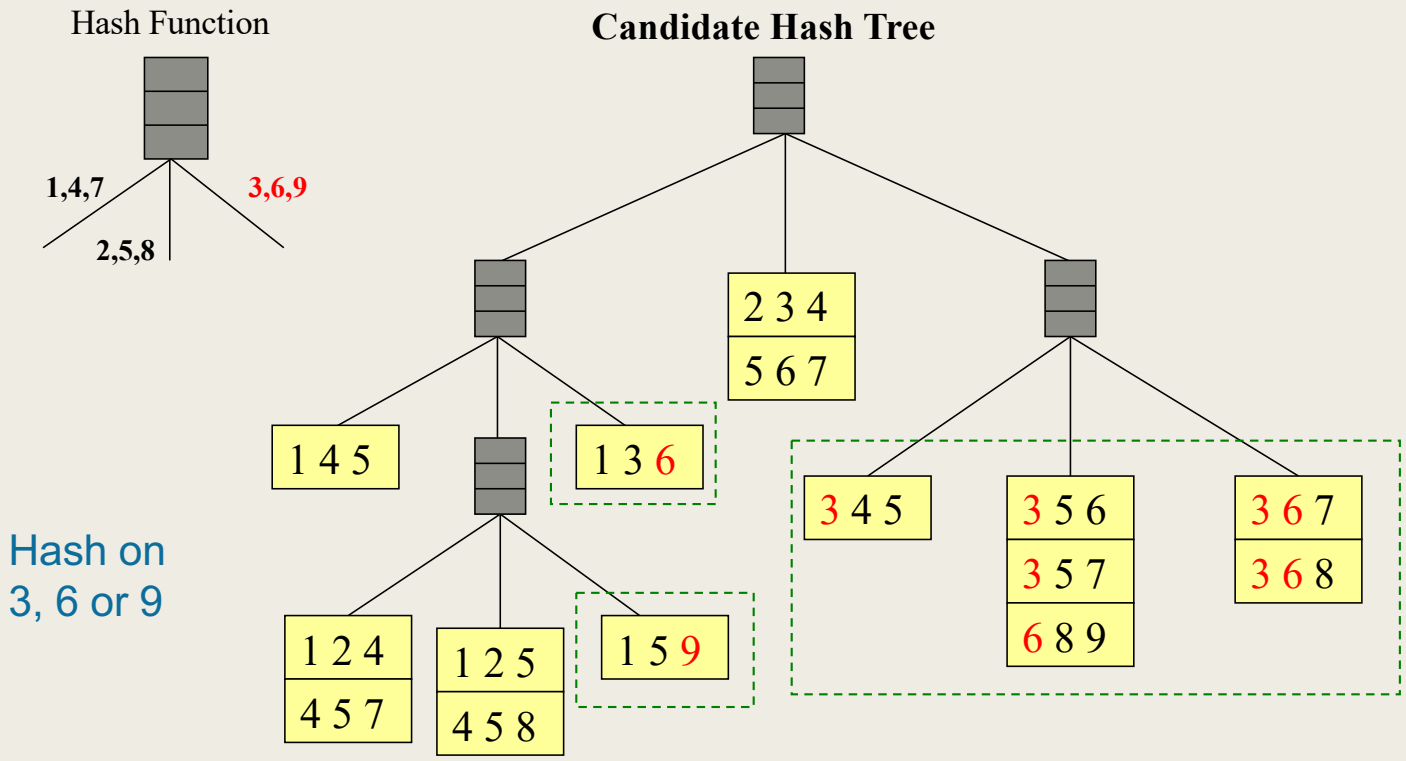
Association Rule Discovery: Hash tree ...



Association Rule Discovery: Hash tree ...



Association Rule Discovery: Hash tree …



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 increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)*

Analysis of the Time Complexity for the Apriori Algorithm

Generation of frequent 1-itemsets For each transaction, we need to update the support count for every item present in the transaction. Assuming that w is the average transaction width, this operation requires $O(Nw)$ time, where N is the total number of transactions.

Candidate generation To generate candidate k-itemsets, pairs of frequent $(k-1)$-itemsets are merged to determine whether they have at least $k-2$ items in common. Each merging operation requires at most $k-2$ equality comparisons. In the best-case scenario, every merging step produces a viable candidate k-itemset. In the worst-case scenario, the algorithm must merge every pair of frequent $(k-1)$-itemsets found in the previous iteration. Therefore, the overall cost of merging frequent itemsets is

$$\sum_{k=2}^{w}(k-2)|C_k| < \text{ Cost of merging } < \sum_{k=2}^{w}(k-2)|F_{k-1}|^2.$$

A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k, the cost for populating the hash tree with candidate itemsets is $O\left(\sum_{k=2}^{w} k|C_k|\right)$. During candidate pruning, we need to verify that the $k-2$ subsets of every candidate k-itemset are frequent. Since the cost for looking up a candidate in a hash tree is $O(k)$, the candidate pruning step requires $O\left(\sum_{k=2}^{w} k(k-2)|C_k|\right)$ time.

Analysis of the Time Complexity for the Apriori Algorithm

Support counting Each transaction of length $|t|$ produces $\binom{|t|}{k}$ itemsets of size k. This is also the effective number of hash tree traversals performed for each transaction. The cost for support counting is $O\left(N \sum_k \binom{w}{k} \alpha_k\right)$, where w is the maximum transaction width and α_k is the cost for updating the support count of a candidate k-itemset in the hash tree.

LECTURE 29

Dr. Vani V

Rule Generation …

- Each frequent k-itemset, Y, can produce up to $2^k - 2$ association rules, ignoring rules that have empty antecedents or consequents ($\emptyset \rightarrow Y$ or $Y \rightarrow \emptyset$).
- An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and Y −X, such that $X \rightarrow Y - X$ satisfies the confidence threshold.
- Note that all such rules must have already met the support threshold because they are generated from a frequent itemset.

Rule Generation…

- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
  - *If {A,B,C,D} is a frequent itemset, candidate rules:*

$ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$, $BCD \rightarrow A$,
$A \rightarrow BCD$, $B \rightarrow ACD$, $C \rightarrow ABD$, $D \rightarrow ABC$
$AB \rightarrow CD$, $AC \rightarrow BD$, $AD \rightarrow BC$, $BC \rightarrow AD$,
$BD \rightarrow AC$, $CD \rightarrow AB$,

- If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

Rule Generation…

- How to efficiently generate rules from frequent itemsets?
  - *In general, confidence does not have an anti-monotone property*

    c(ABC $\rightarrow$D) can be larger or smaller than c(AB $\rightarrow$D)

  - *But confidence of rules generated from the same itemset has an anti-monotone property*
  - *e.g., L = {A,B,C,D}:*

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

    - Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

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

Rule Generation …

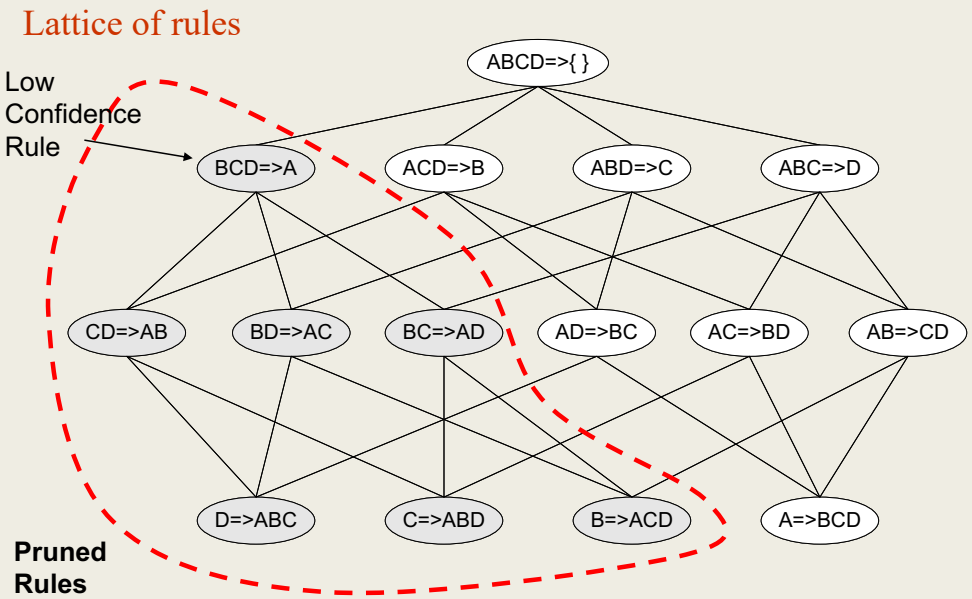
- Example: Let X = {1, 2, 3} be a frequent itemset. There are six candidate association rules that can be generated from X: {1, 2} → {3}, {1, 3} → {2}, {2, 3} → {1}, {1} → {2, 3}, {2} → {1, 3}, and {3} → {1, 2}.
- As each of their support is identical to the support for X, the rules must satisfy the support threshold.
- Computing the confidence of an association rule does not require additional scans of the transaction data set.
- **Consider the rule {1, 2} → {3}, which is generated from the frequent itemset X = {1, 2, 3}.**
- **The confidence for this rule is σ({1, 2, 3})/σ({1, 2}).**
- Because {1, 2, 3} is frequent, the anti-monotone property of support ensures that {1, 2} must be frequent, too.

Rule Generation in Apriori Algorithm ...

- The Apriori algorithm uses a level-wise approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent.
- Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules. For example, if {acd} → {b} and {abd} → {c} are high-confidence rules, then the candidate rule {ad} → {bc} is generated by merging the consequents of both rules.

Rule Generation in Apriori Algorithm …

- Figure shows a lattice structure for the association rules generated from the frequent itemset {a, b, c, d}. If any node in the lattice has low confidence, then entire subgraph spanned by the node can be pruned immediately
- Suppose the confidence for {bcd} → {a} is low. All the rules containing item a in its consequent, including {cd} → {ab}, {bd} → {ac}, {bc} → {ad}, and {d} → {abc} can be discarded.

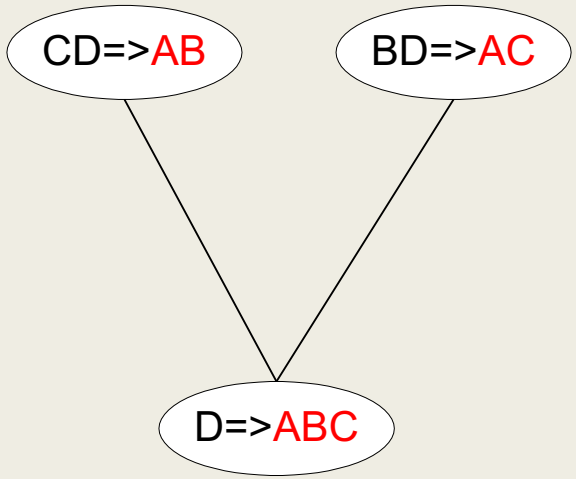


Rule Generation for Apriori Algorithm...

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent

- join(CD=>AB,BD=>AC) would produce the candidate rule D => ABC

- Prune rule D=>ABC if its subset AD=>BC does not have high confidence



Rule Generation

Algorithm 6.2 Rule generation of the *Apriori* algorithm.

1: **for** each frequent k-itemset f_k, $k \geq 2$ **do**
2: $H_1 = \{i \mid i \in f_k\}$ {1-item consequents of the rule.}
3: **call** ap-genrules$(f_k, H_1.)$
4: **end for**

Algorithm 6.3 Procedure ap-genrules(f_k, H_m).

1: $k = |f_k|$ {size of frequent itemset.}
2: $m = |H_m|$ {size of rule consequent.}
3: **if** $k > m + 1$ **then**
4: $H_{m+1} =$ apriori-gen(H_m).
5: **for** each $h_{m+1} \in H_{m+1}$ **do**
6: $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$.
7: **if** $conf \geq minconf$ **then**
8: **output** the rule $(f_k - h_{m+1}) \longrightarrow h_{m+1}$.
9: **else**
10: **delete** h_{m+1} from H_{m+1}.
11: **end if**
12: **end for**
13: **call** ap-genrules$(f_k, H_{m+1}.)$
14: **end if**

Recap

- Question & Answer on Frequent Itemset generation
- Efficiency
- Transaction Storage
- Support Count
- Approaches to compute Support Count
  - *Compare each transaction against every candidate itemset*
  - *Enumerate the itemsets in each transaction (Hash tree)*
- Computation Complexity
- Rule Generation in Apriori Algorithm

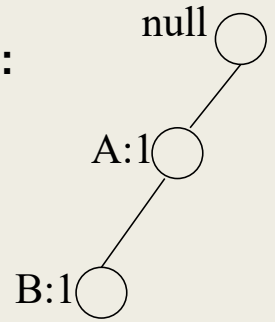
FP-growth Algorithm

- Use a compressed representation of the database using an FP-tree
- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets

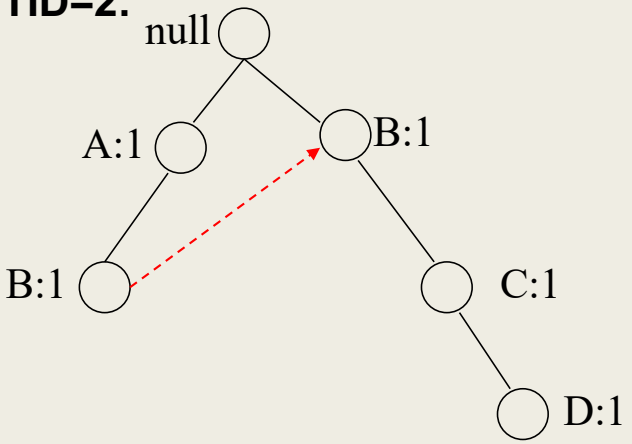
FP-tree construction

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

After reading TID=1:



After reading TID=2:



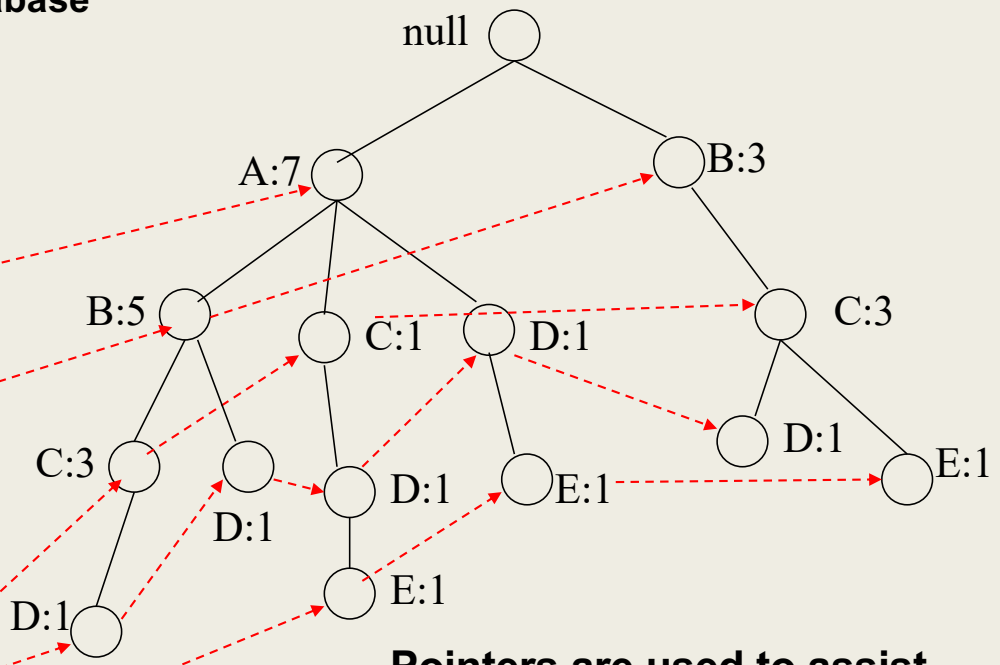
FP-Tree Construction

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

Transaction Database

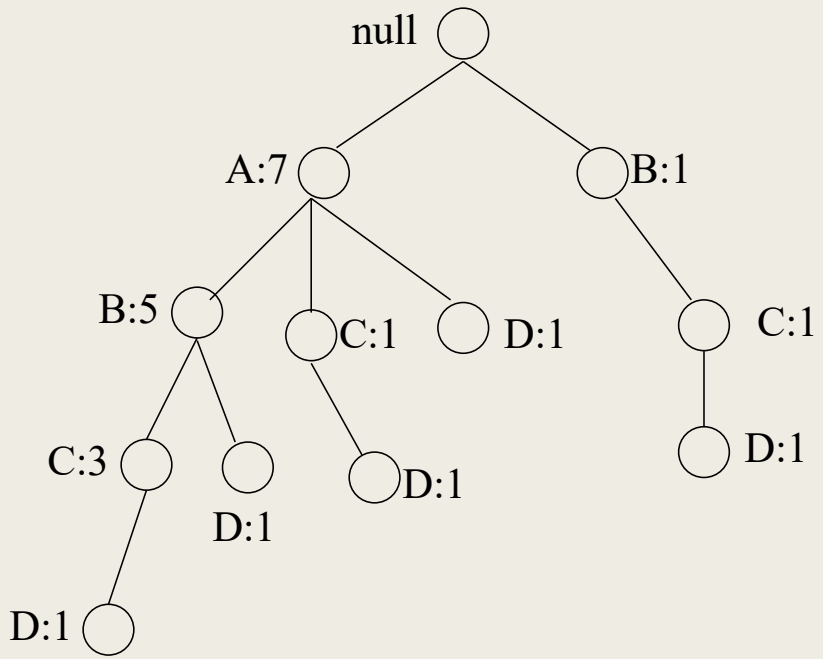
Header table

Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

FP-growth



Conditional Pattern base for D:

P = {(A:1,B:1,C:1),
(A:1,B:1),
(A:1,C:1),
(A:1),
(B:1,C:1)}

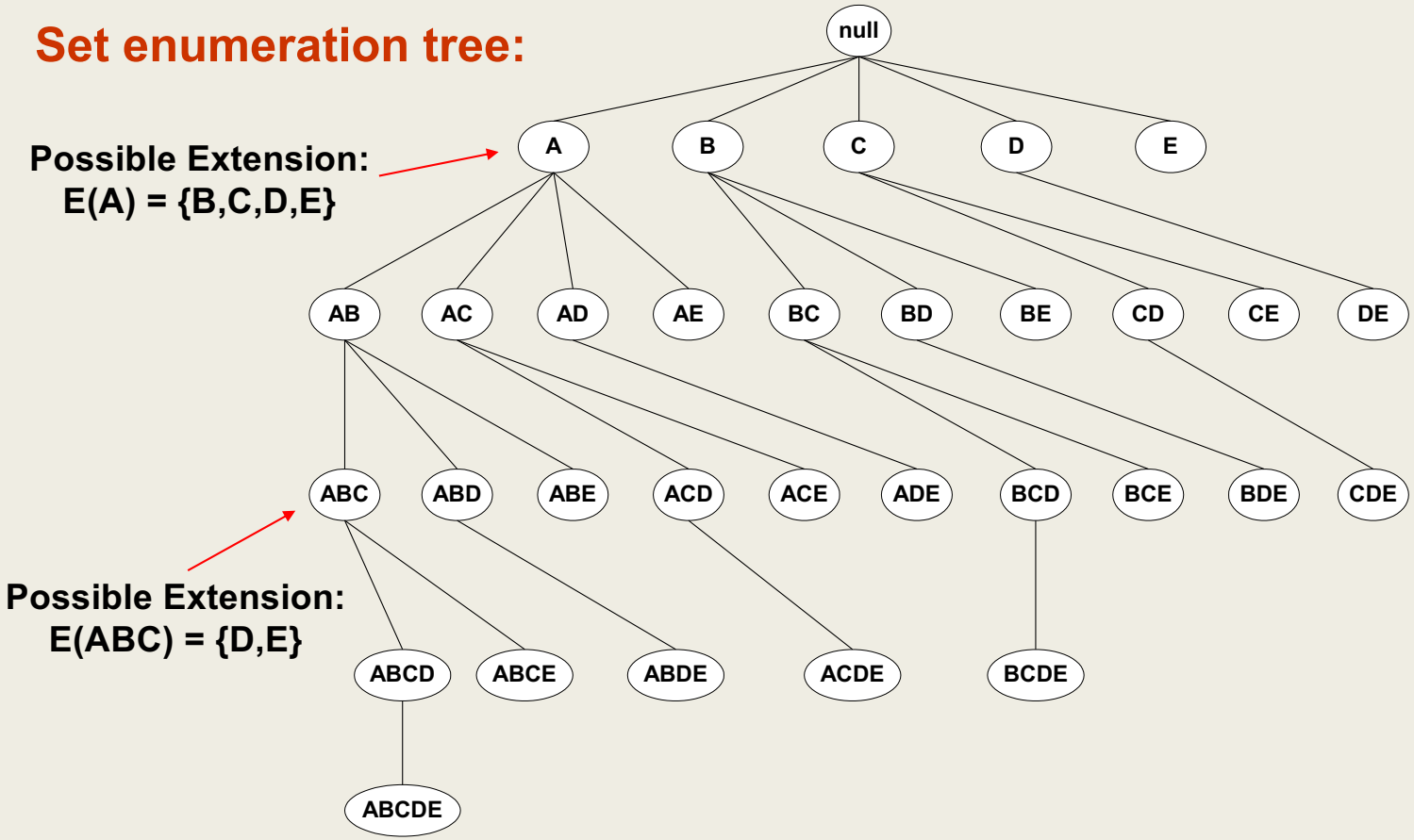
Recursively apply FP-growth on P

Frequent Itemsets found (with sup > 1):

AD, BD, CD, ACD, BCD

Tree Projection

Set enumeration tree:



Tree Projection

- Items are listed in lexicographic order
- Each node P stores the following information:
  - *Itemset for node P*
  - *List of possible lexicographic extensions of P: E(P)*
  - *Pointer to projected database of its ancestor node*
  - *Bitvector containing information about which transactions in the projected database contain the itemset*

Projected Database

Original Database:

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

Projected Database for node A:

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

For each transaction T, projected transaction at node A is $T \cap E(A)$

ECLAT

- For each item, store a list of transaction ids (tids)

Horizontal Data Layout

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

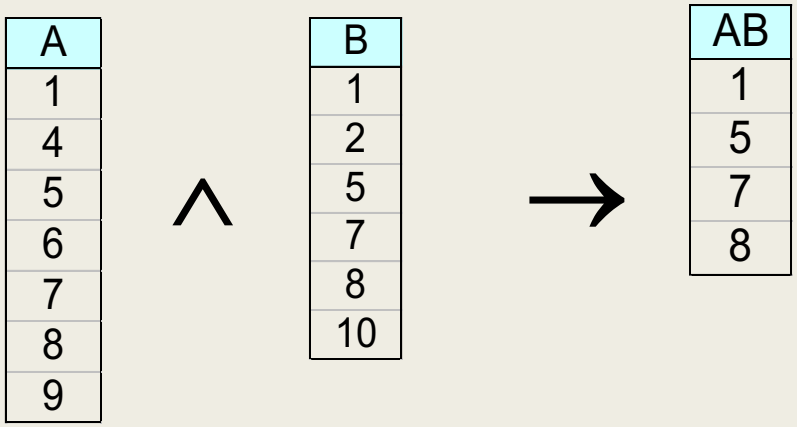
Vertical Data Layout

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-list

ECLAT

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

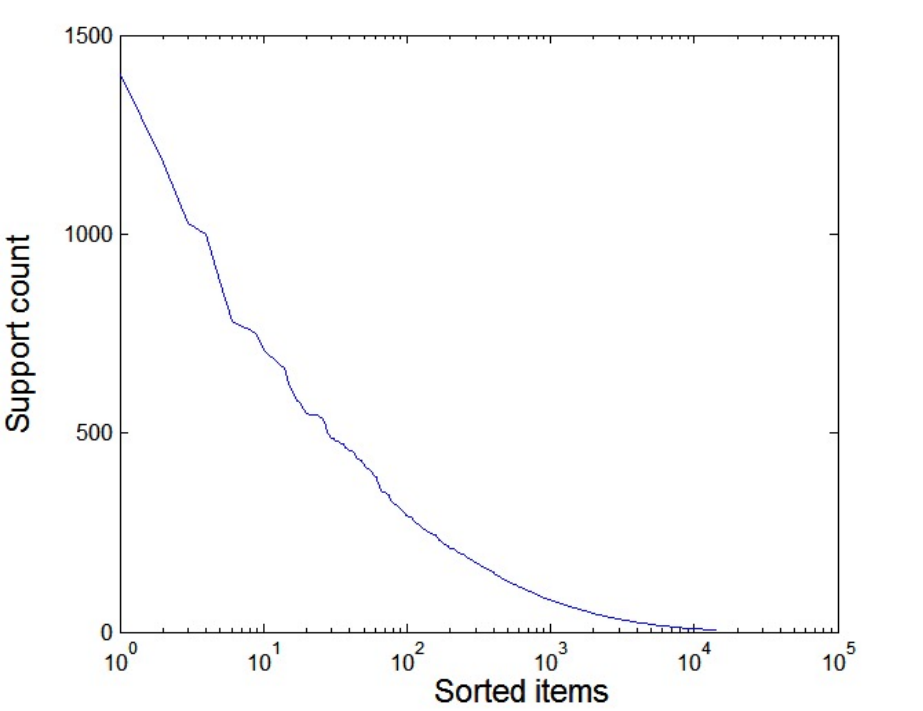


- 3 traversal approaches:
  - *top-down, bottom-up and hybrid*
- Advantage: very fast support counting
- Disadvantage: intermediate tid-lists may become too large for memory

Effect of Support Distribution

- Many real data sets have skewed support distribution

Support distribution of a retail data set



Effect of Support Distribution

- How to set the appropriate *minsup* threshold?
  - *If minsup is set too high, we could miss itemsets involving interesting rare items (e.g., expensive products)*
  - *If minsup is set too low, it is computationally expensive and the number of itemsets is very large*
- Using a single minimum support threshold may not be effective

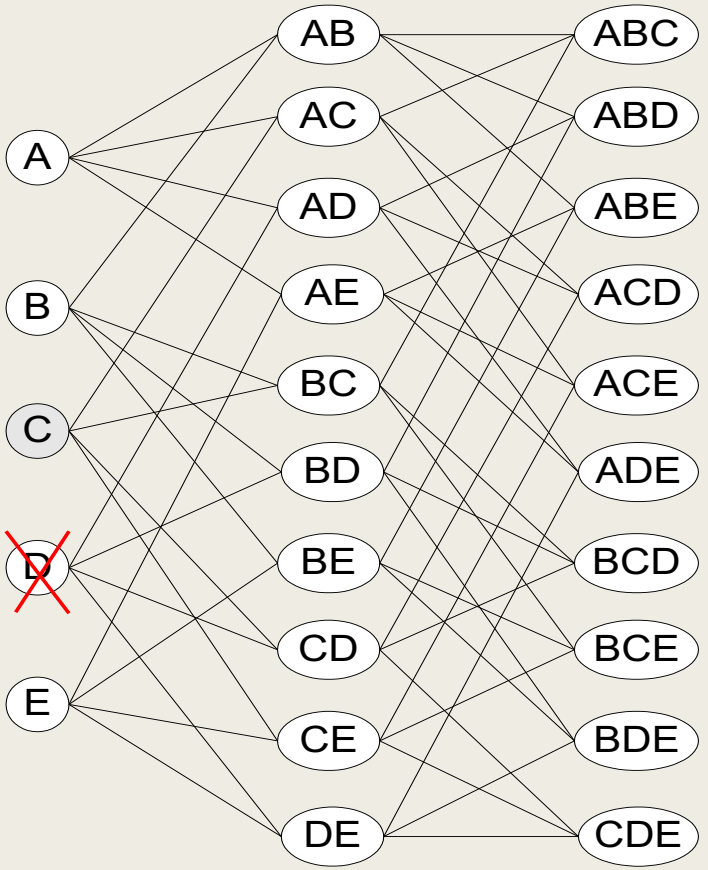
Multiple Minimum Support

- How to apply multiple minimum supports?
  - *MS(i): minimum support for item i*
  - *e.g.: MS(Milk)=5%, MS(Coke) = 3%, MS(Broccoli)=0.1%, MS(Salmon)=0.5%*
  - *MS({Milk, Broccoli}) = min (MS(Milk), MS(Broccoli)) = 0.1%*
  - *Challenge: Support is no longer anti-monotone*
    - Suppose: Support(Milk, Coke) = 1.5% and Support(Milk, Coke, Broccoli) = 0.5%
    - {Milk,Coke} is infrequent but {Milk,Coke,Broccoli} is frequent

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

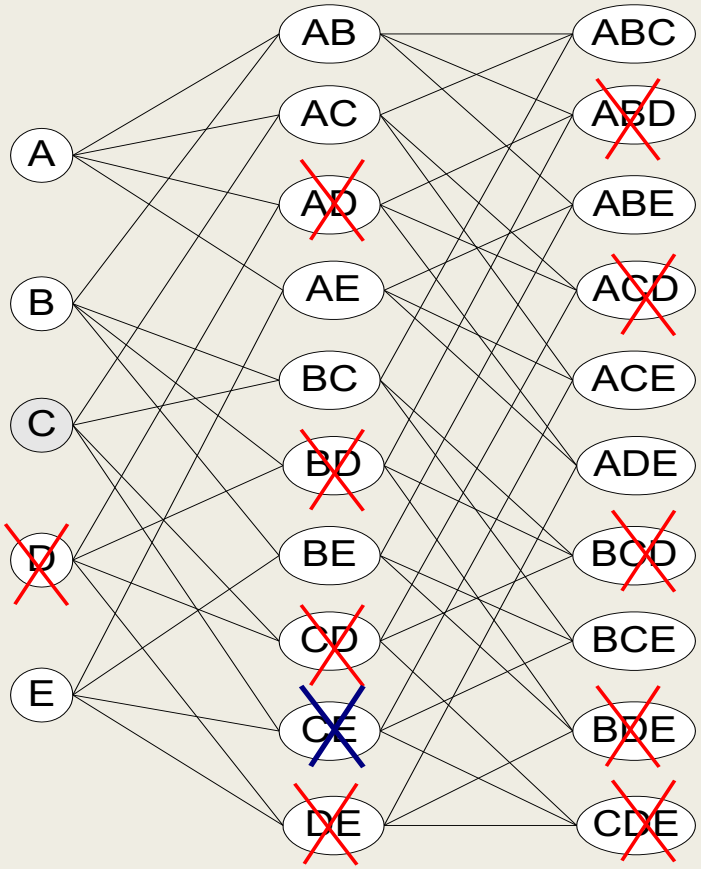
Multiple Minimum Support

Item	MS(I)	Sup(I)
A	0.10%	0.25%
B	0.20%	0.26%
C	0.30%	0.29%
D	0.50%	0.05%
E	3%	4.20%



Multiple Minimum Support

Item	MS(I)	Sup(I)
A	0.10%	0.25%
B	0.20%	0.26%
C	0.30%	0.29%
D	0.50%	0.05%
E	3%	4.20%



Multiple Minimum Support (Liu 1999)

- Order the items according to their minimum support (in ascending order)
  - *e.g.: MS(Milk)=5%, MS(Coke) = 3%, MS(Broccoli)=0.1%, MS(Salmon)=0.5%*
  - *Ordering: Broccoli, Salmon, Coke, Milk*

- Need to modify Apriori such that:
  - L_1 *: set of frequent items*
  - F_1 *: set of items whose support is* $\geq$ *MS(1) where MS(1) is* $min_i(MS(i))$
  - C_2 *: candidate itemsets of size 2 is generated from* F_1 *instead of* L_1

Multiple Minimum Support (Liu 1999)

- Modifications to Apriori:
  - *In traditional Apriori,*
    - A candidate (k+1)-itemset is generated by merging two frequent itemsets of size k
    - The candidate is pruned if it contains any infrequent subsets of size k
  - *Pruning step has to be modified:*
    - Prune only if subset contains the first item
    - e.g.: Candidate={Broccoli, Coke, Milk} (ordered according to minimum support)
    - {Broccoli, Coke} and {Broccoli, Milk} are frequent but {Coke, Milk} is infrequent
      - *Candidate is not pruned because {Coke,Milk} does not contain the first item, i.e., Broccoli.*

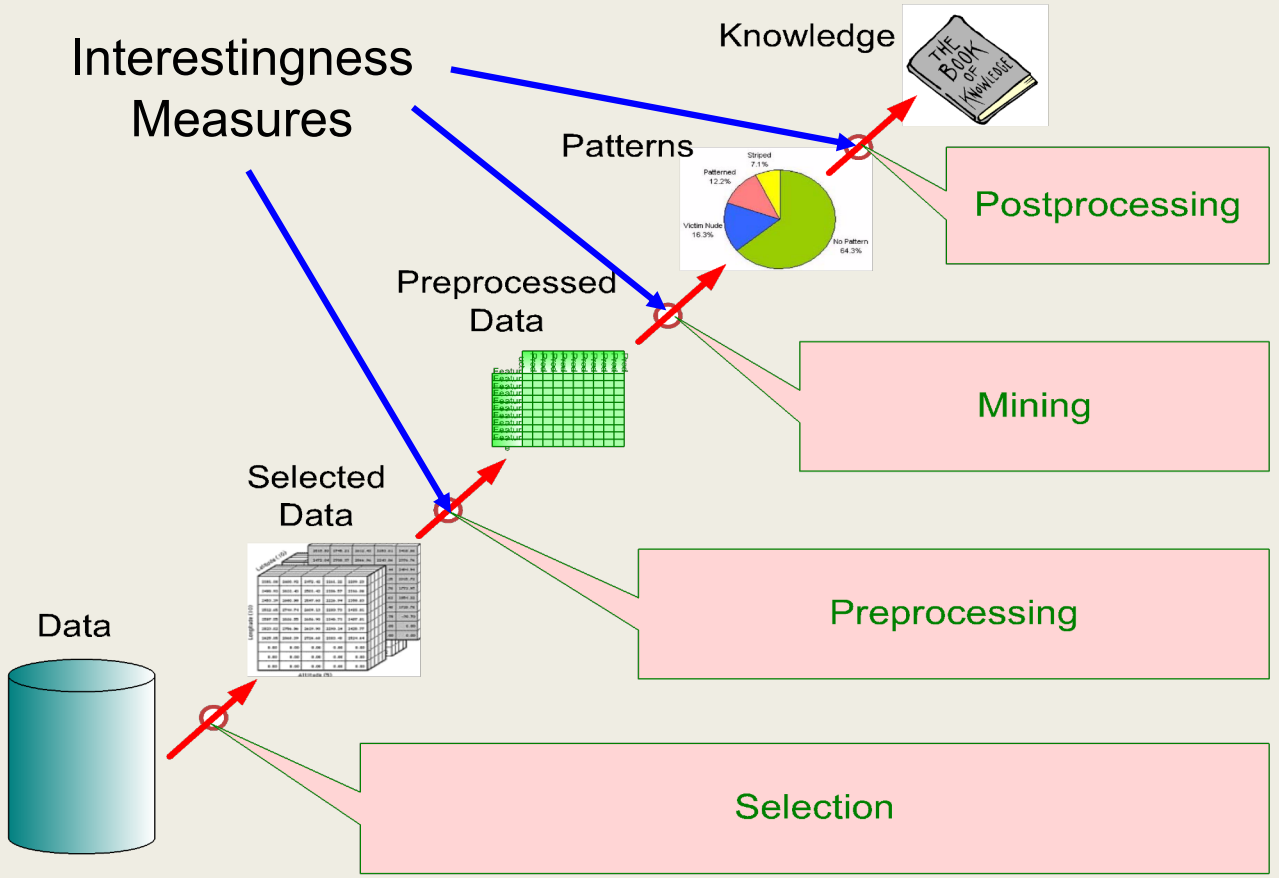
Pattern Evaluation

- Association rule algorithms tend to produce too many rules
  - *many of them are uninteresting or redundant*
  - *Redundant if {A,B,C} → {D} and {A,B} → {D}*
    have same support & confidence

- Interestingness measures can be used to prune/rank the derived patterns

- In the original formulation of association rules, support & confidence are the only measures used

Application of Interestingness Measure



Computing Interestingness Measure

- Given a rule $X \rightarrow Y$, information needed to compute rule interestingness can be obtained from a contingency table

Contingency table for $X \rightarrow Y$

	Y	$\overline{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\overline{X}$	f_{01}	f_{00}	f_{o+}
	f_{+1}	f_{+0}	|T|

f_{11}: support of X and Y
f_{10}: support of X and $\overline{Y}$
f_{01}: support of $\overline{X}$ and Y
f_{00}: support of $\overline{X}$ and $\overline{Y}$

Used to define various measures

- support, confidence, lift, Gini, J-measure, etc.

Drawback of Confidence

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

$\Rightarrow$ Although confidence is high, rule is misleading

$\Rightarrow$ P(Coffee|$\overline{\text{Tea}}$) = 0.9375

Statistical Independence

- Population of 1000 students
  - *600 students know how to swim (S)*
  - *700 students know how to bike (B)*
  - *420 students know how to swim and bike (S,B)*

  - $P(S \wedge B) = 420/1000 = 0.42$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$

  - $P(S \wedge B) = P(S) \times P(B)$ => *Statistical independence*
  - $P(S \wedge B) > P(S) \times P(B)$ => *Positively correlated*
  - $P(S \wedge B) < P(S) \times P(B)$ => *Negatively correlated*

Statistical-based Measures

- Measures that take into account statistical dependence

$$Lift = \frac{P(Y \mid X)}{P(Y)}$$

$$Interest = \frac{P(X,Y)}{P(X)P(Y)}$$

$$PS = P(X,Y) - P(X)P(Y)$$

$$\phi - coefficient = \frac{P(X,Y) - P(X)P(Y)}{\sqrt{P(X)[1-P(X)]P(Y)[1-P(Y)]}}$$

Example: Lift/Interest

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

$\Rightarrow$ Lift = 0.75/0.9= 0.8333 (< 1, therefore is negatively associated)

Drawback of Lift & Interest

	Y	$\bar{Y}$	
X	10	0	10
$\bar{X}$	0	90	90
	10	90	100

$$Lift = \frac{0.1}{(0.1)(0.1)} = 10$$

	Y	$\bar{Y}$	
X	90	0	90
$\bar{X}$	0	10	10
	90	10	100

$$Lift = \frac{0.9}{(0.9)(0.9)} = 1.11$$

Statistical independence:

If P(X,Y)=P(X)P(Y) => Lift = 1

There are lots of measures proposed in the literature

Some measures are good for certain applications, but not for others

What criteria should we use to determine whether a measure is good or bad?

What about Apriori-style support based pruning? How does it affect these measures?

#	Measure	Formula
1	ϕ-coefficient	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
2	Goodman-Kruskal's (λ)	$\frac{\sum_j \max_k P(A_j,B_k)+\sum_k \max_j P(A_j,B_k)-\max_j P(A_j)-\max_k P(B_k)}{2-\max_j P(A_j)-\max_k P(B_k)}$
3	Odds ratio (α)	$\frac{P(A,B)P(\overline{A},\overline{B})}{P(A,\overline{B})P(\overline{A},B)}$
4	Yule's Q	$\frac{P(A,B)P(\overline{A}\overline{B})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{A}\overline{B})+P(A,\overline{B})P(\overline{A},B)} = \frac{\alpha-1}{\alpha+1}$
5	Yule's Y	$\frac{\sqrt{P(A,B)P(\overline{A}\overline{B})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{A}\overline{B})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}} = \frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$
6	Kappa (κ)	$\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
7	Mutual Information (M)	$\frac{\sum_i \sum_j P(A_i,B_j)\log\frac{P(A_i,B_j)}{P(A_i)P(B_j)}}{\min(-\sum_i P(A_i)\log P(A_i), -\sum_j P(B_j)\log P(B_j))}$
8	J-Measure (J)	$\max\left(P(A,B)\log(\frac{P(B\|A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}),\right.$ $\left.P(A,B)\log(\frac{P(A\|B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})})\right)$
9	Gini index (G)	$\max\Big(P(A)[P(B\|A)^2+P(\overline{B}\|A)^2]+P(\overline{A})[P(B\|\overline{A})^2+P(\overline{B}\|\overline{A})^2]$ $-P(B)^2-P(\overline{B})^2,$ $P(B)[P(A\|B)^2+P(\overline{A}\|B)^2]+P(\overline{B})[P(A\|\overline{B})^2+P(\overline{A}\|\overline{B})^2]$ $-P(A)^2-P(\overline{A})^2\Big)$
10	Support (s)	$P(A,B)$
11	Confidence (c)	$\max(P(B\|A), P(A\|B))$
12	Laplace (L)	$\max\left(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2}\right)$
13	Conviction (V)	$\max\left(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})}\right)$
14	Interest (I)	$\frac{P(A,B)}{P(A)P(B)}$
15	cosine (IS)	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
16	Piatetsky-Shapiro's (PS)	$P(A,B)-P(A)P(B)$
17	Certainty factor (F)	$\max\left(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)}\right)$
18	Added Value (AV)	$\max(P(B\|A)-P(B), P(A\|B)-P(A))$
19	Collective strength (S)	$\frac{P(A,B)+P(\overline{A}\overline{B})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{A}\overline{B})}$
20	Jaccard (ζ)	$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$
21	Klosgen (K)	$\sqrt{P(A,B)}\max(P(B\|A)-P(B), P(A\|B)-P(A))$

Properties of A Good Measure

- Piatetsky-Shapiro:
  3 properties a good measure M must satisfy:
  - *M(A,B) = 0 if A and B are statistically independent*
  - *M(A,B) increase monotonically with P(A,B) when P(A) and P(B) remain unchanged*
  - *M(A,B) decreases monotonically with P(A) [or P(B)] when P(A,B) and P(B) [or P(A)] remain unchanged*

Comparing Different Measures

10 examples of contingency tables:

Example	f_{11}	f_{10}	f_{01}	f_{00}
E1	8123	83	424	1370
E2	8330	2	622	1046
E3	9481	94	127	298
E4	3954	3080	5	2961
E5	2886	1363	1320	4431
E6	1500	2000	500	6000
E7	4000	2000	1000	3000
E8	4000	2000	2000	2000
E9	1720	7121	5	1154
E10	61	2483	4	7452

Rankings of contingency tables using various measures:

#	ϕ	λ	α	Q	Y	κ	M	J	G	s	c	L	V	I	IS	PS	F	AV	S	ζ	K
E1	1	1	3	3	3	1	2	2	1	3	5	5	4	6	2	2	4	6	1	2	5
E2	2	2	1	1	1	2	1	3	2	2	1	1	1	8	3	5	1	8	2	3	6
E3	3	3	4	4	4	3	3	8	7	1	4	4	6	10	1	8	6	10	3	1	10
E4	4	7	2	2	2	5	4	1	3	6	2	2	2	4	4	1	2	3	4	5	1
E5	5	4	8	8	8	4	7	5	4	7	9	9	9	3	6	3	9	4	5	6	3
E6	6	6	7	7	7	7	6	4	6	9	8	8	7	2	8	6	7	2	7	8	2
E7	7	5	9	9	9	6	8	6	5	4	7	7	8	5	5	4	8	5	6	4	4
E8	8	9	10	10	10	8	10	10	8	4	10	10	10	9	7	7	10	9	8	7	9
E9	9	9	5	5	5	9	9	7	9	8	3	3	3	7	9	9	3	7	9	9	8
E10	10	8	6	6	6	10	5	9	10	10	6	6	5	1	10	10	5	1	10	10	7

Property under Variable Permutation

	B	$\bar{B}$
A	p	q
$\bar{A}$	r	s

	A	$\bar{A}$
B	p	r
$\bar{B}$	q	s

Does M(A,B) = M(B,A)?

Symmetric measures:

- support, lift, collective strength, cosine, Jaccard, etc

Asymmetric measures:

- confidence, conviction, Laplace, J-measure, etc

Property under Row/Column Scaling

Grade-Gender Example (Mosteller, 1968):

	Male	Female	
High	2	3	5
Low	1	4	5
	3	7	10

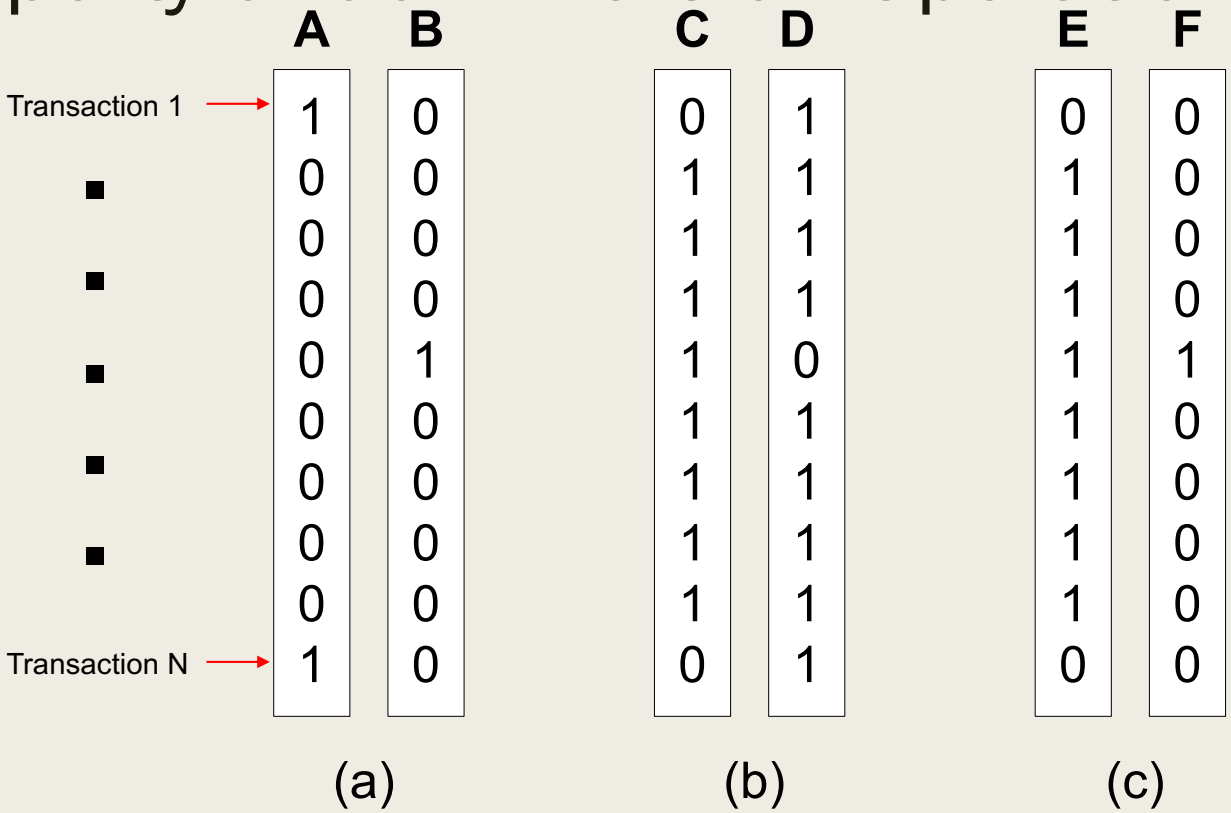
	Male	Female	
High	4	30	34
Low	2	40	42
	6	70	76

2x 10x

Mosteller:

Underlying association should be independent of the relative number of male and female students in the samples

Property under Inversion Operation



Example: ϕ-Coefficient

- ϕ-coefficient is analogous to correlation coefficient for continuous variables

	Y	$\overline{Y}$	
X	60	10	70
$\overline{X}$	10	20	30
	70	30	100

	Y	$\overline{Y}$	
X	20	10	30
$\overline{X}$	10	60	70
	30	70	100

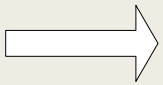
$$\phi = \frac{0.6 - 0.7 \times 0.7}{\sqrt{0.7 \times 0.3 \times 0.7 \times 0.3}} = 0.5238$$

$$\phi = \frac{0.2 - 0.3 \times 0.3}{\sqrt{0.7 \times 0.3 \times 0.7 \times 0.3}} = 0.5238$$

ϕ Coefficient is the same for both tables

Property under Null Addition

	B	$\overline{B}$
A	p	q
$\overline{A}$	r	s



	B	$\overline{B}$
A	p	q
$\overline{A}$	r	s + k

Invariant measures:

- support, cosine, Jaccard, etc

Non-invariant measures:

- correlation, Gini, mutual information, odds ratio, etc

Different Measures have Different Properties

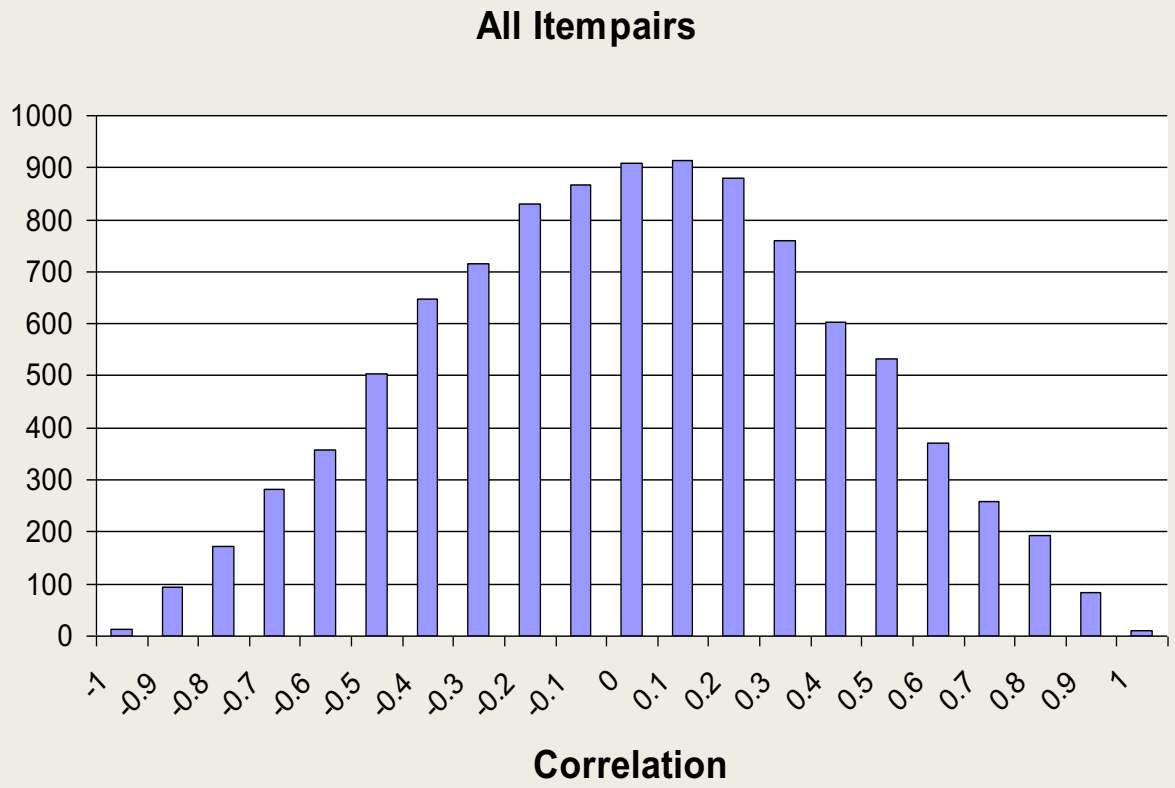
Symbol	Measure	Range	P1	P2	P3	O1	O2	O3	O3'	O4
Φ	Correlation	-1 … 0 … 1	Yes	Yes	Yes	Yes	No	Yes	Yes	No
λ	Lambda	0 … 1	Yes	No	No	Yes	No	No*	Yes	No
α	Odds ratio	0 … 1 … ∞	Yes*	Yes	Yes	Yes	Yes	Yes*	Yes	No
Q	Yule's Q	-1 … 0 … 1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Y	Yule's Y	-1 … 0 … 1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
κ	Cohen's	-1 … 0 … 1	Yes	Yes	Yes	Yes	No	No	Yes	No
M	Mutual Information	0 … 1	Yes	Yes	Yes	Yes	No	No*	Yes	No
J	J-Measure	0 … 1	Yes	No	No	No	No	No	No	No
G	Gini Index	0 … 1	Yes	No	No	No	No	No*	Yes	No
s	Support	0 … 1	No	Yes	No	Yes	No	No	No	No
c	Confidence	0 … 1	No	Yes	No	Yes	No	No	No	Yes
L	Laplace	0 … 1	No	Yes	No	Yes	No	No	No	No
V	Conviction	0.5 … 1 … ∞	No	Yes	No	Yes**	No	No	Yes	No
I	Interest	0 … 1 … ∞	Yes*	Yes	Yes	Yes	No	No	No	No
IS	IS (cosine)	0 .. 1	No	Yes	Yes	Yes	No	No	No	Yes
PS	Piatetsky-Shapiro's	-0.25 … 0 … 0.25	Yes	Yes	Yes	Yes	No	Yes	Yes	No
F	Certainty factor	-1 … 0 … 1	Yes	Yes	Yes	No	No	No	Yes	No
AV	Added value	0.5 … 1 … 1	Yes	Yes	Yes	No	No	No	No	No
S	Collective strength	0 … 1 … ∞	No	Yes	Yes	Yes	No	Yes*	Yes	No
ζ	Jaccard	0 .. 1	No	Yes	Yes	Yes	No	No	No	Yes
K	Klosgen's	$\left(\sqrt{\frac{2}{\sqrt{3}}-1}\right)\left(2-\sqrt{3}-\frac{1}{\sqrt{3}}\right)\ldots 0 \ldots \frac{2}{3\sqrt{3}}$	Yes	Yes	Yes	No	No	No	No	No

Support-based Pruning

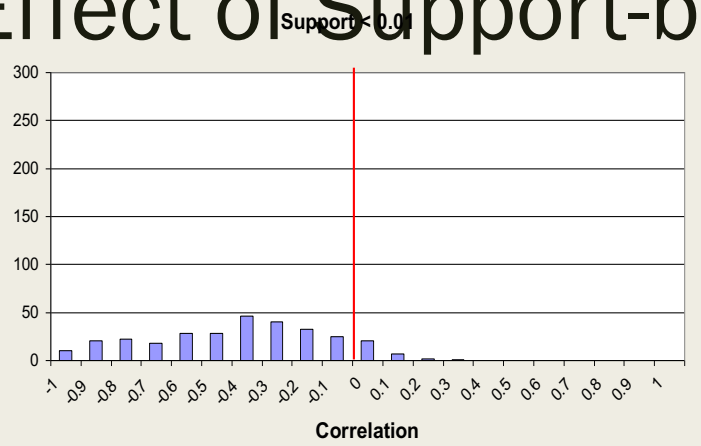
- Most of the association rule mining algorithms use support measure to prune rules and itemsets

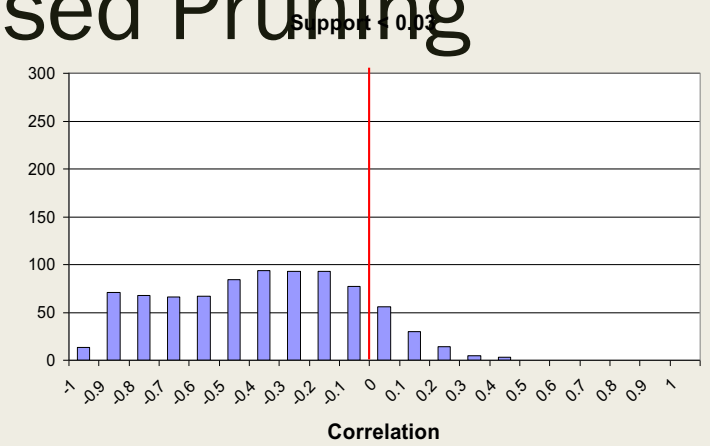
- Study effect of support pruning on correlation of itemsets
  - *Generate 10000 random contingency tables*
  - *Compute support and pairwise correlation for each table*
  - *Apply support-based pruning and examine the tables that are removed*

Effect of Support-based Pruning

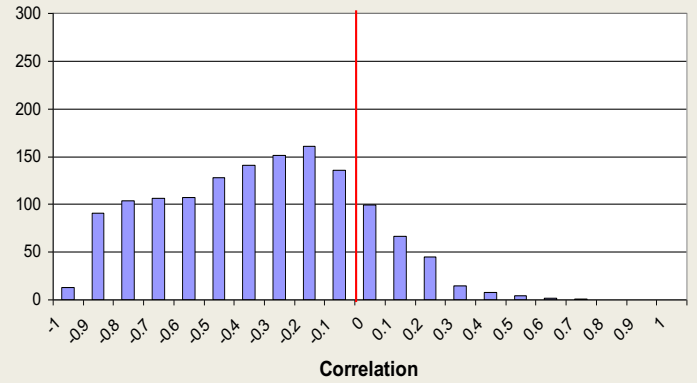


Effect of Support-based Pruning





Support-based pruning eliminates mostly negatively correlated itemsets



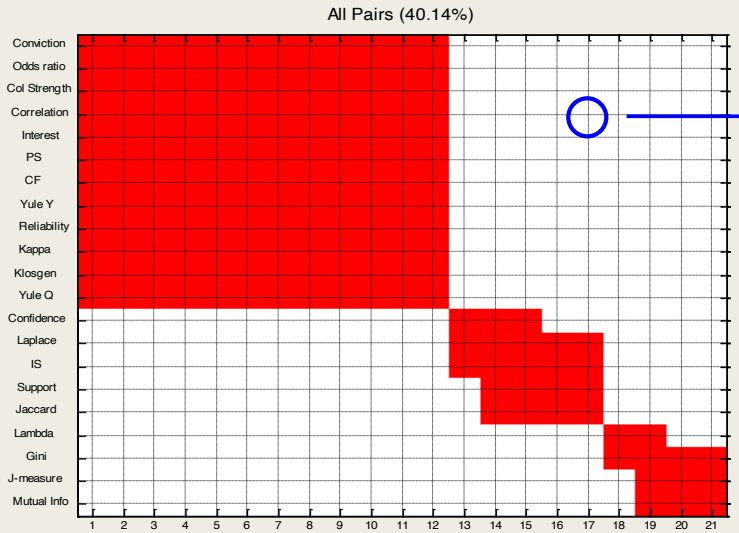
Effect of Support-based Pruning

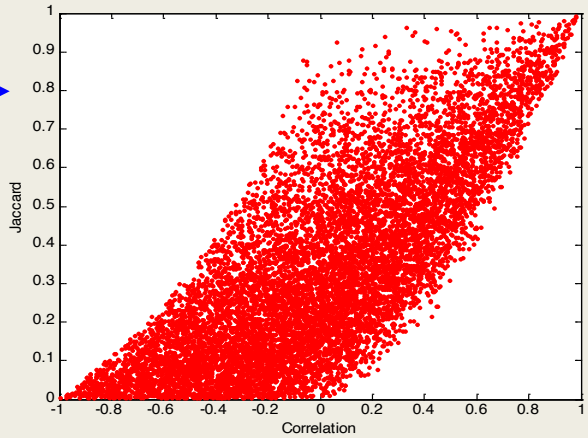
- Investigate how support-based pruning affects other measures

- Steps:
  - *Generate 10000 contingency tables*
  - *Rank each table according to the different measures*
  - *Compute the pair-wise correlation between the measures*

Effect of Support-based Pruning

- Without Support Pruning (All Pairs)



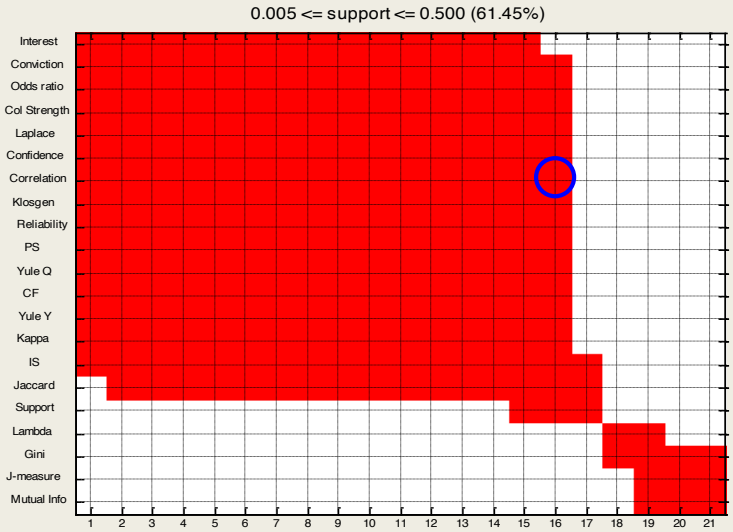


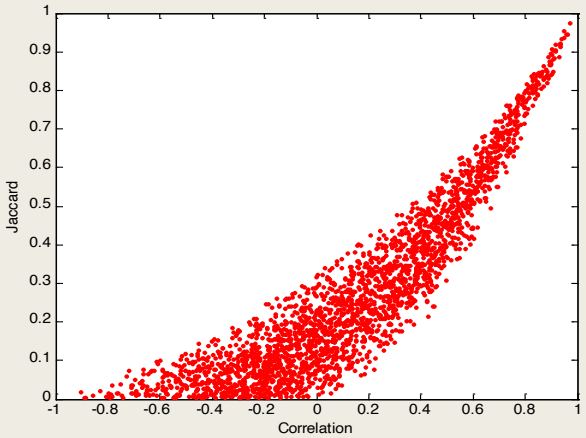
Scatter Plot between Correlation & Jaccard Measure

- Red cells indicate correlation between the pair of measures > 0.85
- 40.14% pairs have correlation > 0.85

Effect of Support-based Pruning

- 0.5% ≤ support ≤ 50%



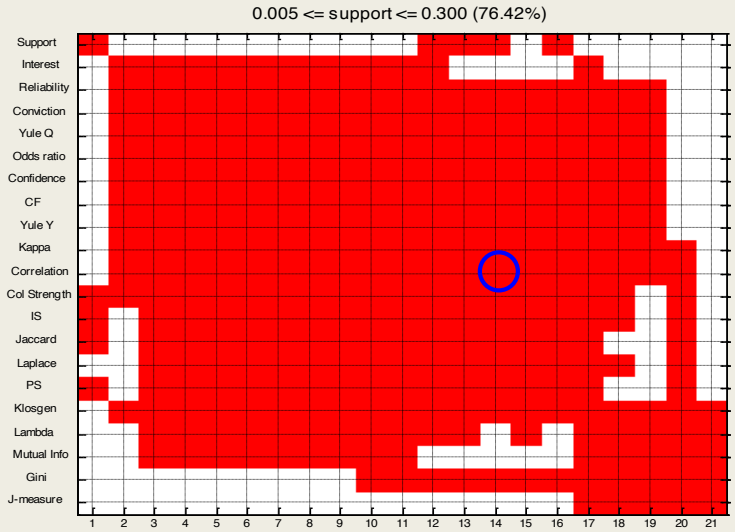


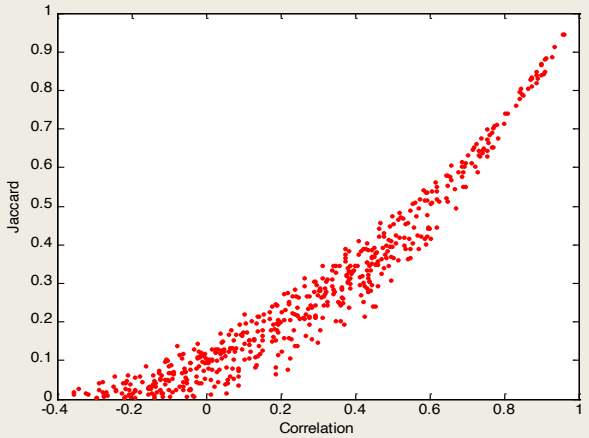
Scatter Plot between Correlation & Jaccard Measure:

- 61.45% pairs have correlation > 0.85

Effect of Support-based Pruning

- 0.5% ≤ support ≤ 30%





Scatter Plot between Correlation & Jaccard Measure

- 76.42% pairs have correlation > 0.85

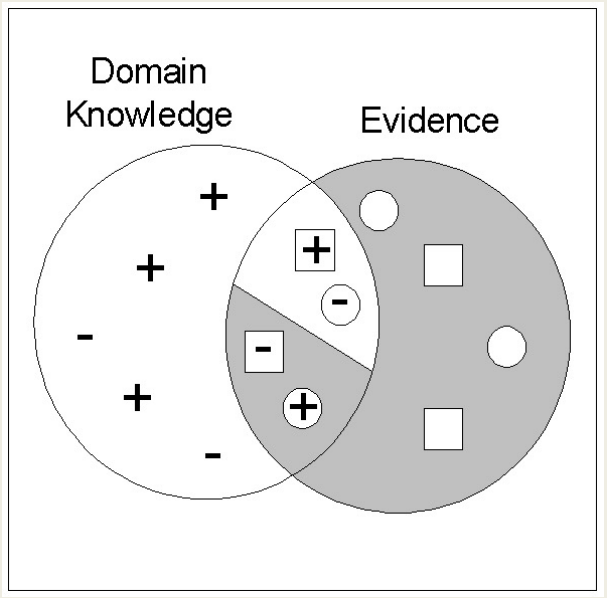
Subjective Interestingness Measure

- Objective measure:
  - *Rank patterns based on statistics computed from data*
  - *e.g., 21 measures of association (support, confidence, Laplace, Gini, mutual information, Jaccard, etc).*

- Subjective measure:
  - *Rank patterns according to user's interpretation*
    - A pattern is subjectively interesting if it contradicts the expectation of a user (Silberschatz & Tuzhilin)
    - A pattern is subjectively interesting if it is actionable (Silberschatz & Tuzhilin)

Interestingness via Unexpectedness

- Need to model expectation of users (domain knowledge)



- Need to combine expectation of users with evidence from data (i.e., extracted patterns)

Interestingness via Unexpectedness

- Web Data (Cooley et al 2001)
  - *Domain knowledge in the form of site structure*
  - *Given an itemset F = {X_1, X_2, …, X_k} (X_i : Web pages)*
    - L: number of links connecting the pages
    - lfactor = L / (k × k-1)
    - cfactor = 1 (if graph is connected), 0 (disconnected graph)
  - *Structure evidence = cfactor × lfactor*
  - *Usage evidence*

$$= \frac{P(X_1 \bigcap X_2 \bigcap \ldots \bigcap X_k)}{P(X_1 \cup X_2 \cup \ldots \cup X_k)}$$

  - *Use Dempster-Shafer theory to combine domain knowledge and evidence from data*

References

- TEXTBOOKS :

1. Pang-Ning Tan, Vipin Kumar, Michael Steinbach: **Chapter 6, Introduction to Data Mining**, Pearson, 2012.
2. Jiawei Han and Micheline Kamber: **Chapter 6, Data Mining - Concepts and Techniques**, 3rd Edition, MorganKaufmann Publisher, 2014