Chapter 4

Association Rules Mining
4.1. Introduction, Association Rules:

The task of association rule mining is to find certain association relationships among a set of objects (called items) in a database. The association relationships are described in association rules. Each rule has two measurements, support and confidence. Confidence is a measure of the rule’s strength, while support corresponds to statistical significance.

The task of discovering association rules was first introduced in 1993 [AIS 93]. Originally, association rule mining is focused on market “basket data” which stores items purchased on a per-transaction basis. A typical example of an association rule on market “basket data” is that 70% of customers who purchase bread also purchase butter. Later, association rule mining is also extended to handle quantitative data.

Finding association rules is valuable for crossing-marketing and attached mailing applications. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns. Besides application on business area, association rule mining can also be applied to other areas, such as medical diagnosis, remotely sensed imagery.

4.2. Formal Definition

Here is the formal definition of association rules [AS 94]:

Let I = {i_1, i_2, ..., i_m} be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that T ⊆ I associated with each transaction is a unique identifier, called its TID. An association rule is an implication of the form X => Y, where X ⊆ I, Y ⊆ I, and X ∩ Y = ∅. X is called antecedent while Y is called consequence of the rule.
There are two measurements for each rule, support and confidence. The rule $X \Rightarrow Y$ has support $s$ in the transaction set $D$ if $s\%$ of transactions in $D$ contains $X \cup Y$. The rule has confidence $c$ if $c\%$ of transactions in $D$ that contains $X$ also contains $Y$. Support indicates how frequently the pattern occurs, while confidence indicates the strength of the rule.

Given a user specified minimum support and minimum confidence, the problem of mining association rules is to find all the association rules whose support and confidence are larger than the respective thresholds. Thus, it can be decomposed into two sub-problems:

(1) Finding the frequent itemsets which have support above the predetermined minimum support.

(2) Deriving all rules, based on each frequent itemset, which have more than predetermined minimum confidence.

The whole performance is mainly determined by the first step, which is the generation of frequent itemsets.

4. 3. Algorithms (History of association rules mining algorithms)

Various algorithms are proposed to discover frequent itemsets.

4.3.1 AIS Algorithm

In AIS algorithms [AIS 93], candidate itemsets are generated and counted on-the-fly as the database is scanned. After reading a trans, it is determined which of the itemsets that were found to be large in the previous pass are contained in this trans. New candidate itemsets are generated by extending these large itemsets with other items in the trans.
4.3.2 SETM Algorithm

This algorithm was motivated by the desire to use SQL to compute large itemsets. Like AIS, the SETM algorithm also generates candidates on-the-fly based on trans read from the database. To use the standard SQL join operation for candidate generation, SETM separates candidate generation from counting.

4.3.3 Apriori Algorithm

4.3.3.1 Basic Apriori Algorithm

The disadvantage of AIS and SETM algorithm is the fact of unnecessarily generating and counting too many candidate itemsets that turn out to be small. To improve the performance, Apriori algorithm was proposed [AS 94]. Apriori algorithm generate the candidate itemsets to be counted in the pass by using only the itemsets found large in the previous pass – without considering the transactions in the database. Apriori beats AIS and SETM by more than an order of magnitude for large datasets.

The key idea of Apriori algorithm lies in the “downward-closed” property of support which means if an itemset has minimum support, then all its subsets also have minimum support. An itemset having minimum support is called frequent itemset (also called large itemset). So any subset of a frequent itemset must also be frequent. The candidate itemsets having k items can be generated by joining frequent itemsets having k-1 items, and deleting those that contain any subset that is not frequent.

Starting by finding all frequent 1-itemsets (itemsets with 1 item), we then consider 2-itemsets, and so forth, so during each iteration, only candidates found to be frequent in the previous iteration are used to generate a new candidate set during the next iteration. The algorithm terminates when there are no frequent k-
itemsets.

Notation is given below in Table 1, while Apriori algorithm in Figure 1.

<table>
<thead>
<tr>
<th>k-itemset</th>
<th>An itemset having k items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_k$</td>
<td>Set of frequent k-itemset (those with minimum support)</td>
</tr>
<tr>
<td>$C_k$</td>
<td>Set of candidate k-itemset (potentially frequent itemsets)</td>
</tr>
</tbody>
</table>

Table 4.1. Notation for mining algorithm
\[
L_1 = \{ \text{frequent 1-itemsets} \}; \\
\text{for (} k = 2; L_{k-1} \neq \emptyset; k++ \text{) do begin} \\
\quad C_k = \text{apriori-gen (} L_{k-1} \text{); // New candidates} \\
\quad \text{forall transactions } t \in D \text{ do begin} \\
\quad \quad C_t = \text{subset (} C_k, t); \quad // \text{Candidates contained in } t \\
\quad \quad \text{forall candidates } c \in C_t \text{ do} \\
\quad \quad \quad c.\text{count}++; \\
\quad \text{end} \\
\quad L_k = \{ c \in C_k | c.\text{count} \geq \text{minsup} \} \\
\text{end} \\
\text{Answer} = \bigcup_k L_k ;
\]

Figure 4.1 Apriori algorithm
Apriori-gen function takes as argument \( L_{k-1} \) and returns a superset of the set of all frequent \( k \)-itemsets. It consists join step and prune step. In join step join \( L_{k-1} \) with \( L_{k-1} \):

\[
\text{insert into } C_k \\
\text{select } p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k-1}, q.\text{item}_k \\
\text{from } L_{k-1} p, L_{k-1} q \\
\text{where } p.\text{item}_1 = q.\text{item}_1, \ldots, p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_k 
\]

In the prune step, delete all itemsets \( c \in C_k \) such that some \( (k-1) \)-subset of \( c \) is not in \( L_{k-1} \).

Subset function is to find all the candidate \( k \)-itemsets in transaction database using hash tree.

### 4.3.3.2 Variation of Apriori Algorithm

AprioriTID and AprioriHybrid are two variation of Apriori Algorithm.

* AprioriTID

The interesting feature of this algorithm is that the database \( D \) is not used for counting the support of candidate itemsets after the first pass. Rather, an encoding of the candidate itemsets used in the previous pass is employed for this purpose. In later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort.

* AprioriHybrid

The performance analysis on execution time shows that in the earlier passes, Apriori does better than AprioriTID, however, AprioriTID beats Apriori in later
passes. In addition, they use the same candidate generation procedure. Based on this observation, AprioriHybrid algorithm is proposed to combine the best features of Apriori and AprioriTid. AprioriHybrid uses Apriori in the initial passes and switches to AprioriTid when it expects that the set C’k (encoded set of original itemset) at the end of the pass will fit in memory. Of course, switching from Apriori to AprioriTid does involve a cost.

4.3.4 DHP (Direct Hashing and Pruning) Algorithm

In frequent itemset generation, the heuristic to construct the candidate set of large itemsets is crucial to performance. The larger the candidate set, the more processing cost required to discover the frequent itemsets. The processing in the initial iterations in fact dominates the total execution cost. It shows the initial candidate set generation, especially for the large 2-itemsets, is the key issue to improve the performance.

Based on the above concern, DHP is proposed [PCY 95]. DHP is a hash-based algorithm and is especially effective for the generation of candidate set of large 2-itemsets.

DHP has two major features, one is efficient generation for large itemsets, the other is effective reduction on trans database sizes.

Instead of including all k-itemsets from L_{k-1} \times L_{k-1} into C_k in Apriori, DHP adds a k-itemset into C_k only if that k-itemset passes the hash filtering, i.e., that k-itemset is hashed into a hash entry whose value is larger than or equal to the min support. Such hash filtering can drastically reduce the size of C_k.
DHP progressively trims the transaction database sizes in two ways, one is to reduce the size of some transactions, the other is to remove some transactions.

The execution time of the first pass of DHP is slightly larger than that of Apriori due to the extra overhead required for generating hash table. However, DHP incurs significantly smaller execution times than Apriori in later passes. The reason is that Apriori scans the full database for every pass, whereas DHP only scans the full database for the first 2 passes and then scans the reduced database thereafter.

4.3.4 PARTITION Algorithm
In [SON 95], Partition algorithm is proposed. Partition dives the database into small partitions such that they can be processed efficiently in memory independently to find out their large itemsets. The large itemsets from the partitions are then combined to form a set of candidate sets. Following that, only one scan of the database is required to find out the large itemsets from the candidates.

4.3.6 DIC (Dynamic Itemset Counting) Algorithm
DIC algorithm, proposed in [MUT 97], counts itemsets of different cardinality simultaneously. The transaction sequence is portioned into blocks. The itemsets are stored in a lattice which is initialized by all singleton sets. While a block is scanned, the count of each itemset in the lattice is adjusted. After a block is processed, an itemset is added to the lattice if and only if all its subsets are potentially large. At the end of the sequence, the algorithm rewinds to the beginning. It terminates when the count of each item in the lattice is determined. Thus after a finite number of scans, the lattice contains a superset of all large itemsets and their counts.
4.4. Quantitative and Categorical ARM

The previous algorithms only deal with Boolean data. To deal with quantitative attributes or categorical attributes, some work needs to be done to partition the values of that attribute and then combine adjacent partitions as necessary [SRI 96]. One example of quantitative association rule is "10% of married people between age 50 and 60 have at least two cares".

Quantitative and Categorical attributes can be converted into <attribute, integer value> pairs. For categorical attribute, the values of the attribute are mapped to a set of consecutive integers. For quantitative attribute with small domain value, map the values to consecutive integers to keep the order. For quantitative attribute with small domain value, map the values to consecutive integers to keep the order. For quantitative attribute with large domain value, first partition into intervals, then map intervals to consecutive integers.

If the intervals are too small, some rules may not have min support. If the intervals are too large, some rules may not have min confidence. There are two solutions to this problem:

- Combine adjacent intervals/values
- Introduce a user-specified "maximum support", stop combining intervals if their combined support exceeds this value.

Partial completeness is one measure to count the information lost by partitioning. The lower the level of partial completeness, the less the information lost. For a given partial completeness level, equi-depth partitioning minimizes the number of intervals required to satisfy that partial completeness level.
A modification of B-tree can be adopted for mining numeric association rules [WTL 98]. The idea is to merge adjacent intervals of numeric values, in a bottom-up manner, on the basis of maximizing the interestingness of a set of association rules.

In [MY 97], the definition of “interest” is proposed for association rules. It takes into account the semantics of interval data. Interval data means ordered data for which the separation between data points has meaning. In the presence of interval data, support and confidence are no longer intuitive measures of the interest of a surel.

4.5. Multi-level Association Rule Mining

Some transaction database may contain data with hierarchy structure. User are interested in generalized association rules that span different levels of the hierarchy, since sometimes more interesting rules can be derived by taking the hierarchy into account.

High-level rules, such as 80% of customers that purchase milk may also purchase bread. Low-level rules, such as 70% of people buy wheat bread if they buy 2% milk.

High-level rules could have high support, low-level rules may not have min support, but they could be more informative.

Let x' denotes an ancestor of x. If a set {x, y} has minimum support, so do {x, y'}, {x', y} and {x', y'}. However, if the rule x=>y has min support and confidence, only the rule x=>y' is guaranteed to have both min support and confidence. While
the rules \( x' \Rightarrow y \) and \( x' \Rightarrow y' \) will have min support, they may not have min confidence.

In multi-level ARM, it's possible to derive redundant rules.

Consider the rule

\[
\text{Milk} \Rightarrow \text{Cereal (8\% sup, 70\% conf)}
\]

If "Milk" is a parent of "Skim Milk", and about a quarter of sales of "Milk" are "Skim Milk", we would expect the rule \( \text{Skim Milk} \Rightarrow \text{Cereal} \) to have 2\% sup and 70\% conf. If the actual sup and conf for this rule are around 2\% and 70\%, the rule can be considered redundant since it does not give any additional information. We only want to find rules whose sup is more R times the expected valued or whose conf is more than R times the expected value, for some user-specified constant R. User can specify "minimum-interest-level" R.

4.6. Parallel ARM Algorithms

To mine association rule in parallel, some trade-off needs to be considered, for example, computation, communication, memory usage, synchronization and the use of problem-specific information. In [AS 96], three algorithms are proposed for mining association rules on a shared-nothing multiprocessor.

- Count Distribution algorithm
- Data Distribution algorithm
- Candidate Distribution algorithm

The focus of the Count Distribution algorithm is on minimizing communication. This algorithm uses a simple principle of allowing "redundant computations in parallel on otherwise idle processors to avoid communication".
The Data Distribution algorithm attempts to utilize the aggregate main memory of the system more effectively. In this algorithm, each processor counts mutually exclusive candidates. Thus, as the number of processors is increased, a large number of candidates can be counted in a pass. The downside of this algorithm is that every processor must broadcast its local data to all other processors in every pass.

The Candidate Distribution algorithm partitions both the data and candidates in such a way that each processor may proceed independently.

The PDM algorithm proposed in [PCY 95-2] tries to parallelize the DHP algorithm. Each node computes the globally large itemsets by exchanging their support counts of the candidate sets. In order to apply the hashing technique, all nodes have to broadcast the hashing result. A technique is proposed to decrease the number of messages. Among all the hash buckets, only those in which the total counts are larger than a threshold are selected for bucket count exchange, so that not all buckets have to be broadcasted.

In [HKK 97], two algorithms are proposed for parallel association rule mining. One is the Intelligent Data Distribution algorithm, the other is Hybrid Distribution algorithm. Intelligent Data Distribution algorithm efficiently uses aggregate memory of the parallel computer by employing intelligent candidate partitioning scheme and uses efficient communication mechanism to move data among the processors. The Hybrid Distribution algorithm dynamically partitions the candidate set to maintain good load balance.

Another algorithm, DMA (Distributed Mining of Association rules), is proposed for mining association rules in distributed databases [CNFF 96]. DMA generates a small number of candidate sets and requires only $O(n)$ messages for support count.
exchange for each candidate set, where \( n \) is the number of sites in a distributed database.

In [SK 98], a parallel algorithm is proposed for mining association rules with classification hierarchy. In this algorithm, the available memory space is fully utilized by identifying the frequently occurring candidate itemsets and copying them over all the processors, through which frequent itemsets can be processed locally without any communication. Thus it can effectively reduce the load skew among the processors.

4.7. Other topics on Association Rule Mining

In [SBM 98], the association rules are generalized to dependence rules which identify statistical dependence in both the presence and absence of items in itemsets. An example of such a rule is "90% of people who do not purchase coffee will purchase tea". A measure of significance of dependence is proposed via the chi-squared test for independence from classical statistics.

Optimized association rule mining, proposed in [RS 98], provides an effective way to focus on the most interesting characteristics involving certain attributes. Optimized association rules are permitted to contain uninstantiated attributes and the problem is to determine instantiations such that either the support, confidence is maximized.

There are some papers discussing how to specify what associations are to be mined by restricting constraints [NLH 98]. Constrained association queries are means of specifying the constraints to be satisfied by the antecedent and consequent of a mined association.
Association rule mining can also be generalized to “query flocks”, that is, parameterized queries with a filter condition to eliminate values of the parameters that are “uninteresting” [TUACMNR 98].

Sampling can also be used in association rule mining [BMUT 97] [LCK 98]. In [BMUT 97], item can be reordered to improve the low-level efficiency. Some term called “implication rules” is proposed, which means those rules normalized based on both the antecedent and the consequent and are truly implications (not simply a measure of co-occurrence). In [LCK 98], sampling technique is applied to find an approximate upper bound on the size of the difference between association rules of a database before and after it is updated. If the bound is low, then the amount of changes in association rules is small. So, the old association rules can be taken as a good approximation of the new ones. If the bound is high, the necessity of updating of the association rules in the database is signalled.

Online association rule mining, proposed in [H 99], means user is free to change the support threshold any time during the first scan of the transaction sequence. The algorithm Carma maintains a superset of all frequent itemsets and for each itemset a shrinking, deterministic interval on its support. After at most 2 scans the algorithm terminates with precise support for each large itemset.

Pruning techniques are always effective for association rule mining. In [CX 99], distributed and global pruning are proposed for data skewness and workload balance in parallel association rule mining.