Lecture 3
Association Rule Mining

CA4010: Data Warehousing and Data Mining
2016/2017 Semester 1

Dr. Mark Roantree
Dublin City University

Agenda

1. **Overview**
2. **Association Rules**
   - Generating Association Rules
3. **The Apriori Algorithm**
4. **Generating Itemsets**
5. **Generating Rules**
   - Measures of Interest: Lift and Leverage

Discovering Association Rules

- *Market Basket Analysis* a special form of **Association Rule Mining**.
- The rules generated for Market Basket Analysis are all of a certain restricted kind.
- Here we are interested in any rules that relate the purchases made by customers in a shop, frequently a large store with many thousands of products, as opposed to those that predict the purchase of one particular item.
- However, these methods are not restricted to the retail industry, but also analysis of items purchased by credit card, patientsâ medical records, crime data and data from satellites.

Transactions and Itemsets

- We will assume that we have a database comprising *n* transactions each of which is *a set of items*.
- Consider each transaction as corresponding to a group of purchases made by a customer, for example *{milk,cheese, bread}* or *{fish, cheese, bread, milk, sugar}*.
- Here *milk, cheese, bread* are **items** and *{milk, cheese, bread}* an **itemset**.
- The aim is finding rules known as **association rules** that apply to purchases made: buying *fish and sugar* is often associated with buying *milk and cheese*.
- However, we want rules that meet certain criteria for being *interesting*.

Search Criteria

- Including an item in a transaction just means that some quantity of it was bought.
- We *do not record* quantity or the items that a customer did *not* buy.
- We are interested in **rules** that include a test of what was not bought, such as customers who buy *milk* but do not buy *cheese* generally buy *bread*.
- We only look for rules that link all the items that were actually bought.

Search Criteria

- Assume that there are *m* possible items that can be bought with the letter *I* to denote the set of all possible items.
- The value of *m* can be very large.
- It partly depends on whether to consider all meat sold as a single item *meat* or as a separate item for each type of meat *(beef, lamb, chicken etc.)* or as a separate item for each *type and weight combination*.
- Clearly the number of different items that could be considered in a basket analysis is potentially very large.

Ordering Itemsets

- While order is not necessary, we write a transaction as {cheese, fish, meat}, not {meat, fish, cheese} etc.
- This does no harm, as the meaning is obviously the same, but has the effect of greatly reducing and simplifying the calculations required to discover all the *interesting* rules that can be extracted from the database.

Sample Transactions

If a database comprises 8 transactions (*n = 8*) and there are 5 different items (unrealistically low), denoted by *a, b, c, d* and *e*, we have *m = 5* and *I = {a, b, c, d, e}*:

Transaction number	Transactions (itemsets)
1	{a, b, c}
2	{a, b, c, d, e}
3	{b}
4	{c, d, e}
5	{c}
6	{b, c, d}
7	{c, d, e}
8	{c, e}

Figure 1: Sample Database of Transactions

All itemsets are subsets of *I*. We do not count the empty set so an itemset can have anything from 1 up to *m* members.

Itemset Count

- We use the term *support count* of an itemset *S*, or just the **count** of an itemset *S*, to mean the number of transactions in the database matched by *S*.
- We say that an itemset *S* matches a transaction *T* (itself an itemset) if *S* is a subset of *T*, i.e. all items in S are also in T.
- For example, itemset *{bread, milk}* matches the transaction *{cheese, bread, fish, milk, wine}*.
- If an itemset *S* = *{bread, milk}* has a support count of 12, written as `count(S)` = 12 or `count({bread, milk})` = 12, it means that 12 of the transactions in the database contain both the items *bread* and *milk*.

Support for an Itemset

- We define the **support** of an itemset S, written as `support(S)`, to be the proportion of itemsets in the database that are matched by S, i.e. the proportion of transactions that contain all the items in S.
- Alternatively we can look at it in terms of the frequency with which the items in S occur together in the database.
- Thus, where n is the number of transactions in the database:

$$support(S) = count(S)/n \tag{1}$$

Prediction

- The aim of Association Rule Mining (ARM) is to examine the contents of the database and find rules, known as *association rules*, in the data.
- For example, we might notice that when items *c* and *d* are bought, item *e* is also purchased.
- We can write this as the rule $cd \rightarrow e$
- We say: cd *implies* e, but we must be careful not to interpret this as meaning that buying *c* and *d* somehow *causes* *e* to be bought.
- It is better to think of rules in terms of *prediction*: if we know that *c* and *d* were bought we can *predict* that *e* was also bought.

Rules and Rulesets

- The rule $cd \rightarrow e$ is typical of many of the rules used in Association Rule Mining in that it is *invariably* incorrect.
- The rule is satisfied for transactions 2, 4 and 7 in Figure 1, but not for transaction 6, i.e. it is satisfied in 75% of cases.
- For basket analysis, it might be interpreted as: if bread and milk are bought, then cheese is also bought in 75% of cases.
- Note that the presence of items *c, d* and *e* in transactions 2, 4, and 7 can also be used to justify other rules such as
  $c \rightarrow ed$ *and* $e \rightarrow cd$
  which again do not have to be always correct.

Rules have a Left-hand side and a Right-hand side

- The number of rules generated from a small database is potentially very large, with many having no value.
- Before deciding which rules to discard and which to retain, some more terminology and notation.
- We can write the set of items appearing on the left- and right-hand sides of a given rule as L and R, respectively, and the rule itself as $L \rightarrow R$.
- L and R must each have at least one member and the two sets must be disjoint, i.e. have no common members.
- The left-hand and right-hand sides of a rule are often called its **antecedent** and **consequent** or its *body* and *head*, respectively.

ARM Terminology and Basics

- Note that with the $L \rightarrow R$ notation the left- and right-hand sides of rules are both sets.
- We should use $\{c, d\} \rightarrow \{e\}$ but we don't bother!
- The **union** of the sets L and R is the set of items that occur in *either* L or R.
- It is written $L \cup R$ (read as L *union* R).
- As L and R are disjoint and each has at least one member, the number of items in the itemset $\mathrm{L} \cup \mathrm{R}$, called the cardinality of $\mathrm{L} \cup \mathrm{R}$, must be at least two.

Calculating Support for Initial Ruleset

- For the rule $cd \rightarrow e$ we have $L = \{c, d\}$, $R = \{e\}$ and $L \cup R = \{c, d, e\}$.
- We can count the number of transactions in the database that are matched by the first two itemsets.
- Itemset L matches four transactions (2,4,6,7) and itemset
  $L \cup R$ matches 3 transactions (2,4,7) so `count(L)` = 4 and `count(L ∪ R)` = 3.
- As there are 8 transactions in the database we can calculate `support(L)` = `count(L)/8` = 4/8 and `support(L ∪ R)` = `count(L ∪ R)/8` = 3/8

Interesting Rules

- A large number of rules can be generated from even quite a small database and we are generally only interested in those that satisfy given criteria for being *interesting*.
- There are many ways in which the *interestingness* of a rule can be measured, but the two most commonly used are **support** and **confidence**.
- There is little point in using rules that only apply to a small proportion of the database or that predict poorly.

Support for Rules

- The **support** for a rule $L \to R$ is the *proportion* of the database to which the rule successfully applies, i.e. the proportion of transactions in which the items in L and the items in R occur together.
- This value is just the support for itemset $L \cup R$, so we define support for a rule:

$$support(L \to R) = support(L \cup R) \tag{2}$$

Predictive Accuracy

- The predictive accuracy of the rule $L \rightarrow R$ is measured by its **confidence**, defined as *the proportion of transactions* for which the rule is satisfied.
- This can be calculated as the number of transactions matched by the left-hand and right-hand sides combined, as a proportion of the number of transactions matched by the left-hand side on its own.
- Confidence is:

$$count(L \cup R)/count(L) \tag{3}$$

Confidence

- Ideally, every transaction matched by L is also matched by $L \cup R$, meaning the value of confidence is 1 and the rule would be called *exact*, i.e. always correct.
- In practice, rules are generally not exact, in which case `count(L ∪ R) < count(L)` and confidence is less than 1.
- Since the support count of an itemset is its support multiplied by the total number of transactions in the database, which is a constant value, confidence is either:

$$confidence(L \rightarrow R) = count(L \cup R)/count(L) \tag{4}$$

$$confidence(L \rightarrow R) = support(L \cup R)/support(L) \tag{5}$$

Thresholds

- It is customary to reject any rule for which the support is below a minimum threshold value called **minsup**, typically 0.01 (i.e. 1%)
- Also reject any rule with confidence below a minimum threshold value called **minconf**, typically 0.8 (i.e. 80%).
- For the rule $cd \rightarrow e$, the confidence is: `count(c, d, e)/count(c, d)` which is 3/4 = 0.75.

Generating Associating Rules

- We will use the term *supported itemset* to mean any itemset for which the value of support is greater than or equal to `minsup`.
- The terms **frequent itemset** and *large itemset* are often used instead of supported itemset.
- A basic but very inefficient method has two stages.
  1. Generate all supported itemsets $L \cup R$ with cardinality at least two.
  2. For each such itemset generate all the possible rules with at least one item on each side and retain those for which confidence $\geq$ `minconf`.

Generating Association Rules: issues

- The main problem is with *step 1* is generating all possible itemsets of cardinality two or greater.
- The number of such itemsets depends on the total number of items *m*.
- For a practical application, this can be hundreds or even thousands.

Get rid of itemsets with size ≤ 1

- The number of possible itemsets $L \cup R$ is the same as the number of possible subsets of I, the set of all items, which has cardinality m.
- There are 2^m such subsets.
- Of these, m itemsets have a single element and one has no element (the empty set).
- Thus, the number of itemsets $L \cup R$ with cardinality at least 2 is $2^m - m - 1$.

Summary

- If m takes the (unrealistically small) value of 20, the number of itemsets $L \cup R$ is 2^{20} - 20 - 1 = 1,048,555.
- If m has a (more realistic but still small) value of 100, the number of itemsets $L \cup R$ is 2^{100} - 100 - 1, which is approximately 10^{30}.
- Generating all the possible itemsets $L \cup R$ and then checking against the transactions in the database to establish which ones are supported is unrealistic.
- Fortunately, a far more efficient method of finding supported itemsets is available which makes the amount of work manageable, although it can still be large in some cases.

Theorem 1

If an itemset is supported, then all of its (non-empty) subsets are also supported.

Proof

- Removing one or more of the items from an itemset cannot reduce and will often increase the number of transactions that it matches.
- Thus, the support for a subset of an itemset must be at least as great as that for the original itemset.
- It follows that any (non-empty) subset of a supported itemset must also be supported.

Theorem 1: Downward Closure

- This result is sometimes called the *downward closure property* of itemsets.
- If we write the set containing all the supported itemsets with cardinality k as L_k, then a second important result follows in **Theorem 2**.
  (The use of the letter L stands for *large itemsets*).

Theorem 2

If $L_k = \Theta$ (the empty set) then L_{k+1}, L_{k+2} etc. must also be empty.

Proof

- If any supported itemsets of cardinality *k+1* or larger exist, they will have subsets of cardinality *k* and it follows from **Theorem 1** that all of these must be supported.
- However, we know that there are no supported itemsets of cardinality *k* as L_k is empty.
- Hence, there are no supported subsets of cardinality *k+1* or larger and L_{k+1}, L_{k+2} etc. must all be empty.

Theorem 2: benefits

- Taking advantage of this result, we generate the supported itemsets in ascending order of cardinality, i.e. all those with one element first, then all those with two elements, then all those with three elements etc.
- At each stage, the set L_k of supported items of cardinality k is generated from the previous set L_{k-1}.
- The benefit of this approach is that if at any stage L_k = Θ, the empty set, we know that L_{k+1}, L_{k+2} etc. must also be empty.
- Itemsets of cardinality *k+1* or greater do not need to be generated and then tested against the transactions in the database as they are guaranteed *not* to be supported.

Apriori Step 1

- We need a method of going from each set L_{k-1} to the next L_k in turn. There are 2 steps
  - First, we use L_{k-1} to form a candidate set C_k containing itemsets of cardinality k.
  - C_k must be constructed in such a way that it is certain to include all the supported itemsets of cardinality k.
  - It may unavoidably contain some other itemsets that are not supported.

Apriori Step 2

- Next, generate L_k as a subset of C_k.
- It is possible to discard some of the members of C_k as possible members of L_k by inspecting the members of L_{k-1}.
- The remainder must be checked against the transactions in the database to establish their support values.
- Only those itemsets with support greater than or equal to `minsup` are copied from C_k into L_k .
- This gives us the **Apriori algorithm** for generating all supported itemsets of cardinality at least 2.

The Apriori Algorithm

Create L_1 = set of supported itemsets of cardinality 1
Set k to 2
while ($L_{k-1} \neq \Theta$)
{ Create C_k from L_{k-1}
Prune all the itemsets in C_k that are not supported, to create L_k
k=k+1 }
The set of all supported itemsets is L1 $\cup L2 \cup \ldots \cup Lk$

Initial Step

- To begin: construct C_1 (the set of all itemsets comprising just a single item) and make a pass through the database counting the number of transactions that match each of these itemsets.
- Dividing each of these counts by the number of transactions in the database gives the value of **support** for each single-element itemset.
- We discard all those with $support < minsup$ to give L_1.
- This process can be represented as Figure 2, continuing until L_k is empty.

Apriori Illustrated

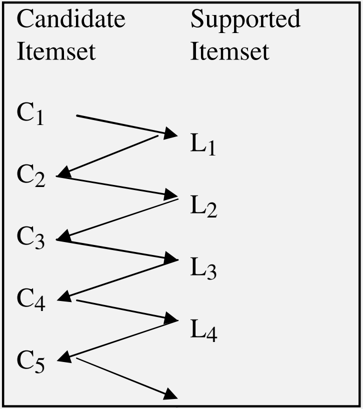

Figure 2: Illustration of the Apriori Algorithm

Apriori-gen Algorithm

- The **Apriori-gen** version takes L_{k-1} and generates C_k without using any of the earlier sets L_{k-1} etc.
- There are two stages, shown on the next slide.
- To illustrate the method, let us assume that L_4 is a list containing 17 itemsets of cardinality 4.

{{p, q, r, s}, {p, q, r, t}, {p, q, r, z}, {p, q, s, z}, {p, r, s, z}, {q, r, s, z}, {r, s,w, x}, {r, s,w, z}, {r, t, v, x}, {r, t, v, z}, {r, t, x, z}, {r, v, x, y}, {r, v, x, z}, {r, v, y, z}, {r, x, y, z}, {t, v, x, z}, {v, x, y, z}}

(Generates C_k from L_{k-1})

Join Step

Compare each member of L_{k-1}, say *A*, with every other member, say *B*, in turn.

If the first *k-2* items in A and B (all but the rightmost elements of the two itemsets) are identical,

place set $A \cup B$ into C_k.

Prune Step

For each member *c* of C_k in turn

{

Examine all subsets of *c* with *k-1* elements

Delete *c* from C_k if any of the subsets is not a member of L_{k-1}

}

Join Step(*k=5, k-2=3*)

- There are only six pairs of elements that have the first three elements in common.
- These are listed below together with the set that each combination causes to be placed into C_5.

First itemset	Second itemset	Contribution to C_5
$\{p,q,r,s\}$	$\{p,q,r,t\}$	$\{p,q,r,s,t\}$
$\{p,q,r,s\}$	$\{p,q,r,z\}$	$\{p,q,r,s,z\}$
$\{p,q,r,t\}$	$\{p,q,r,z\}$	$\{p,q,r,t,z\}$
$\{r,s,w,x\}$	$\{r,s,w,z\}$	$\{r,s,w,x,z\}$
$\{r,t,v,x\}$	$\{r,t,v,z\}$	$\{r,t,v,x,z\}$
$\{r,v,x,y\}$	$\{r,v,x,z\}$	$\{r,v,x,y,z\}$

Figure 3: Initial Version of candidate set C_5

Prune Step

- Here, each of the subsets of cardinality four of the itemsets in C_5 are examined in turn, with the following results.

Itemset in C_5	Subsets all in L_4?
$\{p, q, r, s, t\}$	No, e.g. $\{p, q, s, t\}$ is not a member of L_4
$\{p, q, r, s, z\}$	Yes
$\{p, q, r, t, z\}$	No, e.g. $\{p, q, t, z\}$ is not a member of L_4
$\{r, s, w, x, z\}$	No, e.g. $\{r, s, x, z\}$ is not a member of L_4
$\{r, t, v, x, z\}$	Yes
$\{r, v, x, y, z\}$	Yes

Figure 4: Final Version of candidate set C_5

Conclusion

- We can eliminate the first, third and fourth itemsets from C_5, making the final version of candidate set C_5
- {{p, q, r, s, z}, {r, t, v, x, z}, {r, v, x, y, z}}
- The three itemsets in C_5 now need to be checked against the database to establish which are supported.

Generating *Supported* Itemsets: An Example

- Assume that we have a database with 100 items and a large number of transactions.
- We begin by constructing C_1, the set of itemsets with a single member.
- We make a pass though the database to establish the support count for each of the 100 itemsets in C_1 and from these calculate L_1, the set of supported itemsets that comprise just a single member.
- Assume L_1 has 8 of these members: {a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}.
- We cannot generate rules as they have only one element, but we can form *candidate itemsets* of cardinality two.

Pass 1

- In generating C_2 from L_1, all pairs of (single-item) itemsets in L_1 are considered to match at the join step, since there is nothing to the left of the rightmost element of each one that might fail to match.
- In this case, the candidate generation algorithm gives us as members of C_2 all the itemsets with two members drawn from the eight items a, b, c, . . . h.
- Note that it would be pointless for a candidate itemset of two elements to include any of the other 92 items from the original set of 100, e.g. *{a, z}*, as one of its subsets would be *{z}*, which is not supported.

Generating C_2

There are 28 possible itemsets of cardinality 2 that can be formed from the items a, b, c, . . . , h:
{a, b}, {a, c}, {a, d}, {a, e}, {a, f}, {a, g}, {a, h},
{b, c}, {b, d}, {b, e}, {b, f}, {b, g}, {b, h},
{c, d}, {c, e}, {c, f}, {c, g}, {c, h},
{d, e}, {d, f}, {d, g}, {d, h},
{e, f}, {e, g}, {e, h},
{f, g}, {f, h},
{g, h}.

Pass 2

- We now need to make a second pass through the database to find the support counts of each of these itemsets,
- Then, divide each of the counts by the number of transactions in the database and reject any itemsets that have support less than `minsup`.
- Assume in this case that only 6 of the 28 itemsets with two elements turn out to be supported.
- L_2 = {{a, c}, {a, d}, {a, h}, {c, g}, {c, h}, {g, h}}.
- The algorithm for generating C_3 now gives just four members, i.e. {a, c, d}, {a, c, h}, {a, d, h} and {c, g, h}.

Pass 2 Analysis

- We now check whether each of the candidates meets the condition that all its subsets are supported.
- Itemsets {a, c, d} and {a, d, h} fail this test, because their subsets {c, d} and {d, h} are not members of L_2.
- That leaves just {a, c, h} and {c, g, h} as possible members of L_3.
- We now need a third pass through the database to find the support counts for itemsets {a, c, h} and {c, g, h}.

Pass 3

- We will assume they both turn out to be supported: $L_3 = \{\{a, c, h\}, \{c, g, h\}\}$.
- We now need to calculate C_4.
- It has no members, as the two members of L_3 do not have their first two elements in common.
- As C_4 is empty, L_4 must also be empty, which implies that L_5, L_6 etc. must also be empty and the process ends.

Pass 3 Analysis

- We have found all the itemsets of cardinality at least 2 with just three passes through the database.
- In doing so, we needed to find the support counts for just 100+28+2 = 130 itemsets.
- This is a considerable improvement on checking through the total number of possible itemsets for 100 items, which is approximately 10^{30}.

Overall Analysis

- The set of all supported itemsets with at least two members is the union of L_2 and L_3:
- {{a, c}, {a, d}, {a, h}, {c, g}, {c, h}, {g, h}, {a, c, h}, {c, g, h}}.
- It has eight itemsets as members.
- Now generate the candidate rules from each of these and determine which of them have a confidence value greater than or equal to `minconf`.

Analysis: Closure

- An itemset X is **closed** in a data set S, if there exists no proper super-itemset Y such that Y has the *same support count* as X in S.
- An itemset X is a **closed frequent itemset** in set S, if X is both *closed* and *frequent* in S.
- An itemset X is a **maximal frequent itemset** (or max-itemset) in set S, if X is frequent, and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S.

Closure Example

- Let C be the set of closed frequent itemsets for a data set S satisfying a minimum support threshold, minsup.
- Let M be the set of maximal frequent itemsets for S satisfying minsup.
- Assume we know the support count of each itemset in C and M.
- C and its count information can be used to derive the entire set of frequent itemsets.
- C contains complete information regarding its corresponding frequent itemsets; while M registers only the support of the maximal itemsets.

Closed and Maximal Frequent Itemsets

- Suppose that a transaction database has only two transactions: $\{(a_1, a_2, \ldots, a_{100}); (a_1, a_2, \ldots, a_{50})\}$.
- Let the minimum support count threshold be minsup=1.
- We find two closed frequent itemsets and their support counts: C = $\{\{a_1, a_2, \ldots, a_{100}\} : 1; \{a_1, a_2, \ldots, a_{50}\} : 2\}$.
- There is one maximal frequent itemset: M = $\{\{a_1, a_2, \ldots, a_{100}\} : 1\}$.
- We cannot include $\{a_1, a_2, \ldots, a_{50}\}$ as a maximal frequent itemset because it has a frequent superset: $\{a_1, a_2, \ldots, a_{100}\}$.

Closed and Maximal Frequent Itemsets (2)

- The set of closed frequent itemsets contains complete information regarding the frequent itemsets.
- From C, we can derive (for example):
  (1) $\{a_2, a_{45} : 2\}$ since $\{a_2, a_{45}\}$ is a sub-itemset of the itemset $\{a_1, a_2, \ldots, a_{50} : 2\}$; and
  (2) $\{a_8, a_{55} : 1\}$ since $\{a_8, a_{55}\}$ is not a sub-itemset of the previous itemset but of the itemset $\{a_1, a_2, \ldots, a_{100} : 1\}$.
- However, from the maximal frequent itemset, we can only assert that both itemsets ($\{a_2, a_{45}\}$ and $\{a_8, a_{55}\}$) are frequent, but we cannot assert their actual support counts.

Performance Issues (1)

- Although using the Apriori algorithm is clearly a significant step forward, it can run into substantial efficiency problems when there are a large number of transactions, items or both.
- One of the main problems is the large number of candidate itemsets generated during the early stages of the process.
- If the number of supported itemsets of cardinality one (the members of L_1) is large, say N, the number of candidate itemsets in C_2, which is $N(N - 1)/2$, can be a very large number.

Performance Issues (2)

- A fairly large (but not huge) database may comprise over 1,000 items and 100,000 transactions.
- If there are, say, 800 supported itemsets in L_1, the number of itemsets in C_2 is 800 x 799/2, which is approximately 320,000.
- Since Apriori-gen, current research focus is on efficiency through:
  - reducing the number of passes through all the transactions in the database,
  - reducing the number of unsupported itemsets in C_k,
  - more efficient counting of the number of transactions matched by each of the itemsets in C_k, or some combination of these.

Generating *Rules* for a Supported Itemset

- If supported itemset $L \cup R$ has k elements, we can generate all the possible rules $L \rightarrow R$ systematically from it and then check the value of confidence for each one.
- To do this, it is only necessary to generate all possible right-hand sides in turn.
- Each one must have at least one and at most *k-1* elements.
- Having generated the right-hand side of a rule all the unused items in $L \cup R$ must then be on the left-hand side.
- For itemset {c, d, e} there are 6 possible rules that can be generated, as in figure 6.

Sample Transactions: Reminder

If a database comprises 8 transactions (*n = 8*) and there are 5 different items (unrealistically low), denoted by *a, b, c, d* and *e*, we have *m = 5* and *I = {a, b, c, d, e}*:

Transaction number	Transactions (itemsets)
1	{a, b, c}
2	{a, b, c, d, e}
3	{b}
4	{c, d, e}
5	{c}
6	{b, c, d}
7	{c, d, e}
8	{c, e}

Figure 5: Sample Database of Transactions

Only one of the rules has a confidence value greater than or equal to minconf (0.8)

Rule $L \rightarrow R$	$\text{count}(L \cup R)$	$\text{count}(L)$	$\text{confidence}(L \rightarrow R)$
$de \rightarrow c$	3	3	1.0
$ce \rightarrow d$	3	4	0.75
$cd \rightarrow e$	3	4	0.75
$e \rightarrow cd$	3	4	0.75
$d \rightarrow ce$	3	4	0.75
$c \rightarrow de$	3	7	0.43

Figure 6: Six Rules Generated

The number of ways of selecting i items from the k in a supported itemset of cardinality k for the right-hand side of a rule is denoted by the mathematical expression ${}_kC_i$ which has the value

$$\frac{k!}{(k-1)!i!} \tag{6}$$

The total number of possible right-hand sides L and thus, the total number of possible rules that can be constructed from an itemset $L \cup R$ of cardinality k is
${}_kC_1 + {}_kC_2 + \ldots + {}_kC_{k-i}$.
It can be shown that the value of this sum is 2^k - 2.

Scaling Issues

- Assuming that *k* is reasonably small, say 10, this number is manageable.
- For $k = 10$ there are $2^{10} - 2 = 1{,}022$ possible rules.
- However, as *k* becomes larger the number of possible rules rapidly increases.
- For $k = 20$, it is 1,048,574.
- Fortunately, we can reduce the number of candidate rules considerably using simple logic.

Theorem 3

Transferring members of a supported itemset from the left-hand side of a rule to the right-hand side cannot increase the value of rule confidence.

Proof

For this purpose we will write the original rule as $A \cup B \rightarrow C$, where sets A, B and C all contain at least one element, have no elements in common and the union of the three sets is the supported itemset S.

Transferring the item or items in B from the left to the right-hand side then amounts to creating a new rule $A \rightarrow B \cup C$.

The union of the left- and right-hand sides is the same for both rules, namely the supported itemset S, so we have

$\text{confidence}(A \rightarrow B \cup C) = \frac{\text{support}(S)}{\text{support}(A)}$

$\text{confidence}(A \cup B \rightarrow C) = \frac{\text{support}(S)}{\text{support}(A \cup B)}$

It is clear that the proportion of transactions in the database matched by an itemset A must be at least as large as the proportion matched by a larger itemset $A \cup B$, i.e. $\text{support}(A) \geq \text{support}(A \cup B)$.

Hence it follows that $\text{confidence}(A \rightarrow B \cup C) \leq \text{confidence}(A \cup B \rightarrow C)$.

Using Theorem 3

- If the confidence of a rule $\geq \texttt{minconf}$ we will call the itemset on its righthand side *confident*.
- If not, we will call the right-hand itemset *unconfident*.
- From Theorem 3, we then have two important results that apply whenever the union of the itemsets on the two sides of a rule is fixed:
  - Any superset of an **unconfident** right-hand itemset is **unconfident**
  - Any (non-empty) subset of a **confident** right-hand itemset is **confident**

Generating Rules: Conclusions

- This is very similar to the situation with supported itemsets described in *Apriori*.
- We can generate confident right-hand side itemsets of increasing cardinality in a way similar to *Apriori*, with a considerable reduction in the number of candidate rules for which the confidence needs to be calculated.
- If at any stage there are no more confident itemsets of a certain cardinality, there cannot be any of larger cardinality and the rule generation process can stop.

Interest Measure: Lift

- While generally small in number (compared to overall database), the number of rules with *support* and *confidence* greater than specified threshold values can still be large.
- We would like additional *interestingness* measures we can use to reduce the number to a manageable size, or rank rules in order of importance.
- Two measures are *lift* and *leverage*.
- The **lift** of rule $L \cup R$ measures how many more times the items in L and R occur together in transactions than would be expected if the itemsets L and R were statistically independent.

`lift(L` $\rightarrow R$) Definition

- The number of times the items in L and R occur together in transactions is just `count`(L ∪ R).
- The number of times the items in L occur is `count`(L).
- The proportion of transactions matched by R is `support`(R).
- So if L and R were independent we would expect the number of times the items in L and R occurred together in transactions to be `count(L) X support(R)`.

$$lift(L \rightarrow R) = \frac{count(L \cup R)}{count(L)\ X\ support(R)} \tag{7}$$

`lift(L` $\rightarrow R)$ Definition (2)

- There are a number of ways to express lift.
- This form is equally popular using *n*, the number of transactions in the database.

$$lift(L \rightarrow R) = \frac{n\ X\ confidence(L \rightarrow R)}{count(R)} \tag{8}$$

Calculating Lift: An Example (1)

- Suppose we have a database with 2000 transactions and a rule $L \rightarrow R$ with the following support counts
- `count(L) = 220; count(R) = 250; count(L ∪ R) = 190.`
- We can calculate the values of support and confidence from these:
  - `support(L → R) = count(L ∪ R)/2000 = 0.095`
  - `confidence(L → R) = count(L ∪ R)/count(L) = 0.864`
  - `lift(L → R) = confidence(L ∪ R) X 2000/count(R) = 6.91`

Lift: what does it mean?

- The value of `support(R)` measures the support for *R* if we examine the whole of the database.
- In this example, the itemset matches 250 transactions out of 2000, a proportion of 0.125.
- The value of $\texttt{confidence(L} \rightarrow \texttt{R)}$ measures the support for *R* if we only examine the transactions that match *L*.
- In this case it is 190/220 = 0.864.
- So purchasing the items in *L* makes it 0.864/0.125 = 6.91 times more likely that the items in *R* are purchased.

Interesting Rules

- Lift values greater than 1 are *interesting*.
- They indicate that transactions containing *L* tend to contain *R* more often than transactions that do not contain *L*.
- Although `lift` is a useful measure of interestingness, it is not always the best one to use.
- In some cases, a rule with *higher* **support** and *lower* **lift** can be more interesting than one with lower support and higher lift because it applies to more cases.
- Another measure of interestingness that is sometimes used is *leverage*.

Leverage

- Leverage measures the *difference* between the support for $L \cup R$ and the support that would be expected if L and R were independent.
- The former is just `support(L ∪ R)`.
- The frequencies (i.e. supports) of L and R are `support(L)` and `support(R)`, respectively.
- If L and R were independent, the expected frequency of both occurring in the same transaction would be the product of `support(L)` and `support(R)`.

$$leverage(L \rightarrow R) = support(L \cup R) - support(L)\ X\ support(R). \quad (9)$$

Calculating Leverage: An Example (1)

- The value of the **leverage** of a rule is *always less than* its support.
- The number of rules satisfying the `support` $\geq$ `minsup` and `confidence` $\geq$ `minconf` constraints can be reduced by setting a *leverage constraint*.
- For example, `leverage` $\geq$ `0.0001`, corresponding to an improvement in support of one occurrence per 10,000 transactions in the database.
- Assume a database has 100,000 transactions and we have a rule $L \rightarrow R$ with these support counts
- `count(L) = 8000; count(R) = 9000; count (L U R) = 7000`

Leverage: what does it mean?

- `support = 0.070 confidence = 0.875`
- `lift = 9.722 leverage = 0.063`
- Thus, the rule applies to 7% of the transactions in the database and is satisfied for 87.5% of the transactions that include the items in *L*.
- The latter value is 9.722 times more frequent than would be expected by chance.
- The improvement in support compared with chance is 0.063, corresponding to 6.3 transactions per 100 in the database, i.e. approximately 6300 in the database of 100,000 transactions.