17
Association Rule Mining II

This chapter requires a basic knowledge of mathematical set theory. If you do not already have this, the notes in Appendix A will tell you all you need to know.

17.1 Introduction

This chapter is concerned with a special form of Association Rule Mining, which is known as *Market Basket Analysis*. 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. Although in this chapter ARM will be described in terms of this application, the methods described are not restricted to the retail industry. Other applications of the same kind include analysis of items purchased by credit card, patients' medical records, crime data and data from satellites.

M. Bramer, *Principles of Data Mining*, Undergraduate Topics
in Computer Science, DOI 10.1007/978-1-4471-7307-6_17

17.2 Transactions and Itemsets

We will assume that we have a database comprising n *transactions* (i.e. records), each of which is a set of *items*.

In the case of market basket analysis we can think of 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 etc. are items and we call {milk, cheese, bread} an *itemset*. We are interested in finding rules known as *association rules* that apply to the purchases made by customers, for example 'buying fish and sugar is often associated with buying milk and cheese', but only want rules that meet certain criteria for 'interestingness', which will be specified later.

Including an item in a transaction just means that some quantity of it was bought. For the purposes of this chapter, we are not interested in the quantity of cheese or the number of cans of dog food etc. bought. We do not record the items that a customer did *not* buy and are not 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.

We will assume that there are m possible items that can be bought and will use the letter I to denote the set of all possible items.

In a realistic case the value of m can easily be many hundreds or even many thousands. It partly depends on whether a company decides to consider, say, all the meat it sells 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. It is clear that even in quite a small store the number of different items that could be considered in a basket analysis is potentially very large.

The items in a transaction (or any other itemset) are listed in a standard order, which may be alphabetical or something similar, e.g. we will always 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 we need to do to discover all the 'interesting' rules that can be extracted from the database.

As an example, if a database comprises 8 transactions (so $n = 8$) and there are only 5 different items (an unrealistically low number), denoted by a, b, c, d and e, so we have $m = 5$ and $I = \{a, b, c, d, e\}$, the database might comprise the transactions shown in Figure 17.1.

Note that the details of how the information is actually stored in the database is a separate issue, which is not considered here.

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 17.1 A Database With Eight Transactions

For convenience we write the items in an itemset in the order in which they appear in set I, the set of all possible items, i.e. $\{a, b, c\}$ not $\{b, c, a\}$.

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

17.3 Support for an Itemset

We will 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 (which is itself an itemset) if S is a subset of T, i.e. all the items in S are also in T. For example itemset {bread, milk} matches the transaction {cheese, bread, fish, milk, wine}.

If an itemset $S = \{\text{bread}, \text{milk}\}$ has a support count of 12, written as $\text{count}(S) = 12$ or $\text{count}(\{\text{bread}, \text{milk}\}) = 12$, it means that 12 of the transactions in the database contain both the items bread and milk.

We define the *support* of an itemset S, written as $\text{support}(S)$, 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. So we have $\text{support}(S) = \text{count}(S)/n$, where n is the number of transactions in the database.

17.4 Association Rules

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 often bought too. We can write this as the rule

$$cd \rightarrow e$$

The arrow is read as 'implies', 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.

The rule $cd \rightarrow e$ is typical of most if not all of the rules used in Association Rule Mining in that it is not invariably correct. The rule is satisfied for transactions 2, 4 and 7 in Figure 17.1, 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 bought too 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 invariably correct.

The number of rules that can be generated from quite a small database is potentially very large. In practice most of them are of little if any practical value. We need some way of deciding which rules to discard and which to retain.

First we will introduce 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.

Note that with the $L \rightarrow R$ notation the left- and right-hand sides of rules are both sets. However we will continue to write rules that do not involve variables in a simplified notation, e.g. $cd \rightarrow e$ instead of the more accurate but also more cumbersome form $\{c, d\} \rightarrow \{e\}$.

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 $L \cup R$, called the *cardinality* of $L \cup R$, must be at least two.

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, numbers 2, 4, 6 and 7, and itemset $L \cup R$ matches 3 transactions, numbers 2, 4 and 7, so $\text{count}(L) = 4$ and $\text{count}(L \cup R) = 3$.

As there are 8 transactions in the database we can calculate

$$\text{support}(L) = \text{count}(L)/8 = 4/8$$

and

$$\text{support}(L \cup R) = \text{count}(L \cup R)/8 = 3/8$$

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 *interestingness*. There are many ways in which the interestingness of a rule can be measured, but the two most commonly used are *support* and *confidence*. The justification for this is that there is little point in using rules that only apply to a small proportion of the database or that predict only poorly.

The *support* for a rule $L \rightarrow R$ 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 have

$$\text{support}(L \rightarrow R) = \text{support}(L \cup R).$$

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, i.e. $\text{count}(L \cup R)/\text{count}(L)$.

Ideally, every transaction matched by L would also be matched by $L \cup R$, in which case the value of confidence would be 1 and the rule would be called *exact*, i.e. always correct. In practice, rules are generally not exact, in which case $\text{count}(L \cup R) < \text{count}(L)$ and the 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, the confidence of a rule can be calculated either by

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

or by

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

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%) and also to reject all rules

with confidence below a minimum threshold value called *minconf*, typically 0.8 (i.e. 80%).

For the rule $cd \rightarrow e$, the confidence is $\text{count}(\{c, d, e\})/\text{count}(\{c, d\})$, which is $3/4 = 0.75$.

17.5 Generating Association Rules

There are many ways in which all the possible rules can be generated from a given database. A basic but very inefficient method has two stages.

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.

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*.

Step 2 in this algorithm is fairly straightforward to implement and will be discussed in Section 17.8.

The main problem is with step 1 'generate all supported itemsets $L \cup R$ with cardinality at least 2', assuming we take this to mean that we first generate all possible itemsets of cardinality two or greater and then check which of them are supported. The number of such itemsets depends on the total number of items m. For a practical application this can be very large.

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 have a single element and one has no elements (the empty set). Thus the number of itemsets $L \cup R$ with cardinality at least 2 is $2^m - m - 1$.

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 takes the more realistic but still relatively 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 clearly unrealistic or impossible in practice.

Fortunately, a much 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.

17.6 Apriori

This account is based on the very influential *Apriori* algorithm by Agrawal and Srikant [1], which showed how association rules could be generated in a realistic timescale, at least for relatively small databases. Since then a great deal of effort has gone into looking for improvements on the basic algorithm to enable larger and larger databases to be processed.

The method relies on the following very important result.

Theorem 1
If an itemset is supported, 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. Hence 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.

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 from the above. (The use of the letter L stands for 'large itemsets'.)

Theorem 2
If $L_k = \emptyset$ (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.

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 is $\emptyset$, 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 certain to turn out not to be supported.

We need a method of going from each set L_{k-1} to the next L_k in turn. We can do this in two stages.

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 but may contain some other itemsets that are not supported.

Next we need to generate L_k as a subset of C_k. We can generally discard some of the members of C_k as possible members of L_k by inspecting the members of L_{k-1}. The remainder need to 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 the supported itemsets of cardinality at least 2 (Figure 17.2).

```
Create L_1 = set of supported itemsets of cardinality one
Set k to 2
while (L_{k-1} ≠ ∅) {
    Create C_k from L_{k-1}
    Prune all the itemsets in C_k that are not
        supported, to create L_k
    Increase k by 1
}
The set of all supported itemsets with at least two members is L_2 ∪ ··· ∪ L_{k-2}
```

Figure 17.2 The Apriori Algorithm (adapted from [1])

To start the process we construct C_1, the set of all itemsets comprising just a single item, then 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.

The process involved can be represented diagrammatically as Figure 17.3, continuing until L_k is empty.

Agrawal and Srikant's paper also gives an algorithm *Apriori-gen* which takes L_{k-1} and generates C_k without using any of the earlier sets L_{k-2} etc. There are two stages to this. These are given in Figure 17.4.

To illustrate the method, let us assume that L_4 is the list
$\{\{p,q,r,s\},\{p,q,r,t\},\{p,q,r,z\},\{p,q,s,z\},\{p,r,s,z\},\{q,r,s,z\},$
$\{r,s,w,x\},\{r,s,w,z\},\{r,t,v,x\},\{r,t,v,z\},\{r,t,x,z\},\{r,v,x,y\},$
$\{r,v,x,z\},\{r,v,y,z\},\{r,x,y,z\},\{t,v,x,z\},\{v,x,y,z\}\}$
which contains 17 itemsets of cardinality four.

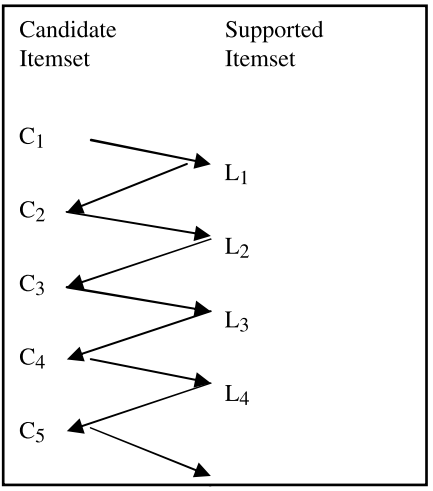

Figure 17.3 Diagram Illustrating the Apriori Algorithm

(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 (i.e. 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}
}

Figure 17.4 The Apriori-gen Algorithm (adapted from [1])

We begin with the join step.

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

The initial version of candidate set C_5 is

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

We now go on to the prune step where 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

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.

17.7 Generating Supported Itemsets: An Example

We can illustrate the entire process of generating supported itemsets from a database of transactions with the following 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.

Let us assume that L_1 has just 8 of these members, namely $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, $\{f\}$, $\{g\}$ and $\{h\}$. We cannot generate any rules from these, as they only have one element, but we can now form candidate itemsets of cardinality two.

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.

There are 28 possible itemsets of cardinality 2 that can be formed from the items a, b, c, ..., h. They are

$\{a,b\},\{a,c\},\{a,d\},\{a,e\},\{a,f\},\{a,g\},\{a,h\},$
$\{b,c\},\{b,d\},\{b,e\},\{b,f\},\{b,g\},\{b,h\},$
$\{c,d\},\{c,e\},\{c,f\},\{c,g\},\{c,h\},$
$\{d,e\},\{d,f\},\{d,g\},\{d,h\},$
$\{e,f\},\{e,g\},\{e,h\},$
$\{f,g\},\{f,h\},$
$\{g,h\}.$

As mentioned previously, it is convenient always to list the elements of an itemset in a standard order. Thus we do not include, say, $\{e, d\}$ because it is the same set as $\{d, e\}$.

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, so $L_2 = \{\{a,c\},\{a,d\},\{a,h\},\{c,g\},\{c,h\},\{g,h\}\}$.

The algorithm for generating C_3 now gives just four members, i.e. $\{a, c, d\}$, $\{a, c, h\}$, $\{a, d, h\}$ and $\{c, g, h\}$.

Before going to the database, we first 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 $\{c, g, h\}$. We will assume they both turn out to be supported, so $L_3 = \{\{a, c, h\}, \{c, g, h\}\}$.

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.

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

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

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.

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 $800 \times 799/2$, which is approximately 320,000.

Since Agrawal and Srikant's paper was published a great deal of research effort has been devoted to finding more efficient ways of generating supported itemsets. These generally involve 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 (perhaps using information collected in previous passes through the database), or some combination of these.

17.8 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 so 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 listed below.

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

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

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 $\frac{k!}{(k-i)!i!}$.

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 ${}_kC_1 + {}_kC_2 + \cdots + {}_kC_{k-1}$. It can be shown that the value of this sum is $2^k - 2$.

Assuming that k is reasonably small, say 10, this number is manageable. For $k = 10$ there are $2^{10} - 2 = 1022$ 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 the following result.

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 $A \cup B \to C$, 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 $A \to B \cup C$.
The union of the left- and right-hand sides is the same for both rules, namely the supported itemset S, so we have
confidence$(A \to B \cup C) = \frac{\text{support}(S)}{\text{support}(A)}$
confidence$(A \cup B \to C) = \frac{\text{support}(S)}{\text{support}(A \cup B)}$
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. support$(A) \geq$ support$(A \cup B)$.
Hence it follows that confidence$(A \to B \cup C) \leq$ confidence$(A \cup B \to C)$.

If the confidence of a rule $\geq$ *minconf* we will call the itemset on its right-hand side *confident*. If not, we will call the right-hand itemset *unconfident*. From the above theorem 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.

This is very similar to the situation with supported itemsets described in Section 17.6. 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.

17.9 Rule Interestingness Measures: Lift and Leverage

Although they are generally only a very small proportion of all the possible rules that can be derived from a 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 that are often used for this are *lift* and *leverage*.

The *lift* of rule $L \rightarrow R$ 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.

The number of times the items in L and R occur together in transactions is just $\text{count}(L \cup R)$. The number of times the items in L occur is $\text{count}(L)$. The proportion of transactions matched by R is $\text{support}(R)$. 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 $\text{count}(L) \times \text{support}(R)$. This gives the formula for lift:

$$\text{lift}(L \rightarrow R) = \frac{\text{count}(L \cup R)}{\text{count}(L) \times \text{support}(R)}$$

This formula can be written in several other forms, including

$$\text{lift}(L \rightarrow R) = \frac{\text{support}(L \cup R)}{\text{support}(L) \times \text{support}(R)}$$

$$\text{lift}(L \rightarrow R) = \frac{\text{confidence}(L \rightarrow R)}{\text{support}(R)}$$

$$\text{lift}(L \to R) = \frac{n \times \text{confidence}(L \to R)}{\text{count}(R)}$$

where n is the number of transactions in the database, and

$$\text{lift}(L \to R) = \frac{\text{confidence}(R \to L)}{\text{support}(L)}$$

Incidentally, from the second of these five formulae, which is symmetric in L and R, we can also see that

$$\text{lift}(L \to R) = \text{lift}(R \to L)$$

Suppose we have a database with 2000 transactions and a rule $L \to R$ with the following support counts

count(L)	count(R)	count($L \cup R$)
220	250	190

We can calculate the values of support and confidence from these:

$$\text{support}(L \to R) = \text{count}(L \cup R)/2000 = 0.095$$

$$\text{confidence}(L \to R) = \text{count}(L \cup R)/\text{count}(L) = 0.864$$

$$\text{lift}(L \to R) = \text{confidence}(L \cup R) \times 2000/\text{count}(R) = 6.91$$

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 confidence($L \to R$) 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.

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*. This measures the difference between the support for $L \cup R$ (i.e. the items in L and R occurring together in the database) and the support that would be expected if L and R were independent.

The former is just support($L \cup R$). 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).

This gives a formula for leverage:

$$\text{leverage}(L \to R) = \text{support}(L \cup R) - \text{support}(L) \times \text{support}(R).$$

The value of the leverage of a rule is clearly 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, e.g. leverage ≥ 0.0001, corresponding to an improvement in support of one occurrence per 10,000 transactions in the database.

If a database has 100,000 transactions and we have a rule $L \to R$ with these support counts

count(L)	count(R)	count($L \cup R$)
8000	9000	7000

the values of support, confidence, lift and leverage can be calculated to be 0.070, 0.875, 9.722 and 0.063 respectively (all to three decimal places).

So 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.

17.10 Chapter Summary

This chapter is concerned with a special form of Association Rule Mining known as *Market Basket Analysis*, the most common application of which is to relate the purchases made by the customers in a shop. An approach to finding rules of this kind, with support and confidence measures above specified thresholds, is described. This is based on the idea of *supported itemsets*. The *Apriori* algorithm for finding supported itemsets is described in detail. Further rule interestingness measures, *lift* and *leverage*, which can be used to reduce the number of rules generated are introduced.

17.11 Self-assessment Exercises for Chapter 17

1. Suppose that L_3 is the list

   $\{\{a,b,c\},\{a,b,d\},\{a,c,d\},\{b,c,d\},\{b,c,w\},\{b,c,x\},$
   $\{p,q,r\},\{p,q,s\},\{p,q,t\},\{p,r,s\},\{q,r,s\}\}$

   Which itemsets are placed in C_4 by the join step of the *Apriori-gen* algorithm? Which are then removed by the prune step?

2. Suppose that we have a database with 5000 transactions and a rule $L \rightarrow R$ with the following support counts

   $$\begin{array}{lcl} \text{count}(L) & = & 3400 \\ \text{count}(R) & = & 4000 \\ \text{count}(L \cup R) & = & 3000 \end{array}$$

   What are the values of support, confidence, lift and leverage for this rule?

Reference

[1] Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke & C. Zaniolo (Eds.), *Proceedings of the 20th international conference on very large databases (VLDB94)* (pp. 487–499). San Mateo: Morgan Kaufmann. http://citeseer.nj.nec.com/agrawal94fast.html.