UNIT 3
ASSOCIATION AND RECOMMENDATION

Advanced Analytical Theory and Methods: Association Rules

Association rules are an unsupervised learning method. This is a descriptive, not predictive, method often used to discover interesting relationships hidden in a large dataset. The disclosed relationships can be represented as rules or frequent itemsets. Association rules are commonly used for mining transactions in databases.

Here are some possible questions that association rules can answer:

- Which products tend to be purchased together?
- Of those customers who are similar to this person, what products do they tend to buy?
- Of those customers who have purchased this product, what other similar products do they tend to view or purchase?

5.1 Overview

Figure 5.1 shows the general logic behind association rules. Given a large collection of transactions (depicted as three stacks of receipts in the figure), in which each transaction consists of one or more items, association rules go through the items being purchased to see what items are frequently bought together and to discover a list of rules that describe the purchasing behavior. The goal with association rules is to discover interesting relationships among the items. (The relationship occurs too frequently to be random and is meaningful from a business perspective, which may or may not be obvious.) The relationships that are interesting depend both on the business context and the nature of the algorithm being used for the discovery.

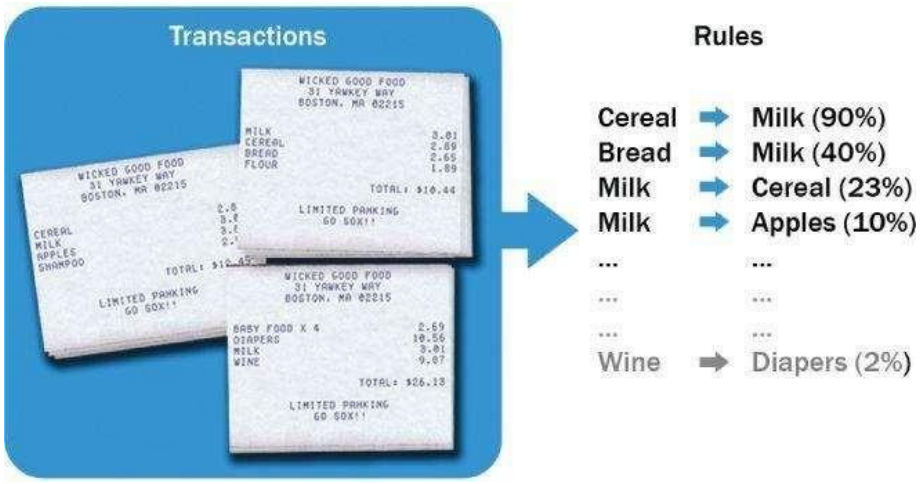


Figure 5.1 The general logic behind association rules

Each of the uncovered rules is in the form X → Y, meaning that when item X is observed, item Y is also observed. In this case, the left-hand side (LHS) of the rule is X, and the right-hand side (RHS) of the rule is Y.

Using association rules, patterns can be discovered from the data that allow the association rule

algorithms to disclose rules of related product purchases. The uncovered rules are listed on the right side of Figure 5.1. The first three rules suggest that when cereal is purchased, 90% of the time milk is purchased also. When bread is purchased, 40% of the time milk is purchased also. When milk is purchased, 23% of the time cereal is also purchased.

In the example of a retail store, association rules are used over transactions that consist of one or more items. In fact, because of their popularity in mining customer transactions, association rules are sometimes referred to as ***market basket analysis***. Each transaction can be viewed as the shopping basket of a customer that contains one or more items. This is also known as an itemset. The term ***itemset*** refers to a collection of items or individual entities that contain some kind of relationship. This could be a set of retail items purchased together in one transaction, a set of hyperlinks clicked on by one user in a single session, or a set of tasks done in one day. An itemset containing ***k*** items is called a ***k-itemset***. This chapter uses curly braces like $\{item\ 1, item\ 2, \ldots\ item\ k\}$ to denote a k- itemset. Computation of the association rules is typically based on itemsets.

The research of association rules started as early as the 1960s. Early research by Hájek et al. introduced many of the key concepts and approaches of association rule learning, but it focused on the mathematical representation rather than the algorithm. The framework of association rule learning was brought into the database community by Agrawal et al. in the early 1990s for discovering regularities between products in a large database of customer transactions recorded by point-of-sale systems in supermarkets. In later years, it expanded to web contexts, such as mining path traversal patterns and usage patterns to facilitate organization of web pages.

This chapter chooses Apriori as the main focus of the discussion of association rules. Apriori is one of the earliest and the most fundamental algorithms for generating association rules. It pioneered the use of support for pruning the itemsets and controlling the exponential growth of candidate itemsets. Shorter candidate itemsets, which are known to be frequent itemsets, are combined and pruned to generate longer frequent itemsets. This approach eliminates the need for all possible itemsets to be enumerated within the algorithm, since the number of all possible itemsets can become exponentially large.

One major component of Apriori is support. Given an itemset L, the ***support*** of L is the percentage of transactions that contain L. For example, if 80% of all transactions contain itemset {bread}, then the support of {bread} is 0.8. Similarly, if 60% of all transactions contain itemset {bread,butter}, then the support of {bread,butter} is 0.6.

A ***frequent itemset*** has items that appear together often enough. The term ―often enough‖ is formally defined with a ***minimum support*** criterion. If the minimum support is set at 0.5, any itemset can be considered a frequent itemset if at least 50% of the transactions contain this itemset. In other words, the support of a frequent itemset should be greater than or equal to the minimum support. For the previous example, both {bread} and {bread,butter} are considered frequent itemsets at the minimum support 0.5. If the minimum support is 0.7, only {bread} is considered a frequent itemset.

If an itemset is considered frequent, then any subset of the frequent itemset must also be frequent. This is referred to as the ***Apriori property*** (or ***downward closure property***). For example, if 60% of the transactions contain {bread,jam}, then at least 60% of all the transactions will contain {bread} or {jam}. In other words, when the support of {bread,jam} is 0.6, the support of {bread} or {jam} is at least 0.6. Figure 5.2 illustrates how the Apriori property works. If

itemset {B,C,D} is frequent, then all the subsets of this itemset, shaded, must also be frequent itemsets. The Apriori property provides the basis for the Apriori algorithm.

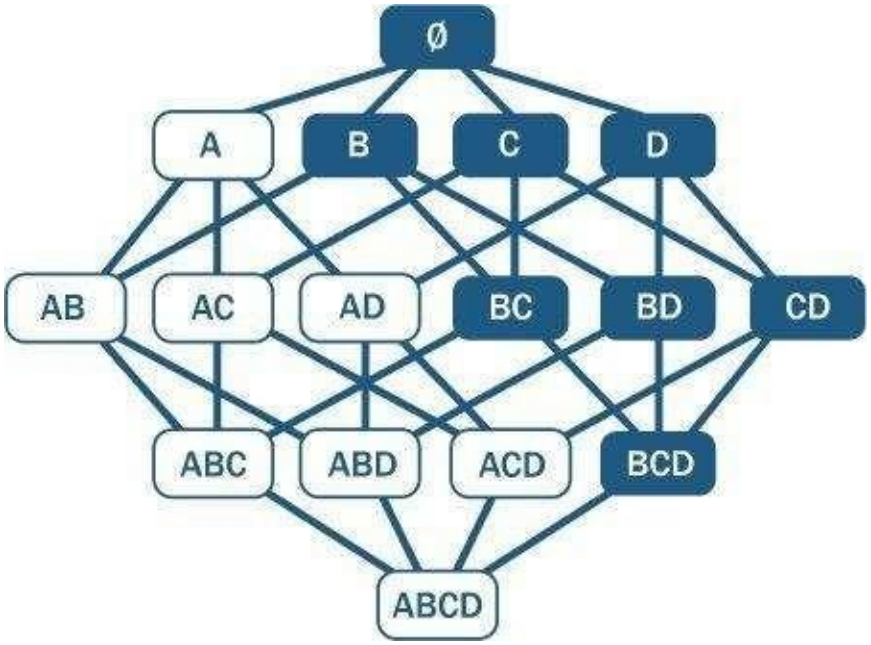


Figure 5.2 Itemset {A,B,C,D} and its subsets

5.2 Apriori Algorithm

The Apriori algorithm takes a bottom-up iterative approach to uncovering the frequent itemsets by first determining all the possible items (or 1-itemsets, for example {bread}, {eggs}, {milk}, …) and then identifying which among them are frequent.

Assuming the minimum support threshold (or the minimum support criterion) is set at 0.5, the algorithm identifies and retains those itemsets that appear in at least 50% of all transactions and discards (or ‖prunes away‖) the itemsets that have a support less than 0.5 or appear in fewer than 50% of the transactions. The word ***prune*** is used like it would be in gardening, where unwanted branches of a bush are clipped away.

In the next iteration of the Apriori algorithm, the identified frequent 1-itemsets are paired into 2-itemsets (for example, {bread,eggs}, {bread,milk}, {eggs,milk}, …) and again evaluated to identify the frequent 2-itemsets among them.

At each iteration, the algorithm checks whether the support criterion can be met; if it can, the algorithm grows the itemset, repeating the process until it runs out of support or until the itemsets reach a predefined length. The Apriori algorithm [5] is given next. Let variable C_k be the set of candidate *k*-itemsets and variable L_k be the set of *k*-itemsets that satisfy the minimum support. Given a transaction database , a minimum support threshold , and an optional parameter indicating the maximum length an itemset could reach, Apriori iteratively computes frequent itemsets L_{k+1} based on L_k.

```
1 Apriori (D, δ, N)
2 k←1
3 L_k ← {1-itemsets that satisfy minimum support δ}
4 while L_k ≠∅
5  if ∄N∨(∃N∧k<N)
6  C_{k+1} ← candidate itemsets generated from L_k
7  for each transaction t in database D do
8   increment the counts of C_{k+1} contained in t
9  L_{k+1} ← candidates in C_{k+1} that satisfy minimum support δ
10  k←k+1
11 return ∪_k L_k
```

The first step of the Apriori algorithm is to identify the frequent itemsets by starting with each item in the transactions that meets the predefined minimum support threshold . These itemsets are 1-itemsets denoted as L_1, as each 1-itemset contains only one item. Next, the algorithm grows the itemsets by joining L_1 onto itself to form new, grown 2-itemsets denoted as $L_2$and determines the support of each 2-itemset in L_2. Those itemsets that do not meet the minimum support threshold are pruned away. The growing and pruning process is repeated until no itemsets meet the minimum support threshold. Optionally, a threshold can be set up to specify the maximum number of items the itemset can reach or the maximum number of iterations of the algorithm. Once completed, output of the Apriori algorithm is the collection of all the frequent *k*-itemsets.

Next, a collection of candidate rules is formed based on the frequent itemsets uncovered in the iterative process described earlier. For example, a frequent itemset `{milk,eggs}` may suggest candidate rules `{milk}→{eggs}` and `{eggs}→{milk}`.

5.3 Evaluation of Candidate Rules

Frequent itemsets from the previous section can form candidate rules such as X implies Y (X → Y). This section discusses how measures such as confidence, lift, and leverage can help evaluate the appropriateness of these candidate rules.

Confidence is defined as the measure of certainty or trustworthiness associated with each discovered rule. Mathematically, confidence is the percent of transactions that contain both X and Y out of all the transactions that contain X (see Equation 5.1).

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Support}(X \wedge Y)}{\text{Support}(X)} \quad \textbf{5.1}$$

For example, if `{bread,eggs,milk}` has a support of 0.15 and `{bread,eggs}` also has a support of 0.15, the confidence of rule `{bread,eggs}→{milk}` is 1, which means 100% of the time a customer buys bread and eggs, milk is bought as well. The rule is therefore correct for 100% of the transactions containing bread and eggs.

A relationship may be thought of as interesting when the algorithm identifies the relationship with a measure of confidence greater than or equal to a predefined threshold. This predefined threshold is called the ***minimum confidence***. A higher confidence indicates that the rule (X → Y) is more interesting or more trustworthy, based on the sample dataset.

So far, this chapter has talked about two common measures that the Apriori algorithm uses: support and confidence. All the rules can be ranked based on these two measures to filter out the

uninteresting rules and retain the interesting ones.

Even though confidence can identify the interesting rules from all the candidate rules, it comes with a problem. Given rules in the form of X → Y, confidence considers only the antecedent (X) and the co-occurrence of X and Y; it does not take the consequent of the rule (Y) into concern. Therefore, confidence cannot tell if a rule contains true implication of the relationship or if the rule is purely coincidental. X and Y can be statistically independent yet still receive a high confidence score. Other measures such as lift and leverage are designed to address this issue.

Lift measures how many times more often X and Y occur together than expected if they are statistically independent of each other. Lift is a measure of how X and Y are really related rather than coincidentally happening together (see Equation 5.2).

$$\mathbf{5.2}\quad Lift(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X) * Support(Y)}$$

Lift is 1 if X and Y are statistically independent of each other. In contrast, a lift of X → Y greater than 1 indicates that there is some usefulness to the rule. A larger value of lift suggests a greater strength of the association between X and Y.

Assuming 1,000 transactions, with {milk,eggs} appearing in 300 of them, {milk} appearing in 500, and {eggs} appearing in 400, then $Lift(milk \rightarrow eggs) = 0.3/(0.5*0.4) = 1.5$. If {bread} appears in 400 transactions and {milk,bread} appears in 400, then $Lift(milk \rightarrow bread) = 0.4/(0.5*0.4) = 2$. Therefore it can be concluded that milk and bread have a stronger association than milk and eggs.

Leverage is a similar notion, but instead of using a ratio, leverage uses the difference (see Equation 5.3). Leverage measures the difference in the probability of X and Y appearing together in the dataset compared to what would be expected if X and Y were statistically independent of each other.

$$\mathbf{5.3}\quad Leverage(X \rightarrow Y) = Support(X \wedge Y) - Support(X) * Support(Y)$$

In theory, leverage is 0 when X and Y are statistically independent of each other. If X and Y have some kind of relationship, the leverage would be greater than zero. A larger leverage value indicates a stronger relationship between X and Y. For the previous example, $Leverage(milk \rightarrow eggs) = 0.3 - (0.5*0.4) = 0.1$ and $Leverage(milk \rightarrow bread) = 0.4 - (0.5*0.4) = 0.2$. It again confirms that milk and bread have a stronger association than milk and eggs.

Confidence is able to identify trustworthy rules, but it cannot tell whether a rule is coincidental. A high-confidence rule can sometimes be misleading because confidence does not consider support of the itemset in the rule consequent. Measures such as lift and leverage not only ensure interesting rules are identified but also filter out the coincidental rules.

This chapter has discussed four measures of significance and interestingness for association rules: support, confidence, lift, and leverage. These measures ensure the discovery of interesting and strong rules from sample datasets. Besides these four rules, there are other alternative measures, such as correlation, collective strength, conviction , and coverage. Refer to the Bibliography to learn how these measures work.

5.4 Applications of Association Rules

The term ***market basket analysis*** refers to a specific implementation of association rules mining that many companies use for a variety of purposes, including these:

- Broad-scale approaches to better merchandising—what products should be included in or excluded from the inventory each month
- Cross-merchandising between products and high-margin or high-ticket items
- Physical or logical placement of product within related categories of products
- Promotional programs— multiple product purchase incentives managed through a loyalty card program

Besides market basket analysis, association rules are commonly used for recommender systems and clickstream analysis.

Many online service providers such as Amazon and Netflix use recommender systems. Recommender systems can use association rules to discover related products or identify customers who have similar interests. For example, association rules may suggest that those customers who have bought product A have also bought product B, or those customers who have bought products A, B, and C are more similar to this customer. These findings provide opportunities for retailers to cross-sell their products.

Clickstream analysis refers to the analytics on data related to web browsing and user clicks, which is stored on the client or the server side. Web usage log files generated on web servers contain huge amounts of information, and association rules can potentially give useful knowledge to web usage data analysts. For example, association rules may suggest that website visitors who land on page X click on links A, B, and C much more often than links D, E, and F. This observation provides valuable insight on how to better personalize and recommend the content to site visitors.

5.5 An Example: Transactions in a Grocery Store

An example illustrates the application of the Apriori algorithm to a relatively simple case that generalizes to those used in practice. Using R and the `arules` and `arulesViz` packages, this example shows how to use the Apriori algorithm to generate frequent itemsets and rules and to evaluate and visualize the rules.

The following commands install these two packages and import them into the current R workspace:

```
install.packages("arules") install.packages("arulesViz") library("arules") library("arulesViz")
```

5.5.1 The Groceries Dataset

The example uses the *Groceries* dataset from the R `arules` package. The *Groceries* dataset is collected from 30 days of real-world point-of-sale transactions of a grocery store. The dataset contains 9,835 transactions, and the items are aggregated into 169 categories.

```
data(Groceries) Groceries
transactions in sparse format with
```

```
9835 transactions (rows) and
169 items (columns)
```

The summary shows that the most frequent items in the dataset include items such as whole milk, other vegetables, rolls/buns, soda, and yogurt. These items are purchased more often than the others.

```
summary(Groceries)
transactions as itemMatrix in sparse format with
9835 rows (elements/itemsets/transactions) and
169 columns (items) and a density of 0.02609146 most frequent items:
  whole milk other vegetables  rolls/buns     soda
   2513    1903    1809    1715
   yogurt   (Other)
   1372   34055
element (itemset/transaction) length distribution:
sizes
1  2  3  4  5  6  7  8  9 10 11  12 13  14
2159 1643 1299  1005 855  645  545  438  350  246  182  117  78  77
15 16  17  18 19  20  21  22  23 24  26  27  28  29
55 46  29  14 14  9  11  4  6  1 1  1  1  3
32
1
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 2.000 3.000 4.409 6.000 32.000
includes extended item information - examples:
  labels level2   level1
1 frankfurter sausage meet and sausage
2  sausage sausage meet and sausage
3 liver loaf sausage meet and sausage
```

The class of the dataset is **transactions**, as defined by the **arules** package. The transactions class contains three slots:

- **transactionInfo:** A data frame with vectors of the same length as the number of transactions
- **itemInfo:** A data frame to store item labels
- **data:** A binary incidence matrix that indicates which item labels appear in every transaction

```
class(Groceries) [1] –transactions‖ attr(,–package‖) [1] —arules‖
```

For the *Groceries* dataset, the **transactionInfo** is not being used. Enter **Groceries@itemInfo** to display all 169 grocery labels as well as their categories. The following command displays only the first 20 grocery labels. Each grocery label is mapped to two levels of categories—**level2** and **level1**—where **level1** is a superset of **level2**. For example, grocery label **sausage** belongs to the **sausage** category in **level2**, and it is part of the **meat** and **sausage** category in **level1**. (Note that **–meet‖** in **level1** is a typo in the dataset.)

```
Groceries@itemInfo[1:20,]
    labels level2 level1
```

```
1  frankfurter sausage  meet and sausage
2   sausage sausage  meet and sausage
3   liver loaf sausage  meet and sausage
4    ham sausage  meet and sausage
5    meat sausage  meet and sausage
6 finished products sausage  meet and sausage
7 organic sausage sausage  meet and sausage
8   chicken poultry  meet and sausage
9    turkey poultry  meet and sausage
10    pork  pork  meet and sausage
11    beef  beef  meet and sausage
12 hamburger meat  beef  meet and sausage
13    fish  fish  meet and sausage
14  citrus  fruit  fruit fruit and vegetables
15 tropical  fruit  fruit fruit and vegetables
16   pip  fruit  fruit fruit and vegetables
17   grapes  fruit fruit and vegetables
18   berries  fruit fruit and vegetables
19  nuts/prunes  fruit fruit and vegetables
20 root vegetables vegetables fruit and vegetables
```

The following code displays the 10th to 20th transactions of the Groceries dataset. The [10:20] can be changed to [1:9835] to display all the transactions.

```
apply(Groceries@data[,10:20], 2,
  function(r) paste(Groceries@itemInfo[r,-labels‖], collapse=‖, -)
  )
```

Each row in the output shows a transaction that includes one or more products, and each transaction corresponds to everything in a customer‘s shopping cart. For example, in the first transaction, a customer has purchased whole milk and cereals.

```
[1] -whole milk, cereals‖
[2] -tropical fruit, other vegetables, white bread, bottled water,
chocolate‖
[3] —citrus fruit, tropical fruit, whole milk, butter, curd, yogurt, flour,
bottled water, dishes‖
[4] -beef‖
[5] -frankfurter, rolls/buns, soda‖ [6] —chicken, tropical fruit‖
[7] -butter, sugar, fruit/vegetable juice, newspapers‖ [8] -fruit/vegetable
juice‖
[9] -packaged fruit/vegetables‖ [10] —chocolate‖
[11] -specialty bar‖
```

The next section shows how to generate frequent itemsets from the *Groceries* dataset.

5.5.2 Frequent Itemset Generation

The apriori() function from the arule package implements the Apriori algorithm to create frequent itemsets. Note that, by default, theapriori() function executes all the iterations at once. However, to illustrate how the Apriori algorithm works, the code examples in this section manually set the parameters of the apriori() function to simulate each iteration of the algorithm.

Assume that the minimum support threshold is set to 0.02 based on management discretion. Because the dataset contains 9,853 transactions, an itemset should appear at least 198 times to be considered a frequent itemset. The first iteration of the Apriori algorithm computes the support of each product in the dataset and retains those products that satisfy the minimum support. The following code identifies 59 frequent 1-itemsets that satisfy the minimum support. The parameters of `apriori()` specify the minimum and maximum lengths of the itemsets, the minimum support threshold, and the target indicating the type of association mined.

```
itemsets <- apriori(Groceries, parameter=list(minlen=1, maxlen=1,
      support=0.02, target="frequent itemsets"))
parameter specification:
confidence minval smax arem aval originalSupport support minlen
  0.8 0.1 1 none FALSE    TRUE 0.02  1 maxlen    target ext
  1 frequent itemsets FALSE
algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm version 4.21
(2004.05.09)  (c) 1996-2004 Christian Borgelt set item appearances ...[0
item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s]. sorting
and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s]. checking subsets of size 1 done
[0.00s]. writing ... [59 set(s)] done [0.00s]. creating S4 object ... done
[0.00s].
```

The summary of the itemsets shows that the support of 1-itemsets ranges from 0.02105 to 0.25552. Because the maximum support of the 1-itemsets in the dataset is only 0.25552, to enable the discovery of interesting rules, the minimum support threshold should not be set too close to that number.

```
summary(itemsets) set of 59 itemsets most frequent items:
frankfurter  sausage   ham  meat  chicken
   1   1   1   1   1 (Other)
   54
element (itemset/transaction) length distribution:sizes
1
59
Min. 1st Qu. Median Mean 3rd Qu. Max.
  1  1  1  1  1  1
summary of quality measures:
support
Min. :0.02105
1st Qu.:0.03015
Median :0.04809
Mean :0.06200
3rd Qu.:0.07666
Max. :0.25552
includes transaction ID lists: FALSE
mining info:
```

```
data ntransactions support confidence
Groceries    9835 0.02    1
```

The following code uses the **inspect()** function to display the top 10 frequent 1-itemsets sorted by their support. Of all the transaction records, the 59 1-itemsets such as **{whole milk}**, **{other vegetables}**, **{rolls/buns}**, **{soda}**, and **{yogurt}** all satisfy the minimum support. Therefore, they are called frequent 1-itemsets.

```
inspect(head(sort(itemsets, by = "support"), 10))
items        support
1 {whole milk}    0.25551601
2 {other vegetables}  0.19349263
3 {rolls/buns}    0.18393493
4 {soda}      0.17437722
5 {yogurt}      0.13950178
6 {bottled water}    0.11052364
7 {root vegetables}  0.10899847
8 {tropical fruit}   0.10493137
9 {shopping bags}    0.09852567
10 {sausage}      0.09395018
```

In the next iteration, the list of frequent 1-itemsets is joined onto itself to form all possible candidate 2-itemsets. For example, 1-itemsets **{whole milk}** and **{soda}** would be joined to become a 2-itemset **{whole milk,soda}**. The algorithm computes the support of each candidate 2-itemset and retains those that satisfy the minimum support. The output that follows shows that 61 frequent 2-itemsets have been identified.

```
itemsets <- apriori(Groceries, parameter=list(minlen=2, maxlen=2,
      support=0.02, target="frequent itemsets"))

parameter specification:
confidence minval smax arem aval originalSupport support minlen
  0.8 0.1 1 none FALSE    TRUE 0.02  2 maxlen    target ext
  2 frequent itemsets FALSE
algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm version 4.21
(2004.05.09)  (c) 1996-2004 Christian Borgelt set item appearances ...[0
item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s]. sorting
and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s]. checking subsets of size 1 2 done
[0.00s]. writing ... [61 set(s)] done [0.00s]. creating S4 object ... done
[0.00s].
```

The summary of the itemsets shows that the support of 2-itemsets ranges from 0.02003 to 0.07483.

```
summary(itemsets) set of 61 itemsets most frequent items:
  whole milk other vegetables    yogurt  rolls/buns
    25     17     9     9
```

```
   soda   (Other)
    9    53
element (itemset/transaction) length distribution:sizes
2
61
Min. 1st Qu. Median Mean 3rd Qu. Max.
  2  2  2  2  2  2
summary of quality measures:
support
Min. :0.02003
1st Qu.:0.02227
Median :0.02613
Mean :0.02951
3rd Qu.:0.03223
Max. :0.07483
includes transaction ID lists: FALSE
mining info:
  data ntransactions support confidence
Groceries   9835 0.02   1
```

The top 10 most frequent 2-itemsets are displayed next, sorted by their support. Notice that whole milk appears six times in the top 10 2-itemsets ranked by support. As seen earlier, {whole milk} has the highest support among all the 1-itemsets. These top 10 2- itemsets with the highest support may not be interesting; this highlights the limitations of using support alone.

```
inspect(head(sort(itemsets, by ="support"),10))
items     support
1 {other vegetables, whole milk}  0.07483477
2 {whole milk, rolls/buns}  0.05663447
3 {whole milk, yogurt}   0.05602440
4 {root vegetables, whole milk}  0.04890696
5 {root vegetables,
other vegetables} 0.04738180
6 {other vegetables, yogurt}   0.04341637
7 {other vegetables, rolls/buns}  0.04260295
8 {tropical fruit,
whole milk}  0.04229792
9 {whole milk,
soda}    0.04006101
10 {rolls/buns, soda}    0.03833249
```

Next, the list of frequent 2-itemsets is joined onto itself to form candidate 3-itemsets. For example {other vegetables,whole milk} and {whole milk,rolls/buns} would be joined as {other vegetables,whole milk,rolls/buns}. The algorithm retains those itemsets that satisfy the minimum support. The following output shows that only two frequent 3-itemsets have been identified.

```
itemsets <- apriori(Groceries, parameter=list(minlen=3, maxlen=3,
      support=0.02, target="frequent itemsets"))
parameter specification:
confidence minval smax arem aval originalSupport support minlen
```

```
  0.8 0.1 1 none FALSE    TRUE 0.02   3 maxlen    target ext
  3 frequent itemsets FALSE
algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm version 4.21
(2004.05.09)  (c) 1996-2004 Christian Borgelt set item appearances …[0
item(s)] done [0.00s].
set transactions …[169 item(s), 9835 transaction(s)] done [0.00s]. sorting
and recoding items … [59 item(s)] done [0.00s].
creating transaction tree … done [0.00s]. checking subsets of size 1 2 3
done [0.00s]. writing … [2 set(s)] done [0.00s].
creating S4 object … done [0.00s].
```

The 3-itemsets are displayed next:

```
inspect(sort(itemsets, by ="support"))
items      support
1 {root vegetables, other vegetables,
whole milk}  0.02318251
2 {other vegetables, whole milk,
yogurt}   0.02226741
```

In the next iteration, there is only one candidate 4-itemset {root vegetables,other vegetables,whole milk,yogurt}, and its support is below 0.02. No frequent 4-itemsets have been found, and the algorithm converges.

```
itemsets <- apriori(Groceries, parameter=list(minlen=4, maxlen=4,
      support=0.02, target="frequent itemsets"))
parameter specification:
confidence minval smax arem aval originalSupport support minlen
  0.8 0.1 1 none FALSE    TRUE 0.02   4 maxlen    target ext
  4 frequent itemsets FALSE
algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm version 4.21
(2004.05.09)  (c) 1996-2004 Christian Borgelt set item appearances …[0
item(s)] done [0.00s].
set transactions …[169 item(s), 9835 transaction(s)] done [0.00s]. sorting
and recoding items … [59 item(s)] done [0.00s].
creating transaction tree … done [0.00s]. checking subsets of size 1 2 3
done [0.00s]. writing … [0 set(s)] done [0.00s].
creating S4 object … done [0.00s].
```

The previous steps simulate the Apriori algorithm at each iteration. For the *Groceries* dataset, the iterations run out of support when $k = 4$. Therefore, the frequent itemsets contain 59 frequent 1-itemsets, 61 frequent 2-itemsets, and 2 frequent 3-itemsets.

When the `maxlen` parameter is not set, the algorithm continues each iteration until it runs out of support or until k reaches the default `maxlen`=10. As shown in the code output that follows, 122 frequent itemsets have been identified. This matches the total number of 59 frequent 1-itemsets, 61 frequent 2-itemsets, and 2 frequent 3-itemsets.

```
itemsets <- apriori(Groceries, parameter=list(minlen=1, support=0.02,
          target="frequent itemsets"))
parameter specification:
confidence minval smax arem aval originalSupport support minlen
  0.8 0.1 1 none FALSE    TRUE 0.02  1 maxlen    target ext
  10  frequent itemsets FALSE
algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm version 4.21
(2004.05.09)  (c) 1996-2004 Christian Borgelt set item appearances ...[0
item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s]. sorting
and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s]. checking subsets of size 1 2 3
done [0.00s]. writing ... [122 set(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

Note that the results are assessed based on the specific business context of the exercise using the specific dataset. If the dataset changes or a different minimum support threshold is chosen, the Apriori algorithm must run each iteration again to retrieve the updated frequent itemsets.

5.5.3 Rule Generation and Visualization

The `apriori()` function can also be used to generate rules. Assume that the minimum support threshold is now set to a lower value 0.001, and the minimum confidence threshold is set to 0.6. A lower minimum support threshold allows more rules to show up. The following code creates 2,918 rules from all the transactions in the *Groceries* dataset that satisfy both the minimum support and the minimum confidence.

```
rules <- apriori(Groceries, parameter=list(support=0.001,
      confidence=0.6, target = "rules"))
parameter specification:
confidence minval smax arem aval originalSupport support minlen
  0.6 0.1 1 none FALSE    TRUE 0.001  1 maxlen target ext
  10 rules FALSE
algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm version 4.21
(2004.05.09)  (c) 1996-2004 Christian Borgelt set item appearances ...[0
item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s]. sorting
and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.01s]. writing ... [2918 rule(s)]
done [0.00s].
creating S4 object ... done [0.01s].
```

The summary of the rules shows the number of rules and ranges of the support, confidence, and lift.

```
summary(rules)
set of 2918 rules
rule length distribution (lhs + rhs):sizes
2 3 4 5 6
3 490 1765 626 34
Min. 1st Qu. Median Mean 3rd Qu. Max.
2.000 4.000 4.000 4.068 4.000 6.000 summary of quality measures:
support confidence lift
Min. :0.001017 Min. :0.6000 Min. : 2.348
1st Qu.:0.001118 1st Qu.:0.6316 1st Qu.: 2.668
Median :0.001220 Median :0.6818 Median : 3.168
Mean :0.001480 Mean :0.7028 Mean : 3.450
3rd Qu.:0.001525 3rd Qu.:0.7500 3rd Qu.: 3.692
Max. :0.009354 Max. :1.0000 Max. :18.996 mining info:
  data ntransactions support confidence
Groceries 9835 0.001 0.6
```

Enter **plot(rules)** to display the scatterplot of the 2,918 rules (Figure 5.3), where the horizontal axis is the support, the vertical axis is the confidence, and the shading is the lift. The scatterplot shows that, of the 2,918 rules generated from the *Groceries* dataset, the highest lift occurs at a low support and a low confidence.

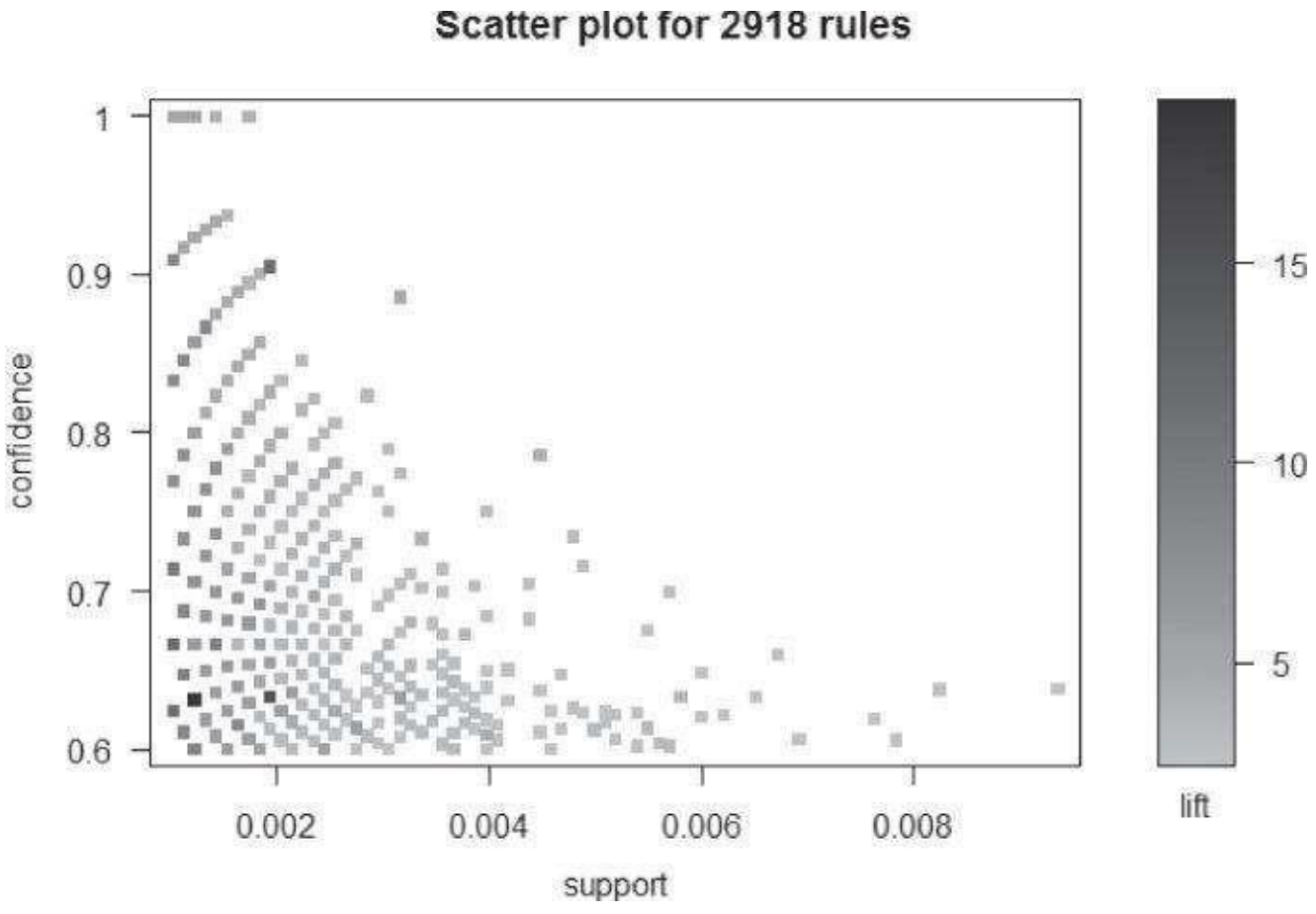


Figure 5.3 Scatterplot of the 2,918 rules with minimum support 0.001 and minimum confidence 0.6

Entering **plot(rules@quality)** displays a scatterplot matrix (Figure 5.4) to compare the support,

confidence, and lift of the 2,918 rules.

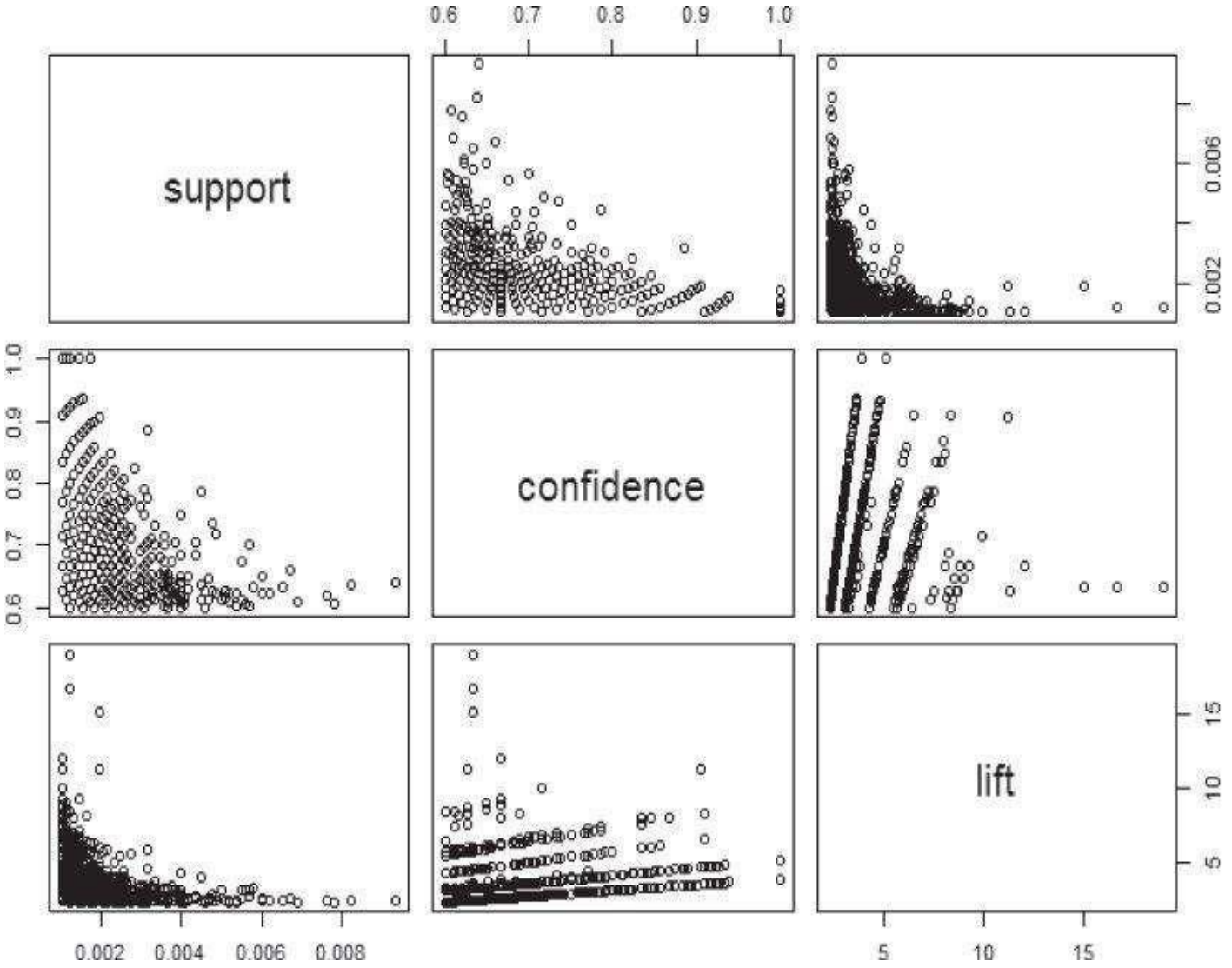


Figure 5.4 Scatterplot matrix on the support, confidence, and lift of the 2,918 rules

Figure 5.4 shows that lift is proportional to confidence and illustrates several linear groupings. As indicated by Equation 5.2 and Equation 5.3, $Lift = Confidence / Support(Y)$. Therefore, when the support of Y remains the same, lift is proportional to confidence, and the slope of the linear trend is the reciprocal of $Support(Y)$. The following code shows that, of the 2,918 rules, there are only 18 different values for $\frac{1}{Support(Y)}$, and the majority occurs at slopes 3.91, 5.17, 7.17, 9.17, and 9.53. This matches the slopes shown in the third row and second column of Figure 5.4, where the x-axis is the confidence and the y-axis is the lift.

```
# compute the 1/Support(Y)
slope <- sort(round(rules@quality$lift / rules@quality$confidence, 2))
# Display the number of times each slope appears in the dataset
unlist(lapply(split(slope,f=slope),length))
3.91 5.17 5.44 5.73 7.17 9.05 9.17 9.53 10.64 12.08
1585  940   12    7  188    1  102   55    1    4
12.42 13.22 13.83 13.95 18.05 23.76 26.44 30.08
1    5    2    9    3    1    1    1
```

The inspect() function can display content of the rules generated previously. The following code shows the top ten rules sorted by the lift. Rule {Instant food products,soda}→{hamburger meat} has the highest lift of 18.995654.

```
inspect(head(sort(rules, by="lift"), 10))
```

```
lhs      rhs
support confidence lift
1 {Instant food products, soda}      => {hamburger meat}
0.001220132 0.6315789 18.995654
2 {soda,
popcorn}    => {salty snack}
0.001220132 0.6315789 16.697793
3 {ham,
processed cheese}  => {white bread}
0.001931876 0.6333333 15.045491
4 {tropical fruit, other vegetables, yogurt,
white bread}   => {butter}
0.001016777 0.6666667 12.030581
5 {hamburger meat, yogurt,
whipped/sour cream} => {butter}
0.001016777 0.6250000 11.278670
6 {tropical fruit, other vegetables, whole milk,
yogurt,
domestic eggs}   => {butter}
0.001016777 0.6250000 11.278670
7 {liquor,
red/blush wine}  => {bottled beer}
0.001931876 0.9047619 11.235269
8 {other vegetables, butter,
sugar}     => {whipped/sour cream}
0.001016777 0.7142857 9.964539
9 {whole milk, butter,
hard cheese}   => {whipped/sour cream}
0.001423488 0.6666667 9.300236
10 {tropical fruit, other vegetables, butter,
fruit/vegetable juice} => {whipped/sour cream}
0.001016777 0.6666667 9.300236
```

The following code fetches a total of 127 rules whose confidence is above 0.9:

```
confidentRules <- rules[quality(rules)$confidence > 0.9]
confidentRules set of 127 rules
```

The next command produces a matrix-based visualization (Figure 5.5) of the LHS versus the RHS of the rules. The legend on the right is a color matrix indicating the lift and the confidence to which each square in the main matrix corresponds.

```
plot(confidentRules, method="matrix", measure=c("lift", "confidence"),
  control=list(reorder=TRUE))
```

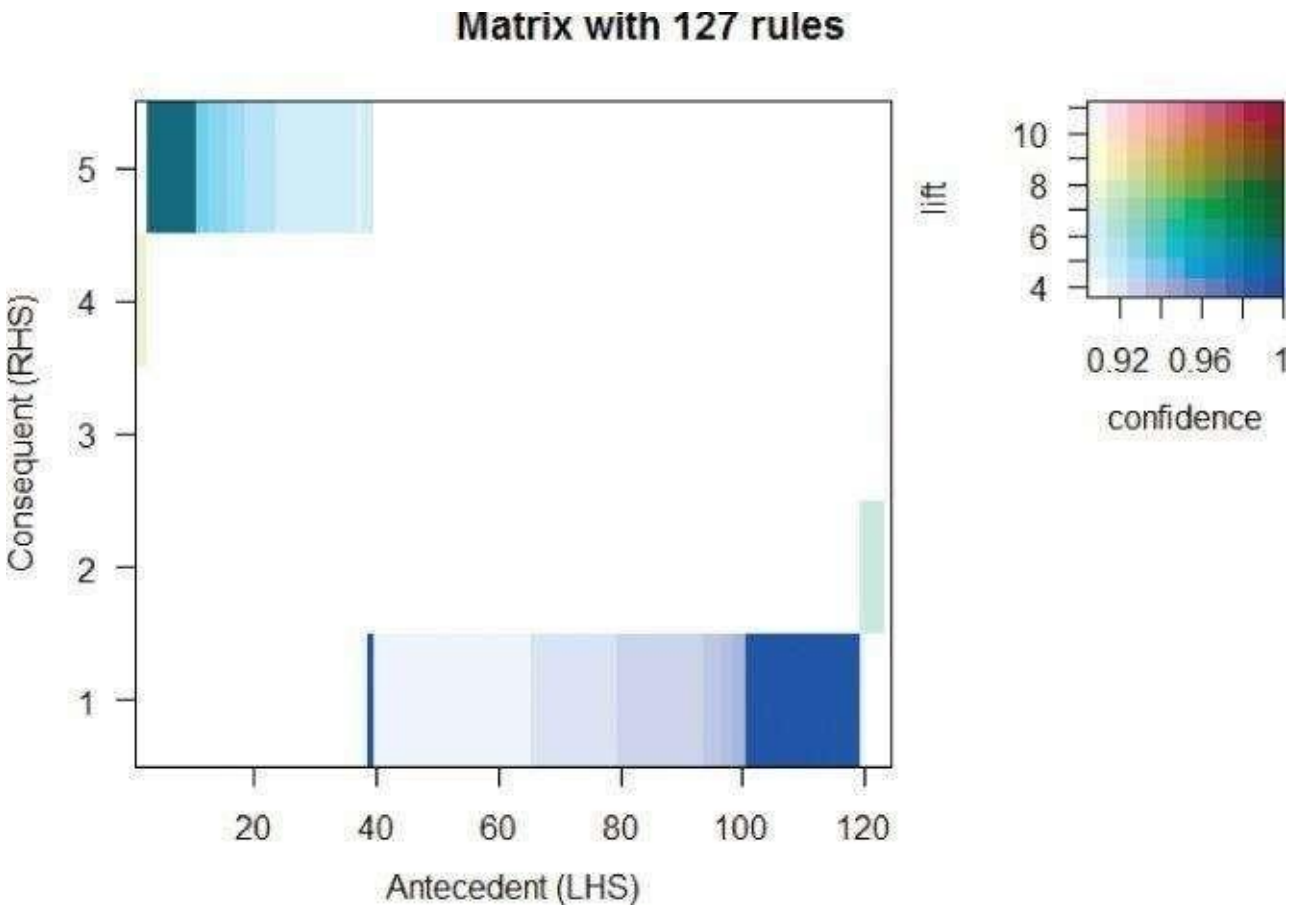


Figure 5.5 Matrix-based visualization of LHS and RHS, colored by lift and confidence

As the previous `plot()` command runs, the R console would simultaneously display a distinct list of the LHS and RHS from the 127 rules. A segment of the output is shown here:

```
Itemsets in Antecedent (LHS)
[1] {citrus fruit,other vegetables,soda,fruit/vegetable juice} [2]
{tropical fruit,other vegetables,whole milk,yogurt,oil}
[3] {tropical fruit,butter,whipped/sour cream,fruit/vegetable juice}
[4] {tropical fruit,grapes,whole milk,yogurt} [5] {ham,tropical
fruit,pip fruit,whole milk}
…
[124] {liquor,red/blush wine} Itemsets in Consequent (RHS)
[1] {whole milk} {yogurt} {root vegetables} [4] {bottled
beer} {other vegetables}
```

The following code provides a visualization of the top five rules with the highest lift. The plot is shown in Figure 5.6. In the graph, the arrow always points from an item on the LHS to an item on the RHS. For example, the arrows that connect ham, processed cheese, and white bread suggest rule `{ham,processed cheese}→{white bread}`. The legend on the top right of the graph shows that the size of a circle indicates the support of the rules ranging from 0.001 to 0.002. The color (or

shade) represents the lift, which ranges from
11.279 to 18.996. The rule with the highest lift is {Instant food products,soda} → {hamburger meat}.

```
highLiftRules <- head(sort(rules, by="lift"), 5)
plot(highLiftRules, method="graph", control=list(type="items"))
```

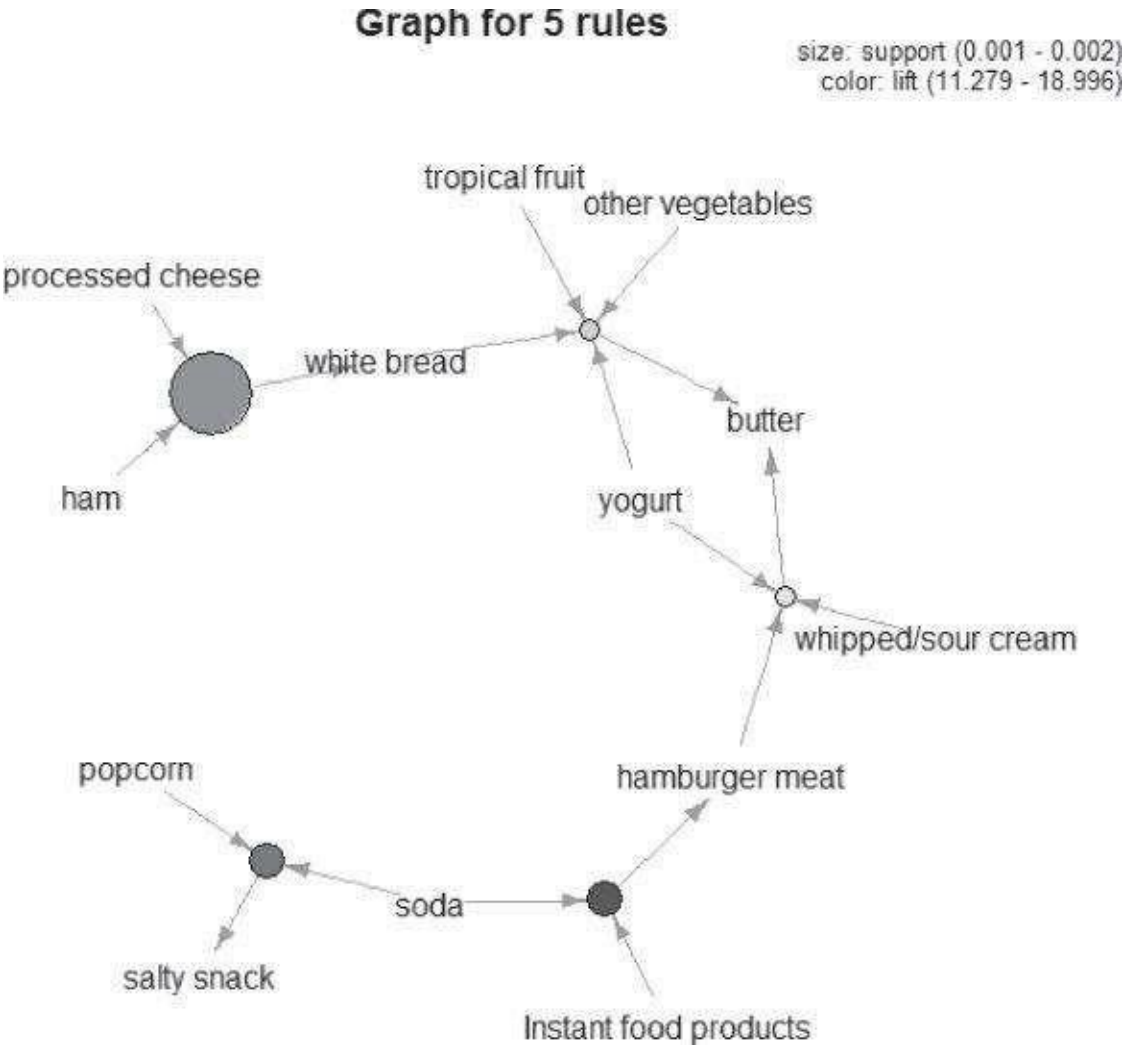


Figure 5.6 Graph visualization of the top five rules sorted by lift

5.6 Validation and Testing

After gathering the output rules, it may become necessary to use one or more methods to validate the results in the business context for the sample dataset. The first approach can be established through statistical measures such as confidence, lift, and leverage. Rules that involve mutually independent items or cover few transactions are considered uninteresting because they may capture spurious relationships.

As mentioned in Section 5.3, confidence measures the chance that X and Y appear together in

relation to the chance X appears. Confidence can be used to identify the interestingness of the rules.

Lift and leverage both compare the support of X and Y against their individual support. While mining data with association rules, some rules generated could be purely coincidental. For example, if 95% of customers buy X and 90% of customers buy Y, then X and Y would occur together at least 85% of the time, even if there is no relationship between the two. Measures like lift and leverage ensure that interesting rules are identified rather than coincidental ones.

Another set of criteria can be established through subjective arguments. Even with a high confidence, a rule may be considered subjectively uninteresting unless it reveals any unexpected profitable actions. For example, rules like {paper}→{pencil} may not be subjectively interesting or meaningful despite high support and confidence values. In contrast, a rule like {diaper}→{beer} that satisfies both minimum support and minimum confidence can be considered subjectively interesting because this rule is unexpected and may suggest a cross-sell opportunity for the retailer. This incorporation of subjective knowledge into the evaluation of rules can be a difficult task, and it requires collaboration with domain experts. As seen in Chapter 2, ―Data Analytics Lifecycle,‖ the domain experts may serve as the business users or the business intelligence analysts as part of the Data Science team. In Phase 5, the team can communicate the results and decide if it is appropriate to operationalize them.

5.7 Diagnostics

Although the Apriori algorithm is easy to understand and implement, some of the rules generated are uninteresting or practically useless. Additionally, some of the rules may be generated due to coincidental relationships between the variables. Measures like confidence, lift, and leverage should be used along with human insights to address this problem.

Another problem with association rules is that, in Phase 3 and 4 of the Data Analytics Lifecycle (Chapter 2), the team must specify the minimum support prior to the model execution, which may lead to too many or too few rules. In related research, a variant of the algorithm [13] can use a predefined target range for the number of rules so that the algorithm can adjust the minimum support accordingly.

Section 5.2 presented the Apriori algorithm, which is one of the earliest and the most fundamental algorithms for generating association rules. The Apriori algorithm reduces the computational workload by only examining itemsets that meet the specified minimum threshold. However, depending on the size of the dataset, the Apriori algorithm can be computationally expensive. For each level of support, the algorithm requires a scan of the entire database to obtain the result. Accordingly, as the database grows, it takes more time to compute in each run. Here are some approaches to improve Apriori‘s efficiency:

- **Partitioning:** Any itemset that is potentially frequent in a transaction database must be frequent in at least one of the partitions of the transaction database.

- **Sampling:** This extracts a subset of the data with a lower support threshold and uses the subset to perform association rule mining.
- **Transaction reduction:** A transaction that does not contain frequent ***k***-itemsets is useless in subsequent scans and therefore can be ignored.
- **Hash-based itemset counting:** If the corresponding hashing bucket count of a ***k***- itemset is below a certain threshold, the ***k***-itemset cannot be frequent.
- **Dynamic itemset counting:** Only add new candidate itemsets when all of their subsets are estimated to be frequent.