Pattern Recognition: Market Basket Analysis

KAHAROU BAWA BOUKARI

2022-07-15

Case study: identifying frequently purchased groceries with association rules

Market basket analysis is used behind the scenes for the recommendation systems used in many brick-and-mortar and online retailers. The learned association rules indicate the combinations of items that are often purchased together. Knowledge of these patterns provides insight into new ways a grocery chain might optimize the inventory, advertise promotions, or organize the physical layout of the store. For instance, if shoppers frequently purchase coffee or orange juice with a breakfast pastry, it may be possible to increase profit by relocating pastries closer to coffee and juice.

In this tutorial, we will perform a market basket analysis of transactional data from a grocery store. However, the techniques could be applied to many different types of problems, from movie recommendations, to dating sites, to finding dangerous interactions among medications. In doing so, we will see how the Apriori algorithm is able to efficiently evaluate a potentially massive set of association rules.

Step 1 – collecting data

Our market basket analysis will utilize the purchase data collected from one month of operation at a real-world grocery store. The data contains 9,835 transactions or about 327 transactions per day (roughly 30 transactions per hour in a 12-hour business day), suggesting that the retailer is not particularly large, nor is it particularly small. The typical grocery store offers a huge variety of items. There might be five brands of milk, a dozen different types of laundry detergent, and three brands of coffee. Given the moderate size of the retailer, we will assume that they are not terribly concerned with finding rules that apply only to a specific brand of milk or detergent. With this in mind, all brand names can be removed from the purchases. This reduces the number of groceries to a more manageable 169 types, using broad categories such as chicken, frozen meals, margarine, and soda.

Step 2 – exploring and preparing the data

Loading the data using the **read.csv()** function as we used to, R would happily comply and read the data into a matrix form as follows:

```
library(readr)
groceries<-read_csv("groceries.csv", col_names =F)
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## Rows: 9835 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): X1, X2, X3, X4
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(groceries)
```

```
## # A tibble: 6 x 4
##   X1               X2                   X3             X4
##   <chr>            <chr>                <chr>          <chr>
## 1 citrus fruit     semi-finished bread  margarine      ready soups
## 2 tropical fruit   yogurt               coffee         <NA>
## 3 whole milk       <NA>                 <NA>           <NA>
## 4 pip fruit        yogurt               cream cheese   meat spreads
## 5 other vegetables whole milk           condensed milk long life bakery product
## 6 whole milk       butter               yogurt         rice,abrasive cleaner
```

The first transaction included four items: citrus fruit, semi-finished bread, margarine, and ready soups. In comparison, the third transaction included only one item: whole milk.

Notice that R created four columns to store the items in the transactional data: X1, X2, X3, and X4. Although this may seem reasonable this, if we use the data in this form, we will encounter problems later may seem reasonable, R chose to create four variables because the first line had exactly four comma-separated values.

Data pre-processing – creating a sparse matrix for transaction data

The sparse matrix has a column (that is, feature) for every item that could possibly appear in someone's shopping bag. Since there are 169 different items in our grocery store data, our sparse matrix will contain 169 columns.

```
library(arules)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'arules'
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
groceries<-read.transactions("groceries.csv", sep = ",")
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   15   16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   29   14   14    9   11    4    6    1    1    1    1    3    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
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
```

The first block of information in the output above provides a summary of the sparse matrix we created. The output 9835 rows refers to the number of transactions, and the output 169 columns refers to the 169 different items that might appear in someone's grocery basket. Each cell in the matrix is 1 if the item was purchased for the corresponding transaction, or 0 otherwise.

The density value of 0.02609146 (2.6 percent) refers to the proportion of nonzero matrix cells. Since there are 9,835 * 169 = 1,662,115 positions in the matrix, we can calculate that a total of 1,662,115 * 0.02609146 = 43,367 items were purchased during the store's 30 days of operation (ignoring the fact that duplicates of the same items might have been purchased). With an additional step, we can determine that the average transaction contained 43,367 / 8,835 = 4.409 distinct grocery items. Of course, if we look a little further down the output, we'll see that the mean number of items per transaction has already been provided.

The next block of the summary() output lists the items that were most commonly found in the transactional data. Since 2,513 / 9,835 = 0.2555, we can determine that whole milk appeared in 25.6 percent of the transactions. The other vegetables, rolls/buns, soda, and yogurt round out the list of other common items, as shown above.

Finally, we are presented with a set of statistics about the size of the transactions. A total of 2,159 transactions contained only a single item, while one transaction had 32 items. The first quartile and median purchase sizes are two and three items, respectively, implying that 25 percent of the transactions contained two or fewer items and the transactions were split in half between those with less than three items and those with more. The mean of 4.409 items per transaction matches the value we calculated by hand.

```
inspect(groceries[1:5])
```

```
##     items
## [1] {citrus fruit,
##      margarine,
##      ready soups,
##      semi-finished bread}
## [2] {coffee,
##      tropical fruit,
##      yogurt}
## [3] {whole milk}
## [4] {cream cheese,
##      meat spreads,
##      pip fruit,
##      yogurt}
## [5] {condensed milk,
##      long life bakery product,
##      other vegetables,
##      whole milk}
```

```
itemFrequency(groceries[,1:3])
```

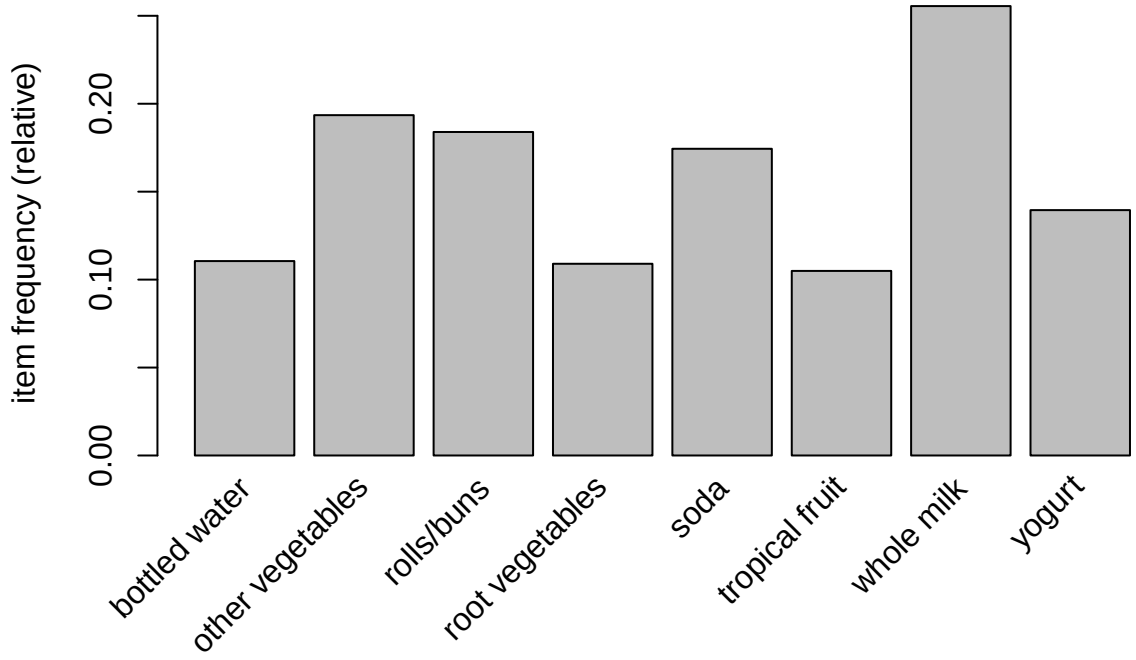
```
## abrasive cleaner artif. sweetener   baby cosmetics
##     0.0035587189     0.0032536858     0.0006100661
```

The proportion of transactions (the support level) for the first three items in the grocery data as shown above.

Visualizing item support – item frequency plots

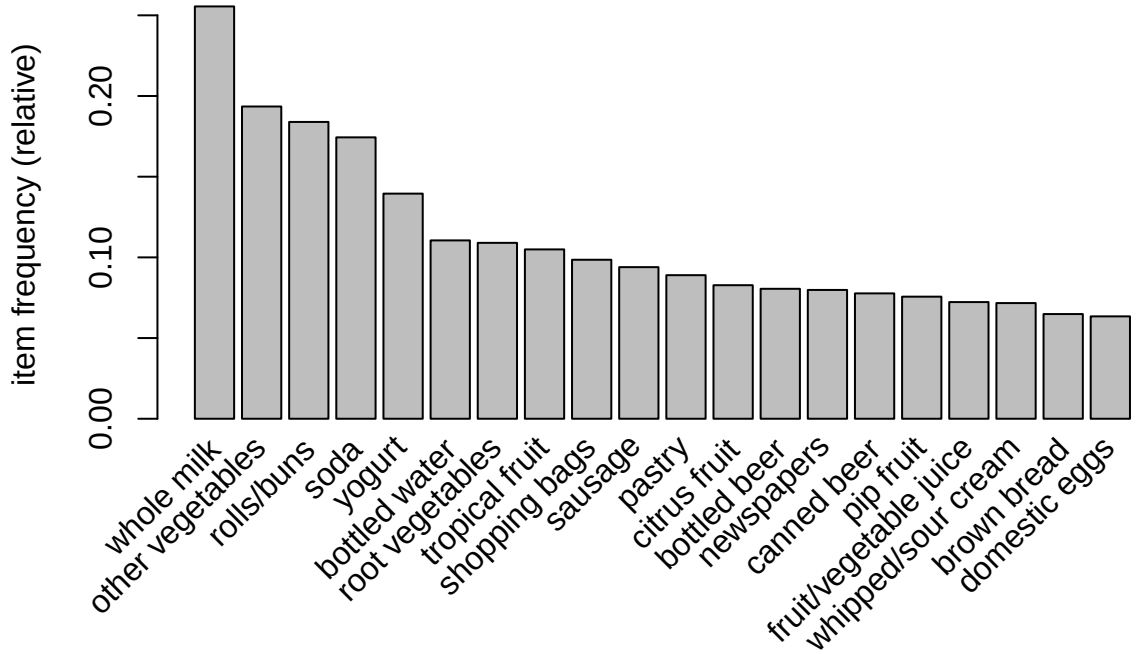
As shown in the following plot, this results in a histogram showing the eight items in the groceries data with at least 10 percent support:

```
itemFrequencyPlot(groceries, support=.1)
```



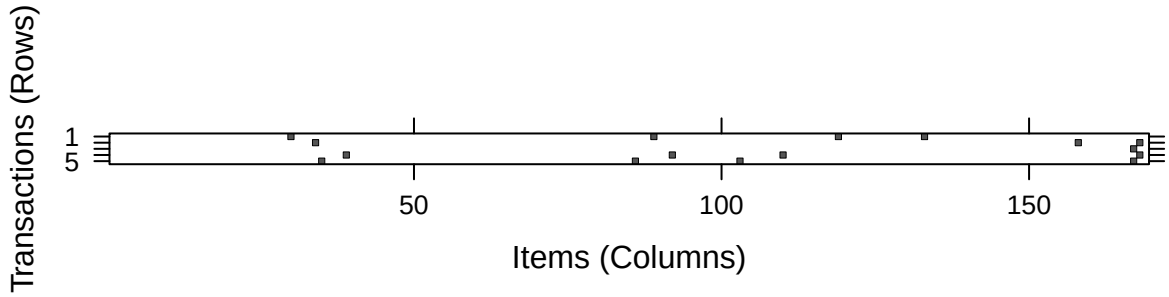
Shown in the following diagram of the top 20 items in the groceries data:

```
itemFrequencyPlot(groceries, topN=20)
```



Visualizing the transaction data – plotting the sparse matrix

```
image(groceries[1:5])
```

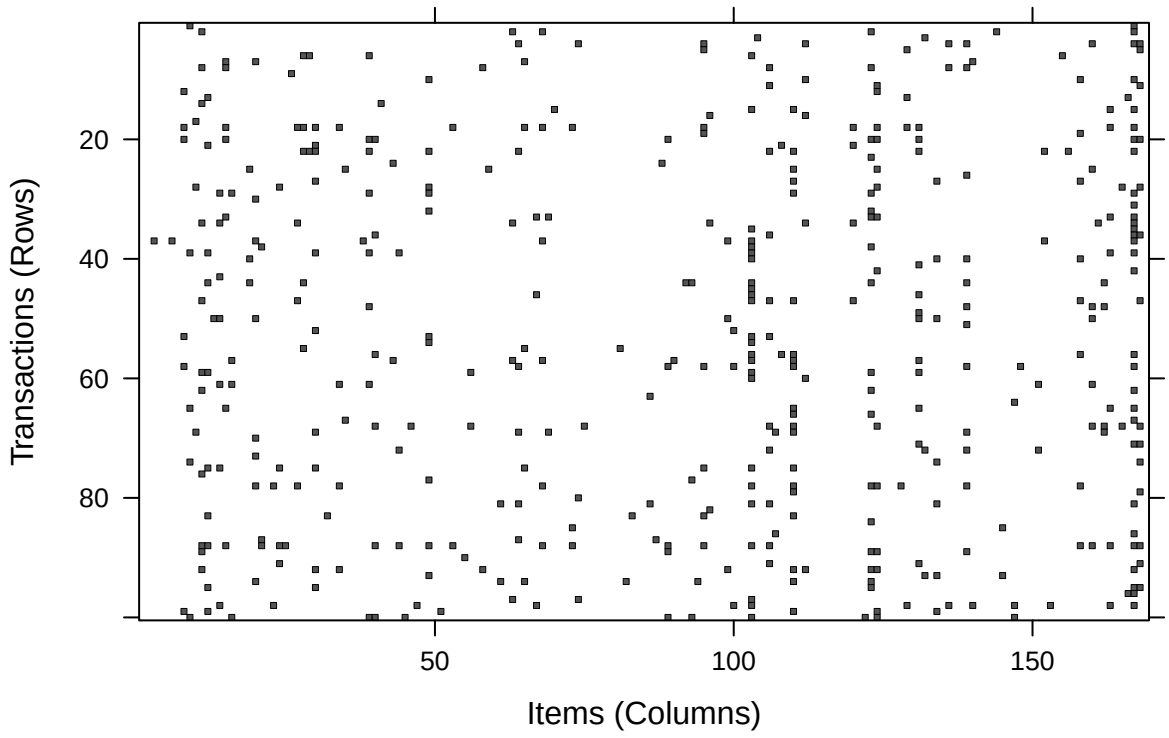


The resulting diagram depicts a matrix with 5 rows and 169 columns, indicating the 5 transactions and 169 possible items we requested. Cells in the matrix are filled with black for transactions (rows) where the item (column) was purchased.

Additionally, patterns in the diagram may help reveal actionable insights within the transactions and items, particularly if the data is sorted in interesting ways. For example, if the transactions are sorted by date, the patterns in black dots could reveal seasonal effects in the number or types of items purchased. Perhaps around Christmas or Hanukkah, toys are more common; around Halloween, perhaps candies become popular. This type of visualization could be especially powerful if the items were also sorted into categories. In most cases, however, the plot will look fairly random, like static on a television screen.

Keep in mind that this visualization will not be as useful for extremely large transaction databases, because the cells will be too small to discern. Still, by combining it with the sample() function, you can view the sparse matrix for a randomly sampled set of transactions. This creates a matrix diagram with 100 rows and 169 columns as shown below:

```
image(sample(groceries, 100))
```



A few columns seem fairly heavily populated, indicating some very popular items at the store. But overall,

the distribution of dots seems fairly random. Given nothing else of note, let's continue with our analysis.

Step 3 – training a model on the data

```
GroceriesRules<-apriori(groceries, parameter = list(support=0.006, confidence=0.25, minlen= 2))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.25    0.1    1 none FALSE            TRUE       5   0.006      2
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 59
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [463 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
GroceriesRules
```

```
## set of 463 rules
```

Our GroceryRules object contains a set of 463 association rules. To determine whether any of them are useful, we'll have to dig deeper.

Step 4 – evaluating model performance

The rule length distribution tells us how many rules have each count of items. In our rule set, 150 rules have only two items, while 297 have three, and 16 have four. The summary statistics associated with this distribution are also given:

```
summary(GroceriesRules)
```

```
## set of 463 rules
##
## rule length distribution (lhs + rhs):sizes
##   2   3   4
## 150 297  16
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.000   3.000   2.711   3.000   4.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.006101   Min.   :0.2500   Min.   :0.009964   Min.   :0.9932
##  1st Qu.:0.007117   1st Qu.:0.2971   1st Qu.:0.018709   1st Qu.:1.6229
```

```
##  Median :0.008744   Median :0.3554   Median :0.024809   Median :1.9332
##  Mean   :0.011539   Mean   :0.3786   Mean   :0.032608   Mean   :2.0351
##  3rd Qu.:0.012303   3rd Qu.:0.4495   3rd Qu.:0.035892   3rd Qu.:2.3565
##  Max.   :0.074835   Max.   :0.6600   Max.   :0.255516   Max.   :3.9565
##      count
##  Min.   : 60.0
##  1st Qu.: 70.0
##  Median : 86.0
##  Mean   :113.5
##  3rd Qu.:121.0
##  Max.   :736.0
##
## mining info:
##       data ntransactions support confidence
##  groceries          9835   0.006       0.25
##                                                                                call
##  apriori(data = groceries, parameter = list(support = 0.006, confidence = 0.25, minlen = 2))
```

The third column is a metric we have not considered yet. The lift of a rule measures how much more likely one item or itemset is purchased relative to its typical rate of purchase, given that you know another item or itemset has been purchased. This is defined by the following equation:

$lift(X->Y) = \frac{confidence(X->Y)}{support(Y)}$

The first three rules in the groceryrules object can be viewed as follows:

```
inspect(GroceriesRules[1:3])
```

```
##     lhs                rhs                support     confidence coverage
## [1] {potted plants} => {whole milk}      0.006914082 0.4000000  0.01728521
## [2] {pasta}         => {whole milk}      0.006100661 0.4054054  0.01504830
## [3] {herbs}         => {root vegetables} 0.007015760 0.4312500  0.01626843
##     lift     count
## [1] 1.565460 68
## [2] 1.586614 60
## [3] 3.956477 69
```

The first rule can be read in plain language as, "if a customer buys potted plants, they will also buy whole milk." With support of 0.007 and confidence of 0.400, we can determine that this rule covers 0.7 percent of the transactions and is correct in 40 percent of purchases involving potted plants. The lift value tells us how much more likely a customer is to buy whole milk relative to the average customer, given that he or she bought a potted plant. Since we know that about 25.6 percent of the customers bought whole milk (support), while 40 percent of the customers buying a potted plant bought whole milk (confidence), we can compute the lift value as 0.40 / 0.256 = 1.56, which matches the value shown.

In spite of the fact that the confidence and lift are high, does {potted plants} -> {whole milk} seem like a very useful rule? Probably not, as there doesn't seem to be a logical reason why someone would be more likely to buy milk with a potted plant. Yet our data suggests otherwise. How can we make sense of this fact? A common approach is to take the association rules and divide them into the following three categories:

- Actionable
- Trivial
- Inexplicable

Obviously, the goal of a market basket analysis is to find actionable rules that provide a clear and useful insight. Some rules are clear, others are useful; it is less common to find a combination of both of these factors. So-called trivial rules include any rules that are so obvious that they are not worth mentioning—they

are clear, but not useful. Suppose you were a marketing consultant being paid large sums of money to identify new opportunities for cross-promoting items. If you report the finding that {diapers} -> {formula}, you probably won't be invited back for another consulting job.

Rules are inexplicable if the connection between the items is so unclear that figuring out how to use the information is impossible or nearly impossible. The rule may simply be a random pattern in the data, for instance, a rule stating that {pickles} -> {chocolate ice cream} may be due to a single customer, whose pregnant wife had regular cravings for strange combinations of foods. The best rules are hidden gems—those undiscovered insights into patterns that seem obvious once discovered. Given enough time, one could evaluate each and every rule to find the gems. However, we (the one performing the market basket analysis) may not be the best judge of whether a rule is actionable, trivial, or inexplicable. In the next section, we'll improve the utility of our work by employing methods to sort and share the learned rules so that the most interesting results might float to the top.

Step 5 – improving model performance

Sorting the set of association rules

```
inspect(sort(GroceriesRules, by="lift")[1:5])
```

```
##     lhs                   rhs                     support confidence   coverage     lift count
## [1] {herbs}            => {root vegetables}    0.007015760  0.4312500 0.01626843 3.956477    69
## [2] {berries}          => {whipped/sour cream} 0.009049314  0.2721713 0.03324860 3.796886    89
## [3] {other vegetables,
##      tropical fruit,
##      whole milk}       => {root vegetables}    0.007015760  0.4107143 0.01708185 3.768074    69
## [4] {beef,
##      other vegetables} => {root vegetables}    0.007930859  0.4020619 0.01972547 3.688692    78
## [5] {other vegetables,
##      tropical fruit}   => {pip fruit}          0.009456024  0.2634561 0.03589222 3.482649    93
```

These rules appear to be more interesting than the ones we looked at previously. The first rule, with a lift of about 3.96, implies that people who buy herbs are nearly four times more likely to buy root vegetables than the typical customer—perhaps for a stew of some sort? Rule two is also interesting. Whipped cream is over three times more likely to be found in a shopping cart with berries versus other carts, suggesting perhaps a dessert pairing?

Taking subsets of association rules

Suppose that given the preceding rule, the marketing team is excited about the possibilities of creating an advertisement to promote berries, which are now in season. Before finalizing the campaign, however, they ask you to investigate whether berries are often purchased with other items. To answer this question, we'll need to find all the rules that include berries in some form.The result is the following set of rules:

```
BerryRules<-subset(GroceriesRules, items %in% "berries")
inspect(BerryRules)
```

```
##     lhs          rhs                  support     confidence coverage  lift
## [1] {berries} => {whipped/sour cream} 0.009049314 0.2721713  0.0332486 3.796886
## [2] {berries} => {yogurt}             0.010574479 0.3180428  0.0332486 2.279848
## [3] {berries} => {other vegetables}   0.010269446 0.3088685  0.0332486 1.596280
## [4] {berries} => {whole milk}         0.011794611 0.3547401  0.0332486 1.388328
##     count
## [1]  89
## [2] 104
## [3] 101
```

```
## [4] 116
```

There are four rules involving berries, two of which seem to be interesting enough to be called actionable. In addition to whipped cream, berries are also purchased frequently with yogurt—a pairing that could serve well for breakfast or lunch as well as dessert.

```
inspect(subset(GroceriesRules, confidence>0.6))
```

```
##     lhs                     rhs              support confidence    coverage     lift count
## [1] {curd,
##      tropical fruit}     => {whole milk} 0.006507372  0.6336634 0.010269446 2.479936    64
## [2] {butter,
##      whipped/sour cream} => {whole milk} 0.006710727  0.6600000 0.010167768 2.583008    66
## [3] {butter,
##      tropical fruit}     => {whole milk} 0.006202339  0.6224490 0.009964413 2.436047    61
## [4] {butter,
##      root vegetables}    => {whole milk} 0.008235892  0.6377953 0.012913066 2.496107    81
## [5] {butter,
##      yogurt}             => {whole milk} 0.009354347  0.6388889 0.014641586 2.500387    92
## [6] {domestic eggs,
##      tropical fruit}     => {whole milk} 0.006914082  0.6071429 0.011387900 2.376144    68
## [7] {other vegetables,
##      tropical fruit,
##      yogurt}             => {whole milk} 0.007625826  0.6198347 0.012302999 2.425816    75
## [8] {other vegetables,
##      root vegetables,
##      yogurt}             => {whole milk} 0.007829181  0.6062992 0.012913066 2.372842    77
```

There are eight rules involving with confidence greater than 60%. All of association are made up of "Whole milk".

Summary

Association rules are frequently used to find useful insights in the massive transaction databases of large retailers. As an unsupervised learning process, association rule learners are capable of extracting knowledge from large databases without any prior knowledge of what patterns to seek. The catch is that it takes some effort to reduce the wealth of information into a smaller and more manageable set of results. The Apriori algorithm, which we studied in this paper, does so by setting minimum thresholds of interestingness, and reporting only the associations meeting these criteria. We put the Apriori algorithm to work while performing a market basket analysis for a month's worth of transactions at a moderately sized supermarket. Even in this small example, a wealth of associations was identified. Among these, we noted several patterns that may be useful for future marketing campaigns. The same methods we applied are used at much larger retailers on databases many times this size.