National University of Singapore
School of Continuing & Lifelong Education (SCALE)

TBA2105 Web Mining
Tutorial/Lab 3

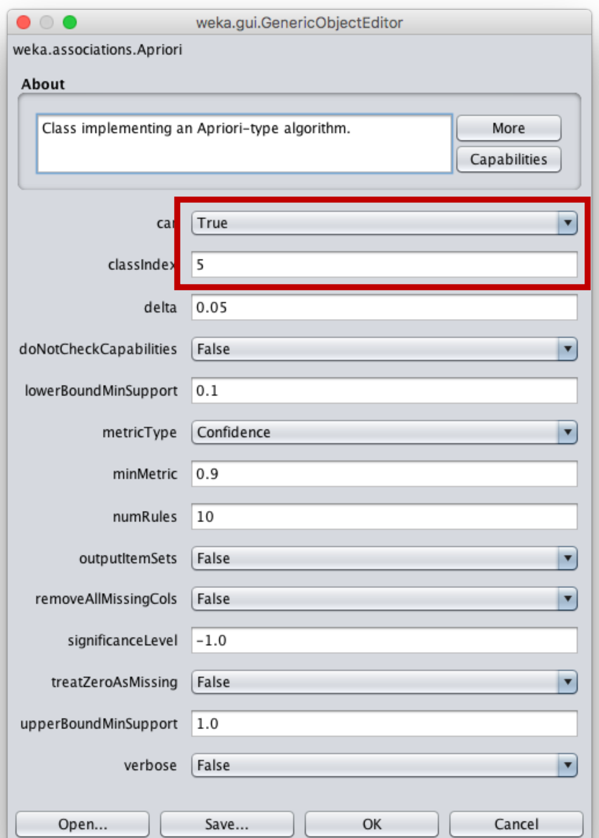
Learning Objectives

- Perform Class Association Rule(CAR) mining using Weka
- Perform association rule mining using R

Class Association Rules Mining

In the previous lab, we have worked with the `weather` dataset to do association rule mining. Since this dataset is mainly used for classification, we would want to make the **play** attribute appear on the RHS. To make use of association rules to do classification, we can mine **Class Association Rules (CAR).**

1. On the apriori property window, set **car** to **True** and enter **5** for the **classIndex**.



2. Now re-run the algorithm that you did in lab 1. We should now see that the **play** no longer appears on the LHS and is present for every RHS. The following is a sample of the rules obtained by keeping other options as default.

```
…
Best rules found:

 1. outlook=overcast 4 ==> play=yes 4    conf:(1)
 2. humidity=normal windy=FALSE 4 ==> play=yes 4    conf:(1)
 3. outlook=sunny humidity=high 3 ==> play=no 3    conf:(1)
 4. outlook=rainy windy=FALSE 3 ==> play=yes 3    conf:(1)
 5. outlook=sunny humidity=normal 2 ==> play=yes 2    conf:(1)
 6. outlook=sunny temperature=hot 2 ==> play=no 2    conf:(1)
 7. outlook=overcast temperature=hot 2 ==> play=yes 2    conf:(1)
 8. outlook=overcast humidity=high 2 ==> play=yes 2    conf:(1)
 9. outlook=overcast humidity=normal 2 ==> play=yes 2    conf:(1)
10. outlook=overcast windy=TRUE 2 ==> play=yes 2    conf:(1)
```

Association Rule Mining using R

For the rest of this lab, we will learn how to perform association rule mining using R. To use the Apriori algorithm in R, we would need to install and load the **arules** package.

Working with Transaction Form

We will first learn how to read in different data formats into the R environment. The following screenshot shows an extract of the first 10 transactions of the `groceries` dataset. Each row is a purchase transaction where the items are separated by comma. This is the **transaction form** that we discussed in the lectures.

```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer)
potted plants
whole milk,cereals
```

We can use the `read.transactions()` function in the **arules** package to read in the transactions.

a) Read up and use the `read.transactions()` function to read in the transactions from the `groceries` dataset into a variable called `transactions`.

To see the details of the transactions, we can use the usual `summary()` function. From the results of the `summary()` function, in the 1st section, we see that there are 9835 transactions and 169 unique items in the dataset. This is represented as a sparse matrix (table form where most of the cells are zero). **Density** shows the percentage of non-zero cells.

Total_Num_Of_Items_Purchased = Density * Num_Transaction * Num_Items

The 2nd section shows the list of the most frequent items and their frequencies.

The 3rd section provides some information about the characteristics of the transactions. Specifically, 2159 transactions are having 1 item only, 1643 transactions are having 2 items only, etc

```
> summary(transactions)
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
            2513             1903             1809
            soda           yogurt          (Other)
            1715             1372            34055

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12
2159 1643 1299 1005  855  645  545  438  350  246  182  117
  13   14   15   16   17   18   19   20   21   22   23   24
  78   77   55   46   29   14   14    9   11    4    6    1
  26   27   28   29   32
   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

…
```

To access the details of the `transactions` object or the `summary(transactions)` object, we can use slot names. Both `transactions` and `summary(transactions)` are S4 class objects in R. Here is an example of how to make use of the slot names to access the values programmatically.

```
> results <- summary(transactions)

> #get the slot names
> slotNames(results)
[1] "Dim"           "density"        "itemSummary"    "lengths"
"lengthSummary" "itemInfo"      "itemsetInfo"

> #to get the density value
> density <- slot(results, "density")
> density
[1] 0.02609146
```

b) Make use of the slot names of `summary(transactions)` to obtain **Num_transaction** and **Num_items** and calculate the **Total_Num_Of_Items_Purchased**.

c) We should get the same result if we were to add up the number of different k-itemsets from the 3rd section. Complete the following codes to verify the results of question b)

```
possibleKValues <- rownames(slot(results, "lengths"))

totalNumberOfItemsPurchased <- 0
for (k in possibleKValues){
  #complete the codes
  ...

  totalNumberOfItemsPurchased <- totalNumberOfItemsPurchased +
(size * as.integer(k))
}
```

d) Using the slot names we can also access other useful information get the S4 slotnames of the `transactions` object. Try getting a vector of the unique items.

The following are other useful codes for accessing the transaction information.

```
> #we can also see the first 10 transaction using inspect()
> inspect(transactions[1:10])
     items
[1]  {citrus fruit,margarine,ready soups,semi-finished bread}
[2]  {coffee,tropical fruit,yogurt}
[3]  {whole milk}
…

> #or if we want to see all the transaction
> df <- as(transactions, "data.frame")
> View(df)

> #we can also get a (sparse) matrix of the transaction
> #(table format)
> mat <- as.matrix(slot(transactions, "data"))
> View(mat)
```

Working with Single Item Form

`read.transactions()` can also work with the single item form. For example, the following is the `bakery` dataset (from the last tutorial) in single item form (`bakery_1000_single.csv`).

```
trans_id,item_id
1,3
1,4
1,2
1,5
2,1
2,2
3,1
3,1
4,1
```

e) Use the `read.transactions()` function to read in the transactions from the `bakery` dataset into a variable called `bakeryTransactions`.

Working with Data Frame

It is also possible to convert an existing data frame which is in a table form into the transaction format.

```
> #not using read.transactions(), instead we cast an
> #existing DF to transactions type
> #the foreign package provides other read.XXX functions
> #for reading other types of datasets
> library(foreign)
> weather <- read.arff("weather.nominal.arff")
> weatherTransactions <- as(weather, "transactions")
```

Apply Apriori Algorithm to mine Rules

To apply the Apriori algorithm and generate the association rules, we can just call the `apriori()` function, supply the `transactions` object from above and the minsup and conf values.

f) Use the `apriori()` function to generate the `rules` for the `transactions` object with the following parameters:
min support = 0.001
min confidence = 0.8
max length = 10 (i.e. only allow maximum of 10 items per itemset)

Similar to before where we use the `inspect()` to view the transactions, we could use also use `inspect()` to view the rules. For each rule, the rule's **LHS**, **RHS**, **support**, **confidence**, and **lift** are given.

```
> inspect(rules[1:5])
    lhs                           rhs              support
confidence lift      count
[1] {liquor,red/blush wine} => {bottled beer} 0.001931876
0.9047619  11.235269 19
[2] {cereals,curd}          => {whole milk}   0.001016777
0.9090909   3.557863 10
[3] {cereals,yogurt}        => {whole milk}   0.001728521
0.8095238   3.168192 17
[4] {butter,jam}            => {whole milk}   0.001016777
0.8333333   3.261374 10
[5] {bottled beer,soups}    => {whole milk}   0.001118454
0.9166667   3.587512 11
```

g) Note that unlike Weka, we cannot supply the lift values to do the filtering during mining, but we could use the `subset()` function to do filtering based on a condition. Use the `subset()` function to filter the rules keeping only those rules with a lift of more than 5.

h) When we have a lot of rules, it is useful to sort the rules by the different measures (support, confidence, lift). We could do this by calling the `sort()` function in the arules package. Use the `sort()` function to sort the rules (from question f) in descending order by support.

i) `apriori()` can also be used to generate the **frequent itemsets** by changing the `target` argument (`target = "frequent itemsets"`). Generate the **frequent itemsets** using the same parameters as question f and sort the results in descending order by support.

Apply Apriori algorithm to generate the Frequent Itemsets

`apriori()` can also be used to generate the frequent itemsets by changing the `target` argument. `sort()` and `inspect()` can also be used for itemsets (name the variable as `sortedFrequentItemsets`).

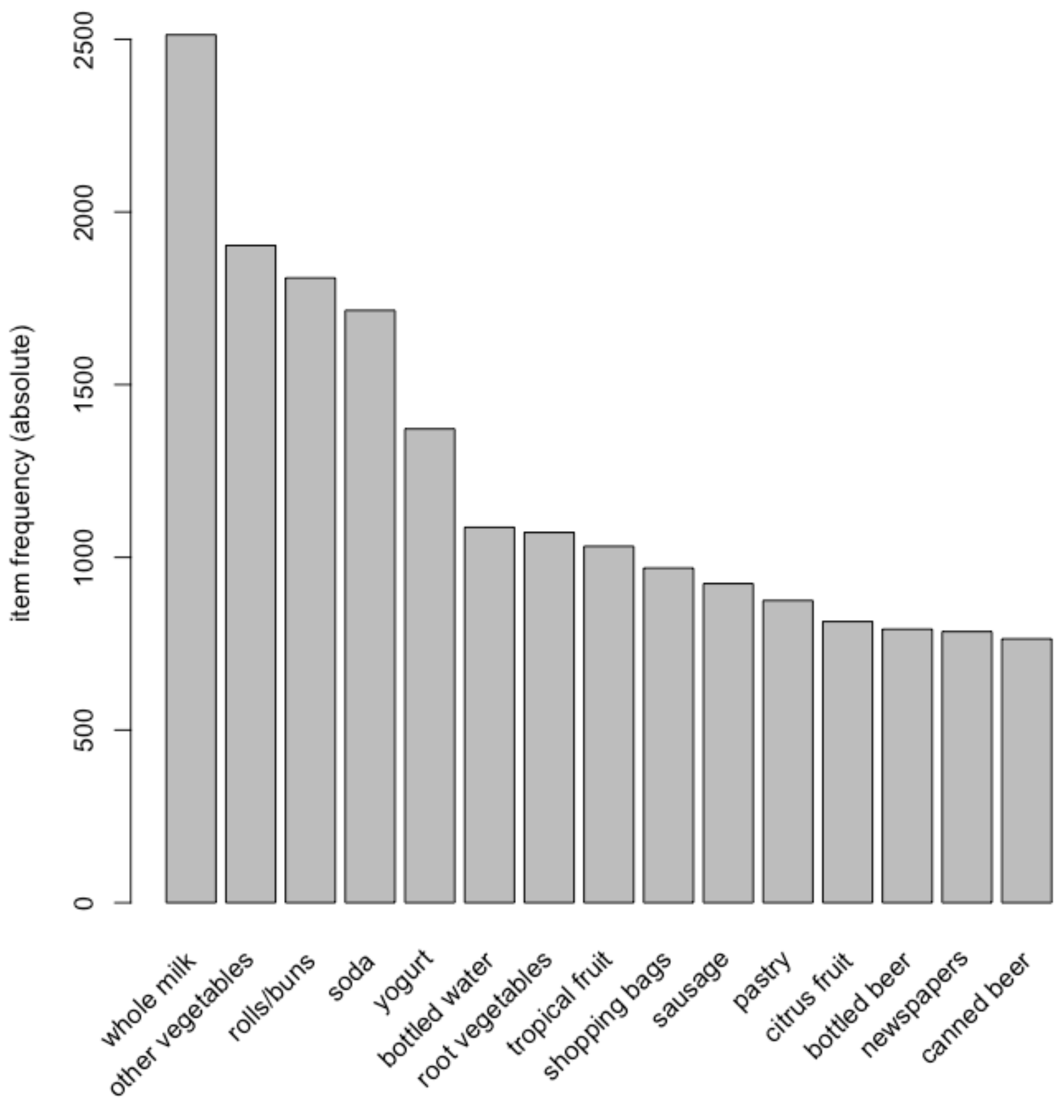
```
…
> inspect(sortedFrequentItemsets[1:10])

     items               support    count
[1]  {whole milk}        0.25551601 2513
[2]  {other vegetables}  0.19349263 1903
[3]  {rolls/buns}        0.18393493 1809
[4]  {soda}              0.17437722 1715
[5]  {yogurt}            0.13950178 1372
[6]  {bottled water}     0.11052364 1087
[7]  {root vegetables}   0.10899847 1072
[8]  {tropical fruit}    0.10493137 1032
[9]  {shopping bags}     0.09852567  969
[10] {sausage}           0.09395018  924
```

Visualizing the data

One of the most powerful features of the R implementation of association mining is the capability of generating relevant visualizations to better appreciate the data.

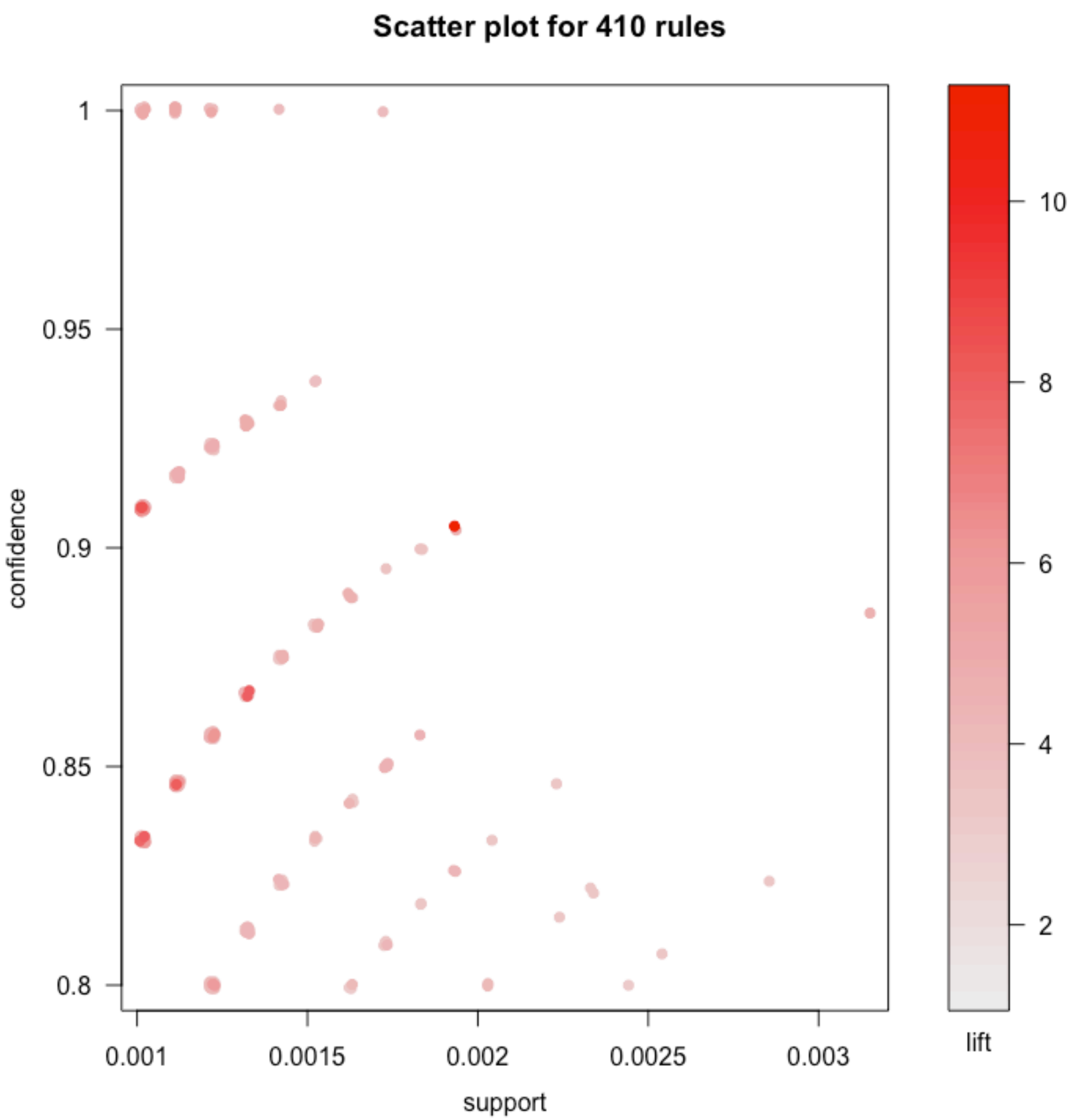
```
#part of the arules library
#possible to generate a column chart of the items
#plot the top 15 items based on absolute count
#type="relative" will plot based on relative frequency
itemFrequencyPlot(transactions, topN=15, type="absolute")
```



Instead of visualizing the items, we could also visualize the rules by using the `arulesViz` package. By calling the `plot()` on the rules, it will generate a scatter plot by looking at the relationships between support, confidence, and lift.

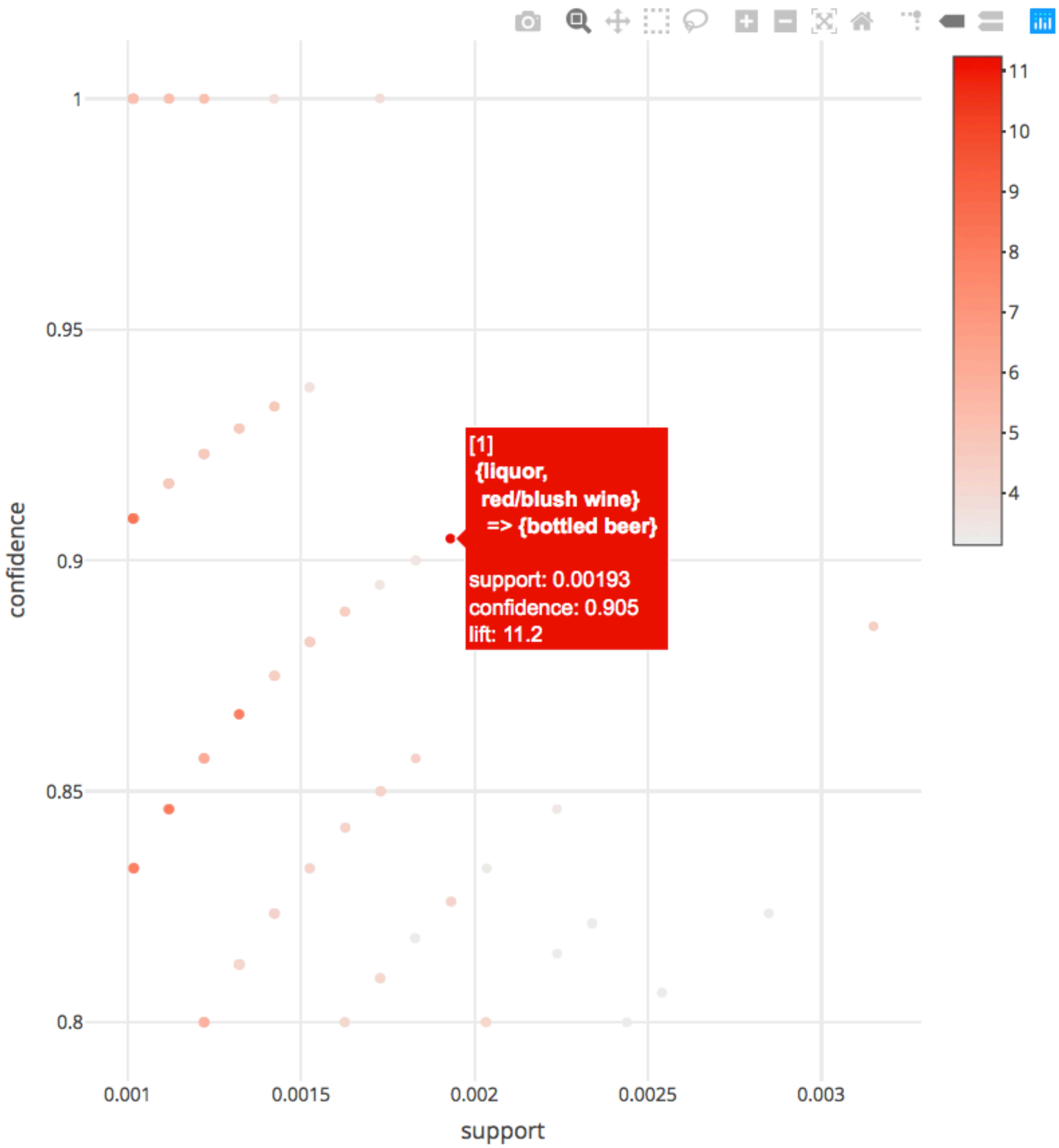
```
library(arulesViz)
plot(rules)

#can change this by defining the "measure" and "shading"
parameters
#lift vs support (color-coded by confidence)
plot(rules, measure = c("support", "lift"), shading =
"confidence")
```



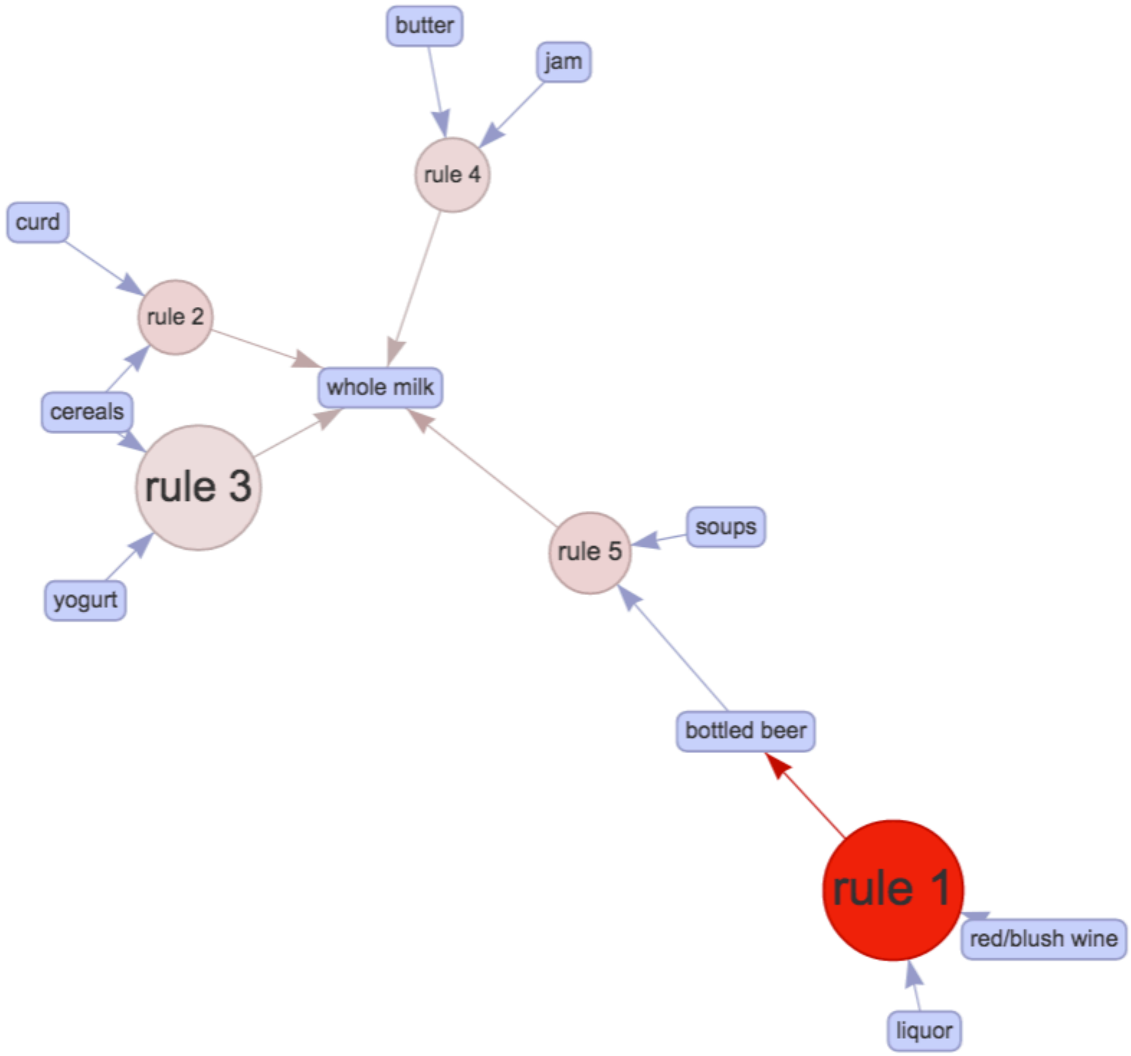
Interactive plots can also be generated where we could interact with the data (e.g. zoom in, view details of points, etc).

```
#the arulesViz package also allows generation of interactive
#plots when we specify the engine="htmlwidget" argument
#it is using plot.ly under the hood
plot(rules, engine = "htmlwidget")
```



The default value of the method argument is scatterplot could also change it to others (e.g. `plot(rules, method="two-key plot")`). One particular interesting plot is when `method="graph"`.

```
#save the first 5 rules using the method graph
#as a html
#so we can interact with it
p <- plot(rules[1:5], method="graph", engine="html")
htmlwidgets::saveWidget(p, "arules.html", selfcontained =
FALSE)
browseURL("arules.html")
```




```
{liquor,red/blush wine} => {bottled beer}
{cereals,curd}          => {whole milk}
{cereals,yogurt}        => {whole milk}
{butter,jam}            => {whole milk}
{bottled beer,soups}    => {whole milk}
```