Tutorial: Association Rules (important)

Q1a: Consider a transactional database where 1, 2, 3, 4, 5, 6, 7 are items.

ID	Items
t 1	1, 2, 3, 5
t 2	1, 2, 3, 4, 5
t 3	1, 2, 3, 7
t 4	1, 3, 6
t 5	1, 2, 4, 5, 6

Suppose the minimum support is 60%. Find all frequent itemsets. Indicate each candidate set C_k, k = 1, 2,.., the candidates that are pruned by each pruning step, and the resulting frequent itemsets L_k.

Use Apriori algorithm to find all frequent itemsets. Itemsets shaded in grey are removed because they fail the minimum support constraint. Those shaded in light yellow are removed because there exists a subset of itemsets that is not frequent. Minimum support count = 5 × 60% = 3

C_1

Itemset	Support
{1}	5
{2}	4
{3}	4
{4}	2
{5}	3
{6}	2
{7}	1

→

L_1

Itemset	Support
{1}	5
{2}	4
{3}	4
{5}	3

C_2

Itemset	Support
{1, 2}	4
{1, 3}	4
{1, 5}	3
{2, 3}	3
{2, 5}	3
{3, 5}	2

→

L_2

Itemset	Support
{1, 2}	4
{1, 3}	4
{1, 5}	3
{2, 3}	3
{2, 5}	3

C_3

Itemset	Support
{1, 2, 3}	3
{1, 2, 5}	3
{1, 3, 5}	
{2, 3, 5}	

→

L_3

Itemset	Support
{1, 2, 3}	3
{1, 2, 5}	3

C_4

Itemset	Support
{1, 2, 3, 5}	

Q1b: Let the minimum support be 60% and minimum confidence be 75%. Show all association rules that are constructed from the same transaction dataset.

All association rules generated from L_2 and L_3 are shown below together with support and confidence. All rows that are not shaded are association rules with confidence ≥ 75%.

Association Rule	Support	Confidence
{1} → {2}	4 (80%)	4/5 (80%)
{2} → {1}	4 (80%)	4/4 (100%)
{1} → {3}	4 (80%)	4/5 (80%)
{3} → {1}	4 (80%)	4/4 (100%)
{1} → {5}	3 (60%)	3/5 (60%)
{5} → {1}	3 (60%)	3/3 (100%)
{2} → {3}	3 (60%)	3/4 (75%)
{3} → {2}	3 (60%)	3/4 (75%)
{2} → {5}	3 (60%)	3/4 (75%)
{5} → {2}	3 (60%)	3/3 (100%)
{1} → {2, 3}	3 (60%)	3/5 (60%)
{2} → {1, 3}	3 (60%)	3/4 (75%)
{3} → {1, 2}	3 (60%)	3/4 (75%)
{1, 2} → {3}	3 (60%)	3/4 (75%)
{1, 3} → {2}	3 (60%)	3/4 (75%)
{2, 3} → {1}	3 (60%)	3/3 (100%)
{1} → {2, 5}	3 (60%)	3/5 (60%)
{2} → {1, 5}	3 (60%)	3/4 (75%)
{5} → {1, 2}	3 (60%)	3/3 (100%)
{1, 2} → {5}	3 (60%)	3/4 (75%)
{1, 5} → {2}	3 (60%)	3/3 (100%)
{2, 5} → {1}	3 (60%)	3/3 (100%)

Q2: Given the transaction database below

ID	Items
1	AB DE
2	BC E
3	AB DE
4	AB CE
5	AB CDE
6	BC D

Apply the Apriori algorithm with minimum support count threshold 3 and find the set M of maximal frequent itemsets.

We need to find first the frequent itemsets in the form candidate & final lists. For example C_1 = {A: 4, …}, L_1={…}.

C1 = {A: 4, B: 6, C: 4, D: 4, E: 5}
L1 = {A, B, C, D, E}
C2 = {AB:4, AC:2, AD:3, AE:4, BC:4, BD:4, BE:5, CD:2, CE:3, DE:3}
L2 = {AB, AD, AE, BC, BD, BE, CE, DE}
C3 = {ABD: 3, ABE: 4, ADE: 3, BCD: 2, BCE: 3, BDE: 3}
L3 = {ABD, ABE, ADE, BCE, BDE}
C4 = {ABDE: 3}
L4 = {ABDE}
C5 = {}
L5 = {}

Maximal frequent itemset: The definition says that an itemset is maximal frequent if none of its immediate supersets is frequent.
From the itemsets of the previous list, we need to find those ones which have all of their immediate supersets infrequent.

M = {ABDE, BCE}
For example ABCE: 2, BCDE: 1, which means that all immediate supersets of BCE are infrequent, for this reason BCE is maximal. For ABDE there is no C5/L5, so also it is maximal.

Q3: Given the transaction database below

TID	Items
1	A,C,D,F
2	B,C,E

	A,B,C,E
	B,D,E
	A,B,C,E
	A,B,C,D

Apply the Apriori algorithm with minimum support count threshold 4 and find (a) the set of frequent itemsets, (b) the set of closed frequent itemsets and (c) the set of maximal frequent itemsets.

We check first the 1-size itemsets

A 4
B 5
C 5
D 3 reject due to less than 4
E 4
F 1

Only 4 1-temsets will be considered as candidates for the creation of 2-size itemsets

AB 3
AC 4
AE 2
BC 4
BE 4
CE 3

3 2-itemsets are suitable; they will create the 3-size itemsets in the next stage

ABC 3 and ab
ACE reject due to ae
ABE 2 and ae
BCE 3 and ce

Thus, none 3-itemsets are frequent.
Remember the definitions:

- **Closed Frequent Itemset:** An itemset is closed if none of its immediate supersets has the same support as that of the itemset.
- **Maximal frequent itemset:** The definition says that an itemset is maximal frequent if none of its immediate supersets is frequent

Let's try to find the closed frequent itemsets

B, C, BE, AC, BC

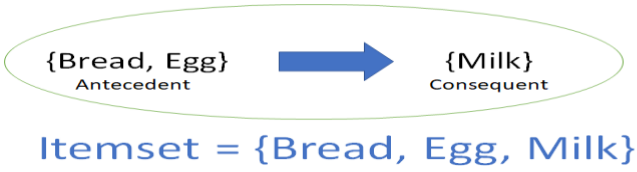
A and E have been rejected as their immediate supersets have the save support. The maximal ones are obviously

BE, AC, BC

R-Implementations (Association Rules)

Association Rules is one of the very important concepts of machine learning being used in market basket analysis. In a store, all vegetables are placed in the same aisle, all dairy items are placed together and cosmetics form another set of such groups. Investing time and resources on deliberate product placements like this not only reduces a customer's shopping time, but also reminds the customer of what relevant items (s)he might be interested in buying, thus helping stores cross-sell in the process. Association rules help uncover all such relationships between items from huge databases. One important thing to note is that Rules do not extract an individual's preference, rather find relationships between set of elements of every distinct transaction. This is what makes them different from collaborative filtering.
To elaborate on this idea, rules do not tie back a users' different transactions over time to identify relationships. List of items with unique transaction IDs (from all users) are studied as one group. This is helpful in placement of products on aisles. On the other hand, collaborative filtering ties back all transactions corresponding to a user ID to identify similarity between users' preferences. This is helpful in recommending items on e-commerce websites, recommending songs on Spotify, etc.

Let's now see what an association rule exactly looks like. It consists of an antecedent and a consequent, both of which are a list of items. Note that implication here is co-occurrence and not causal1ity. For a given rule, itemset is the list of all the items in the antecedent and the consequent.



Various metrics are in place to help us understand the strength of association between these two. Let us go through them all.

- Support: This measure gives an idea of how frequent an itemset is in all the transactions.
- Confidence: This measure defines the likeliness of occurrence of consequent on the cart given that the cart already has the antecedents. The confidence shows how often a rule is found to be true, e.g. if *x* is bought, how often is *y* bought. In this context, rather than *x* and *y*, the terms Left-Hand-Side (LHS) and Right-Hand-Side (RHS) are used.
- Lift: The ratio by which by the confidence of a rule exceeds the expected confidence. Lift controls for the support (frequency) of consequent while calculating the conditional probability of occurrence of {Y} given {X}.Lift provides the information if a rule LHS /RHS is random (LHS and RHS are independent) or not. If Lift > 1, both occurrences are dependent. Only for Lift greater 1 a potential useful rule can be found.

Rule-generation is a two-step process. First is to generate an itemset like {Bread, Egg, Milk} and second is to generate a rule from each itemset like {Bread → Egg, Milk}, {Bread, Egg → Milk} etc.

- Generating itemsets from a list of items: First step in generation of association rules is to get all the frequent itemsets on which binary partitions can be performed to get the antecedent and the consequent.
- Generating all possible rules from the frequent itemsets: Once the frequent itemsets are generated, identifying rules out of them is comparatively less taxing. Rules are formed by binary partition of each itemset.

Association rules use the R arules library. The arulesViz add additional features for graphing and plotting the rules.

Groceries – Our first simple approach to this specific case study.

Association rules reflect regularities of items or elements in a set of items, such as sale items, web link clicks or web page visits. The apriori command in the R package arules mines frequent itemsets, association rules and class association rules using the Apriori algorithm. The apriori command requires input in a transactions class. We normally view a transaction data in a two-dimensional matrix, where each row is one transaction and each column represents one sale item available in a grocery store or any other item objects.
The arules package has a sample dataset, Groceries, which is a transactions class. If we run the command "?Groceries", the R help document will display the data information. The Groceries data set contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The data set contains 9835 transactions and the items are aggregated to 169 categories.

Let's get started by loading up our libraries and data set.

```
Load the libraries
library(arules)
library(arulesViz)
library(datasets)

Load the data set
data(Groceries)

To view the first five transactions,
inspect(head(Groceries))   # see the head function!
```

```
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
## [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 content returned from the inspect command is represented in the common way of describing a transaction by enumerating its items. However, the transactions class stores the transaction data in a special way. Because the transaction data matrix is often very sparse, having a majority of element values being null for the items which haven't been purchased in a transaction. The Groceries data only has around 2% of cells which are not null. To find the components inside the Groceries data, run the str command.

```
str(Groceries)
```

```
## Formal class 'transactions' [package "arules"] with 3 slots
##   ..@ data       :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
##   .. .. ..@ i      : int [1:43367] 13 60 69 78 14 29 98 24 15 29 ...
##   .. .. ..@ p      : int [1:9836] 0 4 7 8 12 16 21 22 27 28 ...
##   .. .. ..@ Dim    : int [1:2] 169 9835
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ factors : list()
##   ..@ itemInfo   :'data.frame':   169 obs. of  3 variables:
##   .. ..$ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ham" ...
##   .. ..$ level2: Factor w/ 55 levels "baby food","bags",..: 44 44 44 44 44 44 44 42 42 41 ...
##   .. ..$ level1: Factor w/ 10 levels "canned food",..: 6 6 6 6 6 6 6 6 6 6 ...
##   ..@ itemsetInfo:'data.frame': 0 obs. of  0 variables
```

A transactions data has three component slots: @data, @itemInfo and @itemsetInfo.
@itemsetInfo Component: Initially, the @itemsetInfo is an empty data frame, which won't be filled with the itemsets until we run the apriori function.
@itemInfo Component: The itemInfo component is a dataframe. The labels column stores the item labels or names. The following command returns the first 20 item names.

```
Groceries@itemInfo$labels[1:20]
```

```
##  [1] "frankfurter"       "sausage"           "liver loaf"
##  [4] "ham"               "meat"              "finished products"
##  [7] "organic sausage"   "chicken"           "turkey"
## [10] "pork"              "beef"              "hamburger meat"
## [13] "fish"              "citrus fruit"      "tropical fruit"
## [16] "pip fruit"         "grapes"            "berries"
## [19] "nuts/prunes"       "root vegetables"
```

@data Component: The @data component holds the transaction data. Groceries@data@i lists the item indexes which are included in each transaction, repeated from the first transaction to the last one. Groceries@data@p indicates the starting position for each transaction when reading its associated item indexes from Groceries@data@i. For instance, we know from the previous inspect command, that the first transaction contains four items: citrus fruit, semi-finished bread, margarine, ready soups. How is the first transaction stored in @data? Firstly, find the item indexes of these four items. A quick way of doing so is the which function which returns the index of the matching element in a vector.

```
which(Groceries@itemInfo$labels == 'citrus fruit')
## [1] 14
which(Groceries@itemInfo$labels == 'semi-finished bread')
## [1] 61
```

Noticeably, the index in @data begins with zero. The first four integers in Groceries@data@i, 13 60 69 78, are the included items for the first transaction. The first integer, 0, in Groceries@data@p indicates the starting position in Groceries@data@i when reading the items for the first transaction. Similarly, the second integer in Groceries@data@p indicates the starting position in Groceries@data@i when reading the items for the second transaction, and so on.

In general, when we wish to mine such type of information (i.e. from transactions), we need to convert our sample data into the Transactions class. The *arules* package provides the function read.transactions which reads basket data into a transactions class. The following script will read basket data as the Transactions class.

```
transactions <- read.transactions(
  file="baskets",
  format = c("basket"),
  sep = ",",
  cols =NULL,
  rm.duplicates = 1,
  skip = 0
)
```

The parameter format is a character string indicating the format of the data set. For 'basket' format, each line in the transaction data file represents a transaction where the items (item labels) are separated by the characters specified by sep. For 'single' format, each line corresponds to a single item, containing at least ids for the transaction and the item. The parameter sep is a character string specifying how fields are separated in the data file. We use ',' in the sample basket data. The parameter skip is number of lines to skip in the file before start reading data.

Let's explore more the data before we make any rules:

```
summary(Groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda           yogurt          (Other)
##             2513             1903             1809             1715             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   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.
##    1.00    2.00    3.00    4.41    6.00   32.00
##
## includes extended item information - examples:
##       labels  level2           level1
## 1 frankfurter sausage meat and sausage
## 2     sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
```

In our language the above means:

- there were 9835 transactions altogether
- 169 different items were bought during the month
- the most frequently bought item was "whole milk": 2513 purchases
- there were 2159 single item baskets, the biggest basket included 32 items
- median basket included 3 items; mean had 4.4 items.

In addition to the command "Groceries@itemInfo$labels[1:20]", we have seen before, items traded at the shop may be shown like this:

```
itemLabels(Groceries)[1:10]     # [1:10] can be dropped to show all items
##  [1] "frankfurter"       "sausage"           "liver loaf"        "ham"               "meat"
```

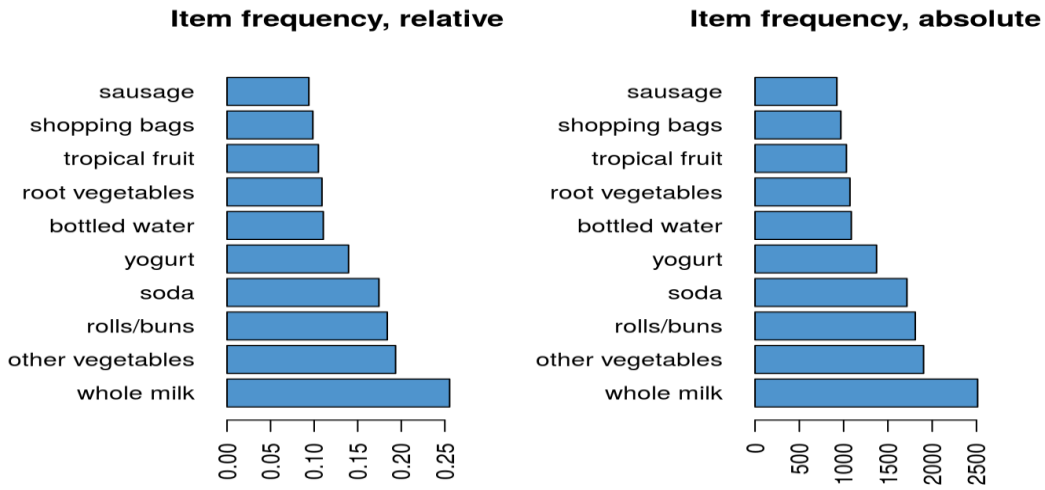
```
##  [6] "finished products" "organic sausage"   "chicken"          "turkey"           "pork"
```

Top 10 most frequent items, both by frequency and absolute counts, may be visualized as follows:

```
par(mfrow=c(1,2))

itemFrequencyPlot(Groceries,
                  type="relative",
                  topN=10, # can be changed to the number of interest
                  horiz=TRUE,
                  col='steelblue3',
                  xlab='',
                  main='Item frequency, relative')

itemFrequencyPlot(Groceries,
                  type="absolute",
                  topN=10,
                  horiz=TRUE,
                  col='steelblue3',
                  xlab='',
                  main='Item frequency, absolute')
```



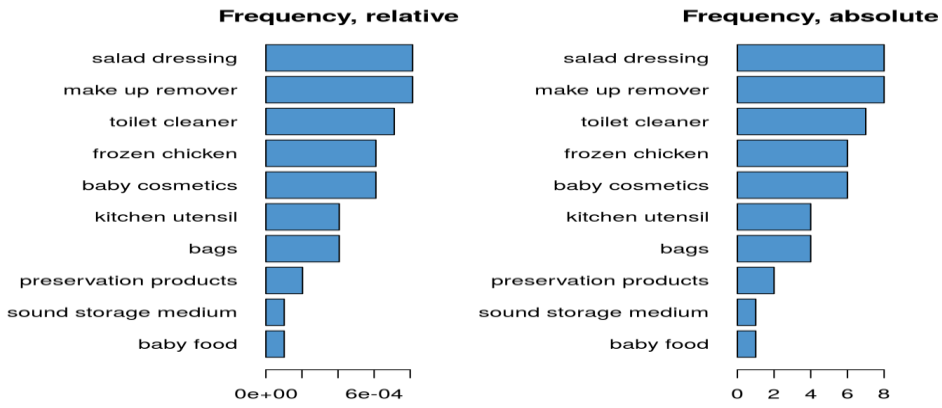
Alternatively, we might show least frequently bought items:

```
par(mar=c(2,10,2,2), mfrow=c(1,2))

barplot(sort(table(unlist(LIST(Groceries))))[1:10]/9835,
      horiz=TRUE,
      las=1,
      col='steelblue3',
      xlab='',
      main='Frequency, relative')

barplot(sort(table(unlist(LIST(Groceries))))[1:10],
```

```
horiz=TRUE,
las=1,
col='steelblue3',
xlab='',
main='Frequency, absolute')
```



which might tell us there are not many babies in the neighbourhood!

Simple contingency table

Sometimes it might be interesting to explore data as a simple contingency table.

```
tbl <- crossTable(Groceries)
tbl[1:5,1:5]
```

```
##              frankfurter sausage liver loaf ham meat
## frankfurter          580      99          7  25   32
## sausage               99     924         10  49   52
## liver loaf             7      10         50   3    0
## ham                   25      49          3 256    9
## meat                  32      52          0   9  254
```

By default, the table shows absolute counts, e.g.:

```
tbl['whole milk','whole milk']
```

```
## [1] 2513
```

shows already familiar number 2513 of purchases of “whole milk” (see the summary above). And:

```
tbl['whole milk','flour']
```

```
## [1] 83
```

would show number of occasions when these two items were purchased together. If we add an additional argument sort=TRUE, we would get items sorted by frequency of purchase (note decreasing counts diagonal-, row-, and column-wise):

```
tbl <- crossTable(Groceries, sort=TRUE)
tbl[1:5,1:5]
```

```
##                whole milk other vegetables rolls/buns soda yogurt
## whole milk           2513            736       557  394    551
## other vegetables      736           1903       419  322    427
## rolls/buns            557            419      1809  377    338
## soda                  394            322       377 1715    269
## yogurt                551            427       338  269   1372
```

Let's try to remember again our useful measures, which would help us in identifying interesting, actionable rules:

- count: Number of times a particular item, or itemset, is encountered in the transactions database. Count for the "whole milk", e.g. is 2513.
- Support: Support of an item, or an itemset consisting of several items, is frequency of occurrence of a specific item. Support (or frequency) is obtained as count divided by number of transactions. Support for "whole milk' e.g. is 2513/9835 = 0.2555. As a rule, but not necessarily, we want items/itemsets with high support, as high frequency would ensure that our potentially valuable finding (i) is not due to chance (ii) might generate enough profit by recycling it many times.
- Confidence: {A} => {B}. Confidence is probability of purchase B, given purchase A happened. For recommending a good rule we prefer higher confidence.
    - Warning 1: Confidence is not defined for contingency table as we are not considering transactions per se here.
    - Warning 2: For substitute products, like "bottled beer"/"canned beer" or "tee"/"coffee", we will observe low confidence, together with lift less than 1.
- Lift: Lift shows how more often the rule under questions happens than if it did simply happen by chance. Lift defined both for itemsets and rules. In general, we prefer higher lift over lower lift.
- ChiSquared: Finally, in certain situations where lift close to 1, but counts are large; or lift is meaningfully different from 1, but counts are low, we may need to turn to statistical chiSquared test to prove that events A and B are statistically dependent (i.e. we did not run into spurious correlation).

Let's see what products tend to complement each other with high lift (i.e. purchase of one product would lead to purchase of another with high probability) and what products tend to be substitutes:

```
crossTable(Groceries, measure='lift', sort=T)[1:5,1:5]
```

```
##                whole milk other vegetables rolls/buns   soda yogurt
## whole milk             NA         1.5136     1.205 0.8991  1.572
## other vegetables     1.5136           NA     1.197 0.9703  1.608
## rolls/buns           1.2050         1.1970        NA 1.1951  1.339
## soda                 0.8991         0.9703     1.195     NA  1.124
## yogurt               1.5717         1.6085     1.339 1.1244     NA
```

It's interesting to see, that "whole milk" goes well with all products, but "soda". So, judged by lift, we are on the way to claim that soda is a substitute for "whole milk" for some people: they tend to buy either one or the other, but buying them together is a relatively rare event. To convince ourselves that lower than 1 lift is not due to chance, let's apply chiSquared test:

```
crossTable(Groceries, measure='chi')['whole milk', 'soda']
## [1] 0.0004535
```

Indeed, the low p-value excludes possibility that lift less than 1 is due to chance. To summarize this section, crossTable() function allows showing and sorting items and pairs of items by:

- count
- support

- lift
- chiSquared

measures, thus identifying most promising two-member itemsets as compliment or substitute candidates.

Apriori: Search for frequent itemsets

Apriori function is a workhorse for arules package that has a lot of flexibility to accommodate almost any practical need of a retail analyst. Apriori allows for:

- mining both frequent itemsets and rules
- that satisfy pre-specified item length, support, confidence and lift
- that might only construct itemsets out of pre-specified items
- that might only include pre-specified items in lhs (left-hand-side of a purchase rule) or rhs (right-hand-side of a purchase rule).

Let's start with mining most frequent itemsets of minimum length equal 2, and frequency of occurrence at least 1 in 1000, i.e. support=.001

```
itemsets <- apriori(Groceries,
                    parameter = list(support=.001,
                                     minlen=2,
                                     target='frequent' # to mine for itemsets
                                     ))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen           target   ext
##          NA    0.1    1 none FALSE           TRUE   0.001      2     10 frequent itemsets FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 6 done [0.06s].
## writing ... [13335 set(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```
summary(itemsets)
```

```
## set of 13335 itemsets
##
## most frequent items:
##       whole milk other vegetables          yogurt  root vegetables   tropical fruit          (Other)
##             3764             3341             2401             1958             1796            27683
##
## element (itemset/transaction) length distribution:sizes
##    2    3    4    5    6
## 2981 6831 3137  376   10
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.00    3.00    3.07    4.00    6.00
##
## summary of quality measures:
##     support
```

```
##  Min.   :0.00102
##  1st Qu.:0.00112
##  Median :0.00142
##  Mean   :0.00226
##  3rd Qu.:0.00224
##  Max.   :0.07483
##
## includes transaction ID lists: FALSE
##
## mining info:
##       data ntransactions support confidence
##  Groceries         9835   0.001          1
```

We see based on pre-specified support 13335 itemsets of maximum length 6 were built out of 157 items (12 were thrown out due to rarity). Support for itemsets is calculated by default, so we can sort by it and print out top itemsets:

```
inspect(sort(itemsets, by='support', decreasing = T)[1:5])
```

```
##              items                              support
## 2981 {other vegetables,whole milk}              0.07483
## 2980 {whole milk,rolls/buns}                    0.05663
## 2978 {whole milk,yogurt}                        0.05602
## 2971 {root vegetables,whole milk}               0.04891
## 2970 {root vegetables,other vegetables}         0.04738
```

Couple of words on the syntax of sort and inspect:

- sort(), as implied by name, sorts itemsets and rules by measure specified in by=... argument. Usually one sorts by support, lift, confidence, chiSqured, or any other measure, that could be calculated with interestMeasure() function.
- inspect() is a command that prints out rules or itemsets of interest.

Should we want to add lift and show top 5 results by lift, we may proceed as follows:

```
quality(itemsets)$lift <- interestMeasure(itemsets, measure='lift', Groceries)
inspect(sort(itemsets, by ='lift', decreasing = T)[1:5])
```

```
##              items                                                                 support    lift
## 13326 {tropical fruit,root vegetables,other vegetables,whole milk,yogurt,oil}     0.001017 459.3
## 13328 {tropical fruit,other vegetables,whole milk,butter,yogurt,domestic eggs}    0.001017 399.6
## 13329 {tropical fruit,root vegetables,other vegetables,whole milk,butter,yogurt}  0.001118 255.9
## 12984 {other vegetables,curd,yogurt,whipped/sour cream,cream cheese }             0.001017 248.7
## 12950 {root vegetables,other vegetables,whole milk,yogurt,rice}                   0.001322 230.6
```

The figures above explain very well why fruits and vegetables departments are often found next to diary departments. Out of curiosity, we can repeat the exercise of building itemsets of length 2 that we did with crossTable, but this time with apriori function:

```
itemsets <- apriori(Groceries,
                    parameter = list(support=.001,
                                     minlen=2,
                                     maxlen=2,
                                     target='frequent' # to mine for itemsets    #pay attention the target option here
                                     ))
```

```
## Apriori
##
## Parameter specification:
```

```
##  confidence minval smax arem  aval originalSupport support minlen maxlen           target   ext
##         NA    0.1    1 none FALSE           TRUE   0.001      2      2 frequent itemsets FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [2981 set(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```
quality(itemsets)$lift <- interestMeasure(itemsets, measure='lift', Groceries)
inspect(sort(itemsets, by ='lift', decreasing = T)[1:10])
```

```
##              items                                  support     lift
## 592  {mayonnaise,mustard}                           0.001423 12.965
## 288  {hamburger meat,Instant food products}         0.003050 11.421
## 93   {detergent,softener}                           0.001118 10.600
## 139  {liquor,red/blush wine}                        0.002135 10.025
## 1408 {flour,sugar}                                   0.004982  8.463
## 210  {salty snack,popcorn}                           0.002237  8.192
## 1113 {ham,processed cheese}                          0.003050  7.071
## 101  {hamburger meat,sauces}                         0.001220  6.684
## 32   {cream cheese ,meat spreads}                   0.001118  6.605
## 404  {detergent,house keeping products}             0.001017  6.346
```

After seeing lift 12.97 for {mayonnaise,mustard}, do not be surprised if you find mustard next to mayonnaise on the shop shelf!

Apriori: Search for rules

When we switch to searching rules we need to change target='frequent itemsets' to target = 'rules'. As well, we should specify confidence (unless we are satisfied with default confidence=0.8)

```
rules <- apriori(Groceries,
                 parameter = list(support=.001,
                                  confidence=.5,
                                  minlen=2,
                                  target='rules' # to mine for rules
                                  ))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen target   ext
##         0.5    0.1    1 none FALSE           TRUE   0.001      2     10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
```

```
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 6 done [0.06s].
## writing ... [5668 rule(s)] done [0.01s].
## creating S4 object  ... done [0.01s].
```

```
summary(rules)
```

```
## set of 5668 rules
##
## rule length distribution (lhs + rhs):sizes
##    2    3    4    5    6
##   11 1461 3211  939   46
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.00    4.00    3.92    4.00    6.00
##
## summary of quality measures:
##     support         confidence        lift
##  Min.   :0.00102   Min.   :0.500   Min.   : 1.96
##  1st Qu.:0.00112   1st Qu.:0.545   1st Qu.: 2.46
##  Median :0.00132   Median :0.600   Median : 2.90
##  Mean   :0.00167   Mean   :0.625   Mean   : 3.26
##  3rd Qu.:0.00173   3rd Qu.:0.684   3rd Qu.: 3.69
##  Max.   :0.02227   Max.   :1.000   Max.   :19.00
##
## mining info:
##       data ntransactions support confidence
##  Groceries         9835   0.001        0.5
```

There were generated 5668 rules of length from 2 to 6. The summary statistics for support, confidence, and lift are self-explanatory. We can sort rules by any of those measures:

```
inspect(sort(rules, by='lift', decreasing = T)[1:5])
```

```
##     lhs                                      rhs                    support    confidence lift
## 53  {Instant food products,soda}          => {hamburger meat}       0.001220 0.6316     19.00
## 37  {soda,popcorn}                       => {salty snack}          0.001220 0.6316     16.70
## 444 {flour,baking powder}                => {sugar}                0.001017 0.5556     16.41
## 327 {ham,processed cheese}               => {white bread}          0.001932 0.6333     15.05
## 55  {whole milk,Instant food products} => {hamburger meat}       0.001525 0.5000     15.04
```

A stand with soda, popcorn, and salty snacks with lift 16.7 looks very promising idea for a department selling DVD's. In case there is a suspicion for spurious correlation chiSquared test is to the rescue:

```
quality(rules)$chi <- interestMeasure(rules, measure='chi', significance=T, Groceries)
inspect(sort(rules, by='lift', decreasing = T)[1:5])
```

```
##     lhs                                      rhs                    support    confidence lift     chi
## 53  {Instant food products,soda}          => {hamburger meat}       0.001220 0.6316     19.00 4.967e-48
## 37  {soda,popcorn}                       => {salty snack}          0.001220 0.6316     16.70 5.279e-42
## 444 {flour,baking powder}                => {sugar}                0.001017 0.5556     16.41 1.703e-34
## 327 {ham,processed cheese}               => {white bread}          0.001932 0.6333     15.05 1.109e-58
## 55  {whole milk,Instant food products} => {hamburger meat}       0.001525 0.5000     15.04 2.865e-46
```

Sub-setting rules and itemsets

Let's consider set of "arbitrary" rules generated with apriori

```
rules <- apriori(Groceries,
                 parameter = list(support=.001,
                                  confidence=.7,
                                  maxlen=5,
                                  target='rules' # to mine for rules
                                  ))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen target   ext
##         0.7    0.1    1 none FALSE            TRUE   0.001      1      5  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 done [0.03s].
## writing ... [1255 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(sort(rules, by="confidence", decreasing = T)[1:5])
```

```
##     lhs                                             rhs                  support  confidence   lift
## 25  {rice,sugar}                                 => {whole milk}         0.001220     1        3.914
## 52  {canned fish,hygiene articles}               => {whole milk}         0.001118     1        3.914
## 147 {root vegetables,butter,rice}                => {whole milk}         0.001017     1        3.914
## 205 {root vegetables,whipped/sour cream,flour} => {whole milk}         0.001729     1        3.914
## 213 {butter,soft cheese,domestic eggs}           => {whole milk}         0.001017     1        3.914
```

The power of subset function can be shown by choosing the following subset:

- rhs should be ‘bottled beer’
- confidence should be above .7
- results should be sorted by lift

```
inspect(sort(subset(rules,
                    subset=rhs %in% 'bottled beer' & confidence > .7),
                    by = 'lift',
                    decreasing = T))
```

```
##   lhs                                    rhs             support  confidence lift
## 2 {liquor,red/blush wine} => {bottled beer}     0.001932 0.9048     11.24
```

By now, we must be well aware of the fact that people buying “liquor” and “red wine” are almost certain to buy “bottled beer” (9 times out of 10), but not “canned beer”:

```
canned_rules <- apriori(Groceries,
                        parameter = list(support=.001,
                                         confidence=.01,
                                         maxlen=5,
```

```
                                target='rules' # to mine for rules
                            ))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen target   ext
##        0.01    0.1    1 none FALSE            TRUE   0.001      1      5  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 done [0.03s].
## writing ... [40827 rule(s)] done [0.01s].
## creating S4 object  ... done [0.01s].
```

```
inspect(subset(canned_rules,
               subset=lhs %ain% c("liquor", "red/blush wine") & rhs %in% 'canned beer' ))
```

Check the outcome of the above function!

Be aware the presence of the %in% operator. There is a set of operators that provide the result of "matching" for associations, transactions and itemMatrices. Important arguments of this match subset:

- lhs - means left hand side, or antecendent
- rhs - mean right hand side, or consequent
- items - items, that make up itemsets
- %in% - matches any
- %ain% - matches all
- %pin% - matches partially
- default - no restrictions applied
- & - additional restrictions on lift, confidence etc.

Example 1

Both "whole milk" and "yogurt" must be present and rule's confidence must be higher than .95

```
inspect(subset(rules, subset=items %ain% c("whole milk","yogurt") & confidence >.95))
```

```
##     lhs                                                rhs              support  confidence lift
## 915 {tropical fruit,grapes,whole milk,yogurt}     => {other vegetables} 0.001017 1          5.168
## 942 {tropical fruit,root vegetables,yogurt,oil}   => {whole milk}       0.001118 1          3.914
## 952 {root vegetables,other vegetables,yogurt,oil} => {whole milk}       0.001423 1          3.914
```

Example 2

Both "whole milk" and "yogurt" must be present in lhs and rule's confidence must be higher than .9

```
inspect(subset(rules, subset=lhs %ain% c("whole milk","yogurt") & confidence >.9))
```

```
##     lhs                                                rhs              support  confidence lift
## 901 {root vegetables,whole milk,yogurt,rice}  => {other vegetables}  0.001322 0.9286     4.799
## 915 {tropical fruit,grapes,whole milk,yogurt} => {other vegetables}  0.001017 1.0000     5.168
## 953 {root vegetables,whole milk,yogurt,oil}   => {other vegetables}  0.001423 0.9333     4.824
```

Example 3

"Bread" must be present in lhs: any type of "bread" – "white bread", "brown bread" – both qualify. "Whole milk" must be present in rhs "as is". Confidence of the rule must be higher than .9

```
inspect(subset(rules, subset= lhs %pin% "bread" & rhs %in% "whole milk" & confidence > .9))
```

```
##      lhs                                                    rhs                 support  confidence lift
## 611  {root vegetables,butter,white bread}                   => {whole milk}     0.001118 0.9167     3.588
## 997  {root vegetables,other vegetables,butter,white bread}  => {whole milk}     0.001017 1.0000     3.914
## 1088 {pip fruit,root vegetables,other vegetables,brown bread}  => {whole milk}  0.001220 0.9231     3.613
## 1095 {root vegetables,other vegetables,rolls/buns,brown bread} => {whole milk}  0.001017 0.9091     3.558
```

It appears from this limited sample, that people buying "whole milk" do not have any preferences for either "white" or "brown bread" (perhaps contingency crossTable would show a different result).

Example 4

Let's see what we can expect at rhs with confidence higher than .7 if we have both "flour" and "whole milk" on the lhs.

```
inspect(subset(rules, subset= lhs %ain% c("flour","whole milk") & confidence>.7))
```

```
##     lhs                                    rhs                  support confidence lift
## 208 {citrus fruit,whole milk,flour} => {other vegetables}  0.00122 0.75       3.876
```

Example 5

Finally, let's go fishing for substitute products. So far we were concerned with complements, i.e. items and itemsets that showed high lift. Let's consider case "Bottled beer Vs. Canned beer" and prove that people tend to buy either one or the other, and rarely do they buy both, qualifying these two as substitute products.

```
rules <- apriori(Groceries,
                 parameter = list(support=.001,
                                  conf = .01,
                                  minlen=2,
                                  maxlen=2,
                                  target='rules'
                                  ))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen target   ext
##        0.01    0.1    1 none FALSE           TRUE   0.001      2      2  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
```

```
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [5818 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Let's only look at the rules where "beer" is present at both left- and right-hand-side of the rule and add chiSquared p-value to prove statistical significance of our findings:

```
quality(rules)$chi  <- interestMeasure(rules, measure='chi', significance=T, Groceries)
inspect(subset(rules, lhs %pin% 'beer' & rhs %pin% 'beer'))
```

```
##     lhs                              rhs            support  confidence       lift   chi
## 4785 {canned beer}  => {bottled beer}  0.002644 0.03403    0.4226 8.743e-07
## 4786 {bottled beer} => {canned beer}   0.002644 0.03283    0.4226 8.743e-07
```

The results so far are quite telling: there are indeed people who buy both bottled and canned beer at once

```
crossTable(Groceries)['canned beer','bottled beer']
## [1] 26
```

- the probability of a consecutive purchase (confidence) is pretty small: ~3%
- this is despite both bottled beer and canned beer being pretty popular purchases

```
crossTable(Groceries)['canned beer','canned beer']
## [1] 764
```

```
crossTable(Groceries)['bottled beer','bottled beer']
## [1] 792
```

All these figures, combined with statistically significant lift below 1 (chi ~ 1e-6) tells us that "bottled beer" and "canned beer" do behave as substitutes.

Visualization

The last step is visualization. Let's say we want to map out the rules in a graph. We can do that with another library called "arulesViz".

```
plot(rules,method="graph",interactive=TRUE,shading=NA)
```