MARKET BASKET ANALYSIS IN GROCERIES DATASET USING R

Business and Data Analytics
Project Report

Submitted by
Laltendu Das [15MCMI22]
Uma Revathi K N[15MCMI20]
Rosni K V [15MCMI15]

Under the Guidance of
Dr. V.Ravi, Professor, IDRBT

November 2015

Table of Contents

List of Figures

List of Tables

1 Abstract

Market Basket Analysis is one of the most common and useful types of data analysis for marketing and retailing. The purpose of market basket analysis is to determine what products customers purchase together. It takes its name from the idea of customers throwing all their purchases into a shopping cart (a "market basket") during grocery shopping. Knowing what products people purchase as a group can be very helpful to a retailer or to any other company. A store could use this information to place products frequently sold together into the same area, while a catalog or World Wide Web merchant could use it to determine the layout of their catalog and order form. Direct marketers could use the basket analysis results to determine what new products to offer their prior customers.

There are many ways to see the similarities between items. These are techniques that fall under the general umbrella of association. We consider Association Mining in the groceries dataset. The outcome of this type of technique, in simple terms, is a set of rules that can be understood as "if this, then that". We then give the description of the Apriori Algorithm and Frequency Pattern Growth. This study illustrates the use of data mining techniques to construct association rules. Also, it illustrates the rules in a graph. The study also include measures for rule strength using R language.

2 Introduction

One of the challenges for companies that have invested heavily in customer data collection is how to extract important information from their vast customer databases and product feature databases, in order to gain competitive advantage. Market basket analysis has been intensively used in many companies as a means to discover product associations and base a retailers promotion strategy on them. Market basket analysis is one of the data mining methods focusing on discovering purchasing patterns by extracting associations or co-occurrences from a stores transactional data[1].

The input for the market basket analysis is a dataset of purchases. A market basket is composed of items bought together in a single trip to a store. The most significant attributes are the transaction identification and item identification. While ignoring the quantity bought and the price. Each transaction represents a purchase, which occurred in a specific time and place. This purchase can be linked to an identified customer(usually carrying a card) or to a non-identified customer.

The dataset with multiple transactions can be shown in a relational table (transaction,item). Corresponding to each attribute there is a set called domain. The table(transaction,item) is a set of all transactions T=$(T_1, T_2, T_3, ...T_n)$ where each transaction contain a subset of items $T_k = (I_a, I_b, I_c, ...)$[2]. Based on the attributes (transaction, item), the market basket will be defined as the N items that are bought together more frequently. Once the market basket with N items is known, we can move on to cross-selling. The next step is to identify all the customers having bought N -m items of the basket and suggest the purchase of some m missing items. In order to make decisions in marketing, the market basket analysis is a powerful tool supporting the implementation of cross-selling strategies. For instance, if a specific customer's buying profile fits into a identified market basket, the next item will be proposed.

Data mining finds interesting patterns from databases such as association rules, correlations, sequences, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems. Association rule mining finds interesting association or correlation relationships among a large set of data items. An association rule is a division of each item set into two subset with one subset, the antecedent, thought of as preceding the other subset, the consequent. There are more association rules than there are itemsets. The Apriori algorithm discussed in coming section deals with the large number of rules problem by using selection criteria that reflect the potential utility of association rules.

Association rules are derived from the frequent itemsets using support and confidence as threshold levels. The interestingness measures like support and confidence plays a vital role in the association analysis. The support is defined as percentage of transactions that contained in the rule and is given by

Support = (# of transactions involving A and B) / (total number of transactions).

The other factor is confidence it is the percentage of transactions that contain B if they contain A.

Confidence = Probability (B if A) = P(B / A)

Confidence = (# of transactions involving A and B) / (total number of transactions that have A).

Item sets that satisfy minimum support and minimum confidence are called strong association rule. For this study we used groceries dataset which contains: a collection of receipts with each line representing 1 receipt and the items purchased.

Although the Apriori algorithm processes data in a different manner from the algorithms FPGrowth and Create Association Rules, gives a lot of insight about our data. A major advantage of the algorithm FP Growth compared to others of the same type is the fact that it uses only two scans of the data and it can be applied to larger data sets. The frequent sets of articles are searched for positive entries from the data base. The entry data set must contain only bionominal attributes. If the data contains other types of attributes preprocessing operators must be used to transform the data set. In RapidMiner the process of exploitation of frequent sets is divided into two parts, first are generated all frequent sets of articles after which are generated the association rules from the frequent sets.

3 Algorithm Description

3.1 Apriori Algorithm

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. It takes all of the transactions in the database into account in order to define the market basket. The market basket can be represented with association rules, with a left and a right side Left $\implies$ Right. For instance, given an itemset $\{A, B, C\}$ the rule $\{B, C\} \implies \{A\}$ should be read as follows if a customer bought $\{B, C\}$ he would probably buy $\{A\}$.

The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties. Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k + 1)-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted L_1. Next,L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3 , and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database.To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property, explained below, is used to reduce the search space.[3]

Apriori property :*All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, *min_sup*, then I is not frequent; that is,$P(I) < min\_sup$. If an item A is added to the itemset I, then the resulting itemset (i.e.,$I \bigcup A$) cannot occur more frequently than I. Therefore, I $\bigcup$ A is not frequent either; that is,$P(I \bigcup A) < min\_sup$.

This property belongs to a special category of properties called antimonotone in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well.* It is called *antimonotone* because the property is monotonic in the context of failing a test. The algorithm pseudo code is given below.

Step 2 generates 1-itemsets, i.e., the frequent items. These 1-itemsets are stored in L_1 list, which will be used to generate C_2. C_2 is the list of the candidate 2-itemsets. The generationoperation of C_{k+1} from L_k is done by *apriori_gen* algorithm. *Apriori_gen* takes two different last items k-itemsets to generate candidate (k+1)-itemset by joining the similar k-1 items with the last different items. For example, let X and Y are two frequent 3-itemsets in L_3, such that X=abc and Y=abd , then the generated candidate 4-itemset is abcd which will be added to C_4 list.

Apriori_gen performs the first complex pruning step to exclude infrequent candidates. This is done in step 24. It degenerates the generated candidate (k+1)-itemset to its k-itemset subsets. If there is any subset not member in L_k, then its candidate will be removed from C_{k+1}. Steps 7 to 10 are responsible for the second pruning step. The core of these steps is the function subset that checks the subsetness of a candidate in a transaction. Subset function depends on a hash tree, which has been built gradually during the progress of mining the large itemsets in each iteration.

The outputs of the Apriori algorithm are easy to understand and many new patterns can be identified. However, the sheer number of association rules may make the interpretation of the results difficult. A

Algorithm 1 Apriori algorithm

```
1: procedure APRIORI–(in D : transactional Database ; minsup: minimum support)
2:     L_1={Large1 − itemsets}
3:     k=2
4:     while ( L_{k−1} ≠ φ) do
5:         C_k=apriori_gen(L_{k−1})
6:         for all transactions t in D do
7:             C^t = subset(C_k, t)
8:             for all candidates c ∈ C^t do
9:                 c.count=c.count+1
10:            end for
11:        end for
12:    end while
13:    L_k={ c ∈ C_k  c.count ≥ minsup }                    ▷ Second Pruning Step
14:    k=k+1
15: end procedure
16: procedure APRIORI_gen(L_{k−1})
17:    C_k=φ
18:    for all itemsets X ∈ L_{k−1} and Y ∈ L_{k−1} do
19:        if X_1=Y_1 ∧ ....∧ X_{k−2}=Y_{k−2} ∧ X_{k−1} < Y_{k−1} then
20:            C=X_1X_2...X_{k−1}Y_{k−1}
21:            add C to C_k
22:        end if                                           ▷ First Pruning Step
23:    end for
24:    Delete candidates itemsets in C_k whose any subset is not in L_{k−1}
25: end procedure
```

Table 1: Transactions

TID	Items
1	{ Bread, Milk }
2	{ Bread, Diapers, Beer, Eggs }
3	{ Milk, Diapers, Beer, Cola }
4	{ Bread, Milk, Diapers, Beer }
5	{ Bread, Milk, Diapers, Cola }

second weakness of the algorithm is the computational times when it searches for large itemsets, due to the exponential complexity of the algorithm. Figure[1] provides a high-level illustration of the frequent itemset generation part of the Apriori algorithm for the transactions shown in Table [1]

3.2 FP Growth algorithm for frequent pattern generation

An interesting method to frequent pattern mining without generating candidate pattern is called frequent-pattern growth, or simply FP-growth, which adopts a divide-and-conquer strategy. First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases (a special kind of projected database), each associated with one frequent item or pattern fragment, and mines each such database separately[7].

It overcomes the two major problems of Apriori algorithm. As in Apriori algorithm.

1. It do not generates large number of candidate items.
2. No repeated scan of original database is required.

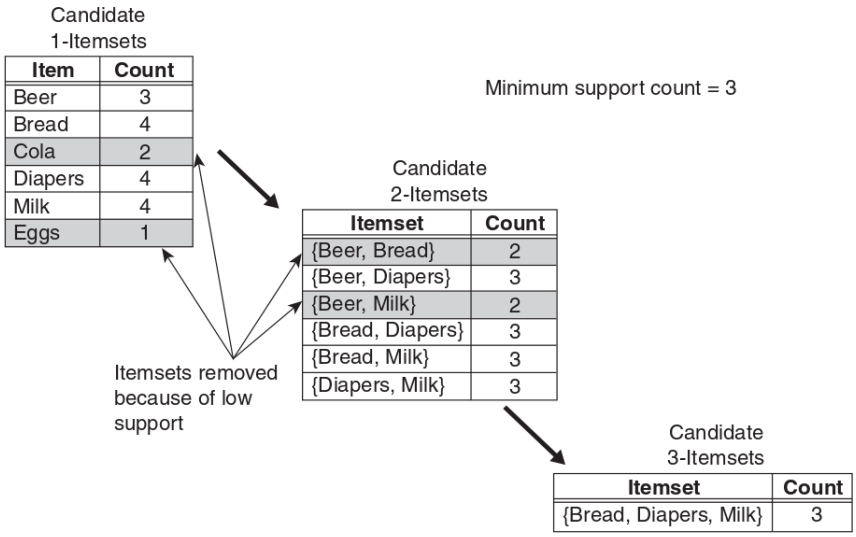


Figure 1: Illustration of frequent itemset generation using the Apriori algorithm.

3.2.1 FP-Tree Representation

An FP-tree is a compressed representation of the input data. It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree. As different transactions can have several items in common, their paths may overlap. The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure. If the size of the FP-tree is small enough to fit into main memory, this will allow us to extract frequent itemsets directly from the structure in memory instead of making repeated passes over the data stored on disk.

Figure[2] shows a data set that contains ten transactions and five items. The structures of the FP-tree after reading the first three transactions are also depicted in the diagram. Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path. Initially, the FP-tree contains only the root node represented by the null symbol. The FP-tree is subsequently extended in the following way:

- The data set is scanned once to determine the support count of each item. Infrequent items are discarded, while the frequent items are sorted in decreasing support counts. For the data set shown in Figure [2], a is the most frequent item, followed by b,c,d, and e.
- The algorithm makes a second pass over the data to construct the FP-tree. After reading the first transaction,{ a, b}, the nodes labeled as a and b are created. A path is then formed from *null* $\Longrightarrow$ a $\Longrightarrow$ b to encode the transaction. Every node along the path has a frequency count of 1.
- After reading the second transaction,{ b,c,d} , a new set of nodes is created for items b , c,and d. A path is then formed to represent the transaction by connecting the nodes *null* $\Longrightarrow$ b $\Longrightarrow$ c $\Longrightarrow$ d. Every node along this path also has a frequency count equal to one. Although the first two transactions have an item in common, which is b, their paths are disjoint because the transactions do not share a common prefix.
- The third transaction,{ a, c, d, e }, shares a common prefix item (which is a) with the first transaction. As a result, the path for the third transaction, *null* $\Longrightarrow$ a $\Longrightarrow$ c $\Longrightarrow$ d $\Longrightarrow$ e, overlaps with the path for the first transaction, *null* $\Longrightarrow$ a $\Longrightarrow$ b. Because of their overlapping path, the frequency count for node a is incremented to two, while the frequency counts for the newly created nodes, c, d, and e, are equal to one.

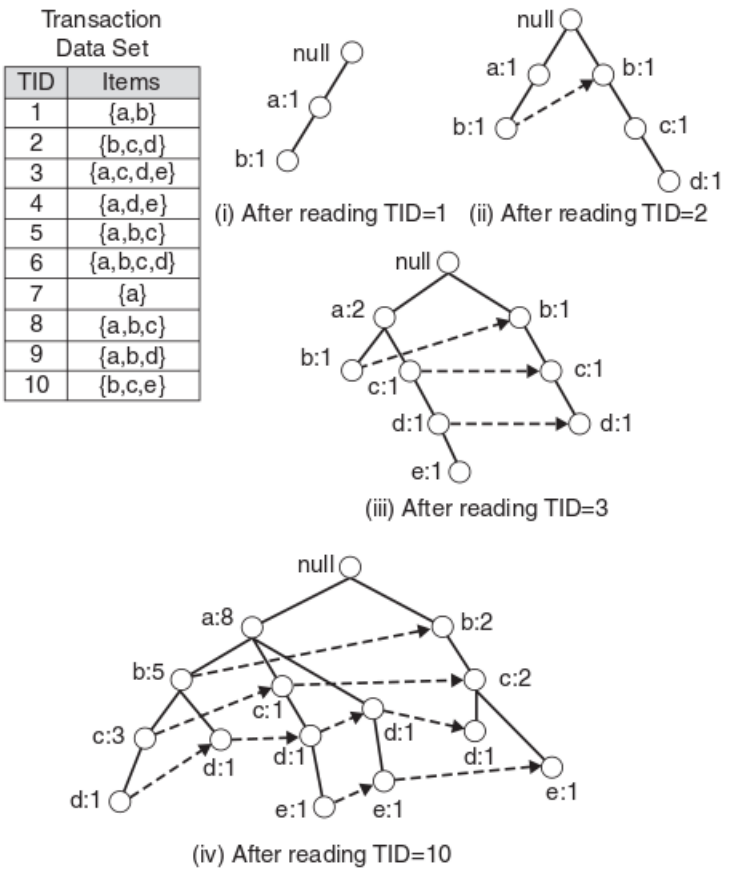


Figure 2: contruction of an fp tree

- This process continues until every transaction has been mapped onto one of the paths given in the FP-tree. The resulting FP-tree after reading all the transactions is shown at the bottom of Figure [2].

3.2.2 Frequent Itemset Generation in FP-Growth Algorithm

FP-growth is an algorithm that generates frequent itemsets from an FP-tree by exploring the tree in a bottom-up fashion. The algorithm pseudo code is given below. FP-growth is an interesting algorithm because it illustrates how a compact representation of the transaction data set helps to efficiently generate frequent itemsets. In addition, for certain transaction data sets, FP-growth outperforms the standard Apriori algorithm by several orders of magnitude. The run time performance of FP-growth depends on the compaction factor of the data set. If the resulting conditional FP-trees are very bushy (in the worst case, a full prefix tree), then the performance of the algorithm degrades significantly because it has to generate a large number of subproblems and merge the results returned by each subproblem.

4 Data set description

The Groceries data set we used for this study contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The data set is provided by Michael Hahsler, Kurt Hornik and Thomas Reutterer[3]. The packages *arules* and *arulesViz* are used here for association mining

Algorithm 2 Algorithm FP-growth

procedure FP-GROWTH–($Tree$, α)
  if Tree contains a single path P **then**
    for each combination (denoted as β) of the node in the path P **do**
      generate pattern $\beta \bigcup \alpha$ with $support\_count$ = minimum support count of nodes in β;
    end for
  else
    for each a_i in the header of Tree **do**
      generate pattern $\beta = a_i \bigcup \alpha$ with $support\_count = a_i.support\_count$;
      construct β's conditional pattern base and then β's conditional $FP\_tree\ Tree_\beta$;
      if $Tree_\beta \neq \phi$ **then**
        call $FP\_growth(\text{Tree}_\beta, \beta)$;
      end if
    end for

tasks.

```
library(arules)
library(arulesViz)
groceries <- read.transactions("/home/freestyler/BDA_project2/groceries.csv", sep = ",")
```

We can see how the items are organized in the dataset by inspect method. The data set contains 9835 transactions and the items are aggregated to 169 categories.

```
inspect(groceries[1:2])

##   items
## 1 {citrus fruit,
##    margarine,
##    ready soups,
##    semi-finished bread}
## 2 {coffee,
##    tropical fruit,
##    yogurt}

print(dim(groceries))

## [1] 9835  169
```

The summary of the groceries dataset is given by:

```
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
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55
##   16   17   18   19   20   21   22   23   24   26   27   28   29   32
##   46   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
```

we can visualize each item with support greater than 0.025 shown in Figure [3].

```
itemFrequencyPlot(groceries,support=0.025,cex.names=0.8,xlim=c(0,0.3),type="relative",
                  horiz=TRUE,col="dark red",las=1,xlab=paste("Proportions of Market
                  Baskets containing Item","\n(Item Relative Frequency or Support)"))
```

5 Result and Discussions

5.1 Apriori algorithm

To mine the rules, we have to pass the minimum required support and confidence. We set the minimum support to 0.001; Support is the fraction of which our item set occurs in our dataset. We set the minimum confidence of 0.8 ;Confidence is the probability that a rule is correct for a new transaction with items on the left. Lift is another measure, the ratio by which by the confidence of a rule exceeds the expected confidence. If the lift is 1 it indicates that the items on the left and right are independent.

```
rules = apriori(groceries, parameter=list(support=0.001, confidence=0.8))
```

```
##
## parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.8    0.1    1 none FALSE            TRUE   0.001      1     10
##  target   ext
##   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 [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
```

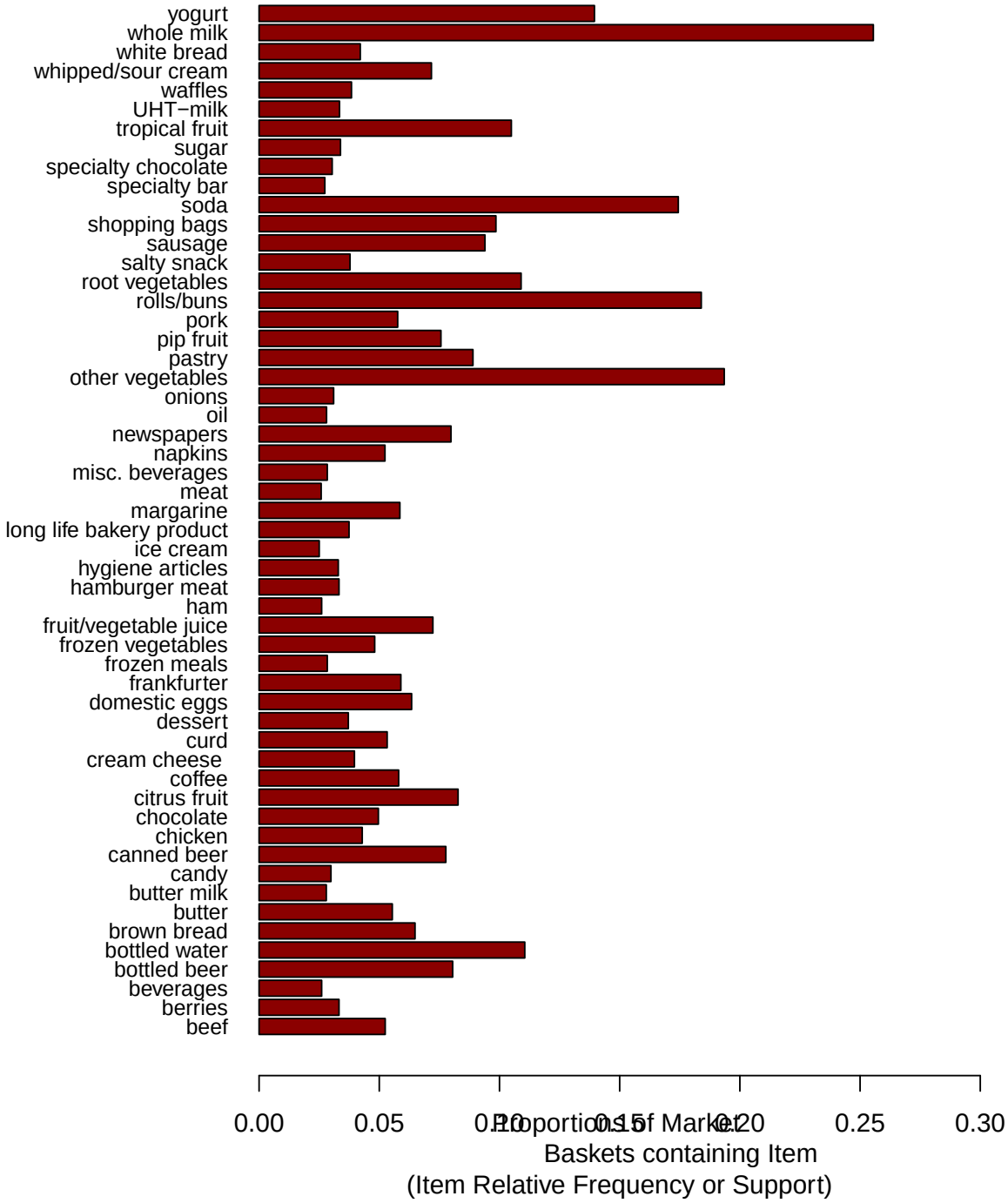


Figure 3: Item frequency

```
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 6 done [0.02s].
## writing ... [410 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

rules<-sort(rules, by="lift", decreasing=TRUE)
```

Sometimes, rules will repeat. Eliminate the redundant rules by:

```
subset.matrix <- is.subset(rules, rules)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1
rules.pruned <- rules[!redundant]
rules<-rules.pruned
```

As a result the number of rules reduced from 410 to 370.

```
summary(rules)

## set of 370 rules
##
## rule length distribution (lhs + rhs):sizes
##   3   4   5   6
##  29 226 111   4
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   4.000   4.000   4.243   5.000   6.000
##
## summary of quality measures:
##     support           confidence          lift
##  Min.   :0.001017   Min.   :0.8000   Min.   : 3.131
##  1st Qu.:0.001017   1st Qu.:0.8333   1st Qu.: 3.312
##  Median :0.001220   Median :0.8462   Median : 3.588
##  Mean   :0.001257   Mean   :0.8659   Mean   : 3.974
##  3rd Qu.:0.001322   3rd Qu.:0.9091   3rd Qu.: 4.373
##  Max.   :0.003152   Max.   :1.0000   Max.   :11.235
##
## mining info:
##       data ntransactions support confidence
##  groceries          9835   0.001        0.8

options(digits=2)
inspect(rules[1:15])

##    lhs                         rhs               support confidence lift
## 1  {liquor,
##     red/blush wine}        => {bottled beer}     0.0019       0.90 11.2
## 2  {citrus fruit,
##     fruit/vegetable juice,
##     other vegetables,
##     soda}                  => {root vegetables}  0.0010       0.91  8.3
## 3  {oil,
##     other vegetables,
##     tropical fruit,
```

```
##     whole milk,
##     yogurt}                => {root vegetables}  0.0010       0.91  8.3
## 4  {citrus fruit,
##     fruit/vegetable juice,
##     grapes}                => {tropical fruit}   0.0011       0.85  8.1
## 5  {other vegetables,
##     rice,
##     whole milk,
##     yogurt}                => {root vegetables}  0.0013       0.87  8.0
## 6  {oil,
##     other vegetables,
##     tropical fruit,
##     whole milk}            => {root vegetables}  0.0013       0.87  8.0
## 7  {ham,
##     other vegetables,
##     pip fruit,
##     yogurt}                => {tropical fruit}   0.0010       0.83  7.9
## 8  {beef,
##     citrus fruit,
##     other vegetables,
##     tropical fruit}        => {root vegetables}  0.0010       0.83  7.6
## 9  {butter,
##     cream cheese ,
##     root vegetables}       => {yogurt}           0.0010       0.91  6.5
## 10 {butter,
##     sliced cheese,
##     tropical fruit,
##     whole milk}            => {yogurt}           0.0010       0.91  6.5
## 11 {cream cheese ,
##     curd,
##     other vegetables,
##     whipped/sour cream}    => {yogurt}           0.0010       0.91  6.5
## 12 {butter,
##     other vegetables,
##     tropical fruit,
##     white bread}           => {yogurt}           0.0010       0.91  6.5
## 13 {pip fruit,
##     sausage,
##     sliced cheese}         => {yogurt}           0.0012       0.86  6.1
## 14 {butter,
##     curd,
##     tropical fruit,
##     whole milk}            => {yogurt}           0.0012       0.86  6.1
## 15 {butter,
##     tropical fruit,
##     white bread}           => {yogurt}           0.0011       0.85  6.1
```

Now we can target items to generate rules. For example, there are two types of targets that we are interested with an example of "whole milk".
1.What are customers likely to buy before buying whole milk
2.What are customers likely to buy if they purchase whole milk
For that we can adjust our apriori function as follows:

```
rules1<-apriori(data=groceries, parameter=list(supp=0.001,conf = 0.08),
               appearance = list(default="lhs",rhs="whole milk"),
               control = list(verbose=F))
rules1<-sort(rules1, decreasing=TRUE,by="lift")
inspect(rules1[1:5])

##   lhs                     rhs          support confidence lift
## 1 {rice,
##    sugar}              => {whole milk}  0.0012          1  3.9
## 2 {canned fish,
##    hygiene articles}   => {whole milk}  0.0011          1  3.9
## 3 {butter,
##    rice,
##    root vegetables}    => {whole milk}  0.0010          1  3.9
## 4 {flour,
##    root vegetables,
##    whipped/sour cream} => {whole milk}  0.0017          1  3.9
## 5 {butter,
##    domestic eggs,
##    soft cheese}        => {whole milk}  0.0010          1  3.9
```

Likewise, we can set the left hand side to be "whole milk" and find its antecedents.

```
rules1<-apriori(data=groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2),
               appearance = list(default="rhs",lhs="whole milk"),
               control = list(verbose=F))
inspect(rules1[1:5])

##   lhs             rhs                support confidence lift
## 1 {whole milk} => {tropical fruit}    0.042       0.17  1.6
## 2 {whole milk} => {root vegetables}   0.049       0.19  1.8
## 3 {whole milk} => {soda}              0.040       0.16  0.9
## 4 {whole milk} => {yogurt}            0.056       0.22  1.6
## 5 {whole milk} => {rolls/buns}        0.057       0.22  1.2
```

We can map out the rules in a graph. We can do that with another library *arulesViz*. Rules with high lift typically have low support. Graph based visualization is shown in Figure [4]. The most interesting rules reside on the support/confidence border which can be clearly seen in this plot.

```
library(arulesViz)
subrules2 <- head(sort(rules, by="lift"), 10)

plot(subrules2, method="graph",control=list(type="items",main=""))
```

The visualization clearly shows some of the rules, and scatter plot also we can use with jitter to reduce occlusion Figure [5]. Both visualizations clearly show that there exists a rule ({liquor, red/blush wine} $\implies$ {bottled beer}) with high support, confidence and lift[5].

```
library("RColorBrewer")
plot(rules,control=list(col=brewer.pal(11,"Spectral")),main="")
```

Mining association rules often results in a very large number of found rules, leaving the analyst with the task to go through all the rules and discover interesting ones. Sifting manually through large sets of rules is time consuming and strenuous. Visualization has a long history of making large amounts of data

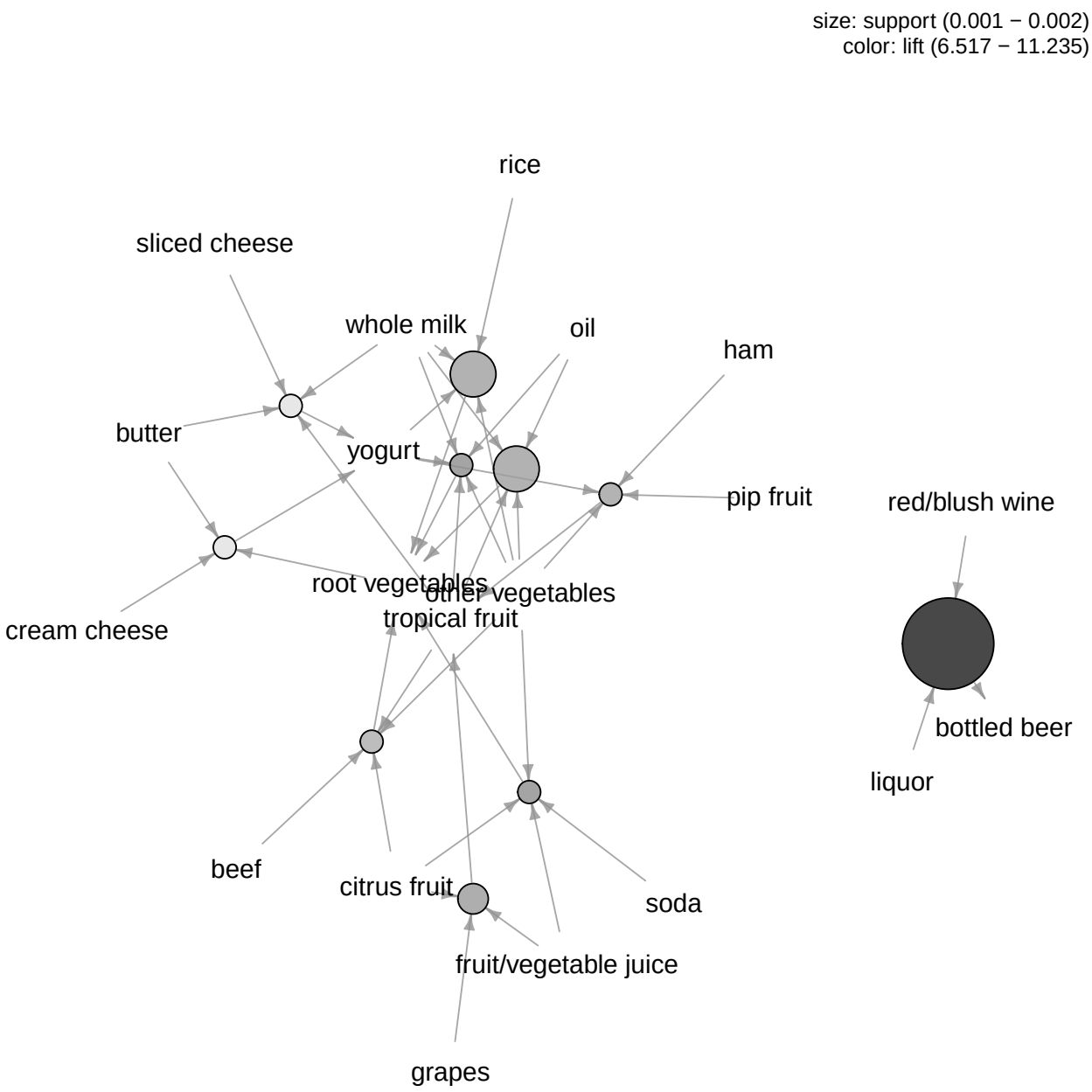


Figure 4: Graph based visualization

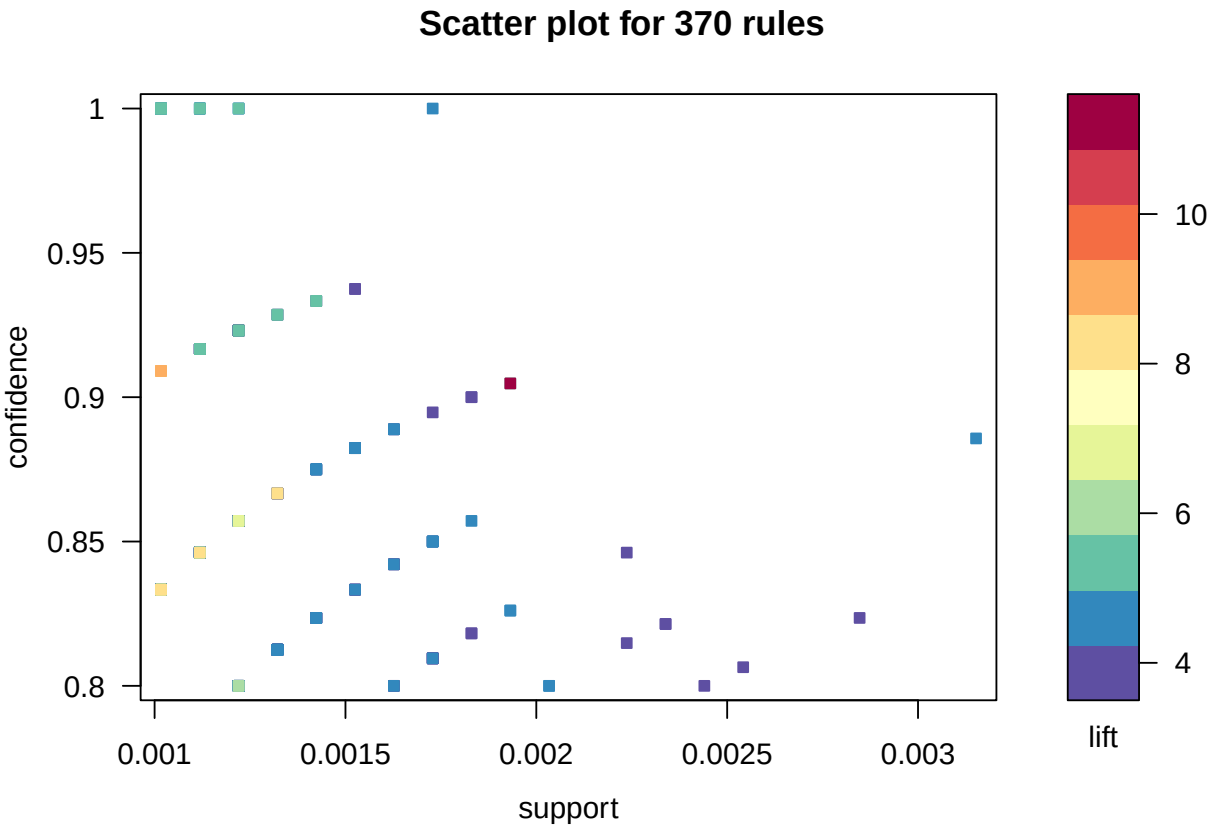
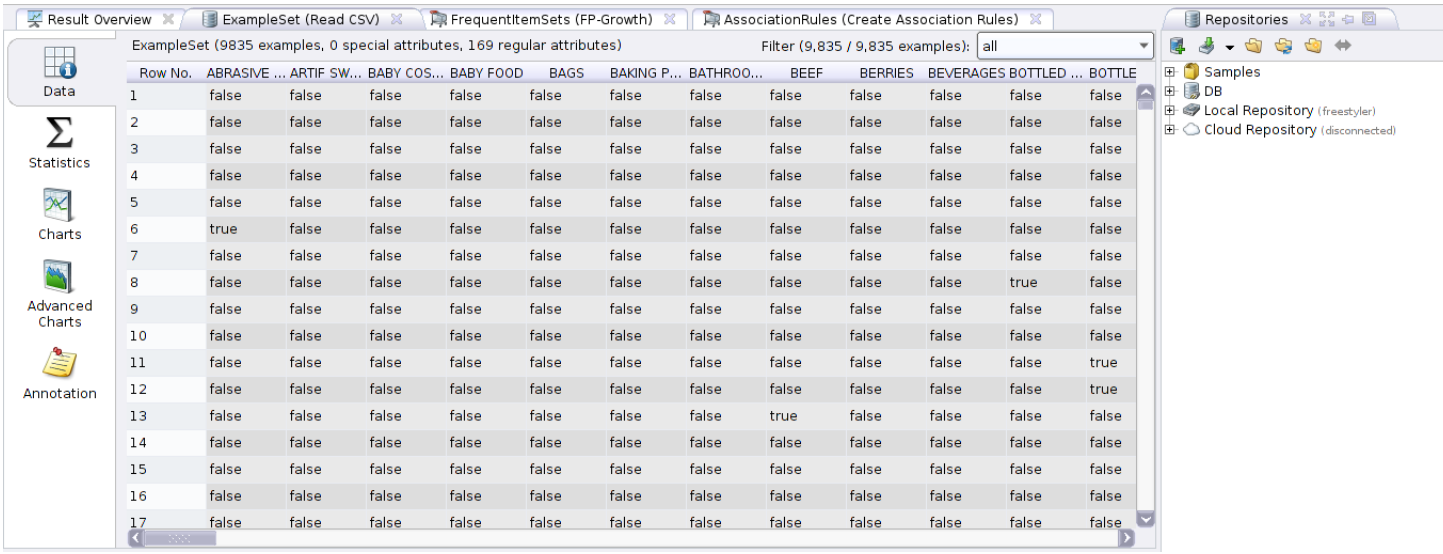


Figure 5: Scatter Plot

better accessible using techniques like selecting and zooming. However, most association rule visualization techniques are still falling short when it comes to a large number of rules[6].

5.2 FP Growth algorithm

The rule mining using FP Growth algorithm is implemented using tool **RAPIDMINER**. But the input data must be in binomial format and original dataset was in transactional data format. Data preprocessing done using R.



Row No.	ABRASIVE ...	ARTIF SW...	BABY COS...	BABY FOOD	BAGS	BAKING P...	BATHROO...	BEEF	BERRIES	BEVERAGES	BOTTLED ...	BOTTLE
1	false	false	false	false	false	false	false	false	false	false	false	false
2	false	false	false	false	false	false	false	false	false	false	false	false
3	false	false	false	false	false	false	false	false	false	false	false	false
4	false	false	false	false	false	false	false	false	false	false	false	false
5	false	false	false	false	false	false	false	false	false	false	false	false
6	true	false	false	false	false	false	false	false	false	false	false	false
7	false	false	false	false	false	false	false	false	false	false	false	false
8	false	false	false	false	false	false	false	false	false	false	true	false
9	false	false	false	false	false	false	false	false	false	false	false	false
10	false	false	false	false	false	false	false	false	false	false	false	false
11	false	false	false	false	false	false	false	false	false	false	false	true
12	false	false	false	false	false	false	false	false	false	false	false	true
13	false	false	false	false	false	false	false	true	false	false	false	false
14	false	false	false	false	false	false	false	false	false	false	false	false
15	false	false	false	false	false	false	false	false	false	false	false	false
16	false	false	false	false	false	false	false	false	false	false	false	false
17	false	false	false	false	false	false	false	false	false	false	false	false

Figure 6: input table in rapid miner

```
groceries <- read.transactions("/home/freestyler/BDA_project2/groceries.csv", sep = ",")
mm <- t(as(groceries,"ngCMatrix"))
new<- mm*1
new<- as.matrix(new)
library("MASS")
write.matrix(format(new, scientific=FALSE),file ="/home/freestyler/dat3.csv", sep=",")
```

Figure [6] shows the table generated in CSV format and then imported to Rapidminer, as we can see in that 9835 examples and 169 regular attributes(items) are present.

For the purpose of rule generation, we need to use FP Growth operator which just accepts the nominal attributes. Hence from the data transformation operators available, selected the text to nominal operator and connected to input data as shown in Figure [7].

For this Frequency Pattern analysis, search field of the operator tab gives the FP Growth operator, which added to our model. One important parameter of this operator is Min Support, it is the number of times that the rule did occur, divided by the number of observations in the data set. Create Association Rule operator added to the model, this operator generates both a set of rules (through the rul port) and a set of associated items (through the ite port). One of the influential parameters of this operator is Min Confidence. Confident percent is a measure of how confident we are that when one attribute is flagged as true, the associated attribute will also be flagged as true. It is gained by dividing the number of times that a rule occurs by the number of times that it could have occurred. The whole model is shown in Figure [8].

As a result of the above model, frequency itemset generated shown in Figure[9]. The rules generated with high lift value along with other performance measures is shown in Figure [10]. And the rule description from

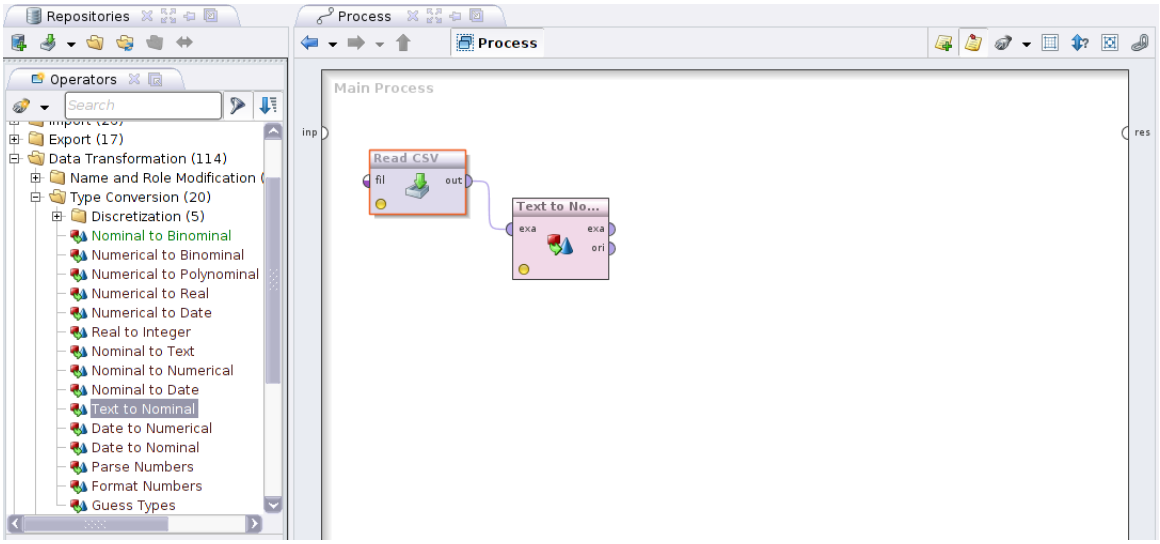


Figure 7: data type conversion

the FP Growth algorithm is given in Figure [11], as a part of visualization the graph generated is shown in Figure[12].

6 Conclusion

In this study we analysed the groceries dataset, With package arules we provide the basic infrastructure which enables us to mine associations and analyze and manipulate the results. Package *arules* gives simple interface to manipulate and analyze transactional data, set of itemsets and rules with subset selection and sorting.

Frequent itemsets discovered depends on the values of parameters like support, confidence and lift. After generating rules from the dataset, eliminated many redundant rules i.e, rules reduced from 410 to 370. Moreover rules generated based on target item also, as discussed in Section 5.1. It is difficult to go through all the rules, hence the visualization methods Scatter plot, Frequency plot, Graph visualization etc explored in this study.

FP Growth is faster than other association mining algorithms. The algorithm reduces the total number candidate itemsets by producing the compressed version of the database in terms of an FP Tree. The FP Tree stores relevant information and allows for the efficient discovery of frequent itemsets. This study concludes that FP growth algorithm generates 83 rules on our dataset with minimum support count 0.95, and apriori algorithm which genrates 370 rules with minimum support count 0.001.

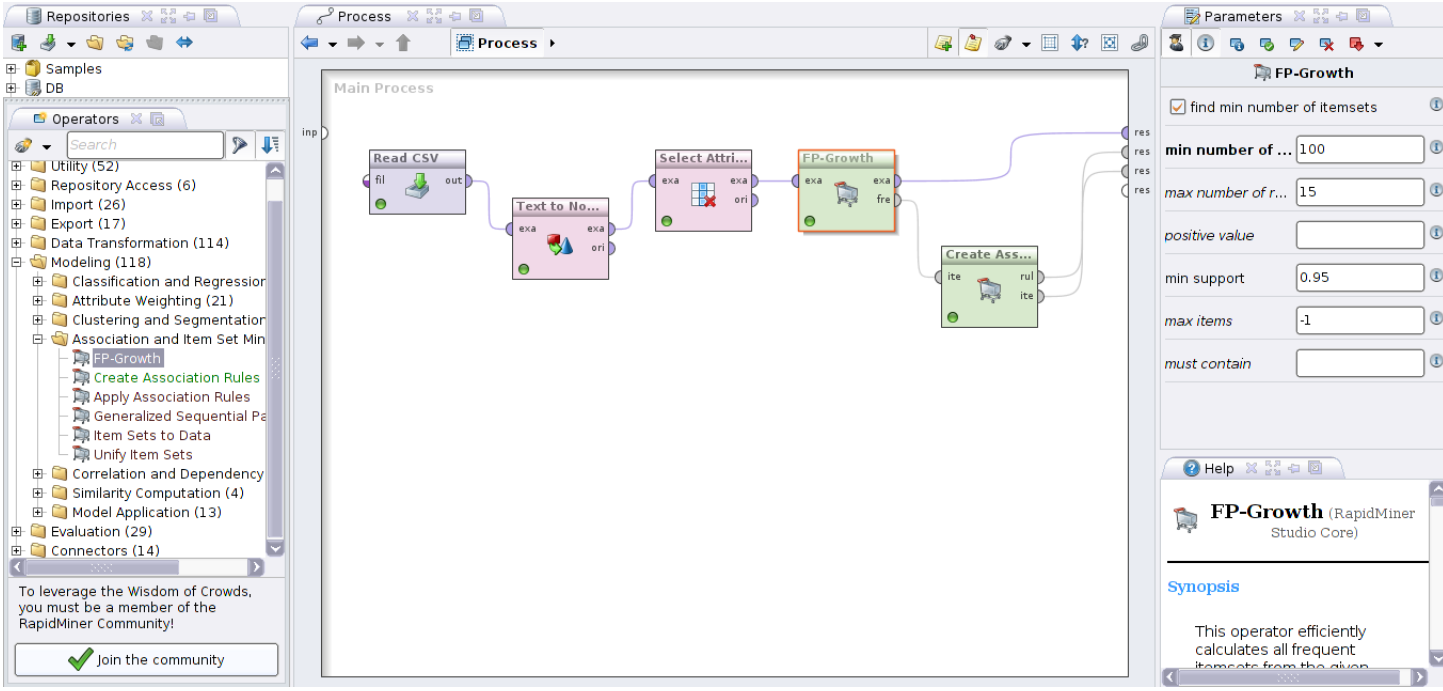


Figure 8: Rule mining using FP Growth

References

[1] Loraine Charlet Annie M.C and Ashok Kumar D *Market Basket Analysis for a Supermarket based on Frequent Itemset Mining* , IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 3, September 2012.

[2] Luis Cavique, *A Scalable Algorithm for the Market Basket Analysis*. ESCS, Instituto Politecnico de Lisboa, Portugal.

[3] Jiawei Han and Micheline Kamber, *Data Mining:Concepts and Techniques*, Second Edition, University of Illinois at Urbana-Champaign.

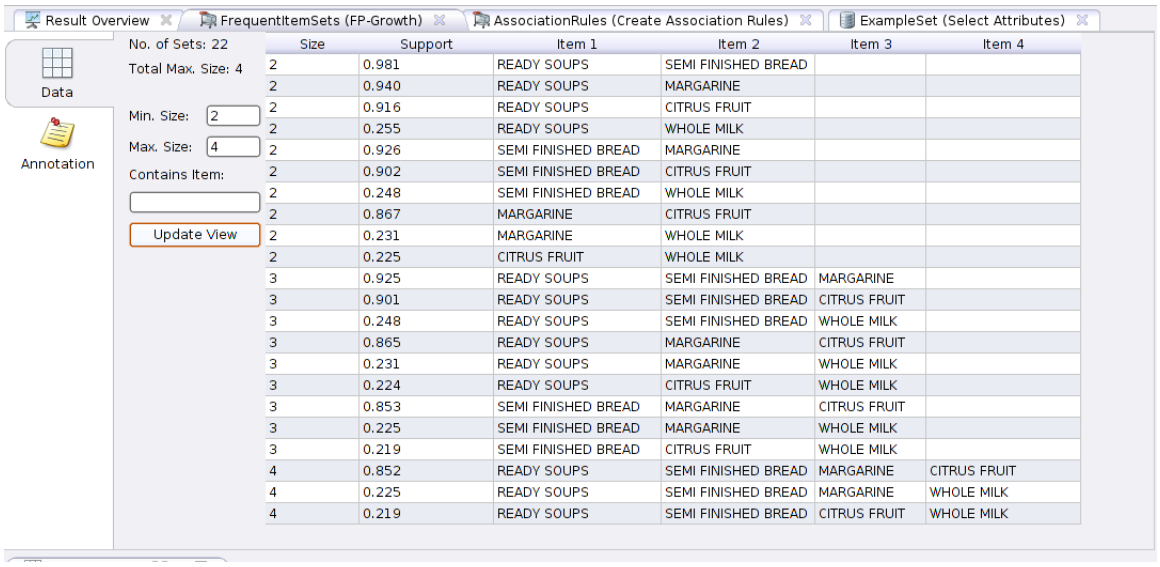
[4] Michael Hahsler, Kurt Hornik and Thomas Reutterer, *R Documentation: Groceries Data Set*, http://finzi.psych.upenn.edu/library/arules/html/Groceries.html, Accessed on 24 October 2015.

[5] Michael Hahsler, Sudheer Chelluboina, Kurt Hornik and Christian Buchta, *Visualizing Association Rules in Hierarchical Groups*, Journal of Machine Learning Research 12 (2011) 2021-2025.

[6] Michael Hahsler and Sudheer Chelluboina, *The arules R-Package Ecosystem: Analyzing Interesting Patterns from Large Transaction Data Sets*, May 19, 2011.

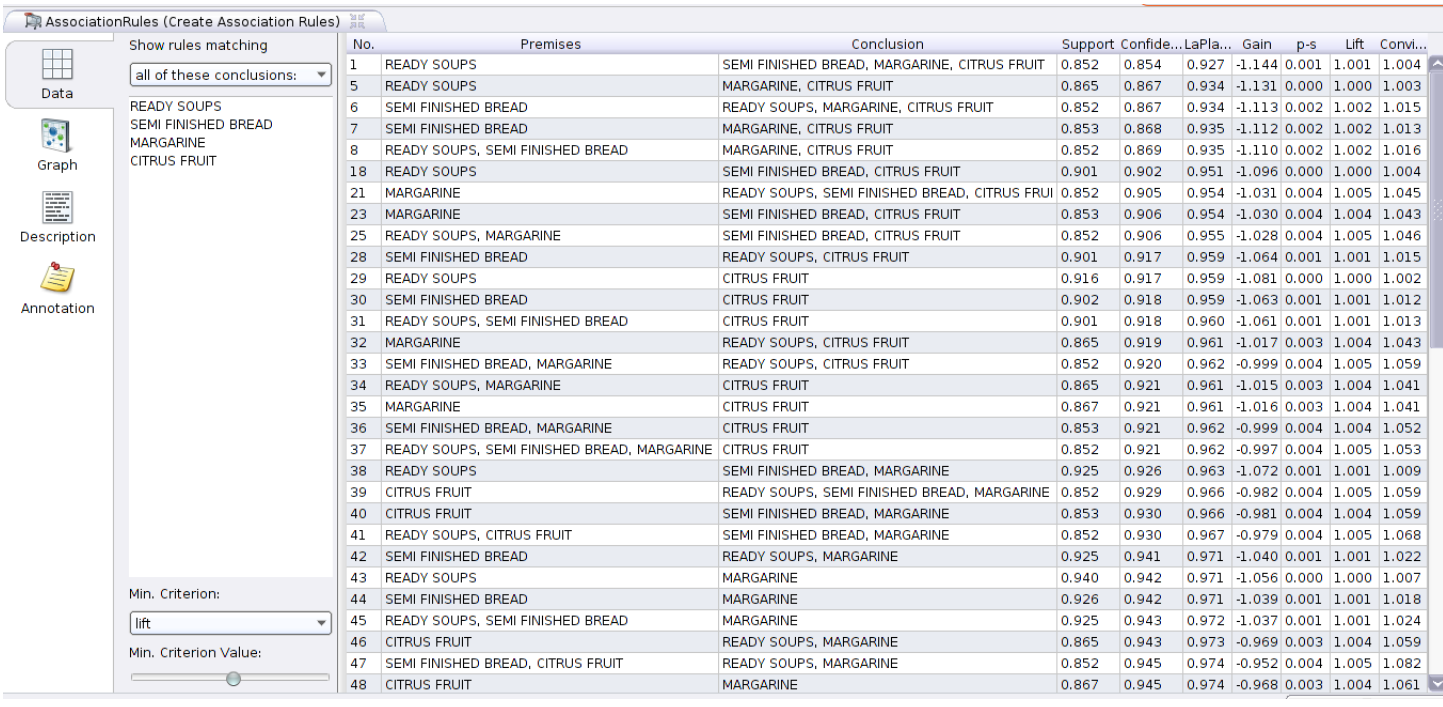
[7] Hossein Tohidi, Hamidah Ibrahim, *A Frequent Pattern Mining Algorithm Based on FP-growth without Generating Tree* , Faculty of Computer Science and Information Technology, Universiti Putra Malaysia .

[8] DANIEL HUNYADI, *Performance comparison of Apriori and FP-Growth algorithms in generating association rules* , Department of Computer Science, Lucian Blaga University of Sibiu, Romania. Proceedings of the European Computing Conference.



Size	Support	Item 1	Item 2	Item 3	Item 4
2	0.981	READY SOUPS	SEMI FINISHED BREAD		
2	0.940	READY SOUPS	MARGARINE		
2	0.916	READY SOUPS	CITRUS FRUIT		
2	0.255	READY SOUPS	WHOLE MILK		
2	0.926	SEMI FINISHED BREAD	MARGARINE		
2	0.902	SEMI FINISHED BREAD	CITRUS FRUIT		
2	0.248	SEMI FINISHED BREAD	WHOLE MILK		
2	0.867	MARGARINE	CITRUS FRUIT		
2	0.231	MARGARINE	WHOLE MILK		
2	0.225	CITRUS FRUIT	WHOLE MILK		
3	0.925	READY SOUPS	SEMI FINISHED BREAD	MARGARINE	
3	0.901	READY SOUPS	SEMI FINISHED BREAD	CITRUS FRUIT	
3	0.248	READY SOUPS	SEMI FINISHED BREAD	WHOLE MILK	
3	0.865	READY SOUPS	MARGARINE	CITRUS FRUIT	
3	0.231	READY SOUPS	MARGARINE	WHOLE MILK	
3	0.224	READY SOUPS	CITRUS FRUIT	WHOLE MILK	
3	0.853	SEMI FINISHED BREAD	MARGARINE	CITRUS FRUIT	
3	0.225	SEMI FINISHED BREAD	MARGARINE	WHOLE MILK	
3	0.219	SEMI FINISHED BREAD	CITRUS FRUIT	WHOLE MILK	
4	0.852	READY SOUPS	SEMI FINISHED BREAD	MARGARINE	CITRUS FRUIT
4	0.225	READY SOUPS	SEMI FINISHED BREAD	MARGARINE	WHOLE MILK
4	0.219	READY SOUPS	SEMI FINISHED BREAD	CITRUS FRUIT	WHOLE MILK

Figure 9: Frequency itemsets



No.	Premises	Conclusion	Support	Confide...	LaPla...	Gain	p-s	Lift	Convi...
1	READY SOUPS	SEMI FINISHED BREAD, MARGARINE, CITRUS FRUIT	0.852	0.854	0.927	-1.144	0.001	1.001	1.004
5	READY SOUPS	MARGARINE, CITRUS FRUIT	0.865	0.867	0.934	-1.131	0.000	1.000	1.003
6	SEMI FINISHED BREAD	READY SOUPS, MARGARINE, CITRUS FRUIT	0.852	0.867	0.934	-1.113	0.002	1.002	1.015
7	SEMI FINISHED BREAD	MARGARINE, CITRUS FRUIT	0.853	0.868	0.935	-1.112	0.002	1.002	1.013
8	READY SOUPS, SEMI FINISHED BREAD	MARGARINE, CITRUS FRUIT	0.852	0.869	0.935	-1.110	0.002	1.002	1.016
18	READY SOUPS	SEMI FINISHED BREAD, CITRUS FRUIT	0.901	0.902	0.951	-1.096	0.000	1.000	1.004
21	MARGARINE	READY SOUPS, SEMI FINISHED BREAD, CITRUS FRUI	0.852	0.905	0.954	-1.031	0.004	1.005	1.045
23	MARGARINE	SEMI FINISHED BREAD, CITRUS FRUIT	0.853	0.906	0.954	-1.030	0.004	1.004	1.043
25	READY SOUPS, MARGARINE	SEMI FINISHED BREAD, CITRUS FRUIT	0.852	0.906	0.955	-1.028	0.004	1.005	1.046
28	SEMI FINISHED BREAD	READY SOUPS, CITRUS FRUIT	0.901	0.917	0.959	-1.064	0.001	1.001	1.015
29	READY SOUPS	CITRUS FRUIT	0.916	0.917	0.959	-1.081	0.000	1.000	1.002
30	SEMI FINISHED BREAD	CITRUS FRUIT	0.902	0.918	0.959	-1.063	0.001	1.001	1.012
31	READY SOUPS, SEMI FINISHED BREAD	CITRUS FRUIT	0.901	0.918	0.960	-1.061	0.001	1.001	1.013
32	MARGARINE	READY SOUPS, CITRUS FRUIT	0.865	0.919	0.961	-1.017	0.003	1.004	1.043
33	SEMI FINISHED BREAD, MARGARINE	READY SOUPS, CITRUS FRUIT	0.852	0.920	0.962	-0.999	0.004	1.005	1.059
34	READY SOUPS, MARGARINE	CITRUS FRUIT	0.865	0.921	0.961	-1.015	0.003	1.004	1.041
35	MARGARINE	CITRUS FRUIT	0.867	0.921	0.961	-1.016	0.003	1.004	1.041
36	SEMI FINISHED BREAD, MARGARINE	CITRUS FRUIT	0.853	0.921	0.962	-0.999	0.004	1.004	1.052
37	READY SOUPS, SEMI FINISHED BREAD, MARGARINE	CITRUS FRUIT	0.852	0.921	0.962	-0.997	0.004	1.005	1.053
38	READY SOUPS	SEMI FINISHED BREAD, MARGARINE	0.925	0.926	0.963	-1.072	0.001	1.001	1.009
39	CITRUS FRUIT	READY SOUPS, SEMI FINISHED BREAD, MARGARINE	0.852	0.929	0.966	-0.982	0.004	1.005	1.059
40	CITRUS FRUIT	SEMI FINISHED BREAD, MARGARINE	0.853	0.930	0.966	-0.981	0.004	1.004	1.059
41	READY SOUPS, CITRUS FRUIT	SEMI FINISHED BREAD, MARGARINE	0.852	0.930	0.967	-0.979	0.004	1.005	1.068
42	SEMI FINISHED BREAD	READY SOUPS, MARGARINE	0.925	0.941	0.971	-1.040	0.001	1.001	1.022
43	READY SOUPS	MARGARINE	0.940	0.942	0.971	-1.056	0.000	1.000	1.007
44	SEMI FINISHED BREAD	MARGARINE	0.926	0.942	0.971	-1.039	0.001	1.001	1.018
45	READY SOUPS, SEMI FINISHED BREAD	MARGARINE	0.925	0.943	0.972	-1.037	0.001	1.001	1.024
46	CITRUS FRUIT	READY SOUPS, MARGARINE	0.865	0.943	0.973	-0.969	0.003	1.004	1.059
47	SEMI FINISHED BREAD, CITRUS FRUIT	READY SOUPS, MARGARINE	0.852	0.945	0.974	-0.952	0.004	1.005	1.082
48	CITRUS FRUIT	MARGARINE	0.867	0.945	0.974	-0.968	0.003	1.004	1.061

Figure 10: Rules with performance measures

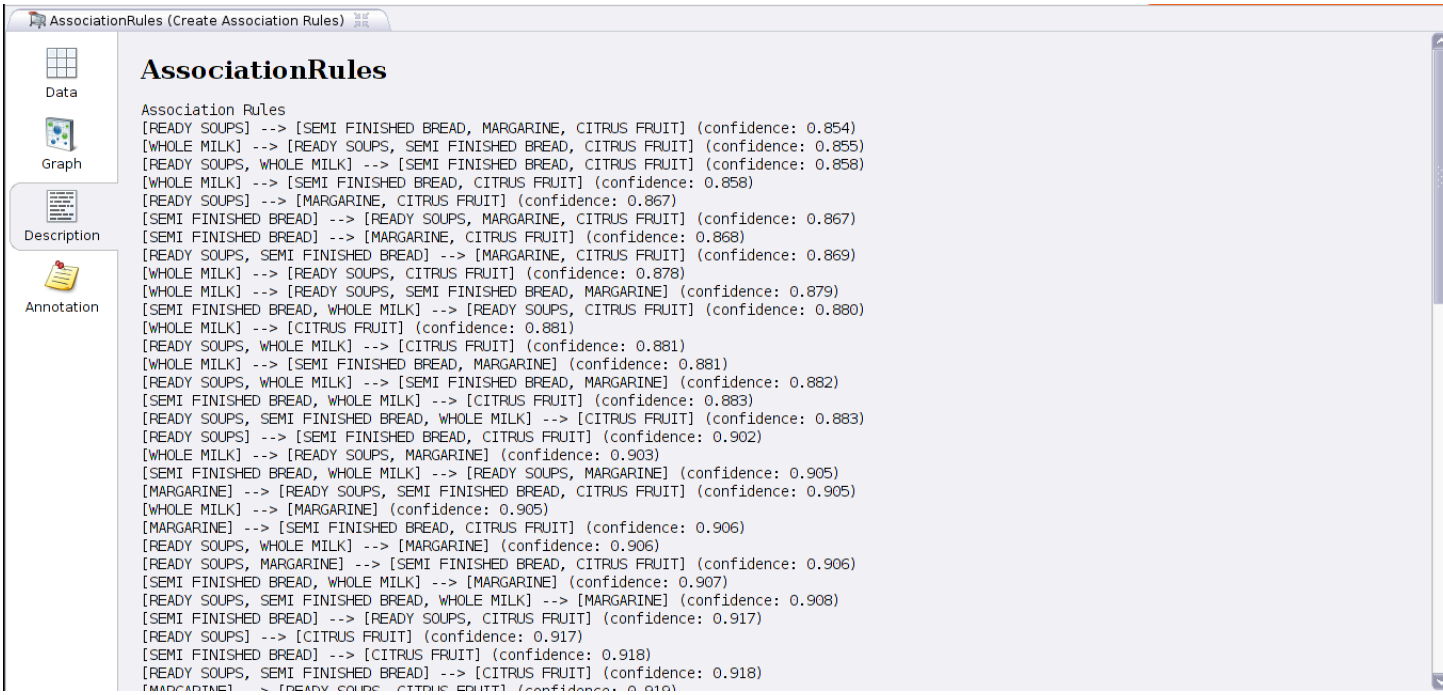


Figure 11: Rules Description

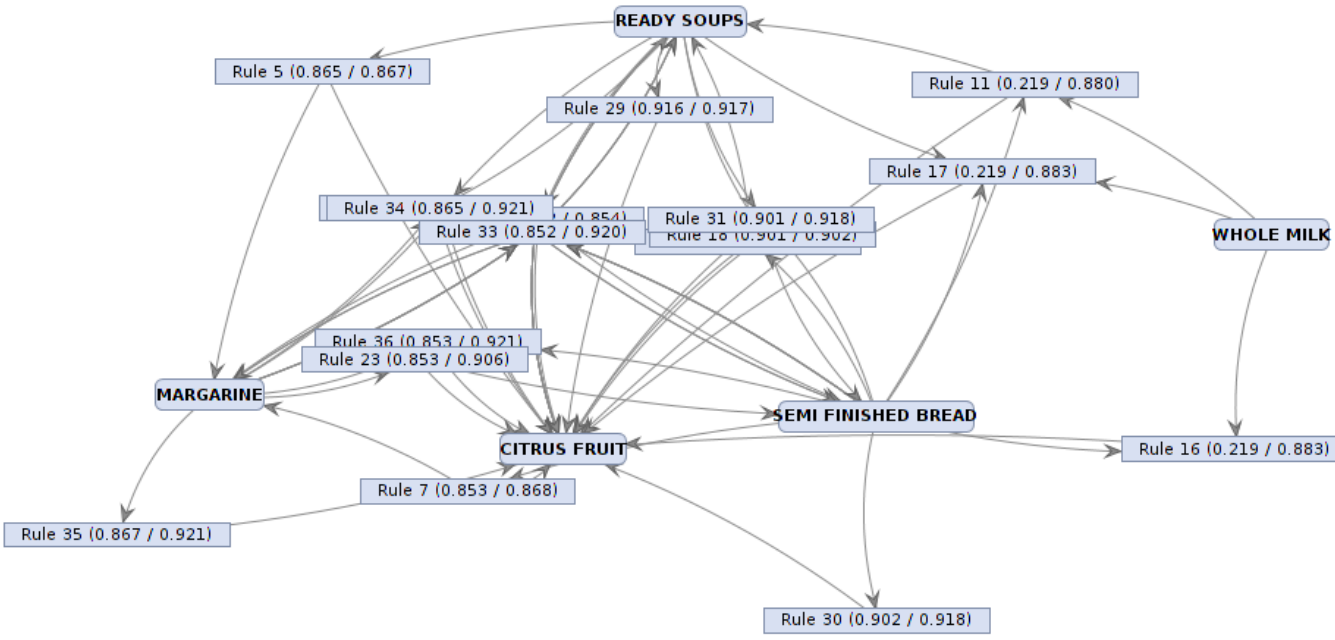


Figure 12: Visualization