Project 3

VADYM DUDARENKO

Association Rules

Market basket analysis

Data

Aim of this report is to analyze relationships between products in the market basket. Data used in the analysis were taken from Kaggle website (https://www.kaggle.com/ekrembayar/apriori-association-rules-grocery-store/notebook) and describe shopping behaviors.

Each row in the data represents one market basket, that means one shopping transaction done by client of the market. Below we can see some statistics about data.

Inspecting data:

Below we see sample of 20 baskets and items stored inside them. As it was said, some of them contain many products like first one.

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

trans1<-read.transactions("C:/Users/Vadym/Desktop/trans1.csv", format="basket",
sep=",", skip=0)
inspect(trans1)
items
[1]  {BISCUIT, BREAD, MILK}
[2]  {BISCUIT, BREAD, CORNFLAKES, MILK}
```

```
[3]  {BOURNVITA, BREAD, TEA}
[4]  {BREAD, JAM, MAGGI, MILK}
[5]  {BISCUIT, MAGGI, TEA}
[6]  {BOURNVITA, BREAD, TEA}
[7]  {CORNFLAKES, MAGGI, TEA}
[8]  {BISCUIT, BREAD, MAGGI, TEA}
[9]  {BREAD, JAM, MAGGI, TEA}
[10] {BREAD, MILK}
[11] {BISCUIT, COCK, COFFEE, CORNFLAKES}
[12] {BISCUIT, COCK, COFFEE, CORNFLAKES}
[13] {BOURNVITA, COFFEE, SUGER}
[14] {BREAD, COCK, COFFEE}
[15] {BISCUIT, BREAD, SUGER}
[16] {COFFEE, CORNFLAKES, SUGER}
[17] {BOURNVITA, BREAD, SUGER}
[18] {BREAD, COFFEE, SUGER}
[19] {BREAD, COFFEE, SUGER}
[20] {COFFEE, CORNFLAKES, MILK, TEA}

size(trans1)
[1] 3 4 3 4 3 3 3 4 4 2 4 4 3 3 3 3 3 3 3 4

length(trans1)
[1] 20

trans2<-read.transactions("C:/Users/Vadym/Desktop/trans2.csv", format="single",
sep=";", cols=c("TRANS","ITEM"), header=TRUE)
trans2
transactions in sparse format with
 20 transactions (rows) and
```

11 items (columns)

size(trans2)

[1] 3 4 3 4 3 3 3 4 4 2 4 4 3 3 3 3 3 3 3 4

Basic descriptive statistics:

round(itemFrequency(trans1),3)

BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI
0.35	0.20	0.65	0.15	0.40	0.30	0.10	0.25

MILK	SUGER	TEA
0.25	0.30	0.35

itemFrequency(trans1, type="absolute")

BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI
7	4	13	3	8	6	2	5

MILK	SUGER	TEA
5	6	7

ctab<-crossTable(trans1, sort=TRUE)

ctab<-crossTable(trans1, measure="count", sort=TRUE)

ctab

	BREAD	COFFEE	BISCUIT	TEA	CORNFLAKES	SUGER	MAGGI	MILK	BOURNVITA	COCK	JAM
BREAD	13	3	4	4	1	4	3	4	3	1	2
COFFEE	3	8	2	1	4	4	0	1	1	3	0
BISCUIT	4	2	7	2	3	1	2	2	0	2	0
TEA	4	1	2	7	2	0	4	1	2	0	1
CORNFLAKES	1	4	3	2	6	1	1	2	0	2	0
SUGER	4	4	1	0	1	6	0	0	2	0	0

```
MAGGI        3    0     2  4       1    0    5   1      0   0  2
MILK         4    1     2  1       2    0    1   5      0   0  1
BOURNVITA    3    1     0  2       0    2    0   0      4   0  0
COCK         1    3     2  0       2    0    0   0      0   3  0
JAM          2    0     0  1       0    0    2   1      0   0  2

stab<-crossTable(trans1, measure="support", sort=TRUE)
round(stab, 3)
          BREAD COFFEE BISCUIT  TEA CORNFLAKES SUGER MAGGI
MILK BOURNVITA COCK  JAM
BREAD       0.65   0.15    0.20 0.20       0.05  0.20  0.15 0.20      0.15 0.05 0.10
COFFEE      0.15   0.40    0.10 0.05       0.20  0.20  0.00 0.05      0.05 0.15 0.00
BISCUIT     0.20   0.10    0.35 0.10       0.15  0.05  0.10 0.10      0.00 0.10 0.00
TEA         0.20   0.05    0.10 0.35       0.10  0.00  0.20 0.05      0.10 0.00 0.05
CORNFLAKES  0.05   0.20    0.15 0.10       0.30  0.05  0.05 0.10      0.00 0.10 0.00
SUGER       0.20   0.20    0.05 0.00       0.05  0.30  0.00 0.00      0.10 0.00 0.00
MAGGI       0.15   0.00    0.10 0.20       0.05  0.00  0.25 0.05      0.00 0.00 0.10
MILK        0.20   0.05    0.10 0.05       0.10  0.00  0.05 0.25      0.00 0.00 0.05
BOURNVITA   0.15   0.05    0.00 0.10       0.00  0.10  0.00 0.00      0.20 0.00 0.00
COCK        0.05   0.15    0.10 0.00       0.10  0.00  0.00 0.00      0.00 0.15 0.00
JAM         0.10   0.00    0.00 0.05       0.00  0.00  0.10 0.05      0.00 0.00 0.10

ptab<-crossTable(trans1, measure="probability", sort=TRUE) # jak support
round(ptab,3)
        BREAD COFFEE BISCUIT  TEA CORNFLAKES SUGER MAGGI MILK
BOURNVITA COCK  JAM
BREAD       0.65   0.15    0.20 0.20       0.05  0.20  0.15 0.20      0.15 0.05 0.10
COFFEE      0.15   0.40    0.10 0.05       0.20  0.20  0.00 0.05      0.05 0.15 0.00
BISCUIT     0.20   0.10    0.35 0.10       0.15  0.05  0.10 0.10      0.00 0.10 0.00
TEA         0.20   0.05    0.10 0.35       0.10  0.00  0.20 0.05      0.10 0.00 0.05
```

CORNFLAKES	0.05	0.20	0.15	0.10	0.30	0.05	0.05	0.10	0.00	0.10	0.00
SUGER	0.20	0.20	0.05	0.00	0.05	0.30	0.00	0.00	0.10	0.00	0.00
MAGGI	0.15	0.00	0.10	0.20	0.05	0.00	0.25	0.05	0.00	0.00	0.10
MILK	0.20	0.05	0.10	0.05	0.10	0.00	0.05	0.25	0.00	0.00	0.05
BOURNVITA	0.15	0.05	0.00	0.10	0.00	0.10	0.00	0.00	0.20	0.00	0.00
COCK	0.05	0.15	0.10	0.00	0.10	0.00	0.00	0.00	0.00	0.15	0.00
JAM	0.10	0.00	0.00	0.05	0.00	0.00	0.10	0.05	0.00	0.00	0.10

Association Rules

Now we will search for association rules. Association rules define relationship between occurance of two or more products. They are characterized by a few of parameters.

Analogically to support level of a product, support level of a rule means how many times rule appears in the dataset in compare to the total number of transactions. Confidence level of a rule between products is defined by the percentage of times when both consequent product (or products) and antecedent product (or products) appear in a transaction in compare to all times when antecedent product (or products) appear in a transaction. We can also define it as a ratio of support level of both consequent and antecedent items to support level of antecedent item (or items).

Lift level stands for the ratio of the confidence level of a rule to the support level of the consequent product (or products) from this rule. It can also be described as probality of consequent occurring in the transaction where antecedent occurs too compared to probability of consequent occurring in the whole set of transactions. Lift values higher than one stand for positive relationship of two products (or sets of products) and lower than one for negative relationship. If lift level is equal to one, products are independent.

Eclat & Apriori

The Eclat algorithm does not create rules - it digs through frequent sets to limit the data set. It works by using eclat(). As a result, we obtain frequent sets and measure values determined for them (e.g. support). When specifying search restrictions, the minimum support for the set is usually specified (e.g. supp = 0.1). One can also limit the maximum length of the set (e.g. to 10 elements maxlen=10). To create rules, use the ruleInduction() function. Displaying sets and rules is with the inspect() command.

```
freq.items<-eclat(trans1, parameter=list(supp=0.25, maxlen=15))
Eclat
parameter specification:
 tidLists support minlen maxlen           target  ext
    FALSE    0.25      1     15 frequent itemsets TRUE
algorithmic control:
 sparse sort verbose
      7   -2    TRUE
Absolute minimum support count: 5
create itemset ...
set transactions ...[11 item(s), 20 transaction(s)] done [0.00s].
sorting and recoding items ... [8 item(s)] done [0.00s].
creating bit matrix ... [8 row(s), 20 column(s)] done [0.00s].
writing  ... [8 set(s)] done [0.00s].
Creating S4 object  ... done [0.00s].

inspect(freq.items)
items        support count
[1] {BREAD}      0.65    13
[2] {COFFEE}     0.40     8
[3] {BISCUIT}    0.35     7
[4] {TEA}        0.35     7
[5] {CORNFLAKES} 0.30     6
[6] {MAGGI}      0.25     5
```

[7] {SUGER}   0.30   6

[8] {MILK}   0.25   5

Vector of support values:

round(support(items(freq.items), trans1) , 2)

[1] 0.65 0.40 0.35 0.35 0.30 0.25 0.30 0.25

Obtaining the result :

freq.rules<-ruleInduction(freq.items, trans1, confidence=0.9)

freq.rules

set of 0 rules

The apriori algorithm creates frequent item sets and based on these created item sets it creates rules. Default minimum values: minimum support (supp = 0.1), minimum confidence (conf = 0.8).

inspect(freq.rules)

rules.trans1<-apriori(trans1, parameter=list(supp=0.1, conf=0.5))

Apriori

Parameter specification:

confidence minval smax arem aval originalSupport maxtime support minlen maxlen target ext

0.5   0.1   1 none FALSE   TRUE   5   0.1   1   10 rules TRUE

Algorithmic control:

filter tree heap memopt load sort verbose

0.1 TRUE TRUE FALSE TRUE   2   TRUE

Absolute minimum support count: 2

set item appearances ...[0 item(s)] done [0.00s].

set transactions ...[11 item(s), 20 transaction(s)] done [0.00s].

sorting and recoding items ... [11 item(s)] done [0.00s].

creating transaction tree ... done [0.00s].

checking subsets of size 1 2 3 4 done [0.00s].

writing ... [55 rule(s)] done [0.00s].

creating S4 object ... done [0.00s].

Sorting rules by confidence + displaying

rules.by.conf<-sort(rules.trans1, by="confidence", decreasing=TRUE)

inspect(head(rules.by.conf))

```
lhs                 rhs       support confidence coverage lift     count
[1] {JAM}           => {MAGGI}   0.10    1          0.10     4.000000 2
[2] {JAM}           => {BREAD}   0.10    1          0.10     1.538462 2
[3] {COCK}          => {COFFEE}  0.15    1          0.15     2.500000 3
[4] {JAM, MAGGI}    => {BREAD}   0.10    1          0.10     1.538462 2
[5] {BREAD, JAM}    => {MAGGI}   0.10    1          0.10     4.000000 2
[6] {COCK, CORNFLAKES} => {BISCUIT} 0.10    1          0.10     2.857143 2
```

rules.by.lift<-sort(rules.trans1, by="lift", decreasing=TRUE) # sorting by lift

inspect(head(rules.by.lift))

```
   lhs                          rhs     support confidence coverage lift     count
[1] {BISCUIT, COFFEE}           => {COCK}  0.1     1.0000000  0.10     6.666667
2
[2] {BISCUIT, COFFEE, CORNFLAKES} => {COCK}  0.1     1.0000000  0.10
6.666667 2
[3] {BREAD, MAGGI}              => {JAM}   0.1     0.6666667  0.15     6.666667
2
[4] {BISCUIT, CORNFLAKES}       => {COCK}  0.1     0.6666667  0.15
4.444444 2
[5] {JAM}                       => {MAGGI} 0.1     1.0000000  0.10     4.000000 2
[6] {BREAD, JAM}                => {MAGGI} 0.1     1.0000000  0.10     4.000000
2
```

```
rules.by.count<- sort(rules.trans1, by="count", decreasing=TRUE) # sorting by count
inspect(head(rules.by.count))
    lhs         rhs      support confidence coverage lift     count
[1] {}       => {BREAD}  0.65    0.6500000  1.00     1.000000 13
[2] {MILK}   => {BREAD}  0.20    0.8000000  0.25     1.230769  4
[3] {MAGGI}  => {TEA}    0.20    0.8000000  0.25     2.285714  4
[4] {TEA}    => {MAGGI}  0.20    0.5714286  0.35     2.285714  4
[5] {SUGER}  => {COFFEE} 0.20    0.6666667  0.30     1.666667  4
[6] {COFFEE} => {SUGER}  0.20    0.5000000  0.40     1.666667  4
```

```
rules.by.supp<-sort(rules.trans1, by="support", decreasing=TRUE)
inspect(head(rules.by.supp))
    lhs         rhs      support confidence coverage lift     count
[1] {}       => {BREAD}  0.65    0.6500000  1.00     1.000000 13
[2] {MILK}   => {BREAD}  0.20    0.8000000  0.25     1.230769  4
[3] {MAGGI}  => {TEA}    0.20    0.8000000  0.25     2.285714  4
[4] {TEA}    => {MAGGI}  0.20    0.5714286  0.35     2.285714  4
[5] {SUGER}  => {COFFEE} 0.20    0.6666667  0.30     1.666667  4
[6] {COFFEE} => {SUGER}  0.20    0.5000000  0.40     1.666667  4
```

Digging the rules

1. in the context of induction - what is the cause / consequence of a given purchase
2. looking for rules for closed item sets
3. finding significant rules
4. looking for maximal rules
5. looking for redundant rules
6. searching subsets and supersets
7. by searching for transactions that support the rules

```
rules.bread<-apriori(data=trans1, parameter=list(supp=0.001,conf = 0.08),
                     appearance=list(default="lhs", rhs="BREAD"),
control=list(verbose=F))
# sorting and displaying the rules
rules.bread.byconf<-sort(rules.bread, by="confidence", decreasing=TRUE)
inspect(head(rules.bread.byconf))
lhs                  rhs     support confidence coverage lift     count
[1] {JAM}            => {BREAD} 0.10    1         0.10     1.538462 2
[2] {JAM, MILK}      => {BREAD} 0.05    1         0.05     1.538462 1
[3] {JAM, MAGGI}     => {BREAD} 0.10    1         0.10     1.538462 2
[4] {JAM, TEA}       => {BREAD} 0.05    1         0.05     1.538462 1
[5] {BOURNVITA, TEA} => {BREAD} 0.10    1         0.10     1.538462 2
[6] {MAGGI, MILK}    => {BREAD} 0.05    1         0.05     1.538462 1

rules.bread<-apriori(data=trans1, parameter=list(supp=0.001,conf = 0.08),
                     appearance=list(default="rhs",lhs="BREAD"),
control=list(verbose=F))

rules.bread.byconf<-sort(rules.bread, by="confidence", decreasing=TRUE)
inspect(head(rules.bread.byconf))
  lhs        rhs       support confidence coverage lift      count
[1] {}      => {COFFEE}  0.40    0.4000000  1.00     1.0000000 8
[2] {}      => {TEA}     0.35    0.3500000  1.00     1.0000000 7
[3] {}      => {BISCUIT} 0.35    0.3500000  1.00     1.0000000 7
[4] {BREAD} => {MILK}    0.20    0.3076923  0.65     1.2307692 4
[5] {BREAD} => {SUGER}   0.20    0.3076923  0.65     1.0256410 4
[6] {BREAD} => {TEA}     0.20    0.3076923  0.65     0.8791209 4
```

Closed transactions with the apriori() and eclat() command

- the apriori algorithm allows you to search for rules for closed frequent item sets
- the options should be set: parameter=list(target="closed frequent item sets") or parameter=list(target="maximally frequent item sets") - closed transactions are the most complex and common - incorrectly set rule parameters may cause the set of closed rules to be empty.

```
trans1.closed<-apriori(trans1, parameter=list(target="closed frequent itemsets",
support=0.25))
Apriori
Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
maxlen
         NA    0.1    1 none FALSE            TRUE       5    0.25      1     10
                   target  ext
 closed frequent itemsets TRUE
Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE
Absolute minimum support count: 5
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[11 item(s), 20 transaction(s)] done [0.00s].
sorting and recoding items ... [8 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
filtering closed item sets ... done [0.00s].
sorting transactions ... done [0.00s].
writing ... [8 set(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
inspect(trans1.closed)
items        support count
[1] {MILK}       0.25     5
[2] {MAGGI}      0.25     5
[3] {SUGER}      0.30     6
[4] {CORNFLAKES} 0.30     6
[5] {TEA}        0.35     7
[6] {BISCUIT}    0.35     7
[7] {COFFEE}     0.40     8
[8] {BREAD}      0.65    13

is.closed(trans1.closed)
{MILK}     {MAGGI}     {SUGER} {CORNFLAKES}       {TEA}
{BISCUIT}     {COFFEE}
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
TRUE
     {BREAD}
       TRUE

freq.closed<-eclat(trans1, parameter=list(supp=0.15, maxlen=15, target="closed frequent itemsets"))
Eclat
parameter specification:
 tidLists support minlen maxlen                  target  ext
    FALSE    0.15      1     15 closed frequent itemsets TRUE
algorithmic control:
 sparse sort verbose
      7   -2    TRUE
Absolute minimum support count: 3
```

create itemset ...

set transactions ...[11 item(s), 20 transaction(s)] done [0.00s].

sorting and recoding items ... [10 item(s)] done [0.00s].

creating bit matrix ... [10 row(s), 20 column(s)] done [0.00s].

writing ... [21 set(s)] done [0.00s].

Creating S4 object ... done [0.00s].

inspect(freq.closed)

	items	support	count
[1]	{COCK, COFFEE}	0.15	3
[2]	{BOURNVITA, BREAD}	0.15	3
[3]	{BREAD, MILK}	0.20	4
[4]	{BREAD, SUGER}	0.20	4
[5]	{COFFEE, SUGER}	0.20	4
[6]	{BREAD, MAGGI}	0.15	3
[7]	{MAGGI, TEA}	0.20	4
[8]	{COFFEE, CORNFLAKES}	0.20	4
[9]	{BISCUIT, CORNFLAKES}	0.15	3
[10]	{BREAD, TEA}	0.20	4
[11]	{BISCUIT, BREAD}	0.20	4
[12]	{BREAD, COFFEE}	0.15	3
[13]	{BREAD}	0.65	13
[14]	{COFFEE}	0.40	8
[15]	{BISCUIT}	0.35	7
[16]	{TEA}	0.35	7
[17]	{CORNFLAKES}	0.30	6
[18]	{MAGGI}	0.25	5
[19]	{SUGER}	0.30	6
[20]	{MILK}	0.25	5

```
is.closed(freq.closed)
 {COCK,COFFEE}    {BOURNVITA,BREAD}        {BREAD,MILK}
{BREAD,SUGER}
            TRUE               TRUE               TRUE               TRUE
    {COFFEE,SUGER}       {BREAD,MAGGI}        {MAGGI,TEA}
{COFFEE,CORNFLAKES}
            TRUE               TRUE               TRUE               TRUE
{BISCUIT,CORNFLAKES}        {BREAD,TEA}     {BISCUIT,BREAD}
{BREAD,COFFEE}
            TRUE               TRUE               TRUE               TRUE
          {BREAD}          {COFFEE}          {BISCUIT}             {TEA}
            TRUE               TRUE               TRUE               TRUE
      {CORNFLAKES}           {MAGGI}           {SUGER}            {MILK}
            TRUE               TRUE               TRUE               TRUE
       {BOURNVITA}
            TRUE

freq.max<-eclat(trans1, parameter=list(supp=0.15, maxlen=15, target="maximally
frequent itemsets"))
Eclat
parameter specification:
 tidLists support minlen maxlen                    target  ext
    FALSE    0.15      1     15 maximally frequent itemsets TRUE
algorithmic control:
 sparse sort verbose
      7   -2    TRUE
Absolute minimum support count: 3
create itemset ...
set transactions ...[11 item(s), 20 transaction(s)] done [0.00s].
sorting and recoding items ... [10 item(s)] done [0.00s].
```

```
creating bit matrix ... [10 row(s), 20 column(s)] done [0.00s].
writing  ... [12 set(s)] done [0.00s].
Creating S4 object  ... done [0.00s].

inspect(freq.max) # not clear output, ham is not more frequent than individual
baskets
items                  support count
[1]  {COCK, COFFEE}        0.15    3
[2]  {BOURNVITA, BREAD}    0.15    3
[3]  {BREAD, MILK}         0.20    4
[4]  {BREAD, SUGER}        0.20    4
[5]  {COFFEE, SUGER}       0.20    4
[6]  {BREAD, MAGGI}        0.15    3
[7]  {MAGGI, TEA}          0.20    4
[8]  {COFFEE, CORNFLAKES}  0.20    4
[9]  {BISCUIT, CORNFLAKES} 0.15    3
[10] {BREAD, TEA}          0.20    4
[11] {BISCUIT, BREAD}      0.20    4
[12] {BREAD, COFFEE}       0.15    3

inspect(rules.bread[is.maximal(rules.bread)==TRUE])
   lhs        rhs          support confidence coverage lift      count
[1]  {}      => {COCK}       0.15    0.1500000  1.00     1.0000000 3
[2]  {}      => {CORNFLAKES} 0.30    0.3000000  1.00     1.0000000 6
[3]  {BREAD} => {JAM}        0.10    0.1538462  0.65     1.5384615 2
[4]  {BREAD} => {BOURNVITA}  0.15    0.2307692  0.65     1.1538462 3
[5]  {BREAD} => {MILK}       0.20    0.3076923  0.65     1.2307692 4
[6]  {BREAD} => {MAGGI}      0.15    0.2307692  0.65     0.9230769 3
[7]  {BREAD} => {SUGER}      0.20    0.3076923  0.65     1.0256410 4
[8]  {BREAD} => {TEA}        0.20    0.3076923  0.65     0.8791209 4
```

```
[9]  {BREAD} => {BISCUIT}    0.20    0.3076923  0.65     0.8791209 4
[10] {BREAD} => {COFFEE}     0.15    0.2307692  0.65     0.5769231 3
```

```
inspect(rules.bread[is.redundant(rules.bread)==FALSE])
lhs        rhs          support confidence coverage lift     count
[1]  {}      => {JAM}        0.10    0.1000000  1.00     1.000000 2
[2]  {}      => {COCK}       0.15    0.1500000  1.00     1.000000 3
[3]  {}      => {BOURNVITA}  0.20    0.2000000  1.00     1.000000 4
[4]  {}      => {MILK}       0.25    0.2500000  1.00     1.000000 5
[5]  {}      => {MAGGI}      0.25    0.2500000  1.00     1.000000 5
[6]  {}      => {SUGER}      0.30    0.3000000  1.00     1.000000 6
[7]  {}      => {CORNFLAKES} 0.30    0.3000000  1.00     1.000000 6
[8]  {}      => {TEA}        0.35    0.3500000  1.00     1.000000 7
[9]  {}      => {BISCUIT}    0.35    0.3500000  1.00     1.000000 7
[10] {}      => {COFFEE}     0.40    0.4000000  1.00     1.000000 8
[11] {BREAD} => {JAM}        0.10    0.1538462  0.65     1.538462 2
[12] {BREAD} => {BOURNVITA}  0.15    0.2307692  0.65     1.153846 3
[13] {BREAD} => {MILK}       0.20    0.3076923  0.65     1.230769 4
[14] {BREAD} => {SUGER}      0.20    0.3076923  0.65     1.025641 4
```

A subset is a set that is contained within another (existing) set.

A superset is a set that is not contained in another (existing) set.

```
is.superset(rules.bread)
{JAM}             | . . . . . . . . . . . . . . . . .
{COCK}            . | . . . . . . . . . . . . . . . .
{BOURNVITA}       . . | . . . . . . . . . . . . . . .
{MILK}            . . . | . . . . . . . . . . . . . .
{MAGGI}           . . . . | . . . . . . . . . . . . .
{SUGER}           . . . . . | . . . . . . . . . . . .
```

```
{CORNFLAKES}     . . . . . . | . . . . . . . . . . .
{TEA}            . . . . . . . | . . . . . . . . . .
{BISCUIT}        . . . . . . . . | . . . . . . . . .
{COFFEE}         . . . . . . . . . | . . . . . . . .
{BREAD,JAM}      | . . . . . . . . . | . . . . . . .
{BOURNVITA,BREAD} . . | . . . . . . . . | . . . . . .
{BREAD,MILK}     . . . | . . . . . . . . | . . . . .
{BREAD,MAGGI}    . . . . | . . . . . . . . | . . . .
{BREAD,SUGER}    . . . . . | . . . . . . . . | . . .
{BREAD,TEA}      . . . . . . . | . . . . . . . | . .
{BISCUIT,BREAD}  . . . . . . . . | . . . . . . . | .
{BREAD,COFFEE}   . . . . . . . . . | . . . . . . . |
```

```
is.subset(rules.bread)
is.superset(rules.bread, sparse=FALSE)
supportingTransactions(rules.bread, trans1)
tidLists in sparse format with
 18 items/itemsets (rows) and
 20 transactions (columns)
```

The Jaccard Index can be derived from the dissimilarity() function in the arules package. It can be designated for sets or transactions.

The starting point is always the same: for sets / transactions A and B it is counted how many times it occurs:

- A and B; A but not B; B but not A; neither A nor B;
- general A in all sets / transactions
- generally B in all sets / transactions

Jaccard Index is the number in both sets / the number in one of the sets

The formal Jaccard index notation J (X, Y) = | X∩Y | / | XUY | -> similarity

Alternative notation is Jaccard distance = 1-Jaccard coefficient -> dissimilarity

J index = 100% | distance J = 0 - when all products are in X and Y

J index = 50% | distance J = 0.5 - when every second product is shared in X and Y

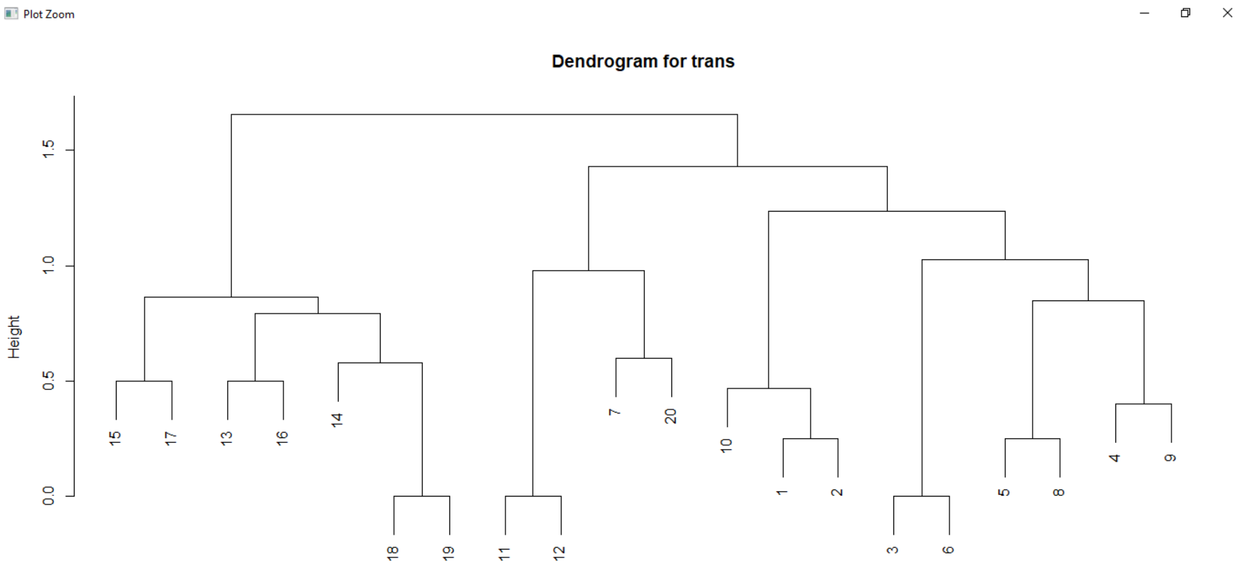
J index = 0% | distance J = 1 - when all products in X and Y are different

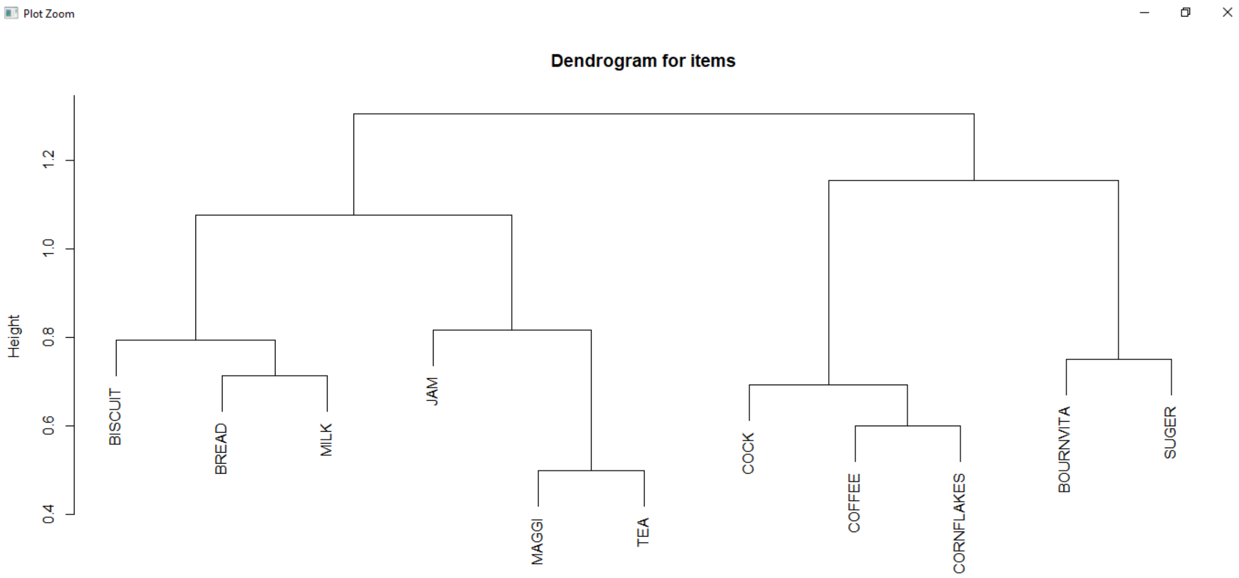
```
trans.sel<-trans1[,itemFrequency(trans1)>0.05] # selected transations
d.jac.i<-dissimilarity(trans.sel, which="items") # Jaccard as default
round(d.jac.i,2)
```

	BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI	MILK	SUGER
BOURNVITA	1.00									
BREAD	0.75	0.79								
COCK	0.75	1.00	0.93							
COFFEE	0.85	0.91	0.83	0.62						
CORNFLAKES	0.70	1.00	0.94	0.71	0.60					
JAM	1.00	1.00	0.85	1.00	1.00	1.00				
MAGGI	0.80	1.00	0.80	1.00	1.00	0.90	0.60			
MILK	0.80	1.00	0.71	1.00	0.92	0.78	0.83	0.89		
SUGER	0.92	0.75	0.73	1.00	0.60	0.91	1.00	1.00	1.00	
TEA	0.83	0.78	0.75	1.00	0.93	0.82	0.88	0.50	0.91	1.00

plot(hclust(d_jac.t, method="ward.D2"), main="Dendrogram for trans")

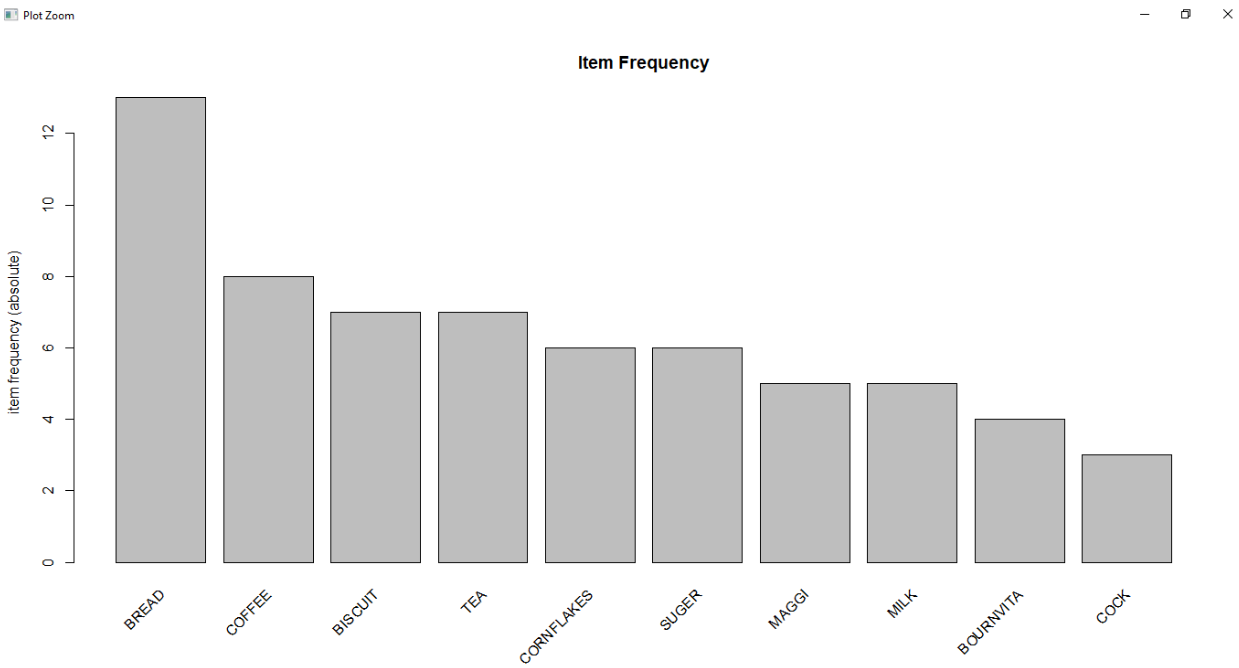
plot(hclust(d.jac.i, method="ward.D2"), main="Dendrogram for items")

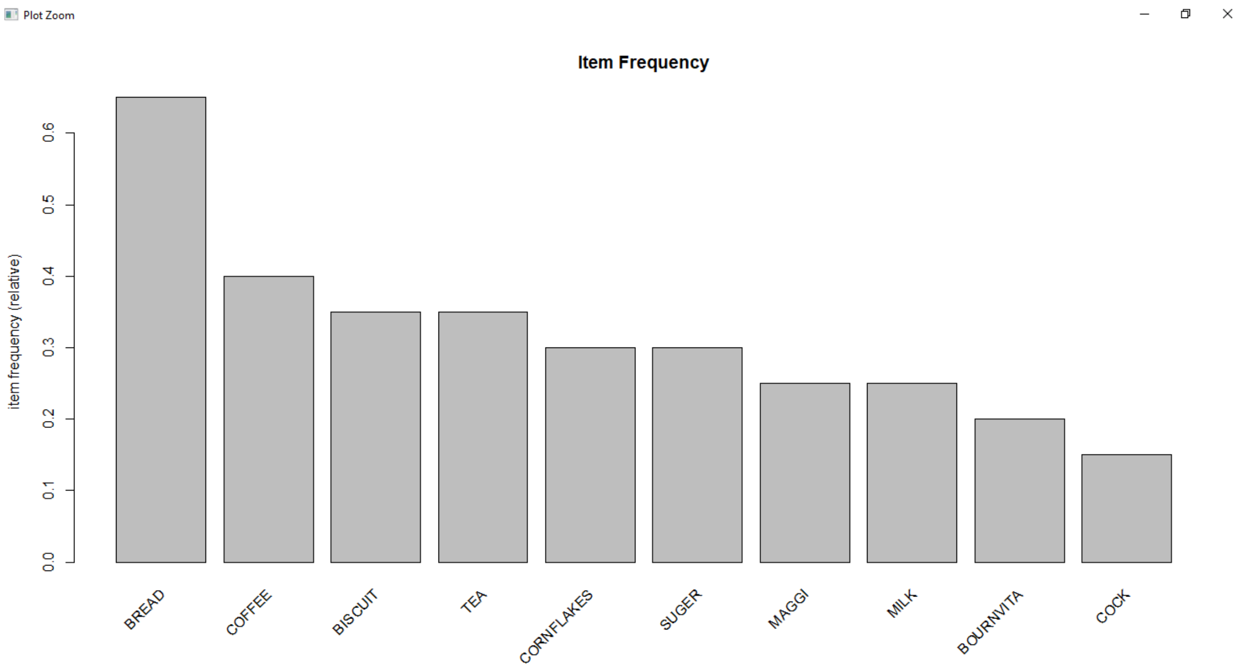




itemFrequencyPlot(trans1, topN=10, type="absolute", main="Item Frequency")

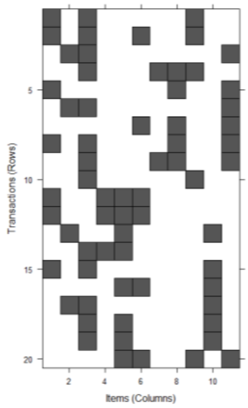
itemFrequencyPlot(trans1, topN=10, type="relative", main="Item Frequency")





We can easily observe the number of items in our basket and their frequency. It is obvious that bread appears the most, than we can see that coffee, biscuit and tea have almost the same frequency.

image(trans1)



plot(rules.trans1, method="matrix", measure="lift")

Itemsets in Antecedent (LHS)

[1] "{BISCUIT,COFFEE,CORNFLAKES}" "{BISCUIT,COFFEE}"

"{BREAD,MAGGI}"

[4] "{BREAD,JAM}" "{BISCUIT,TEA}"

"{BISCUIT,COCK,COFFEE}"

[7] "{BISCUIT,CORNFLAKES}" "{BISCUIT,COCK}"

"{BISCUIT,MAGGI}"

[10] "{COCK,COFFEE,CORNFLAKES}" "{JAM}"

"{COCK,CORNFLAKES}"

[13] "{BISCUIT,COCK,CORNFLAKES}" "{COFFEE,CORNFLAKES}"

"{BREAD,TEA}"

[16] "{BREAD,COFFEE}" "{COCK}" "{COCK,COFFEE}"

[19] "{BISCUIT,BREAD}" "{BOURNVITA,BREAD}"

"{COFFEE}"

[22] "{MAGGI}" "{TEA}" "{CORNFLAKES}"

[25] "{JAM,MAGGI}" "{BOURNVITA,TEA}"

"{BISCUIT,MILK}"

[28] "{BREAD,MILK}" "{BOURNVITA}" "{SUGER}"

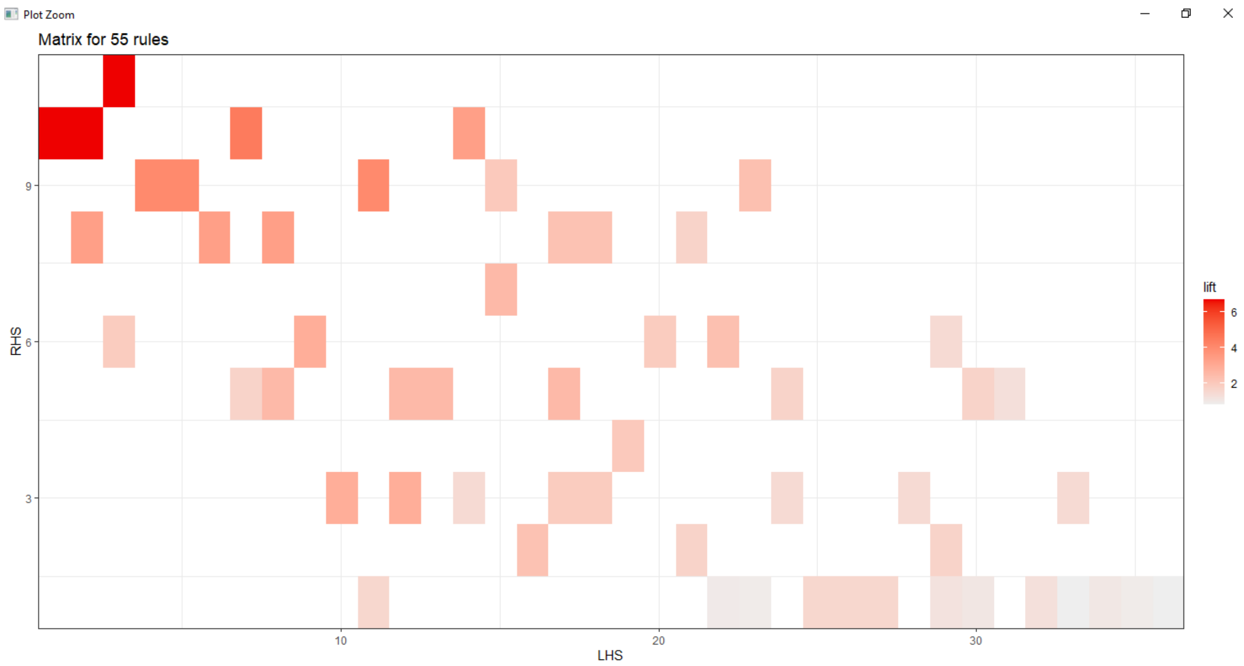
[31] "{BREAD,SUGER}" "{MILK}" "{MAGGI,TEA}"

[34] "{}" "{BISCUIT}" "{COFFEE,SUGER}"

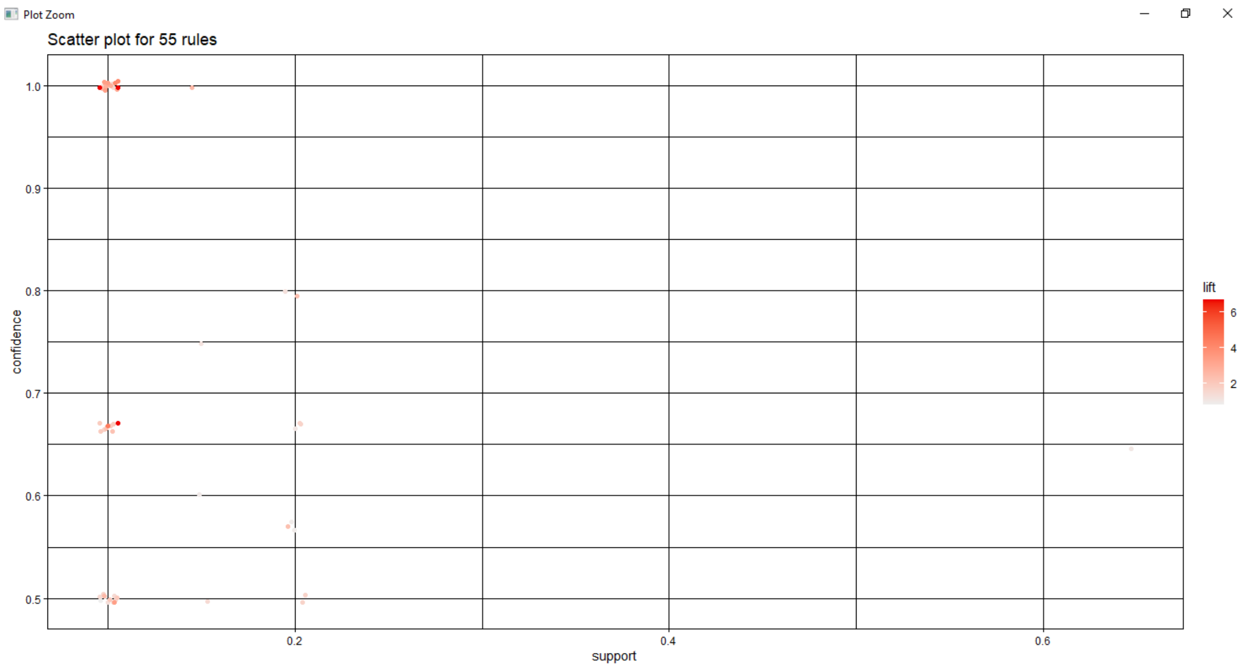
Itemsets in Consequent (RHS)

[1] "{BREAD}" "{SUGER}" "{BISCUIT}" "{MILK}" "{COFFEE}" "{TEA}"

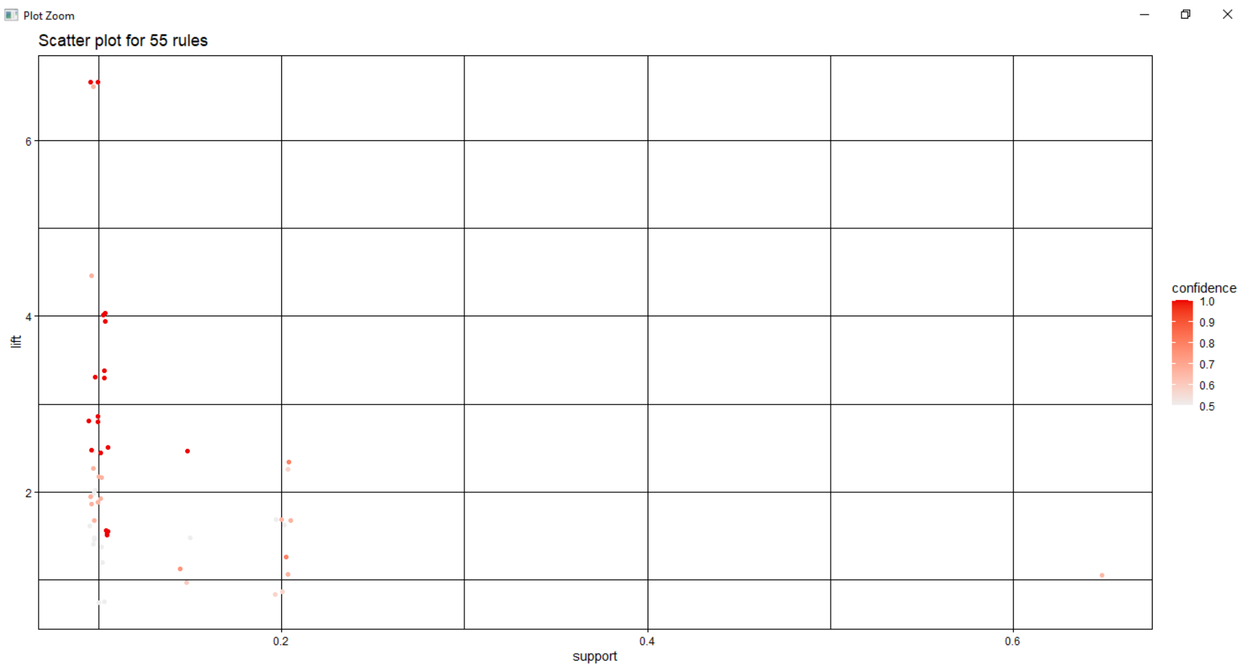
[7] "{BOURNVITA}" "{CORNFLAKES}" "{MAGGI}" "{COCK}" "{JAM}"



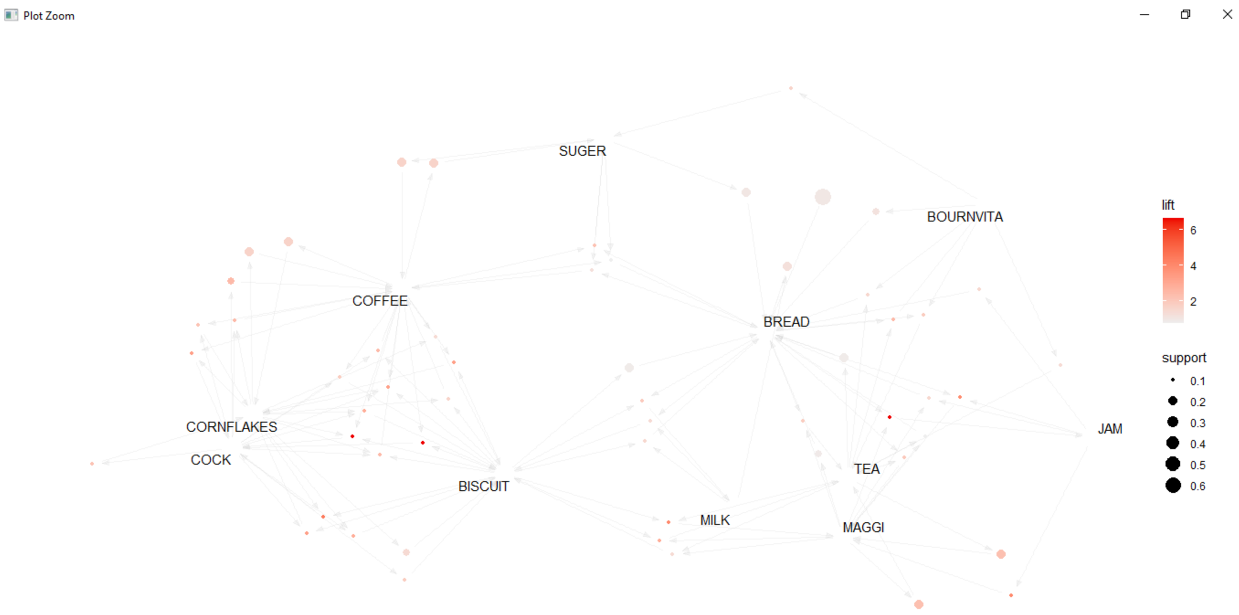
plot(rules.trans1)



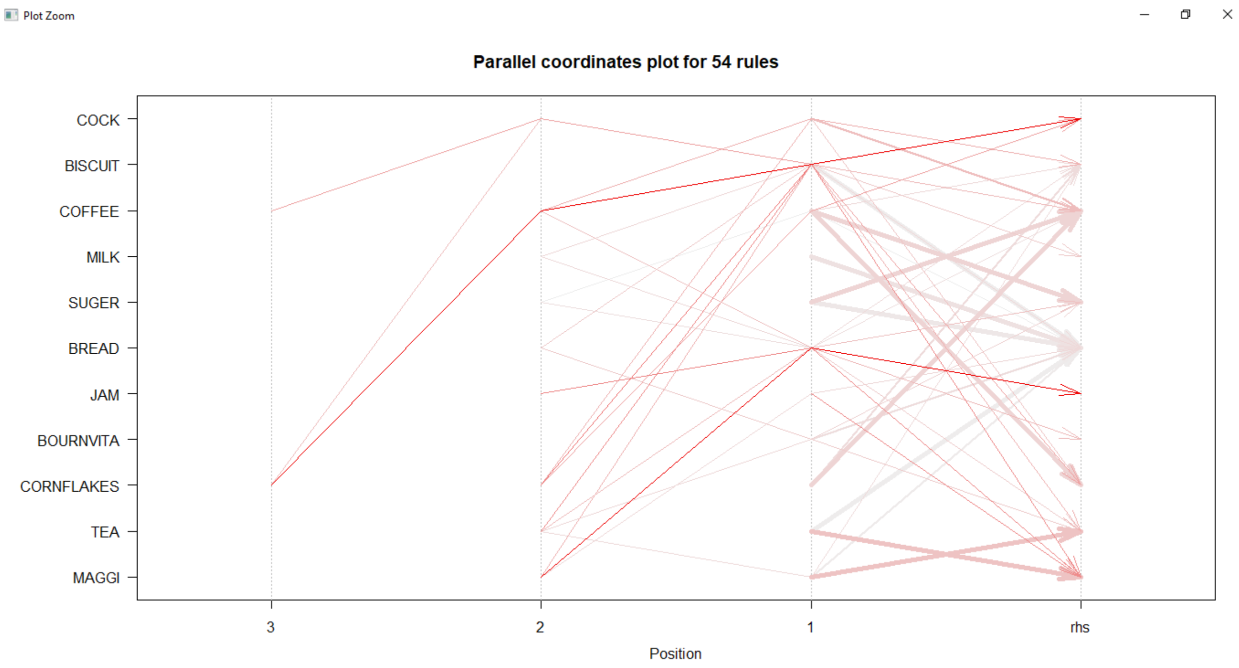
plot(rules.trans1, measure=c("support","lift"), shading="confidence")



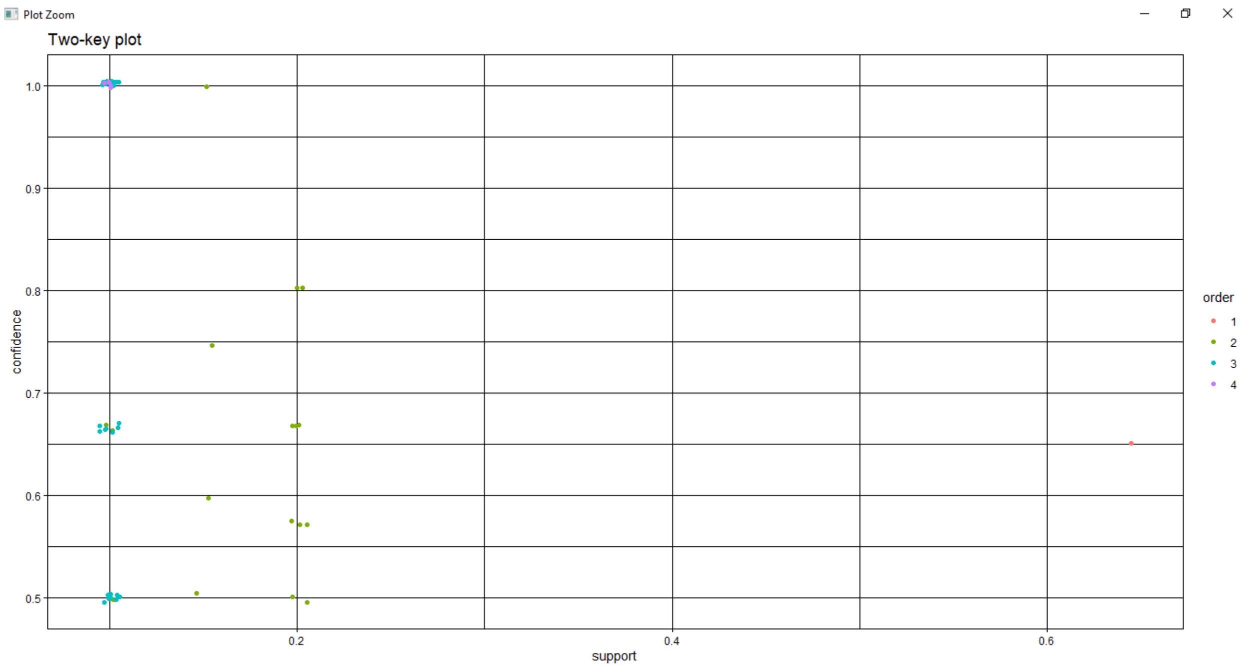
plot(rules.trans1, method="graph", control=list(type="items"))



plot(rules.trans1, method="paracoord", control=list(reorder=TRUE))



plot(rules.trans1, shading="order", control=list(main="Two-key plot"))



names(itemFrequency(trans1)) # info on product categories

[1] "BISCUIT"   "BOURNVITA" "BREAD"   "COCK"   "COFFEE"   "CORNFLAKES" "JAM"

[8] "MAGGI"   "MILK"   "SUGER"   "TEA"

names.real<-c("BISCUIT", "BOURNVITA" , "BREAD", "COCK", "COFFEE", "CORNFLAKES", "JAM", "MAGGI", "MILK", "SUGER", "TEA") # old names

```
names.level1<-c("cake", "cake", "breakfast", "breakfast", "drink", "breakfast", "breakfast","cake", "drink", "cake", "drink") # new names

itemInfo(trans1)<-data.frame(labels = names.real, level1 = names.level1)
itemInfo(trans1)
  labels    level1
1     BISCUIT      cake
2   BOURNVITA      cake
3       BREAD breakfast
4        COCK breakfast
5      COFFEE     drink
6  CORNFLAKES breakfast
7         JAM breakfast
8       MAGGI      cake
9        MILK     drink
10      SUGER      cake
11        TEA     drink

trans1_level2<-aggregate(trans1, by="level1")
trans1
inspect(trans1) # transactions with old names
inspect(trans1_level2) # transactions with new names
items
[1]  {breakfast, cake, drink}
[2]  {breakfast, cake, drink}
[3]  {breakfast, cake, drink}
[4]  {breakfast, cake, drink}
[5]  {cake, drink}
[6]  {breakfast, cake, drink}
[7]  {breakfast, cake, drink}
```

[8] {breakfast, cake, drink}
[9] {breakfast, cake, drink}
[10] {breakfast, drink}
[11] {breakfast, cake, drink}
[12] {breakfast, cake, drink}
[13] {cake, drink}
[14] {breakfast, drink}
[15] {breakfast, cake}
[16] {breakfast, cake, drink}
[17] {breakfast, cake}
[18] {breakfast, cake, drink}
[19] {breakfast, cake, drink}
[20] {breakfast, drink}

The following analysis of the rules at a higher level of aggregation shows that it is easier to draw conclusions about purchasing patterns

```
rules.trans1_lev2<-apriori(trans1_level2, parameter=list(supp=0.1, conf=0.5))
Apriori
Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
maxlen target  ext
        0.5    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules TRUE
Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE
Absolute minimum support count: 2

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[3 item(s), 20 transaction(s)] done [0.00s].
```

```
sorting and recoding items ... [3 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [12 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].

rules.by.conf<-sort(rules.trans1_lev2, by="confidence", decreasing=TRUE)
inspect(head(rules.by.conf))
lhs            rhs         support confidence coverage lift      count
[1] {}          => {breakfast} 0.90    0.9000000  1.00     1.0000000 18
[2] {}          => {drink}     0.90    0.9000000  1.00     1.0000000 18
[3] {breakfast} => {drink}     0.80    0.8888889  0.90     0.9876543 16
[4] {drink}     => {breakfast} 0.80    0.8888889  0.90     0.9876543 16
[5] {cake}      => {breakfast} 0.75    0.8823529  0.85     0.9803922 15
[6] {cake}      => {drink}     0.75    0.8823529  0.85     0.9803922 15

trans<-random.transactions(nItems=10, nTrans=20, method="independent",
verbose=FALSE)
image(trans)
inspect(trans)
items                                     transactionID
[1]  {item4, item6}                             trans1
[2]  {item3, item5, item7}                      trans2
[3]  {item3, item4, item9}                      trans3
[4]  {item3}                                    trans4
[5]  {item2, item7}                             trans5
[6]  {item1, item2, item3, item4, item7, item10} trans6
[7]  {item9}                                    trans7
[8]  {item1, item6, item10}                     trans8
[9]  {item1, item4, item5, item7, item9}        trans9
```

[10] {item5, item7, item8}   trans10

[11] {item6, item10}   trans11

[12] {item1, item2, item3, item10}   trans12

[13] {item3, item10}   trans13

[14] {item1, item3, item5, item9}   trans14

[15] {item2, item3, item4, item5, item6}   trans15

[16] {item1, item5}   trans16

[17] {item2, item3}   trans17

[18] {item3}   trans18

[19] {item2, item4, item9}   trans19

[20] {item7}   trans20

Based on the drawn data, we can create rules (apriori()) and view them (inspect(), sort()), check their length (size()) and how many were created (legnth()), create data sets (eclat())

```
rules.random<-apriori(trans, parameter=list(supp=0.05, conf=0.3))
inspect(rules.random)
rules.by.conf<-sort(rules.random, by="confidence", decreasing=TRUE)
inspect(rules.by.conf)
size(rules.by.conf)
length(rules.by.conf)
freq.items<-eclat(trans, parameter=list(supp=0.25, maxlen=15)) # basic eclat
inspect(freq.items)
items    support count
[1] {item3}  0.50    10
[2] {item4}  0.30     6
[3] {item1}  0.30     6
[4] {item5}  0.30     6
[5] {item2}  0.30     6
```

[6] {item7} 0.30   6

[7] {item10} 0.25   5

[8] {item9} 0.25   5

Conclusion

Association rules allow us to find interesting patterns in customers preferences. In this study the dataset from market basket has been analyzed. The most popular item in this basket is bread, that is a pretty expected result. Method of search ECLAT algorithm provides faster solution for founding association rules but uses more memory than Apriori algorithm.