Association Rule Mining using R

CIS400/600 Fundamentals of Data and Knowledge Mining

1. **Introduction**

   (a) Install **"arules"** package and import **"arules"** using

   ```
   > install.packages("arules")
   > library(arules)
   ```

   (b) CSV files are read in R using

   ```
   > read.csv("sample.csv")
   ```

   (c) Datasets having transactions in CSV format (viz. **groceries.csv**) are read as

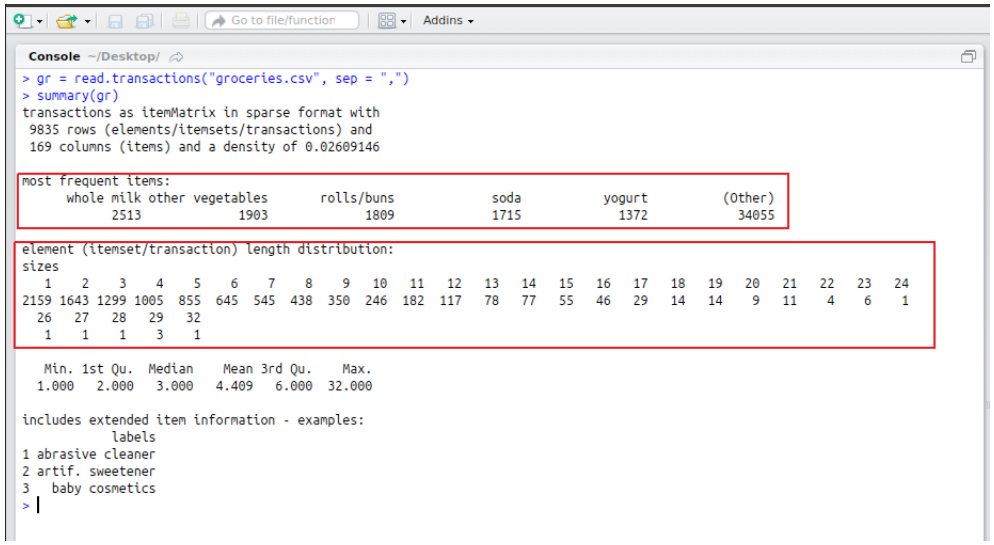
   ```
   > gr = read.transactions("groceries.csv", sep = ",")
   ```

   read.transactions() works when using **arules** package. Therefore it is necessary to first import **arules** package as mentioned in (a)

   (d) Summary of dataset having transactions is available using

   ```
   > summary(gr)
   ```

   (e) Item frequency and number of transactions having one item, two items, three items and so on are all available in summary.



```
> gr = read.transactions("groceries.csv", sep = ",")
> summary(gr)
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           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.
  1.000   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
>
```

It can be seen that **whole milk** is the most frequent item among all the transactions. Also, number of transactions with only one item is 2159, no. of transactions having two items is 1645

2. **Understanding item frequencies**

   (a) Frequencies of items in a transaction can be well understood using functions viz. **itemFrequency()**, **itemFrequencyPlot()** etc.

   (b) Items along with their frequencies are output using the command

```
> itemFrequency(gr)
```

   (c) **sort()** function is used to sort items according to frequencies. Sorting by default is in ascending order. To print top 10 items with minimum frequency

```
> sort(itemFrequency(gr))[1:10]
```

   Likewise, to print top 10 items with maximum frequency

```
> sort(itemFrequency(gr), decreasing = TRUE)[1:10]
```

   (d) Some interesting functions to be noted are

```
> itemFrequencyPlot(gr, support = 0.05)
```

```
> itemFrequencyPlot(gr, topN = 10)
```

The first command helps in generating histograms listing items with support value of 0.05. The function also takes "lift" as parameter.
The second command outputs histogram with top 10 highest frequency items

3. **Apriori algorithm**

   (a) To generate rules using **"Apriori"** algorithm, the command used is

```
> rules = apriori(gr)
```

It has to be noted that, the default parameters for **apriori()** are

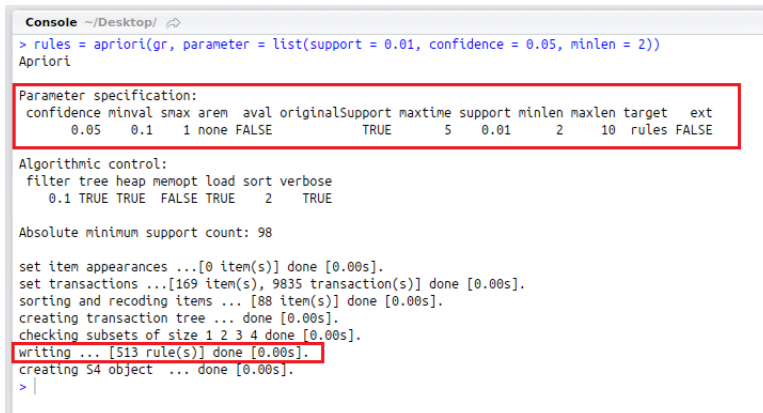
```
> rules = apriori(gr)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 983

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 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].
writing ... [0 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
>
>
```

Note: For these default parameters, there are no rules generated

```
> rules = apriori(gr)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 983

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 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].
writing ... [0 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
>
>
```

(b) Rules are generated with parameter specifications as below

```
> rules = apriori(gr, parameter = list(support =
    0.01, confidence = 0.05, minlen = 2))
```



```
Console ~/Desktop/
> rules = apriori(gr, parameter = list(support = 0.01, confidence = 0.05, minlen = 2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
       0.05    0.1    1 none FALSE            TRUE       5    0.01      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: 98

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [513 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
>
```

It can be seen that the default parameters have now been changed to user-specified parameters and the number of rules generated is 513

(c) Rules are printed using

```
> inspect(rules)
```

(d) Rules can be sorted according to support, confidence and lift and printed. Sorting is done in descending order

```
> inspect(sort(rules, by = "support"))
> inspect(sort(rules, by = "confidence"))
> inspect(sort(rules, by = "list"))
```

(e) **arules::subset** viz. **subset()** function helps in identifying rules with specific items

Among the generated rules **"rules"**, the below command selects rules that have **whole milk** in RHS part of the rule X => Y

```
> whole_milk_rules = subset(rules, rhs %in% "whole
    milk")
> inspect(whole_milk_rules)
```

Among the generated rules **"rules"**, the below command selects rules that have **whole milk** in LHS part of the rule X => Y

```
> whole_milk_rules = subset(rules, lhs %in% "whole
  milk")
> inspect(whole_milk_rules)
```

Among the generated rules **"rules"**, the below command selects rules that have only **whole milk** and **soda** in the LHS part of the rule X => Y

```
> whole_milk_soda_rules = subset(rules, lhs %ain%
  c("whole milk", "soda"))
> inspect(whole_milk_soda_rules)
```

Read through **arules::subset** for more details