4DATA
GRADED PROJECT

Opportunjuray Kouka
Samia Lettat
Thomas Atger-Lecaplain

Table of Contents

Part 1: Analyse a data set

1. Download the arules package from CRAN site https://cran.r-project.org/ (Links to an external site.) and load it in RStudio

install.packages("arules")

library("arules")

2. Load the Groceries data set that comes with the arules package

data("Groceries")

3. Use the R functions that you have already seen in class to explore the data set in order to answer the following questions:
   a. How many items are in the data set?

ncol(Groceries)

Or

length(Groceries@itemInfo$labels)

Result: 169

   b. How many transactions are in the data set?

nrow(Groceries)

Result: 9835

   c. Calculate the density value

summary(Groceries)

Result: 0.02609146

   d. Give the interpretation of this density value

The matrix contains 2,6% of non-zero cells.

   e. Which are the most commonly found items in the data set?

items_freq = as.data.frame(table(Groceries@data@i))

items_freq$label = Groceries@itemInfo$labels

head(items_freq[order(-items_freq$Freq),],10)

   f. What percentage of transactions contain yogurt ?

itemFrequency(Groceries)[["yogurt"]]*100

Result: 13,95%

Or

items_freq$Freq[which(items_freq$label=="yogurt")]/nrow(Groceries)*100

Result: 13,95%

g. How many transactions have only seven items ?

Groceries[which(size(Groceries) == 7)]

Result: 545

h. How many transactions have two items ?

Groceries[which(size(Groceries) == 2)]

Result: 1643

i. What is the average number of items in a transaction ?

mean(size(Groceries))

Result: 4.409456

j. Bonus: Own analysis

<u>Most items in a transaction:</u>

max(size(Groceries))

Result: 32

<u>Median amount of items in transactions:</u>

median(size(Groceries))

Result: 3

4. Which R function will you use to display transactions 4 to 8.

LIST(Groceries)[4:8]

Or

as(Groceries,"list")[4:8]

5. Which R function will you use to see the proportion of transactions that contain the first item:

subset(Groceries, items %in% as(Groceries,"list")[[1]][1])

Result: 814

6. Which R function will you use to view the proportion of transactions that contain a number of other items (for example items 3 to 7)

Inspect(Groceries[3:7])

7. Which R function will you use to visually display the proportion of transactions containing items that have at least 15% support

itemFrequency(Groceries,type="relative")[as.numeric(itemFrequency(Groceries,type="relative")) > 0.15]

Result:

```
other vegetables      whole milk      rolls/buns         soda
      0.1934926       0.2555160       0.1839349       0.1743772
```

8. Display the proportions of the top 10 items

```
itemFrequency(Groceries,type="absolute")[1:10]
```

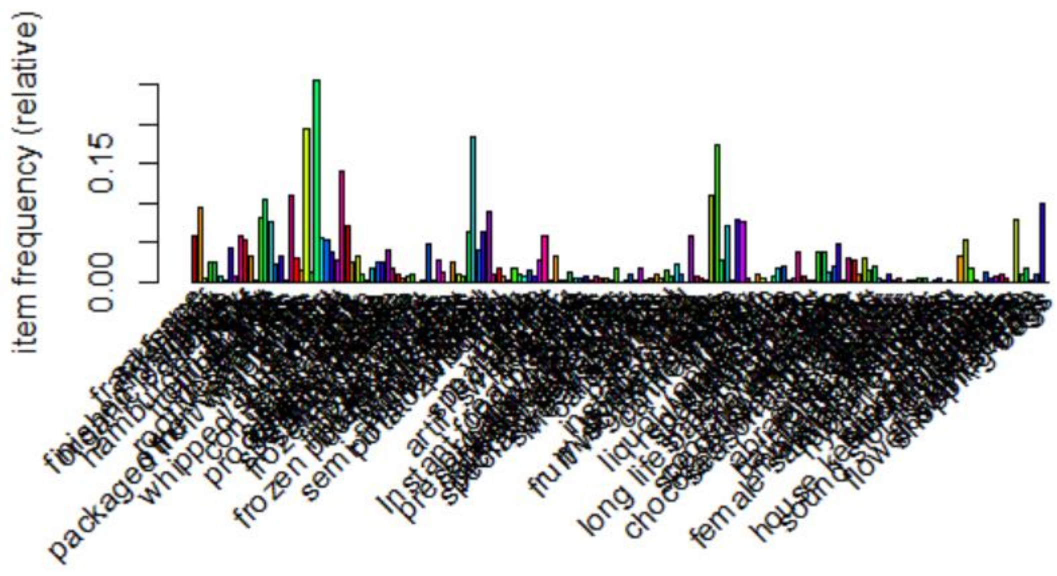
Result:

```
frankfurter        sausage       liver loaf          ham          meat finished products
          580           924            50           256           254            64
 organic sausage        chicken          turkey          pork
           22            422            80           567
```

9. Visualize the entire sparse matrix, including all the items of the data set

```
itemFrequencyPlot(Groceries, col=rainbow(10))
```

It is also possible to use the ***sample*** *function, but it is barely readable with every single transaction represented.*



Part 2: Create the association rules

1. Load Data and preprocess text; Find the association items in the Groceries data set using the default support and confidence

Definitions:

- Support: How popular an itemset is, as measured by the proportion of transactions in which an itemset appears.
- Confidence: How likely item Y is purchased when item X is purchased.
- Lift: How likely item Y is purchased when item X is purchased, while controlling for how popular item Y is.

Source

Default support and confidence:

Default minimum support: 0.1

Default minimum confidence: 0.8

Source (also appears in the output when running apriori without parameters)

Association items with default parameters using apriori:

rules <- apriori(Groceries)

Result: **No rules!**

2. Try different support and confidence levels to generate different numbers of rules and explain what it means

Example of Apriori with different support and confidence:
rules <- apriori(Groceries, parameter = list(supp = 0.01, conf = 0.5))

summary(rules)

Result: Set of 15 rules

Explanation:

By changing support from 0.1 to 0.01, we are including products that are in 1 in 100 transactions instead of 1 in 10 in our rules.

When using a confidence of 0.5 instead of 0.8, we are lowering the support threshold required for a product purchased at the same time as another: in our case, it means that more groceries will be included in the rules when they are present in the same transaction.

3. Which R function will you use to get a high-level overview of the generated rules and to answer the following questions:

inspect(rules)

   a. How many rules have 3 items (lhs and rhs)?

rules[size(rules) == 3]

Result: set of x rules

To display the rules themselves:

```
inspect(rules[size(rules) == 3])
```

b. Take a closer look at the first 10 rules.

```
inspect(rules[1:10])
```

c. Are these rules interesting? What three categories are used to define "interesting"?

An interesting rule could be characterized by its lift, support and confidence.

Since our dataset contains just short of ten thousand transactions, the top 10 rules of all those we created in question 2 (support and confidence of at least 0.01 and 0.5 respectively) represent popular products, and those rules have a decently high lift (i.e. >2), which means they exceed the expected confidence of independent products taken on their own. As such, they could indeed be defined as interesting.

4. Take a closer look at the ten best rules.

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

```
inspect(rules[1:10])
```

Result: The top 10 rules with the highest lift in the ruleset

5. Find any rules that contain the word "chocolate" and store it in chocrules object

```
chocrules = subset(rules, subset = lhs %in% "chocolate" | rhs %in% "chocolate")
```

6. View the rules

```
inspect(chocrules)
```

Part 3: Create a recommendation engine

- Identify unique users and items from the data set
- Use frequently co-occurring items to make recommendations to users about items that frequently co-exist with items they have given high ratings to

To do this, you need to use the package: plyr

Hints:

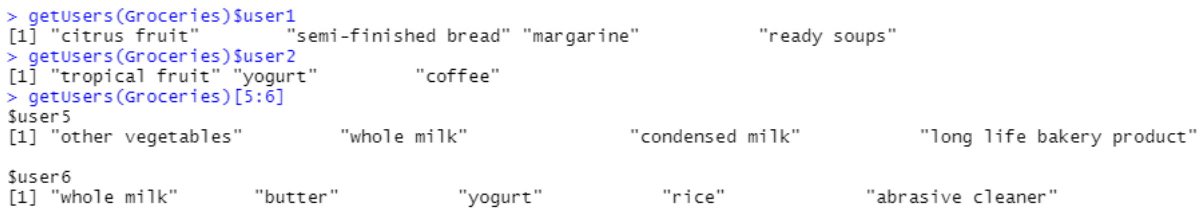
- Create a function to calculate a list of unique users (Think carefully about how to do it)

```
getUsers <- function(tsct){
  tmp <- list()
  toList <- as(tsct, 'list')
  for (i in 1:tsct@data@Dim[2]){
    tmp[[paste("user", i, sep = '')]] <- toList[[i]]
  }
  return(tmp)
}
```

Usage:

getUsers(Groceries) returns a list indexing each users.

A single user's basket can be displayed with *getUsers(Groceries)$userX* where X is the user's number.



```
> getUsers(Groceries)$user1
[1] "citrus fruit"        "semi-finished bread" "margarine"           "ready soups"
> getUsers(Groceries)$user2
[1] "tropical fruit" "yogurt"         "coffee"
> getUsers(Groceries)[5:6]
$user5
[1] "other vegetables"         "whole milk"               "condensed milk"           "long life bakery product"

$user6
[1] "whole milk"        "butter"            "yogurt"            "rice"              "abrasive cleaner"
```

- Create a function to calculate a list of unique items.

A list composed of each items can be created by using the following statement:

itemlist <- Groceries@itemInfo$labels

- Establish Product List Index.
- Create a recommendation function
- Generate recommendations for each of the users

Python Recommendation Engine

Presentation

Due to various difficulties with using R, we have opted to use Python instead to create the recommendation engine.

Dubbed “recomengine.py”, the script does the following actions:

1. Imports the dataset from a csv file using the *pandas* library
2. Parses the dataset and converts it into a list
3. Generates association rules using the apyori library
4. Prompts the user for a client number, and calculates recommended products by the following method:
    a. **Every possible** combinations of items from the client’s basket are calculated using the *itertools* library
    b. Each combination is tested against **every** generated rules’ LHS values (both are sorted just beforehand)
    c. If a rule matches, its RHS value is added to a list
    d. Duplicates are removed from the matched item list, and its content is printed out

Usage

The script must be placed in the same directory as the “groceries.csv” file.

Dependencies can easily be installed using the requirements.txt file, with the following command (ideally in a virtual environment):

```
pip install -r requirements.txt
```

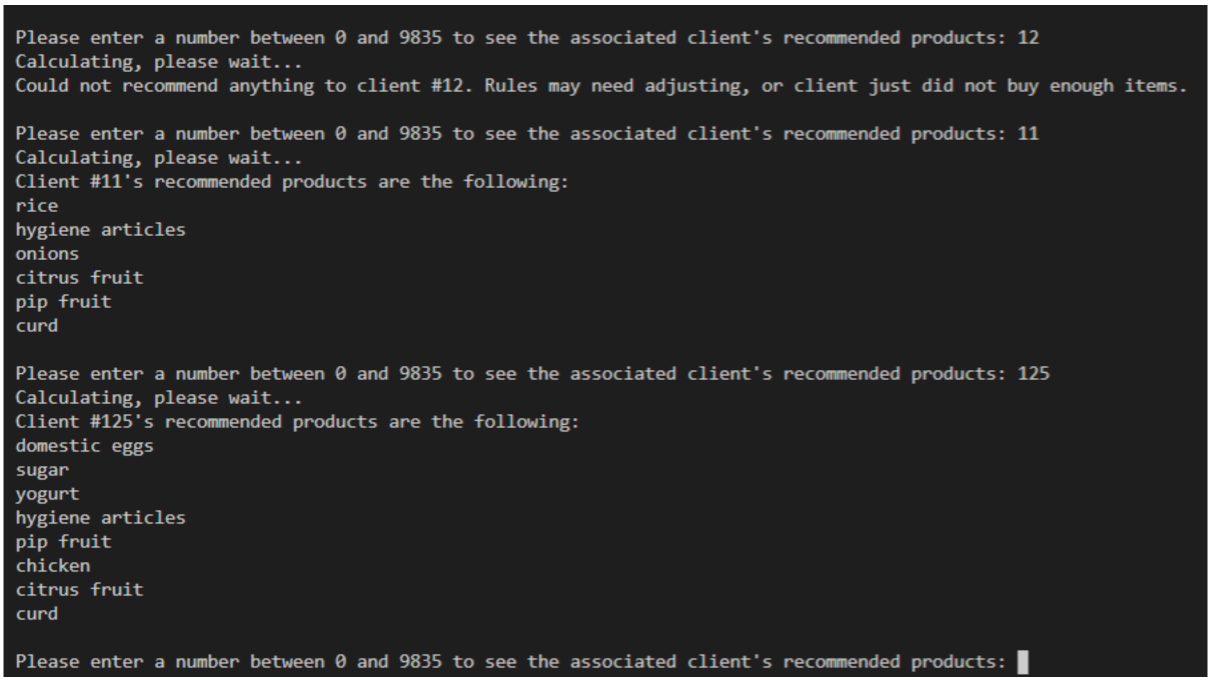
The script can then be started with:

```
python recomengine.py
```

Once the script prompts for a client number, it can be stopped by using Ctrl+C.

The rules generation parameters (i.e. minimum support, confidence, lift, number of items…) can be edited at line 54.

Screenshot



```
Please enter a number between 0 and 9835 to see the associated client's recommended products: 12
Calculating, please wait...
Could not recommend anything to client #12. Rules may need adjusting, or client just did not buy enough items.

Please enter a number between 0 and 9835 to see the associated client's recommended products: 11
Calculating, please wait...
Client #11's recommended products are the following:
rice
hygiene articles
onions
citrus fruit
pip fruit
curd

Please enter a number between 0 and 9835 to see the associated client's recommended products: 125
Calculating, please wait...
Client #125's recommended products are the following:
domestic eggs
sugar
yogurt
hygiene articles
pip fruit
chicken
citrus fruit
curd

Please enter a number between 0 and 9835 to see the associated client's recommended products: █
```

Potential improvements

As it stands, the engine is somewhat barebones and not very optimized.

The following additions or modifications would have been welcome, given the time:

- A proper configuration method (i.e. using arguments, a configuration file, …)
- A better "man-machine interface" (either through a GUI, or a command-line interface)
- Display the reason why a product is recommended (with the related rule)
- Optimization, such as
  - Only testing combinations that could fit a rule's minimum LHS as defined in Apriori's parameters
  - Only testing rules that contains items that actually are in the client's basket (through a check beforehand)
- Make the script conform to PEP8 and document it further (although it is nearly entirely explained through comments)
- Using the *logging* library to help debugging the engine