

R Notebook

Problem 1

Build an R Notebook of the concrete strength example in the textbook on pages 232 to 239. Show each step and add appropriate documentation.

```
#setwd('~/Documents/ML/')  
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(dplyr)
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
```

```
##      filter, lag
```

```
## The following objects are masked from 'package:base':
```

```
##
```

```
##      intersect, setdiff, setequal, union
```

```
library(neuralnet)
```

```
##
```

```
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
```

```
##
```

```
##      compute
```

```
library(kernlab)
```

```
##
```

```
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
```

```
##
```

```
##      alpha
```

```
#install.packages("arules")
```

```
library(arules)
```

```
## Loading required package: Matrix
```

```
##
```

```
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:kernlab':
```

```
##
```

```
##      size
```

```
## The following object is masked from 'package:dplyr':
```

```
##
```

```
##      recode
```

```
## The following objects are masked from 'package:base':
##
## abbreviate, write
```

Step 1 – exploring and preparing the data

```
concrete <- read.csv("concrete.csv")
str(concrete)
```

```
## 'data.frame': 1030 obs. of 9 variables:
## $ cement : num 540 540 332 332 199 ...
## $ slag : num 0 0 142 142 132 ...
## $ ash : num 0 0 0 0 0 0 0 0 0 ...
## $ water : num 162 162 228 228 192 228 228 228 228 ...
## $ superplastic: num 2.5 2.5 0 0 0 0 0 0 0 ...
## $ coarseagg : num 1040 1055 932 932 978 ...
## $ fineagg : num 676 676 594 594 826 ...
## $ age : int 28 28 270 365 360 90 365 28 28 28 ...
## $ strength : num 80 61.9 40.3 41 44.3 ...
```

```
# Normalize the data
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
concrete_norm <- as.data.frame(lapply(concrete, normalize))
summary(concrete_norm$strength)
```

```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.0000 0.2664 0.4001 0.4172 0.5457 1.0000
```

```
summary(concrete$strength)
```

```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 2.33 23.71 34.45 35.82 46.13 82.60
```

```
# Partition the data into a training set with 75 percent of the examples and a testing set with 25 percent
concrete_train <- concrete_norm[1:773, ]
concrete_test <- concrete_norm[774:1030, ]
```

Step 2 – training a model on the data

```
# We'll begin by training the simplest multilayer feedforward network with only a single hidden node:
concrete_model <- neuralnet(strength ~ cement + slag
+ ash + water + superplastic + coarseagg + fineagg + age,
data = concrete_train)
plot(concrete_model)
```

Step 3 – evaluating model performance

```
# Generate predictions on the test dataset
model_results <- compute(concrete_model, concrete_test[1:8])
# Get the predicted values
predicted_strength <- model_results$net.result

# Find correlation between our predicted concrete strength and the true value
cor(predicted_strength, concrete_test$strength)
```

```
## [1,]
## [1,] 0.7251732948
```

Step 4 – improving model performance

```
# Increase the number of hidden nodes to five

concrete_model2 <- neuralnet(strength ~ cement + slag +
ash + water + superplastic +
coarseagg + fineagg + age,
data = concrete_train, hidden = 5)
plot(concrete_model2)
model_results2 <- compute(concrete_model2, concrete_test[1:8])
predicted_strength2 <- model_results2$net.result
cor(predicted_strength2, concrete_test$strength)

##           [,1]
## [1,] 0.5738298702
```

Problem 2

Build an R Notebook of the optical character recognition example in the textbook on pages 249 to 257. Show each step and add appropriate documentation.

Step 1 – exploring and preparing the data

```
letters <- read.csv("letterdata.csv")
str(letters)

## 'data.frame': 20000 obs. of 17 variables:
## $ letter: Factor w/ 26 levels "A","B","C","D",...: 20 9 4 14 7 19 2 1 10 13 ...
## $ xbox : int 2 5 4 7 2 4 4 1 2 11 ...
## $ ybox : int 8 12 11 11 1 11 2 1 2 15 ...
## $ width : int 3 3 6 6 3 5 5 3 4 13 ...
## $ height: int 5 7 8 6 1 8 4 2 4 9 ...
## $ onpix : int 1 2 6 3 1 3 4 1 2 7 ...
## $ xbar : int 8 10 10 5 8 8 8 8 10 13 ...
## $ ybar : int 13 5 6 9 6 8 7 2 6 2 ...
## $ x2bar : int 0 5 2 4 6 6 6 2 2 6 ...
## $ y2bar : int 6 4 6 6 6 9 6 2 6 2 ...
## $ xybar : int 6 13 10 4 6 5 7 8 12 12 ...
## $ x2ybar: int 10 3 3 4 5 6 6 2 4 1 ...
## $ xy2bar: int 8 9 7 10 9 6 6 8 8 9 ...
## $ xedge : int 0 2 3 6 1 0 2 1 1 8 ...
## $ xedgey: int 8 8 7 10 7 8 8 6 6 1 ...
## $ yedge : int 0 4 3 2 5 9 7 2 1 1 ...
## $ yedgex: int 8 10 9 8 10 7 10 7 7 8 ...

# create training and testing data frames
letters_train <- letters[1:16000, ]
letters_test <- letters[16001:20000, ]
```

Step 2 – training a model on the data

```
letter_classifier <- ksvm(letter ~ ., data = letters_train,
kernel = "vanilladot")

## Setting default kernel parameters
letter_classifier

## Support Vector Machine object of class "ksvm"
##
```

```
## SV type: C-svc (classification)
## parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 7037
##
## Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524 -32.7694 -49.9786 -18.1824 -62
## Training error : 0.130062
```

Step 3 – evaluating model performance

```
# Make predictions
letter_predictions <- predict(letter_classifier, letters_test)
head(letter_predictions)
```

```
## [1] U N V X N H
## Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

```
table(letter_predictions, letters_test$letter)
```

```
##
## letter_predictions  A  B  C  D  E  F  G  H  I  J  K  L  M  N
##                   A 144  0  0  0  0  0  0  0  0  1  0  0  1  2
##                   B  0 121  0  5  2  0  1  2  0  0  1  0  1  0
##                   C  0  0 120  0  4  0 10  2  2  0  1  3  0  0
##                   D  2  2  0 156  0  1  3 10  4  3  4  3  0  5
##                   E  0  0  5  0 127  3  1  1  0  0  3  4  0  0
##                   F  0  0  0  0  0 138  2  2  6  0  0  0  0  0
##                   G  1  1  2  1  9  2 123  2  0  0  1  2  1  0
##                   H  0  0  0  1  0  1  0 102  0  2  3  2  3  4
##                   I  0  1  0  0  0  1  0  0 141  8  0  0  0  0
##                   J  0  1  0  0  0  1  0  2  5 128  0  0  0  0
##                   K  1  1  9  0  0  0  2  5  0  0 118  0  0  2
##                   L  0  0  0  0  2  0  1  1  0  0  0 133  0  0
##                   M  0  0  1  1  0  0  1  1  0  0  0  0 135  4
##                   N  0  0  0  0  0  1  0  1  0  0  0  0  0 145
##                   O  1  0  2  1  0  0  1  2  0  1  0  0  0  1
##                   P  0  0  0  1  0  2  1  0  0  0  0  0  0  0
##                   Q  0  0  0  0  0  0  8  2  0  0  0  3  0  0
##                   R  0  7  0  0  1  0  3  8  0  0 13  0  0  1
##                   S  1  1  0  0  1  0  3  0  1  1  0  1  0  0
##                   T  0  0  0  0  3  2  0  0  0  0  1  0  0  0
##                   U  1  0  3  1  0  0  0  2  0  0  0  0  0  0
##                   V  0  0  0  0  0  1  3  4  0  0  0  0  1  2
##                   W  0  0  0  0  0  0  1  0  0  0  0  0  2  0
##                   X  0  1  0  0  2  0  0  1  3  0  1  6  0  0
##                   Y  3  0  0  0  0  0  0  1  0  0  0  0  0  0
##                   Z  2  0  0  0  1  0  0  0  3  4  0  0  0  0
##
## letter_predictions  O  P  Q  R  S  T  U  V  W  X  Y  Z
##                   A  2  0  5  0  1  1  1  0  1  0  0  1
##                   B  0  2  2  3  5  0  0  2  0  1  0  0
##                   C  2  0  0  0  0  0  0  0  0  0  0  0
##                   D  5  3  1  4  0  0  0  0  0  3  3  1
##                   E  0  0  2  0 10  0  0  0  0  2  0  3
```

```
##           F    0  16    0    0    3    0    0    1    0    1    2    0
##           G    1    2    8    2    4    3    0    0    0    1    0    0
##           H   20    0    2    3    0    3    0    2    0    0    1    0
##           I    0    1    0    0    3    0    0    0    0    5    1    1
##           J    1    1    3    0    2    0    0    0    0    1    0    6
##           K    0    1    0    7    0    1    3    0    0    5    0    0
##           L    0    0    1    0    5    0    0    0    0    0    0    1
##           M    0    0    0    0    0    0    3    0    8    0    0    0
##           N    0    0    0    3    0    0    1    0    2    0    0    0
##           O   99    3    3    0    0    0    3    0    0    0    0    0
##           P    2  130    0    0    0    0    0    0    0    0    1    0
##           Q    3    1  124    0    5    0    0    0    0    0    2    0
##           R    1    1    0  138    0    1    0    1    0    0    0    0
##           S    0    0   14    0  101    3    0    0    0    2    0   10
##           T    0    0    0    0    3  133    1    0    0    0    2    2
##           U    1    0    0    0    0    0  152    0    0    1    1    0
##           V    1    0    3    1    0    0    0  126    1    0    4    0
##           W    0    0    0    0    0    0    4    4  127    0    0    0
##           X    1    0    0    0    1    0    0    0    0  137    1    1
##           Y    0    7    0    0    0    3    0    0    0    0  127    0
##           Z    0    0    0    0   18    3    0    0    0    0    0  132
```

```
# The following command returns a vector of TRUE or FALSE values, indicating whether the model's predic
agreement <- letter_predictions == letters_test$letter
table(agreement)
```

```
## agreement
## FALSE  TRUE
##   643  3357
```

```
prop.table(table(agreement))
```

```
## agreement
##   FALSE    TRUE
## 0.16075 0.83925
```

Step 4 – improving model performance

```
# Choose the Gaussian RBF kernel
letter_classifier_rbf <- ksvm(letter ~ ., data = letters_train,
kernel = "rbfdot")
# Make prediction
letter_predictions_rbf <- predict(letter_classifier_rbf,
letters_test)
# Compare the accuracy to our linear SVM
agreement_rbf <- letter_predictions_rbf == letters_test$letter
table(agreement_rbf)
```

```
## agreement_rbf
## FALSE  TRUE
##   277  3723
```

```
prop.table(table(agreement_rbf))
```

```
## agreement_rbf
##   FALSE    TRUE
## 0.06925 0.93075
```

Problem 3

Build an R Notebook of the grocery store transactions example in the textbook on pages 266 to 284. Show each step and add appropriate documentation.

```
groceries <- read.transactions("groceries.csv", sep = ",")
summary(groceries)

## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609145577
##
## 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.
## 1.000000  2.000000  3.000000  4.409456  6.000000 32.000000
##
## includes extended item information - examples:
##      labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3  baby cosmetics

inspect(groceries[1:5])

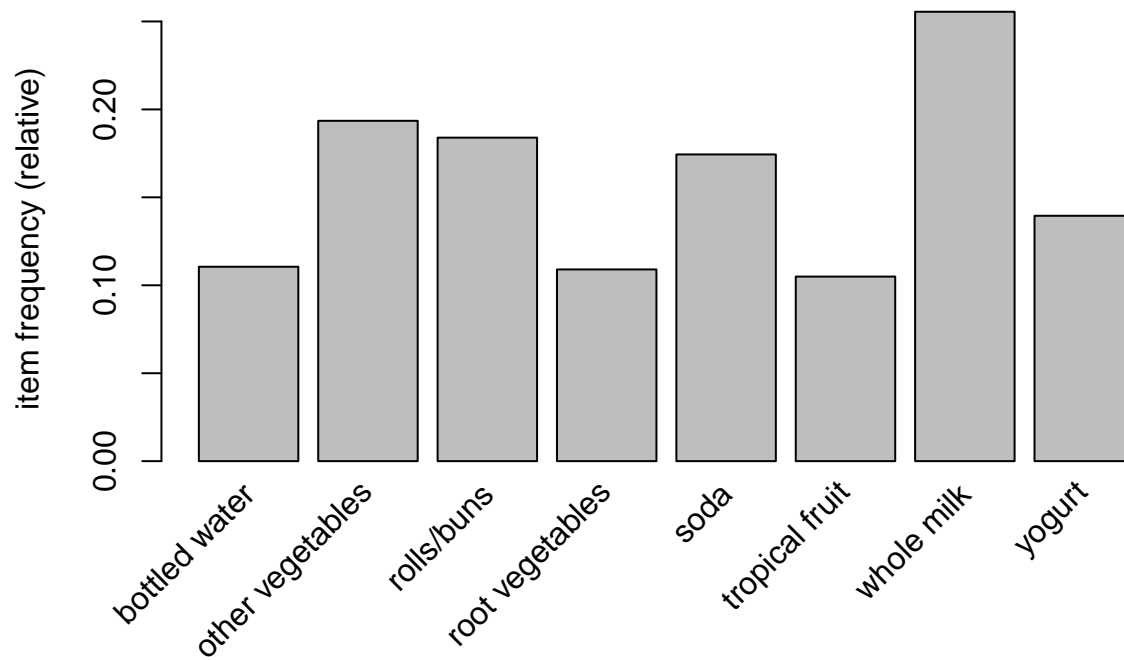
##      items
## [1] {citrus fruit,
##      margarine,
##      ready soups,
##      semi-finished bread}
## [2] {coffee,
##      tropical fruit,
##      yogurt}
## [3] {whole milk}
## [4] {cream cheese,
##      meat spreads,
##      pip fruit,
##      yogurt}
## [5] {condensed milk,
##      long life bakery product,
##      other vegetables,
##      whole milk}

# View the support level for the first three items in the grocery data
itemFrequency(groceries[, 1:3])
```

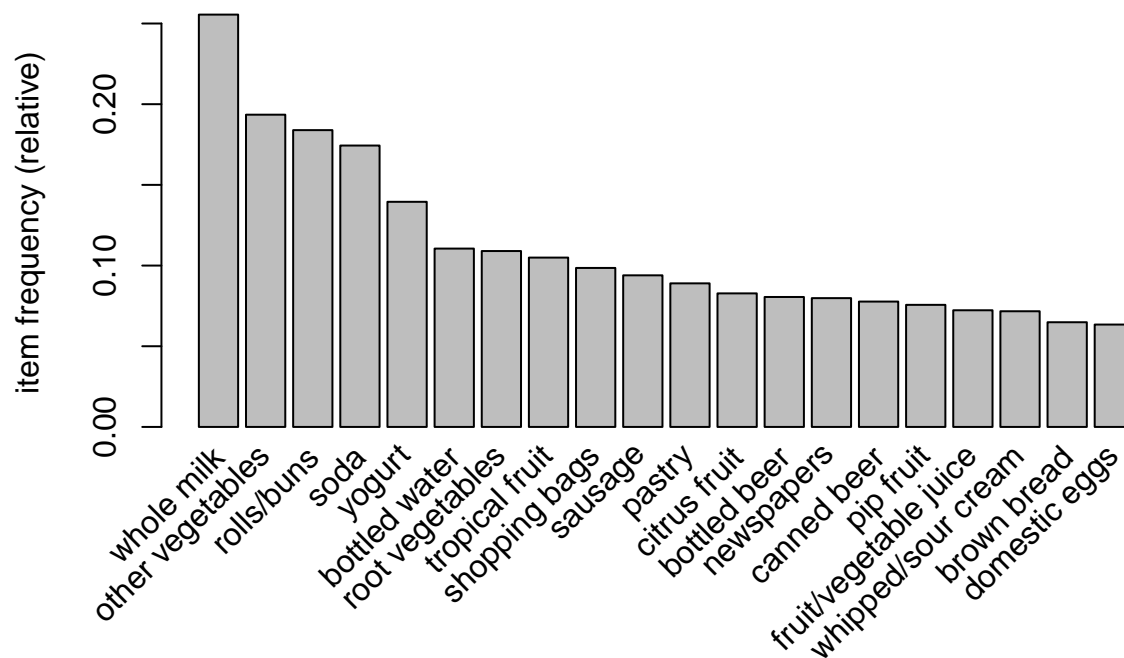
```
## abrasive cleaner artif. sweetener baby cosmetics
## 0.0035587188612 0.0032536858160 0.0006100660905
```

Visualizing item support – item frequency plots

```
# Visualize items appearing in a minimum proportion of transactions, with the support parameter using i
itemFrequencyPlot(groceries, support = 0.1)
```

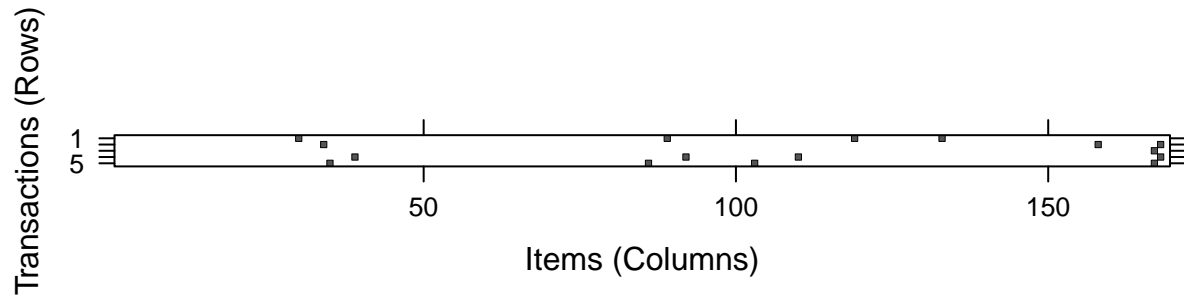


```
# See the top 20 items with the topN parameter
itemFrequencyPlot(groceries, topN = 20)
```



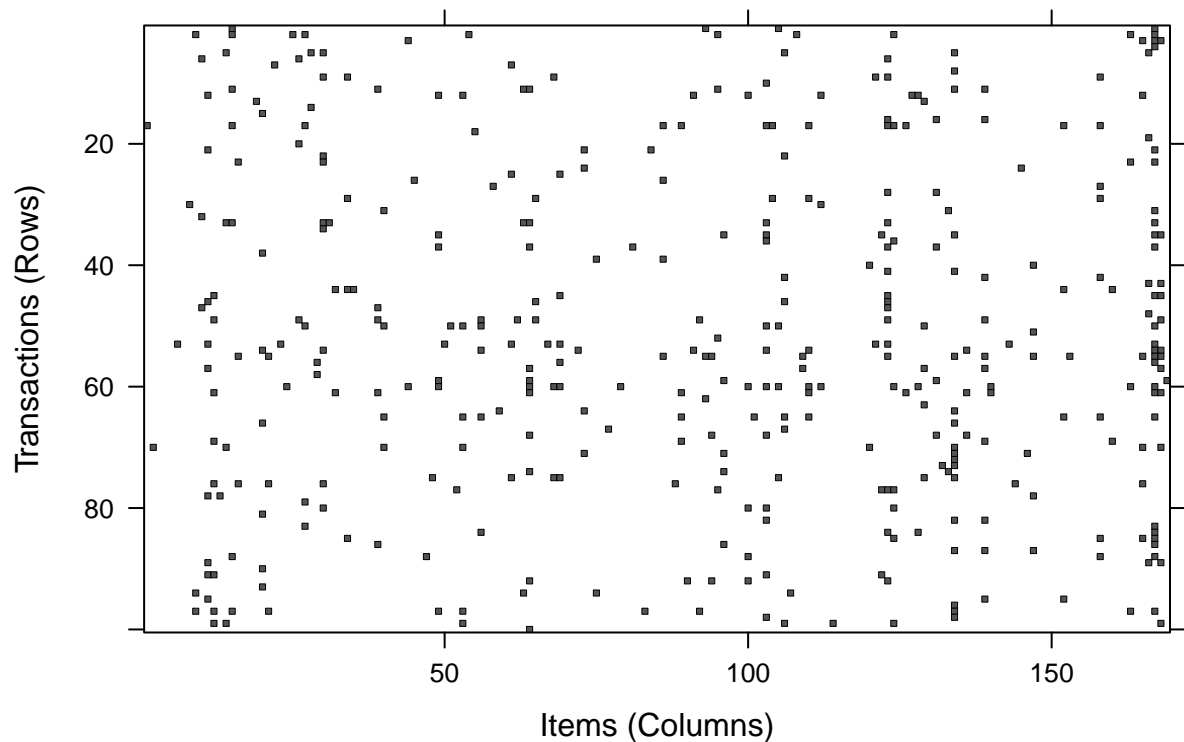
Visualizing the transaction data – plotting the sparse matrix

```
image(groceries[1:5])
```



```
# create random selection of 100 transactions
```

```
image(sample(groceries, 100))
```



Step 3 – training a model on the data

```
# Use the default settings of support = 0.1 and confidence = 0.8
```

```
apriori(groceries)
```

```
## Apriori
```

```
##
```

```
## Parameter specification:
```

```
## confidence minval smax arem aval originalSupport maxtime support minlen
```

```
##           0.8    0.1    1 none FALSE             TRUE     5    0.1    1
```

```
## maxlen target   ext
```

```
##      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].

## set of 0 rules
# Find a set of association rules using the Apriori algorithm
groceryrules <- apriori(groceries, parameter = list(support =
0.006, confidence = 0.25, minlen = 2))

## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
## 0.25 0.1 1 none FALSE TRUE 5 0.006 2
## maxlen target ext
## 10 rules FALSE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE
##
## Absolute minimum support count: 59
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [463 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].

groceryrules

## set of 463 rules

summary(groceryrules)

## set of 463 rules
##
## rule length distribution (lhs + rhs):sizes
## 2 3 4
## 150 297 16
##
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 2.000000 2.000000 3.000000 2.710583 3.000000 4.000000
##
## summary of quality measures:
## support confidence lift
## Min. :0.006100661 Min. :0.2500000 Min. :0.9932367

```

```
## 1st Qu.:0.007117438 1st Qu.:0.2970711 1st Qu.:1.6229230
## Median :0.008744281 Median :0.3553719 Median :1.9332351
## Mean :0.011539429 Mean :0.3785573 Mean :2.0350922
## 3rd Qu.:0.012302999 3rd Qu.:0.4494849 3rd Qu.:2.3564791
## Max. :0.074834774 Max. :0.6600000 Max. :3.9564774
## count
## Min. : 60.0000
## 1st Qu.: 70.0000
## Median : 86.0000
## Mean :113.4903
## 3rd Qu.:121.0000
## Max. :736.0000
##
## mining info:
## data ntransactions support confidence
## groceries 9835 0.006 0.25
```

```
inspect(groceryrules[1:3])
```

```
## lhs rhs support confidence
## [1] {pot plants} => {whole milk} 0.006914082359 0.4000000000
## [2] {pasta} => {whole milk} 0.006100660905 0.4054054054
## [3] {herbs} => {root vegetables} 0.007015760041 0.4312500000
## lift count
## [1] 1.565459610 68
## [2] 1.586614470 60
## [3] 3.956477379 69
```

Step 5 – improving model performance

Sorting the set of association rules

```
inspect(sort(groceryrules, by = "lift")[1:5])
```

```
## lhs rhs support confidence lift count
## [1] {herbs} => {root vegetables} 0.007015760041 0.4312500000 3.956477379 69
## [2] {berries} => {whipped/sour cream} 0.009049313676 0.2721712538 3.796885505 89
## [3] {other vegetables,
## tropical fruit,
## whole milk} => {root vegetables} 0.007015760041 0.4107142857 3.768073694 69
## [4] {beef,
## other vegetables} => {root vegetables} 0.007930859176 0.4020618557 3.688692491 78
## [5] {other vegetables,
## tropical fruit} => {pip fruit} 0.009456024403 0.2634560907 3.482648725 93
```

Taking subsets of association rules

```
# Find any rules with berries appearing in the rule
berryrules <- subset(groceryrules, items %in% "berries")
inspect(berryrules)
```

```
## lhs rhs support confidence
## [1] {berries} => {whipped/sour cream} 0.009049313676 0.2721712538
## [2] {berries} => {yogurt} 0.010574478902 0.3180428135
## [3] {berries} => {other vegetables} 0.010269445857 0.3088685015
## [4] {berries} => {whole milk} 0.011794611083 0.3547400612
## lift count
## [1] 3.796885505 89
```

```
## [2] 2.279847719 104
## [3] 1.596280459 101
## [4] 1.388328095 116
```

Saving association rules to a file or data frame

```
# Save the rules to a CSV file
write(groceryrules, file = "groceryrules.csv",
sep = ",", quote = TRUE, row.names = FALSE)
# Convert the rules into an R data frame
groceryrules_df <- as(groceryrules, "data.frame")
str(groceryrules_df)
```

```
## 'data.frame':   463 obs. of  5 variables:
## $ rules      : Factor w/ 463 levels "{baking powder} => {other vegetables}",...: 340 302 207 206 208 ...
## $ support    : num  0.00691 0.0061 0.00702 0.00773 0.00773 ...
## $ confidence: num  0.4 0.405 0.431 0.475 0.475 ...
## $ lift       : num  1.57 1.59 3.96 2.45 1.86 ...
## $ count      : num  68 60 69 76 76 69 70 67 63 88 ...
```