Practice Problems 7

Jordan Lian

3/28/2021

```r
library(tidyverse)
```

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.

1. Collecting Data

http://archive.ics.uci.edu/ml

2. Exploring and preparing the data

```r
# Load in data set
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 0 ...
##  $ water       : num  162 162 228 228 192 228 228 228 228 228 ...
##  $ superplastic: num  2.5 2.5 0 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 ...
```

```r
# Normalize function
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Normalize data
concrete_norm <- as.data.frame(lapply(concrete, normalize))

# Confirm that normalization worked
summary(concrete_norm$strength)
```

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

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

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

```r
# Split into training and test
concrete_train <- concrete_norm[1:773, ]
concrete_test <- concrete_norm[774:1030, ]
```

3. Training a model on the data

```r
# Train the simplest multiplayer feedforward network
library(neuralnet)
```

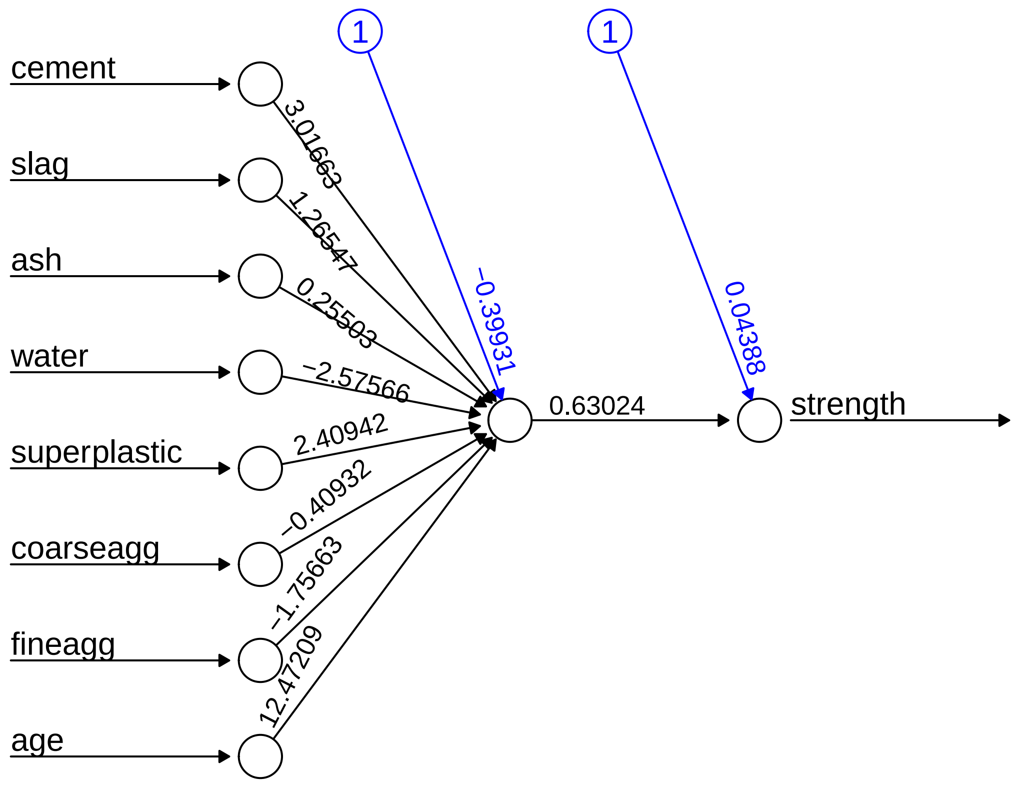
```
## Warning: package 'neuralnet' was built under R version 4.0.4
```

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

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

```r
concrete_model <- neuralnet(strength ~ cement + slag + ash + water + superplastic +
                              coarseagg + fineagg + age, data = concrete_train)

# Visualize network topology
plot(concrete_model, rep="best")
```



Error: 5.669069 Steps: 1471

In this simple model, there is one input node for each of the eight features, followed by a single hidden node and a single output node that predicts the concrete strength. The weights for each of the connections are also depicted, as are the bias terms (indicated by the nodes labeled with the number 1). The bias terms are numeric constants that allow the value at the indicated nodes to be shifted upward or downward, much like the intercept in a linear equation.

At the bottom of the figure, R reports the number of training steps and an error measure called the Sum of Squared Errors (SSE), which as you might expect, is the sum of the squared predicted minus actual values. A lower SSE implies better predictive performance. This is helpful for estimating the model's performance on the training data, but tells us little about how it will perform on unseen data.

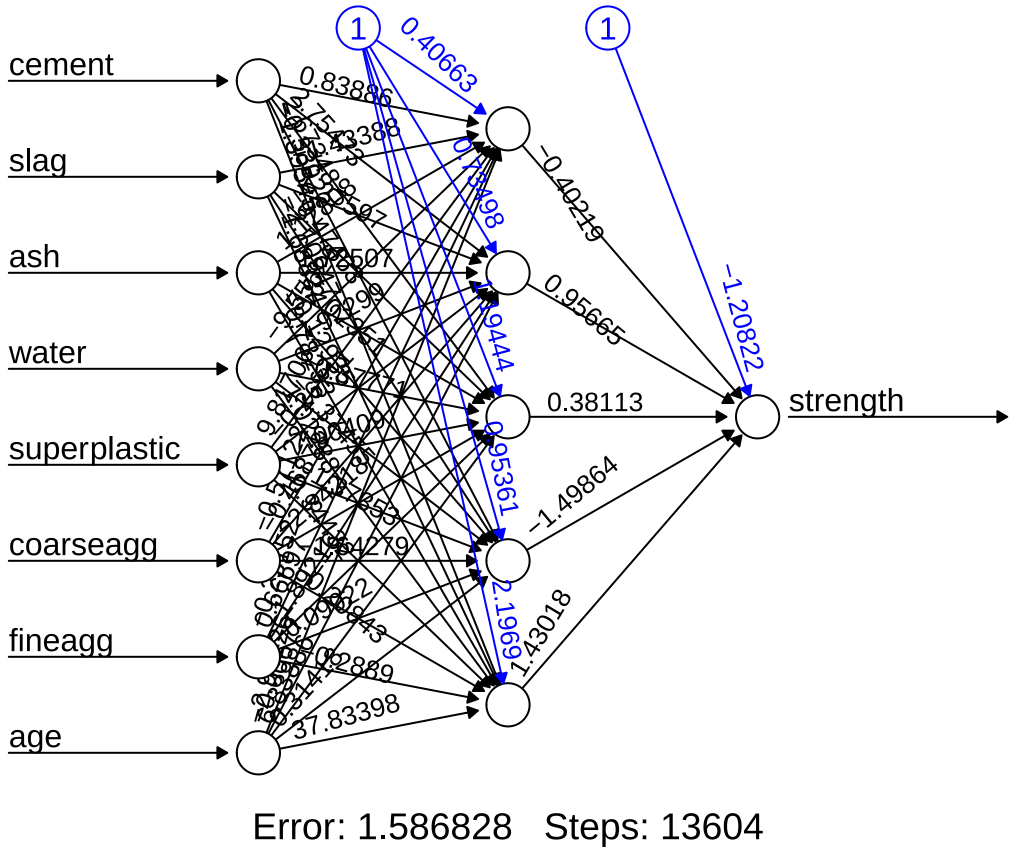
4. Evaluating model performance

```r
model_results <- compute(concrete_model, concrete_test[1:8])
predicted_strength <- model_results$net.result
cor(predicted_strength, concrete_test$strength)
```

```
##           [,1]
## [1,] 0.7184887
```

5. Improving model performance

```r
# Change hidden value
concrete_model2 <- neuralnet(strength ~ cement + slag + ash + water + superplastic +
                              coarseagg + fineagg + age, data = concrete_train, hidden = 5)
plot(concrete_model2, rep="best")
```



```r
model_results2 <- compute(concrete_model2, concrete_test[1:8])
predicted_strength2 <- model_results2$net.result
cor(predicted_strength2, concrete_test$strength)
```

```
##           [,1]
## [1,] 0.797479
```

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.

1. Collecting Data

http://archive.ics.uci.edu/ml

2. Exploring and preparing the data

```
# Load in dataset
letters <- read.csv("letterdata.csv")
str(letters)
```

```
## 'data.frame':    20000 obs. of  17 variables:
##  $ letter: chr  "T" "I" "D" "N" ...
##  $ 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 ...
```

```
# Make certain columns factors
col_names <- names(letters)
letters[,col_names] <- lapply(letters[,col_names] , factor)

# Split into training/test datasets
letters_train <- letters[1:16000, ]
letters_test  <- letters[16001:20000, ]
```

3. Training a model on the data

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.3
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 4.0.3
```

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

```
## The following object is masked from 'package:purrr':
##
##     cross
```

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

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.3
```

```r
library(klaR)
```

```
## Warning: package 'klaR' was built under R version 4.0.3
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

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

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

```
##  Setting default kernel parameters
```

```r
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 : 8802
##
## Objective Function Value : -8.7027 -7.0448 -12.0957 -7.8373 -6.9396 -15.8701 -14.8544 -8.45 -13.5164
## Training error : 0.02475
```

In this simple model, there is one input node for each of the eight features, followed by a single hidden node and a single output node that predicts the concrete strength. The weights for each of the connections are also depicted, as are the bias terms (indicated by the nodes labeled with the number 1). The bias terms are numeric constants that allow the value at the indicated nodes to be shifted upward or downward, much like the intercept in a linear equation.

At the bottom of the figure, R reports the number of training steps and an error measure called the Sum of Squared Errors (SSE), which as you might expect, is the sum of the squared predicted minus actual values. A lower SSE implies better predictive performance. This is helpful for estimating the model's performance on the training data, but tells us little about how it will perform on unseen data.

4. Evaluating model performance

```r
# Predictions
letter_predictions <- predict(letter_classifier, letters_test)
head(letter_predictions)

# Table
table(letter_predictions, letters_test$letter)

# Table and proportion
agreement <- letter_predictions == letters_test$letter
table(agreement)
prop.table(table(agreement))
```

5. Improving model performance

```r
# Change kernel
letter_classifier_rbf <- ksvm(letter ~ ., data = letters_train, kernel = "rbfdot")
letter_predictions_rbf <- predict(letter_classifier_rbf, letters_test)

# Get table and percentages
agreement_rbf <- letter_predictions_rbf == letters_test$letter
table(agreement_rbf)
```

```
## agreement_rbf
## FALSE  TRUE
##   460  3540
```

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

```
## agreement_rbf
## FALSE  TRUE
## 0.115 0.885
```

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.

1. Collecting Data

http://archive.ics.uci.edu/ml

2. Exploring and preparing the data

Data Preparation - creating a sparse matrix for transaction data

```
# Load in dataset
library(arules)
```

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

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## 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
```

```
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 0.02609146
##
## 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   16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   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
```

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

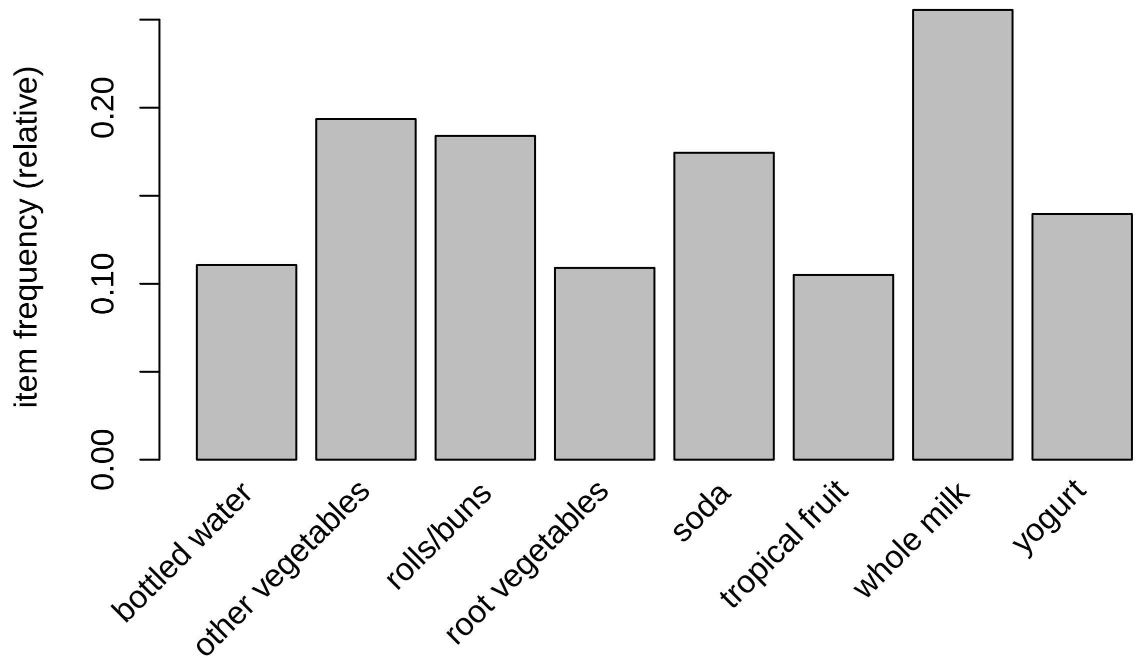
```
##     items
## [1] {citrus fruit,
##      margarine,
##      ready soups,
##      semi-finished bread}
## [2] {coffee,
##      tropical fruit,
##      yogurt}
## [3] {whole milk}
## [4] {cream cheese,
##      meat spreads,
##      pip fruit,
##      yogurt}
## [5] {condensed milk,
##      long life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

```
itemFrequency(groceries[, 1:3])
```

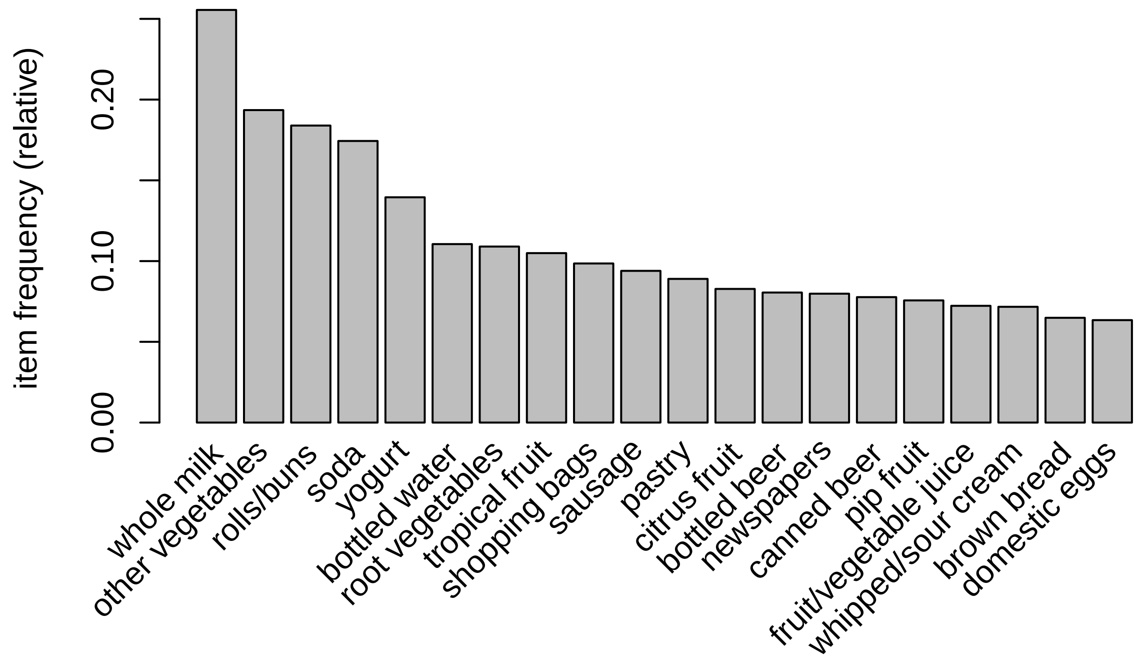
```
## abrasive cleaner artif. sweetener   baby cosmetics
##     0.0035587189     0.0032536858     0.0006100661
```

Visualizing item support - item frequency plots

```
itemFrequencyPlot(groceries, support = 0.1)
```

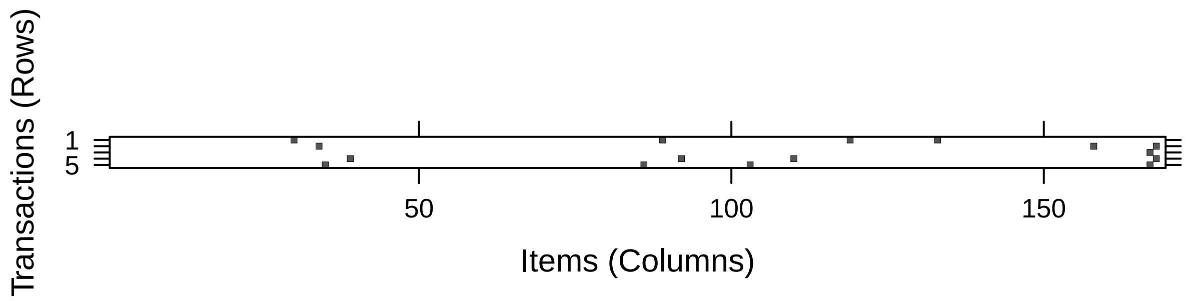


```r
itemFrequencyPlot(groceries, topN = 20)
```

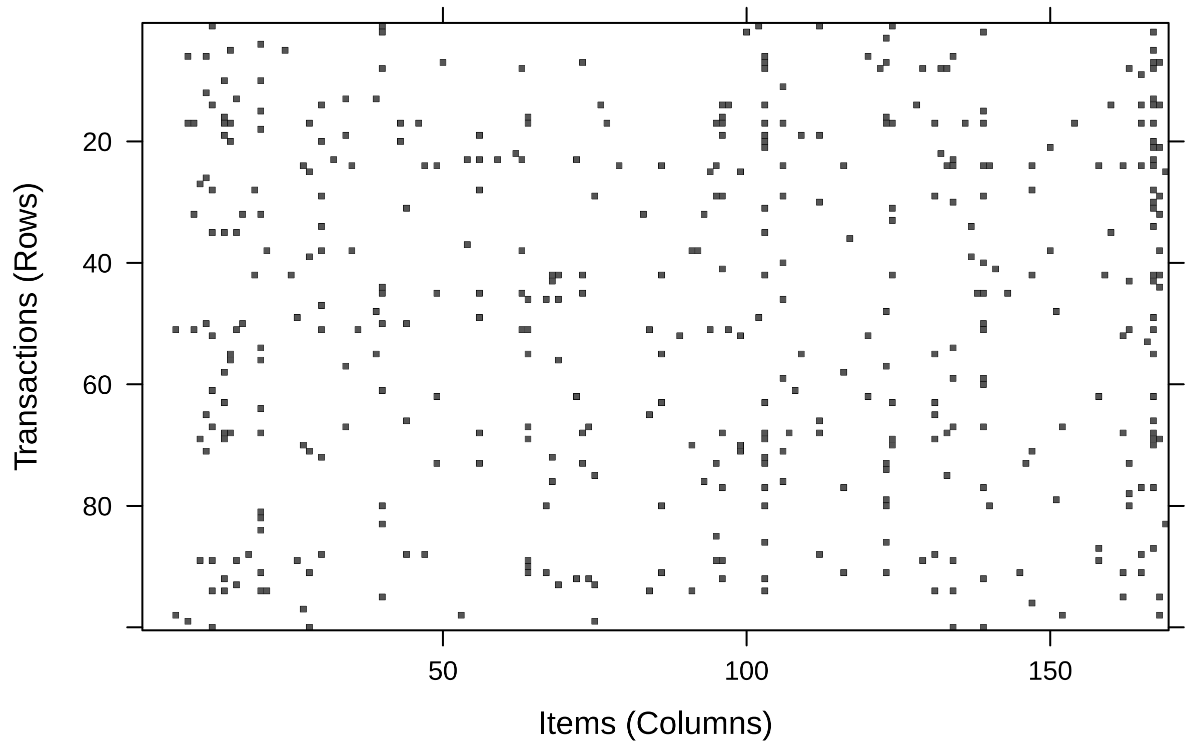


Visualizing the transaction data - plotting the sparse matrix

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



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



3. Training a model on the data

```r
# Use apriori()
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 TRUE
##
## 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
```

```r
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 TRUE
##
## 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 [0.01s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.01s].
## writing ... [463 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
groceryrules
```

```
## set of 463 rules
```

The Sum of Squared Errors (SSE) is the sum of the squared predicted minus actual values. A lower SSE implies better predictive performance. This is helpful for estimating the model's performance on the training data, but it doesn't tell how it will perform on unseen data.

4. Evaluating model performance

```r
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.
##   2.000   2.000   3.000   2.711   3.000   4.000
##
```

```
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.006101   Min.   :0.2500   Min.   :0.009964   Min.   :0.9932
##  1st Qu.:0.007117   1st Qu.:0.2971   1st Qu.:0.018709   1st Qu.:1.6229
##  Median :0.008744   Median :0.3554   Median :0.024809   Median :1.9332
##  Mean   :0.011539   Mean   :0.3786   Mean   :0.032608   Mean   :2.0351
##  3rd Qu.:0.012303   3rd Qu.:0.4495   3rd Qu.:0.035892   3rd Qu.:2.3565
##  Max.   :0.074835   Max.   :0.6600   Max.   :0.255516   Max.   :3.9565
##      count
##  Min.   : 60.0
##  1st Qu.: 70.0
##  Median : 86.0
##  Mean   :113.5
##  3rd Qu.:121.0
##  Max.   :736.0
##
## mining info:
##       data ntransactions support confidence
##  groceries          9835   0.006       0.25
```

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

```
##     lhs             rhs                support     confidence coverage
## [1] {pot plants} => {whole milk}       0.006914082 0.4000000  0.01728521
## [2] {pasta}      => {whole milk}       0.006100661 0.4054054  0.01504830
## [3] {herbs}      => {root vegetables} 0.007015760 0.4312500  0.01626843
##     lift     count
## [1] 1.565460 68
## [2] 1.586614 60
## [3] 3.956477 69
```

The lift of a rule measures how much more likely one item or item set is purchased relative to its typical rate of purchase, given that you know another item or item set has been purchased. This is defined by the following equation:

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}$$

Unlike confidence where the item order matters, $lift(X \rightarrow Y)$ is the same as $lift(Y \rightarrow X)$.

5. Improving model performance

Sorting the set of association rules

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

```
##     lhs                     rhs                      support confidence   coverage     lift count
## [1] {herbs}              => {root vegetables}    0.007015760   0.4312500 0.01626843 3.956477    69
## [2] {berries}            => {whipped/sour cream} 0.009049314   0.2721713 0.03324860 3.796886    89
## [3] {other vegetables,
##      tropical fruit,
##      whole milk}       => {root vegetables}    0.007015760  0.4107143 0.01708185 3.768074    69
## [4] {beef,
##      other vegetables} => {root vegetables}    0.007930859  0.4020619 0.01972547 3.688692    78
## [5] {other vegetables,
##      tropical fruit}   => {pip fruit}          0.009456024  0.2634561 0.03589222 3.482649    93
```

Taking subsets of association rules

```
berryrules <- subset(groceryrules, items %in% "berries")
inspect(berryrules)
```

```
##     lhs          rhs                  support     confidence coverage  lift
## [1] {berries} => {whipped/sour cream} 0.009049314 0.2721713  0.0332486 3.796886
## [2] {berries} => {yogurt}             0.010574479 0.3180428  0.0332486 2.279848
## [3] {berries} => {other vegetables}   0.010269446 0.3088685  0.0332486 1.596280
## [4] {berries} => {whole milk}         0.011794611 0.3547401  0.0332486 1.388328
##     count
## [1]  89
## [2] 104
## [3] 101
## [4] 116
```

Saving association rules to a file or data frame

```
write(groceryrules, file = "groceryrules.csv", sep = ",", quote = TRUE, row.names = FALSE)
groceryrules_df <- as(groceryrules, "data.frame")
str(groceryrules_df)
```

```
## 'data.frame':    463 obs. of  6 variables:
##  $ rules     : chr  "{pot plants} => {whole milk}" "{pasta} => {whole milk}" "{herbs} => {root veget
##  $ support   : num  0.00691 0.0061 0.00702 0.00773 0.00773 ...
##  $ confidence: num  0.4 0.405 0.431 0.475 0.475 ...
##  $ coverage  : num  0.0173 0.015 0.0163 0.0163 0.0163 ...
##  $ lift      : num  1.57 1.59 3.96 2.45 1.86 ...
##  $ count     : int  68 60 69 76 76 69 70 67 63 88 ...
```