PeerReview_7

Minxin Cheng

```
#install.packages("kernlab")
#install.packages("neuralnet")
#install.packages("arules")
library(readxl)
library(kernlab)
library(neuralnet)
library(arules)
```

Problem 1

1. Read data file

```
concrete <- read_excel("Concrete_Data.xls")
```

2. Change column names

```
# change column names
colnames(concrete) <- c("cement", "slag", "ash", "water", "superplastic",
                        "coarseagg", "fineagg", "age", "strength")
# check data structure
str(concrete)
```

```
## tibble [1,030 x 9] (S3: tbl_df/tbl/data.frame)
##  $ cement      : num [1:1030] 540 540 332 332 199 ...
##  $ slag        : num [1:1030] 0 0 142 142 132 ...
##  $ ash         : num [1:1030] 0 0 0 0 0 0 0 0 0 0 ...
##  $ water       : num [1:1030] 162 162 228 228 192 228 228 228 228 228 ...
##  $ superplastic: num [1:1030] 2.5 2.5 0 0 0 0 0 0 0 0 ...
##  $ coarseagg   : num [1:1030] 1040 1055 932 932 978 ...
##  $ fineagg     : num [1:1030] 676 676 594 594 826 ...
##  $ age         : num [1:1030] 28 28 270 365 360 90 365 28 28 28 ...
##  $ strength    : num [1:1030] 80 61.9 40.3 41.1 44.3 ...
```

3. Normalize the data

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.2663  0.4000  0.4172  0.5457  1.0000
```

```r
# compare with the original data to make sure the data was normalized
summary(concrete$strength)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.332  23.707  34.443  35.818  46.136  82.599
```

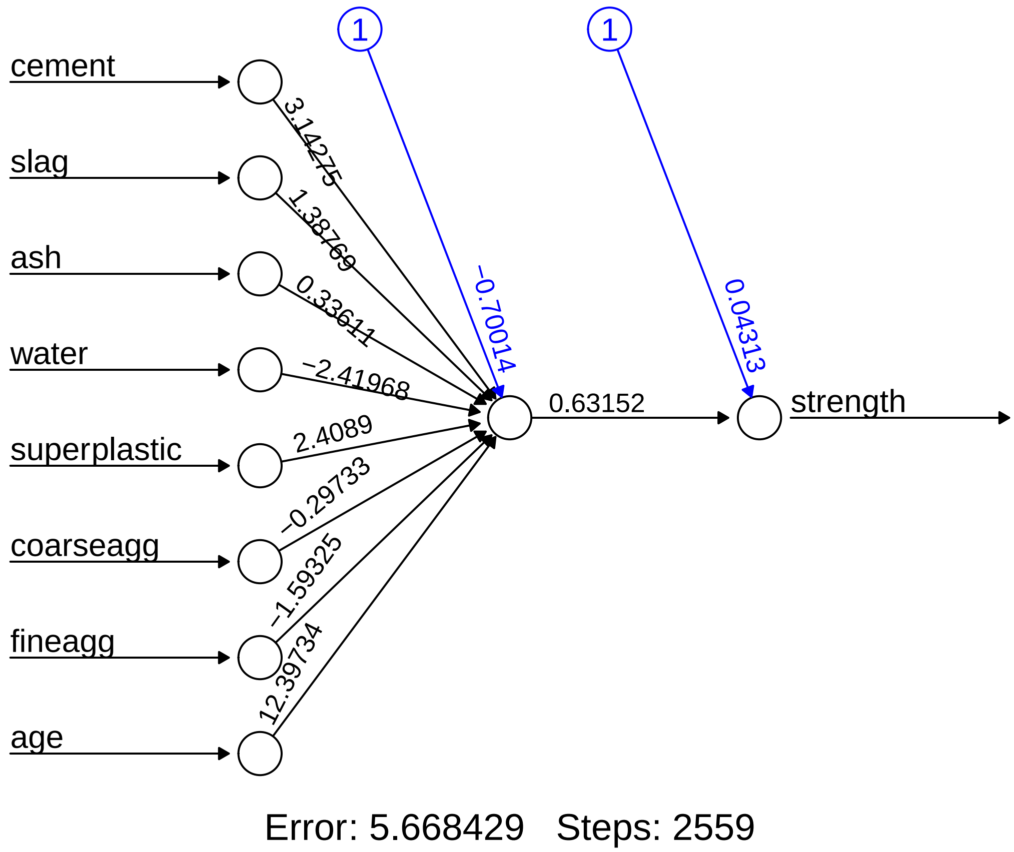
4. Split the data set

Since the data has already been randomized, we directly split it by row numbers

```r
concrete_train <- concrete_norm[1:773, ]
concrete_test <- concrete_norm[774:1030, ]
```

5. Train a model

```r
# train a model using training data
set.seed(123)
concrete_model <- neuralnet(strength ~ cement + slag + ash + water +
                              superplastic + coarseagg + fineagg + age,
                            data = concrete_train)
# visualize the model
plot(concrete_model, rep = "best")
```



This plot illustrated the eight features as the input and the weights of each connections. The blue line indicated the bias terms, which are numeric constants that allow the value at the indicated nodes to be shifted upward or downward. There is also a Sum of Squared Errors (SSE) at the bottom gives a general idea of the performance of the data.

6. Make prediction

The compute() function is slightly different from predict() function in the way that it returns a list with two components: 1)neurons, which stores the neurons for each layer in the network, and 2)net.result, which stores the predicted values

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

```
##          [,1]
## 774 0.3895896
## 775 0.2449998
## 776 0.2500528
## 777 0.2271870
## 778 0.3316256
## 779 0.1822438
```

7. Evaluate the model performance

Given this is a numeric prediction problem rather than a classification, we can't use confusion matrix to examine model accuracy, will use the cor() function to get a correlation between variables

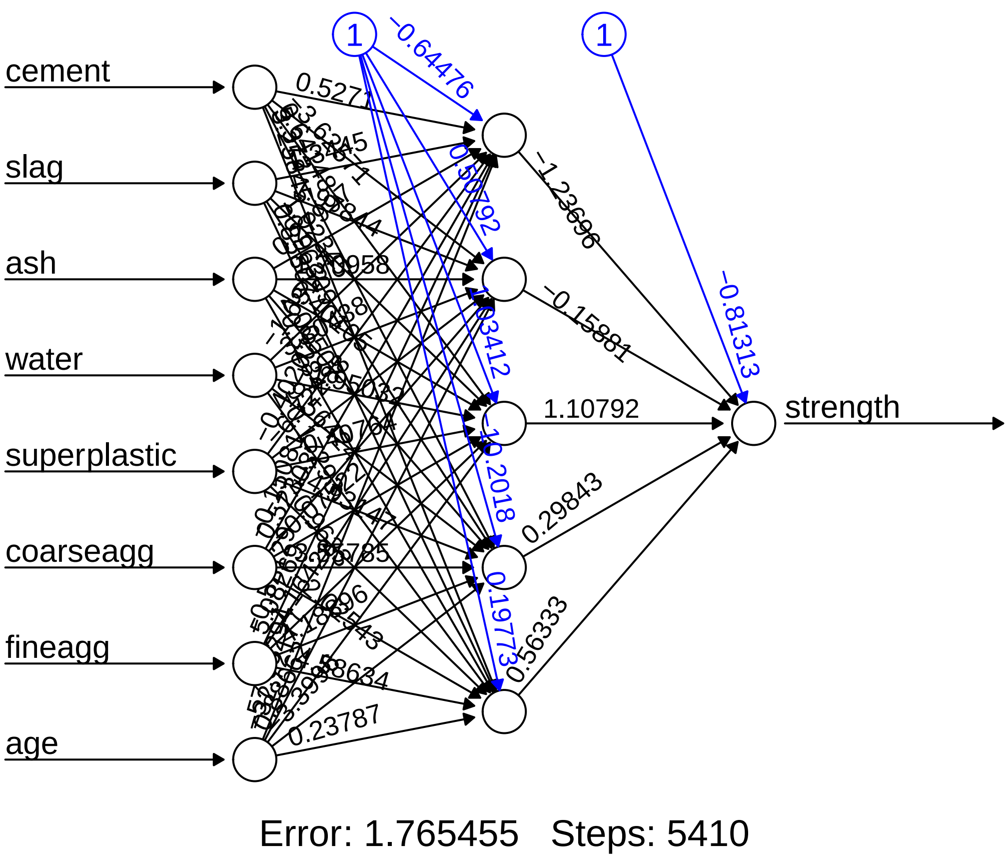
```
cor(predicted_strength, concrete_test$strength)
```

```
##          [,1]
## [1,] 0.72136
```

The correlation is 0.72, which is a fairly strong linear relationship. Next we can try to improve the performance by using more hidden nodes.

8. Improve model performance

```
# train another model using more hidden nodes
set.seed(123)
concrete_model2 <- neuralnet(strength ~ cement + slag + ash + water +
                              superplastic + coarseagg + fineagg +age,
                            data = concrete_train, hidden = 5)
# plot the model
plot(concrete_model2, rep = "best")
```



From the figure above, the SSE has been reduced from 5.67 to 1.77 and the number of training steps increased from 2559 to 5410. A more complex networks will take more iterations to find the optimal weights.

9. Make prediction using the improved model

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

10. Evaluate the improved model

```
cor(predicted_strength2, concrete_test$strength)
```

```
##           [,1]
## [1,] 0.7961801
```

The correlation has been improved from 0.72 to 0.80.

Problem 2

1. Read in data file

```
letters <- read.csv("letter-recognition.data")
```

2. Change column names and check the data set

```
# change column names
colnames(letters) <- c("letter", "xbox", "ybox", "width", "height", "onpix",
                       "xbar", "ybar", "x2bar", "y2bar", "xybar", "x2ybar",
                       "xy2bar", "xedge", "xedgey", "yedge", "yedgex")
# check data structure
str(letters)
```

```
## 'data.frame':    19999 obs. of  17 variables:
##  $ letter: chr  "I" "D" "N" "G" ...
##  $ xbox  : int  5 4 7 2 4 4 1 2 11 3 ...
##  $ ybox  : int  12 11 11 1 11 2 1 2 15 9 ...
##  $ width : int  3 6 6 3 5 5 3 4 13 5 ...
##  $ height: int  7 8 6 1 8 4 2 4 9 7 ...
##  $ onpix : int  2 6 3 1 3 4 1 2 7 4 ...
##  $ xbar  : int  10 10 5 8 8 8 8 10 13 8 ...
##  $ ybar  : int  5 6 9 6 8 7 2 6 2 7 ...
##  $ x2bar : int  5 2 4 6 6 6 2 2 6 3 ...
##  $ y2bar : int  4 6 6 6 9 6 2 6 2 8 ...
##  $ xybar : int  13 10 4 6 5 7 8 12 12 5 ...
##  $ x2ybar: int  3 3 4 5 6 6 2 4 1 6 ...
##  $ xy2bar: int  9 7 10 9 6 6 8 8 9 8 ...
##  $ xedge : int  2 3 6 1 0 2 1 1 8 2 ...
##  $ xedgey: int  8 7 10 7 8 8 6 6 1 8 ...
##  $ yedge : int  4 3 2 5 9 7 2 1 1 6 ...
##  $ yedgex: int  10 9 8 10 7 10 7 7 8 7 ...
```

```r
# check column types
sapply(letters, class)
```

```
##      letter        xbox        ybox       width      height       onpix
## "character"   "integer"   "integer"   "integer"   "integer"   "integer"
##        xbar        ybar       x2bar       y2bar       xybar      x2ybar
##   "integer"   "integer"   "integer"   "integer"   "integer"   "integer"
##      xy2bar       xedge      xedgey       yedge      yedgex
##   "integer"   "integer"   "integer"   "integer"   "integer"
```

```r
# change the letter column from chacater to factor
letters$letter <- as.factor(letters$letter)
```

3. Split the data set to training and testing

data has already been randomized and the r package will recale the data automatically

```r
letters_train <- letters[1:16000, ]
letters_test <- letters[16001:19999, ]
```

4. Train a model

```r
set.seed(123)
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 : 7039
##
## Objective Function Value : -14.1747 -20.007 -23.5629 -6.2007 -7.5523 -32.7693 -49.9788 -18.1824 -62.
## Training error : 0.13025
```

5. Make predictions

```r
# make predictions
letter_predictions <- predict(letter_classifier, letters_test)
# check the first few predicted letters
head(letter_predictions)
```

```
## [1] N V X N H E
## 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
```

```r
# use table() function to compare the predicted letter with
# the true letter in the testing dataset
table(letter_predictions, letters_test$letter)
```

```
##
## letter_predictions   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
##                  A 144   0   0   0   0   0   0   0   0   1   0   0   1   2   2
##                  B   0 121   0   5   2   0   1   2   0   0   1   0   1   0   0
##                  C   0   0 120   0   4   0  10   2   2   0   1   3   0   0   2
##                  D   2   2   0 156   0   1   3  10   4   3   4   3   0   5   5
##                  E   0   0   5   0 127   3   1   1   0   0   3   4   0   0   0
##                  F   0   0   0   0   0 138   2   2   6   0   0   0   0   0   0
##                  G   1   1   2   1   9   2 123   2   0   0   1   2   1   0   1
##                  H   0   0   0   1   0   1   0 102   0   2   3   2   3   4  20
##                  I   0   1   0   0   0   1   0   0 141   8   0   0   0   0   0
##                  J   0   1   0   0   0   1   0   2   5 128   0   0   0   0   1
##                  K   1   1   9   0   0   0   2   5   0   0 118   0   0   2   0
##                  L   0   0   0   0   2   0   1   1   0   0   0 134   0   0   0
##                  M   0   0   1   1   0   0   1   1   0   0   0   0 135   4   0
##                  N   0   0   0   0   0   1   0   1   0   0   0   0   0 145   0
##                  O   1   0   2   1   0   0   1   2   0   1   0   0   0   1  99
##                  P   0   0   0   1   0   2   1   0   0   0   0   0   0   0   2
##                  Q   0   0   0   0   0   0   8   2   0   0   0   3   0   0   3
##                  R   0   7   0   0   1   0   3   8   0   0  13   0   0   1   1
##                  S   1   1   0   0   1   0   3   0   1   1   0   1   0   0   0
##                  T   0   0   0   0   3   2   0   0   0   0   1   0   0   0   0
##                  U   1   0   3   1   0   0   0   2   0   0   0   0   0   0   1
##                  V   0   0   0   0   0   1   3   4   0   0   0   0   1   2   1
##                  W   0   0   0   0   0   0   1   0   0   0   0   0   2   0   0
##                  X   0   1   0   0   2   0   0   1   3   0   1   5   0   0   1
##                  Y   3   0   0   0   0   0   0   1   0   0   0   0   0   0   0
##                  Z   2   0   0   0   1   0   0   0   3   4   0   0   0   0   0
##
## letter_predictions   P   Q   R   S   T   U   V   W   X   Y   Z
##                  A   0   5   0   1   1   1   0   1   0   0   1
##                  B   2   2   3   5   0   0   2   0   1   0   0
##                  C   0   0   0   0   0   0   0   0   0   0   0
##                  D   3   1   4   0   0   0   0   0   3   3   1
##                  E   0   2   0  10   0   0   0   0   2   0   3
##                  F  16   0   0   3   0   0   1   0   1   2   0
##                  G   2   8   2   4   3   0   0   0   1   0   0
##                  H   0   2   3   0   3   0   2   0   0   1   0
##                  I   1   0   0   3   0   0   0   0   5   1   1
##                  J   1   3   0   2   0   0   0   0   1   0   6
##                  K   1   0   7   0   1   3   0   0   5   0   0
##                  L   0   1   0   5   0   0   0   0   0   0   1
##                  M   0   0   0   0   0   3   0   8   0   0   0
##                  N   0   0   3   0   0   1   0   2   0   0   0
##                  O   3   3   0   0   0   3   0   0   0   0   0
##                  P 130   0   0   0   0   0   0   0   0   1   0
##                  Q   1 124   0   5   0   0   0   0   0   2   0
```

```
##                    R    1    0  138    0    1    0    1    0    0    0    0
##                    S    0   14    0  101    3    0    0    0    2    0   10
##                    T    0    0    0    3  133    1    0    0    0    2    2
##                    U    0    0    0    0    0  151    0    0    1    1    0
##                    V    0    3    1    0    0    0  126    1    0    4    0
##                    W    0    0    0    0    0    4    4  127    0    0    0
##                    X    0    0    0    1    0    0    0    0  137    1    1
##                    Y    7    0    0    0    3    0    0    0    0  127    0
##                    Z    0    0    0   18    3    0    0    0    0    0  132
```

From the table above, most of the predictions are correct. We can also see that the most common uncorrect prediction is H-O, Z-S, F-P, R-K, which make sense. Next we will summarize the predictions.

6. Evaluate model performance

```r
agreement <- letter_predictions == letters_test$letter
table(agreement)
```

```
## agreement
## FALSE  TRUE
##   642  3357
```

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

```
## agreement
##     FALSE      TRUE
## 0.1605401 0.8394599
```

From the counting above, the overall accuracy is 0.84.

7. Improve model performance

Improve the model by using a more complex kernel function to map the data into a higher dimensional space.

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

8. Evaluate the improved model

```r
agreement_rbf <- letter_predictions_rbf == letters_test$letter
table(agreement_rbf)
```

```
## agreement_rbf
## FALSE  TRUE
##   275  3724
```

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

```
## agreement_rbf
##      FALSE       TRUE
## 0.06876719 0.93123281
```

The overall accuracy improved from 0.84 to 0.93, which is pretty good.

Problem 3

1. Read in data file

```r
groceries <- read.transactions("Groceries.csv", header = FALSE, sep = ",")
```

```
## Warning in readLines(file, encoding = encoding): incomplete final line found on
## 'Groceries.csv'
```

2. Summary the data set

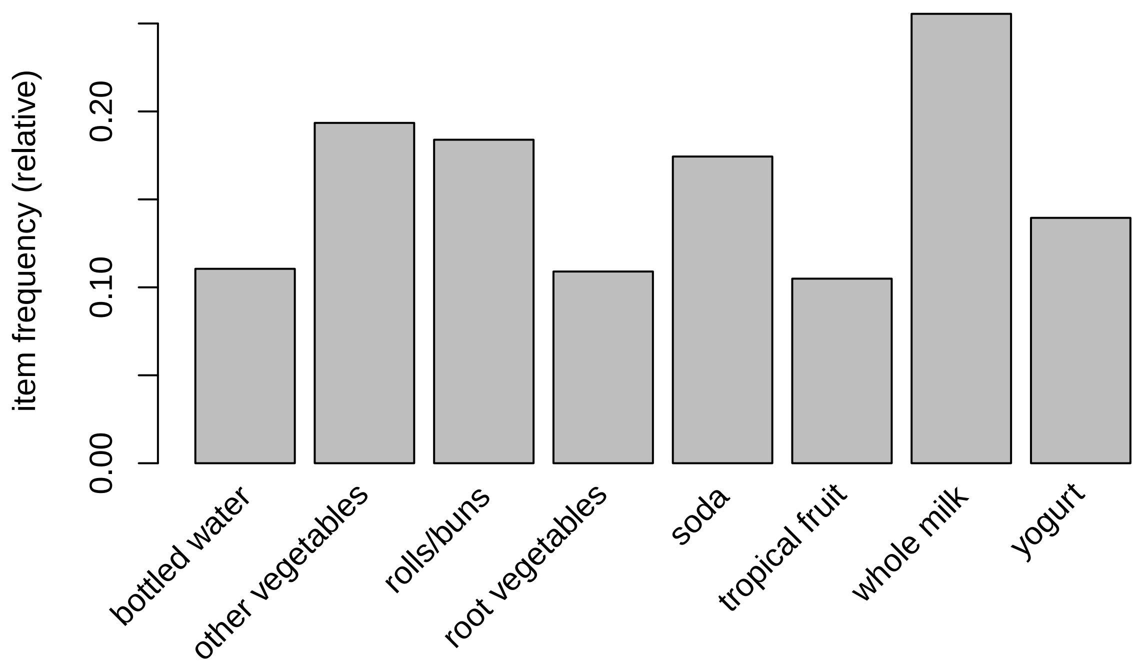
```r
summary(groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9836 rows (elements/itemsets/transactions) and
##  201 columns (items) and a density of 0.02195155
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34087
##
## 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    2
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.412   6.000  32.000
##
## includes extended item information - examples:
##             labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
```

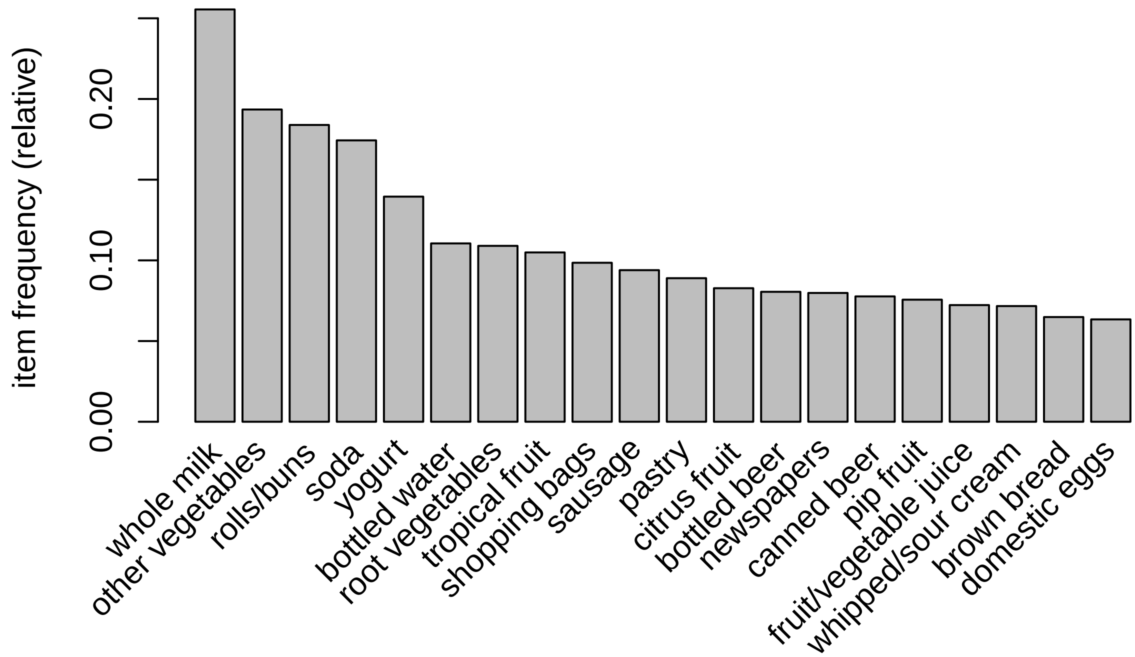
From the table, there are 9837 transactions and 233 different items. Each cell in the matrix is 1 if the item was purchased for the corresponding transaction, or else is 0. Density indicates the proportion of nonzero matrix cells is 0.023. There are 9838 x 233 = 2193874 positions in the matrix, therefore, there are 2193874 x 0.0232271 = 50957 items were purchased. On average, each transaction contains 50957 / 9838 = 5.18 items. The most frequent items was whole milk, vegetables, rolls/buns, and soda.

3. Visualize the data set

```r
# plot the items that with at least 10% support
itemFrequencyPlot(groceries, support = 0.1)
```




```r
# plot the top 20 items
itemFrequencyPlot(groceries, topN = 20)
```



4. Train a model

First set the confidence threshold of 0.25, meaning that in order to be included in the results, the rule has to be correct at least 25 percent of the time. This will eliminate the most unreliable rules. The minlen is set as 2 to eliminate rules that contain fewer than two items.

```
set.seed(123)
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 ...[201 item(s), 9836 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
```

The groceryrules contains a set of 463 association rules.

5. Evaluate model performance

```
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.   :0.006100   Min.   :0.2500   Min.   :0.009963   Min.   :0.9933
##  1st Qu.:0.007117   1st Qu.:0.2971   1st Qu.:0.018707   1st Qu.:1.6231
##  Median :0.008743   Median :0.3554   Median :0.024807   Median :1.9334
##  Mean   :0.011538   Mean   :0.3786   Mean   :0.032604   Mean   :2.0353
##  3rd Qu.:0.012302   3rd Qu.:0.4495   3rd Qu.:0.035889   3rd Qu.:2.3567
##  Max.   :0.074827   Max.   :0.6600   Max.   :0.255490   Max.   :3.9569
##      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          9836   0.006       0.25
```

Lift of a rule measures how much more likely one item or itemset is purchased relative to its typical rate of purchase. A large lift value is a stronger indicator that a rule is important, and reflects a true connection between the items. We can then look at specific rules. The first three rules in the groceryrules are:

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

```
##     lhs                 rhs                support     confidence coverage
## [1] {potted plants} => {whole milk}      0.006913379 0.4000000  0.01728345
## [2] {pasta}         => {whole milk}      0.006100041 0.4054054  0.01504677
## [3] {herbs}         => {root vegetables} 0.007015047 0.4312500  0.01626678
##     lift     count
## [1] 1.565619 68
## [2] 1.586776 60
## [3] 3.956880 69
```

Take the first row as an example, if a customer buys a potted plants, they will also likely to buy whole milk. The support and confidence for this rule is 0.007 and 0.4. This rule covers 0.017 percent of the transactions.

6. Improve model performance

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

```
##     lhs                   rhs                      support confidence   coverage     lift count
## [1] {herbs}            => {root vegetables}    0.007015047  0.4312500 0.01626678 3.956880    69
## [2] {berries}          => {whipped/sour cream} 0.009048394  0.2721713 0.03324522 3.797272    89
## [3] {other vegetables,
##      tropical fruit,
##      whole milk}       => {root vegetables}    0.007015047  0.4107143 0.01708011 3.768457    69
## [4] {beef,
##      other vegetables} => {root vegetables}    0.007930053  0.4020619 0.01972346 3.689068    78
## [5] {other vegetables,
##      tropical fruit}   => {pip fruit}          0.009455063  0.2634561 0.03588857 3.483003    93
```

Use berry as an example, if we want to know all the associations about berry

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

```
##     lhs          rhs                  support     confidence coverage
## [1] {berries} => {whipped/sour cream} 0.009048394 0.2721713  0.03324522
## [2] {berries} => {yogurt}             0.010573404 0.3180428  0.03324522
## [3] {berries} => {other vegetables}   0.010268402 0.3088685  0.03324522
## [4] {berries} => {whole milk}         0.011793412 0.3547401  0.03324522
##     lift     count
## [1] 3.797272  89
## [2] 2.280080 104
## [3] 1.596443 101
## [4] 1.388469 116
```

7. Save association rules to a data frame

```r
groceryrules_df <- as(groceryrules, "data.frame")
str(groceryrules_df)
```

```
## 'data.frame':    463 obs. of  6 variables:
##  $ rules     : chr  "{potted plants} => {whole milk}" "{pasta} => {whole milk}" "{herbs} => {root veg
##  $ 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 2.46 1.86 ...
##  $ count     : int  68 60 69 76 76 69 70 67 63 88 ...
```