Holsenbeck_S_7

Stephen Synchronicity

2018-03-27

```r
# This code will extract the assignment HTML and print the output formatted for
# this Rmd document. Set Assignment html below Use if assignment has blue font
# headers, and lists of questions
library(rvest)
Q <- xml2::read_html("https://da5030.weebly.com/assignment-7.html") %>% rvest::html_nodes(xpath = "//fo
    rvest::html_children()
Qs <- vector("list", sum(stringr::str_detect(Q, "Problem")))
for (i in seq_along(Q)) {
    if (Q[i] %>% html_text() %>% stringr::str_detect("Problem")) {
        n <- Q[i] %>% html_text() %>% stringr::str_extract("(?<=Problem\\s)\\d") %>%
            as.numeric
        Qs[[n]][["h1"]] <- paste("#", rvest::html_text(Q[i]), "\n")
        print(Qs[[n]][["h1"]])
        next
    } else if (rvest::html_attrs(Q[i]) %>% grepl("paragraph", ., ignore.case = T) &
        html_children(Q[i]) %>% html_attrs() %>% grepl("rgb\\(85", ., ignore.case = T)) {
        Qs[[n]][["q"]] <- paste("<div class='q'>", html_text(Q[i]), "</div>\n```{r  '",
            n, "'}\n```\n<p class='a'></p>\n\n", sep = "")
    } else {
        next
    }

    if (n == length(Qs)) {
        break
    }
}
# grep(pattern=substr(html_text(n),1,20),x=xml_parent(n),ignore.case = T) Qtext
# <- xml2::read_html('https://da5030.weebly.com/assignment-7.html') %>%
# rvest::html_nodes(xpath='//font[contains(@color,'#24678d')]/ancestor::div[1]/following-sibling::div[c
# Q.form <- vector('list',length(Q)) for (i in seq_along(Q)) { Q.form[[i]] <-
# list(title=NA,Qs=NA) Q.form[[i]][['title']] <- Q[i] %>%
# rvest::html_node(css='font') %>% rvest::html_text() %>% paste('#
# ',.,'\n',sep='') if(length(Qtext)>0){ li <- xml2::xml_contents(Qtext[[i]]) %>%
# xml2::xml_children() %>% rvest::html_text() for (l in seq_along(li)) {
# Q.form[[i]][['Qs']][l] <- paste('## ',i,letters[l],'\n<div
# class='q'>',li[l],'\n</div>\n```{r '',i,letters[l],''}\n```\n<p
# class='a'>\n</p>',sep='') } }else { } }
lapply(Qs, FUN = "cat", sep = "\n")
detach("package:rvest")
```

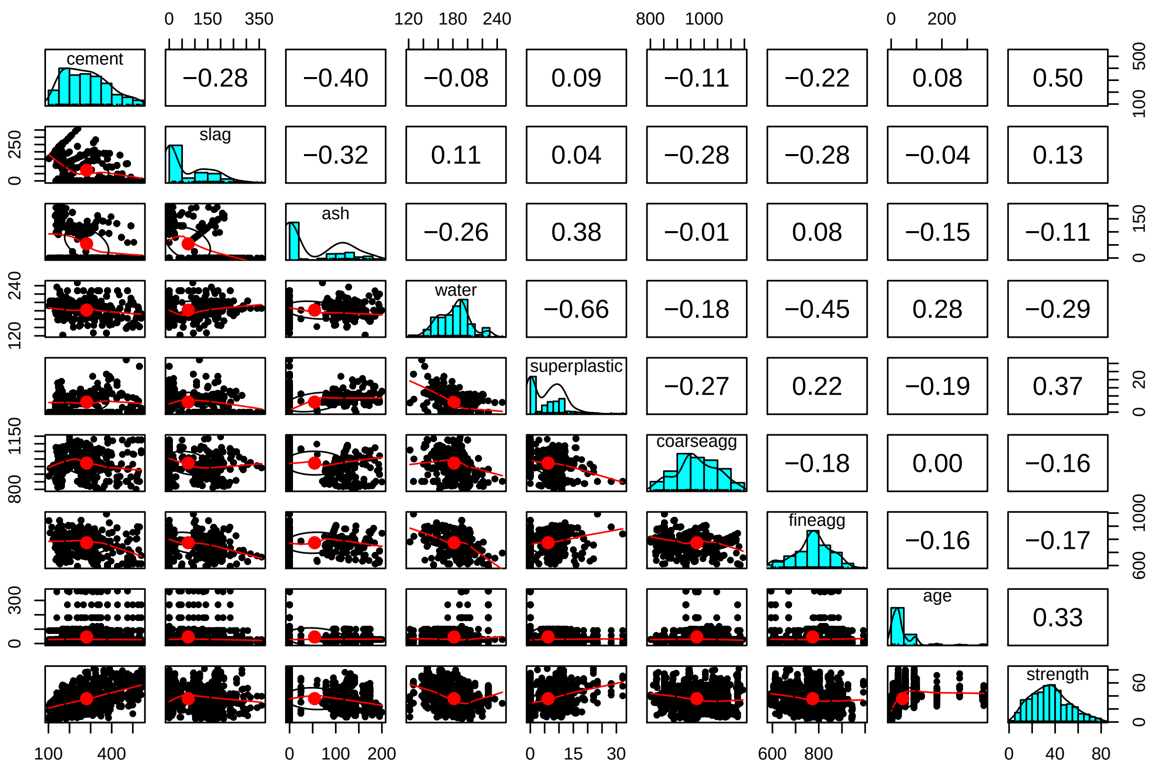
Problem 1

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

```r
# ------------------------ Sat Mar 24 16:02:47 2018 ------------------------# Load
# Data
cs <- read.csv("concrete.csv")
str(cs)
```

```
## 'data.frame':    1030 obs. of  9 variables:
##  $ cement      : num  141 169 250 266 155 ...
##  $ slag        : num  212 42.2 0 114 183.4 ...
##  $ ash         : num  0 124.3 95.7 0 0 ...
##  $ water       : num  204 158 187 228 193 ...
##  $ superplastic: num  0 10.8 5.5 0 9.1 0 0 6.4 0 9 ...
##  $ coarseagg   : num  972 1081 957 932 1047 ...
##  $ fineagg     : num  748 796 861 670 697 ...
##  $ age         : int  28 14 28 28 28 90 7 56 28 28 ...
##  $ strength    : num  29.9 23.5 29.2 45.9 18.3 ...
```

```r
# Evaluate normality
psych::pairs.panels(cs)
```




```r
# About 5 look normal, while 4 look skew ------------------------ Sat Mar 24
# 16:20:23 2018 ------------------------# min max normalization function will
# work better because 4/9 features are not normally distributed.
normalize <- function(x) {
    return((x - min(x))/(max(x) - min(x)))
}
# Apply min-max normalizations to all columns.
cs.norm <- as.data.frame(lapply(cs, normalize))
```

```r
# Verify that it worked.
cs$strength %>% summary
```

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

```r
cs.norm$strength %>% summary
```

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

```r
# ----------------------- Sat Mar 24 16:30:37 2018 ------------------------# Test
# and training data sets
cs.train <- cs.norm[1:773, ]
cs.test <- cs.norm[774:1030, ]
# install.packages('neuralnet') ----------------------- Sat Mar 24 17:36:38 2018
# ------------------------# Train a net with the default 1 neuron in the hidden
# layer

library(neuralnet)
cs.model <- neuralnet(formula = strength ~ cement + slag + ash + water + superplastic +
    coarseagg + fineagg + age, data = cs.train)
# Visualize the outcome
plot(cs.model)
# Age>cement>slag. SSE:~5.08 Make a prediction
cs.pred <- compute(cs.model, cs.test[-9])
# Evaluate the Pearson correlation of the prediction with the actual
cor(cs.pred$net.result, cs.test$strength)
```

```
##              [,1]
## [1,] 0.8063811202
```

```r
# ----------------------- Sat Mar 24 17:46:25 2018 ------------------------#
# Change the number of layers in an iterative fashion, determine optimal number
# of neurons in hidden layer
n <- 1:10
tune <- vector()
for (i in n) {
    cs.model <- neuralnet(formula = strength ~ cement + slag + ash + water + superplastic +
        coarseagg + fineagg + age, data = cs.train, hidden = i)
    cs.pred <- compute(cs.model, cs.test[-9])
    tune[i] <- cor(cs.pred$net.result, cs.test$strength)
}
# plot the relationship of neurons in hidden layer to the accuracy of the
# prediction
plot(x = 1:length(tune), y = tune, type = "b", main = "Number of neurons in hidden layer v Prediction Ac
text(1:length(tune), tune, labels = round(tune, 3), adj = c(1, 0))
# Equivalent performance at 5 & 8, with 10 showing about ~1.5% improvement Top
# performancing model
c(which.max(tune), tune[which.max(tune)])
```

```
## [1] 8.0000000000 0.9425725883
```

```r
plot(cs.model)
# Yikes It looks like there are various algorithms with which computations can be
# made. We will try two of the other algorithms here and see how they perform
# runs <- expand.grid(algo=c('sag', 'slr'), n=c(8,10)) -----------------------
```

```r
# Sat Mar 24 18:11:56 2018 ------------------------# To improve the speed at
# which it executes we can just compare errors mods <- vector('list',nrow(runs))
# for (i in 1:nrow(runs)) { mods[[i]] <- neuralnet(formula = strength ~ cement +
# slag + ash + water + superplastic + coarseagg + fineagg + age, data = cs.train,
# hidden = runs[i,'n'], algorithm = runs[i,'algo'], threshold= .01) tune[i] <-
# mods[[i]]$result.matrix['error',] } tune <- lapply(mods,function(x){ cs.pred <-
# compute(x,cs.test[-9]) cor(cs.pred$net.result,cs.test$strength) })
# ----------------------- Sat Mar 24 18:42:49 2018 ------------------------# Only
# the first model seemed to work properly and provide an error.
```

The weights in this implementation provide the user with a better understanding of how much correlation each attribute has with the response variable. I am interested to know how one might be able to customize the function in each neuron of the a hidden layer. To use stock indicators as an example, one neuron might be an RSI (relative strength indicator), one an SMA (simple moving average), another would be an DM+ - DM- (directional momentum up or down compared to one another), and lastly a parabolic SAR (stop and reverse). I suppose one could compute each value as a column in an extended timeseries object, convert that to a dataframe, and then run the neural net on the df.

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.

```r
# ----------------------- Sat Mar 24 20:35:45 2018 ------------------------# Read
# 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 ...
```

```r
# divide into training and test data
letters.train <- letters[1:16000, ]
letters.test <- letters[16001:20000, ]
# ----------------------- Sat Mar 24 20:42:34 2018 -----------------------#
# Create a model
library(kernlab)
svm.linear.time <- system.time({
```

```r
    letter.classifier <- ksvm(letter ~ ., data = letters.train, kernel = "vanilladot")
    # The results of printing the object directly are quite verbose, so we will print
    # the error here, which we can hopefully use to calibrate future models.
    letter.classifier@error
    cM <- caret::confusionMatrix(predict(letter.classifier, letters.test), letters.test$letter)
})
```

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

```r
cM$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   0.8392500000   0.8327971944   0.8274954781   0.8505066430   0.0420000000
## AccuracyPValue  McnemarPValue
##   0.0000000000            NaN
```

```r
# The prediction indicates an accuracy of ~84% ----------------------- Sat Mar 24
# 20:50:26 2018 ------------------------# Use a different kernel to improve
# accuracy
svm.radial.time <- system.time({
    letter.classifier.rbf <- ksvm(letter ~ ., data = letters.train, kernel = "rbfdot")
    cM.rbf <- caret::confusionMatrix(predict(letter.classifier.rbf, letters.test),
        letters.test$letter)
})
cM.rbf$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   0.9307500000   0.9279685740   0.9224380811   0.9384257635   0.0420000000
## AccuracyPValue  McnemarPValue
##   0.0000000000            NaN
```

```r
(cM.rbf$overall[1] - cM$overall[1])/cM$overall[1]
```

```
##    Accuracy
## 0.109025916
```

```r
# About an 11% improvement in accuracy
svm.linear.time
```

```
##    user  system elapsed
##   14.11    0.62   17.56
```

```r
svm.radial.time
```

```
##    user  system elapsed
##  129.08    0.41  131.01
```

```r
(svm.radial.time[3] - svm.linear.time[3])/svm.linear.time[3]
```

```
##    elapsed
## 6.46070615
```

```r
# With a cost of about ~11x the time to compute. Worth taking into consideration
# if we were to use larger datasets.
```

Let's set up an optimization using caret.

```r
library(caret)
ltr.train <- createDataPartition(letters$letter, times = 1, p = 0.8)
ltr.train <- trainControl(method = "repeatedcv", repeats = 1, index = ltr.train,
```

```r
    verboseIter = T, allowParallel = T)

# svmRadial.time <- system.time(ltr.mod <-
# train(letter~.,data=letters,method='svmRadial',tuneLength=9,metric='Kappa',trCtrl=ltr.train))

library(doParallel)
# make a cluster with all possible threads (not cores)
cl <- makeCluster(detectCores() - 1)
# register the number of parallel workers (here all CPUs)
registerDoParallel(cl)
# return number of parallel workers
getDoParWorkers()
svmRadial.par.time <- system.time(ltr.par.mod <- train(letter ~ ., data = letters,
    method = "svmRadial", tuneLength = 8, metric = "Accuracy", trCtrl = ltr.train))
# insert parallel calculations here stop the cluster and remove Rscript.exe
# childs (WIN)
stopCluster(cl)
registerDoSEQ()
train.results <- list(Time = svmRadial.par.time, Model = ltr.par.mod)
save(train.results, file = "svmRadial.RData")
```

```r
load("svmRadial.RData")
svmRadial.par.time <- train.results$Time
ltr.par.mod <- train.results$Model
svmRadial.par.time
```

```
##    user  system elapsed
##   93.60    9.81 9009.60
```

```r
svmRadial.par.time[3]/3600  # Hours
```

```
##     elapsed
## 2.502666667
```

```r
svm.radial.time
```

```
##    user  system elapsed
##  129.08    0.41  131.01
```

```r
# ----------------------- Tue Mar 27 14:28:03 2018 -----------------------# The
# time involved to train the model with caret took approximately 81 times as long
(svmRadial.par.time[3] - svm.radial.time[3])/svm.radial.time[3]
```

```
##     elapsed
## 67.77032288
```

```r
# The best tune
ltr.par.mod$bestTune
```

	sigma	C
8	0.0473692272	32

```r
# Model validation
cM.par <- caret::confusionMatrix(predict(ltr.par.mod$finalModel, newdata = letters.test[,
    -1], type = "response"), letters.test$letter)
cM.par$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   0.9955000000   0.9953191607   0.9928973496   0.9973309003   0.0420000000
## AccuracyPValue  McnemarPValue
##   0.0000000000            NaN
```

```r
# 99.5 % accuracy - possibly overfitted, but excellent accuracy nonetheless.
(cM.par$overall[1] - cM$overall[1])/cM$overall[1]  # ~ 19% improvement over the linear model
```

```
##     Accuracy
## 0.1861781352
```

```r
(cM.par$overall[1] - cM.rbf$overall[1])/cM.rbf$overall[1]  # ~7% improvement over the single train radi
```

```
##      Accuracy
## 0.06956755305
```

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.

```r
# ----------------------- Tue Mar 27 14:58:23 2018 ------------------------# Load
# Data
library(arules)
groceries <- arules::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
```

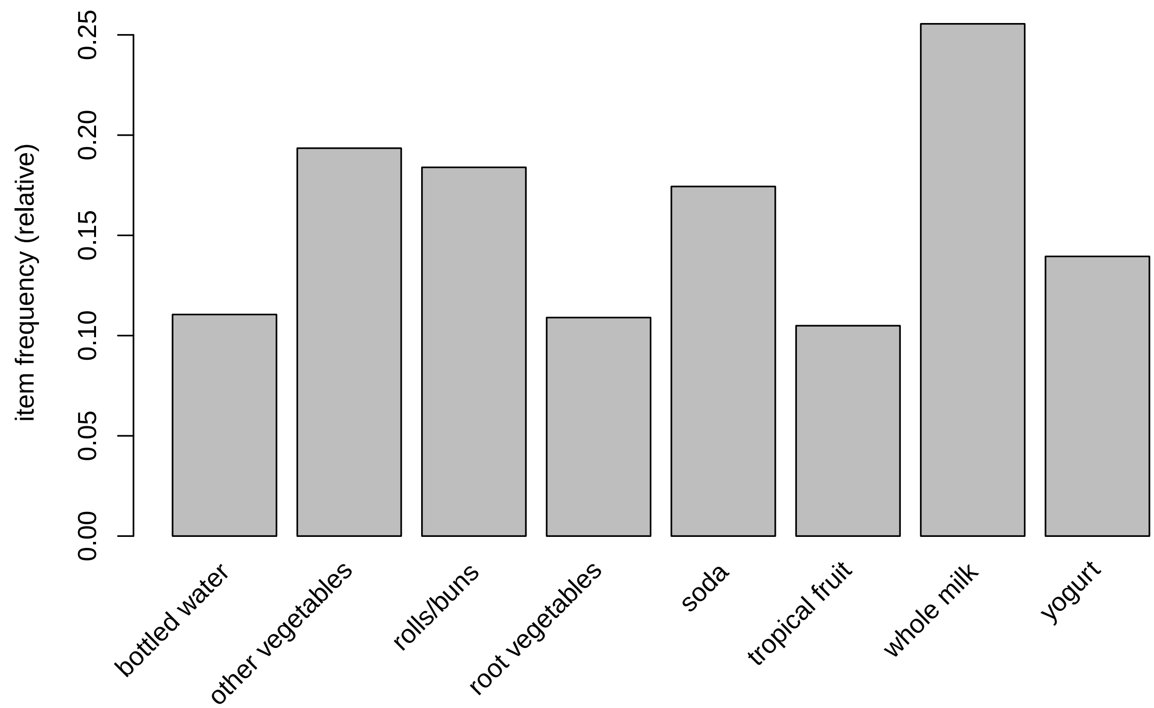
```r
# View the first 5
inspect(groceries[1:5])
```

```
##     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}
```

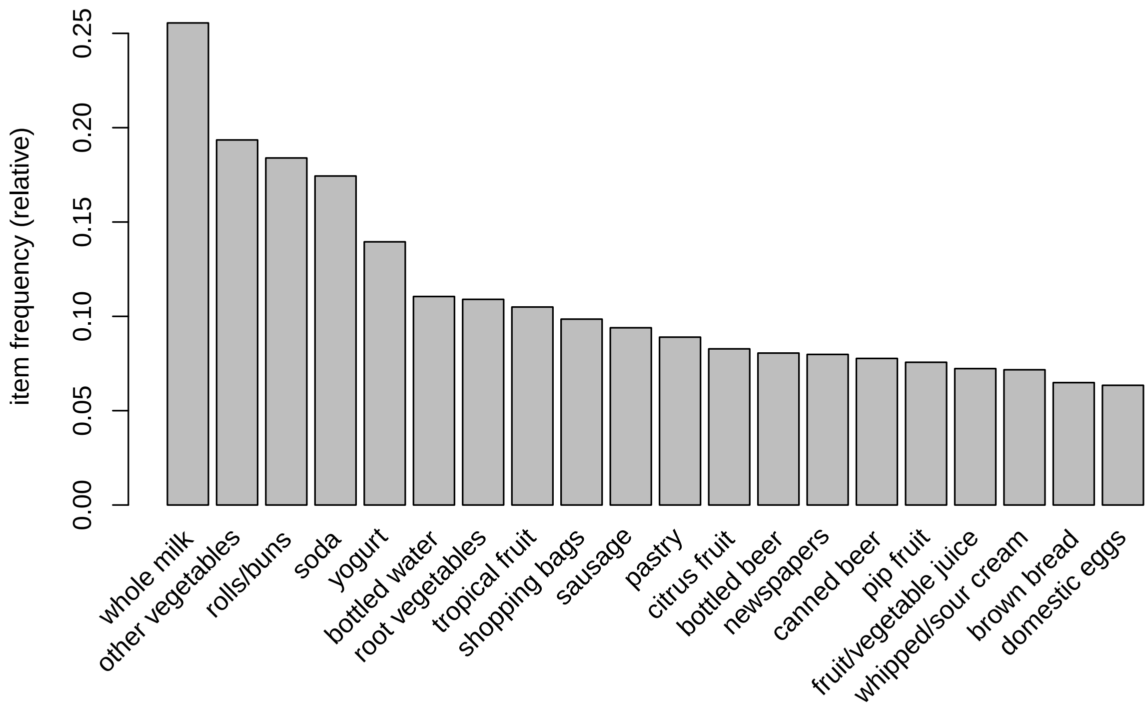
```r
# View ordered Item frequencies in the top quantile
groc.iF <- itemFrequency(groceries)
(groc.q3 <- groc.iF[groc.iF > quantile(groc.iF)[4]][order(groc.iF[groc.iF > quantile(groc.iF)[4]],
    decreasing = T)])
```

```
##               whole milk         other vegetables               rolls/buns
##               0.25551601423            0.19349262837            0.18393492628
##                    soda                   yogurt            bottled water
##               0.17437722420            0.13950177936            0.11052364006
##          root vegetables           tropical fruit            shopping bags
##               0.10899847483            0.10493136756            0.09852567361
##                 sausage                   pastry             citrus fruit
##               0.09395017794            0.08896797153            0.08276563294
##             bottled beer               newspapers              canned beer
##               0.08052872395            0.07981698017            0.07768174886
##                pip fruit    fruit/vegetable juice       whipped/sour cream
##               0.07564819522            0.07229283172            0.07168276563
##              brown bread             domestic eggs              frankfurter
##               0.06487036096            0.06344687341            0.05897305541
##                margarine                   coffee                     pork
##               0.05856634469            0.05805795628            0.05765124555
##                   butter                     curd                     beef
##               0.05541433655            0.05327910524            0.05246568378
##                  napkins                chocolate        frozen vegetables
##               0.05236400610            0.04961870869            0.04809354347
##                  chicken              white bread             cream cheese
##               0.04290798170            0.04209456024            0.03965429588
##                  waffles             salty snack long life bakery product
##               0.03843416370            0.03782409761            0.03741738688
##                  dessert                    sugar                 UHT-milk
##               0.03711235384            0.03385866802            0.03345195730
##                  berries            hamburger meat         hygiene articles
##               0.03324860193            0.03324860193            0.03294356889
```

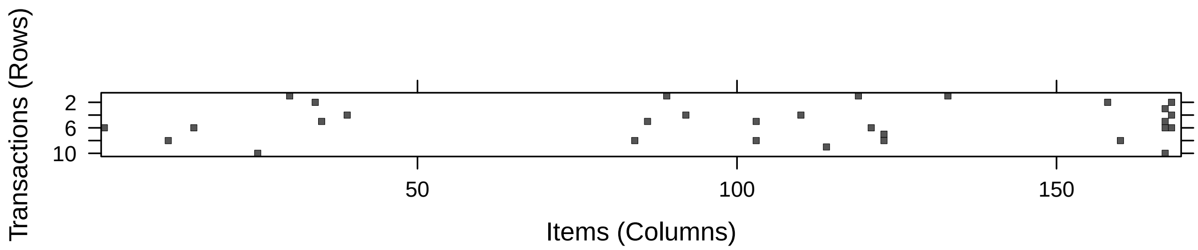
```r
# Visualize the frequencies with support greater than 10%
itemFrequencyPlot(groceries, support = 0.1)  # This matches the output above
```



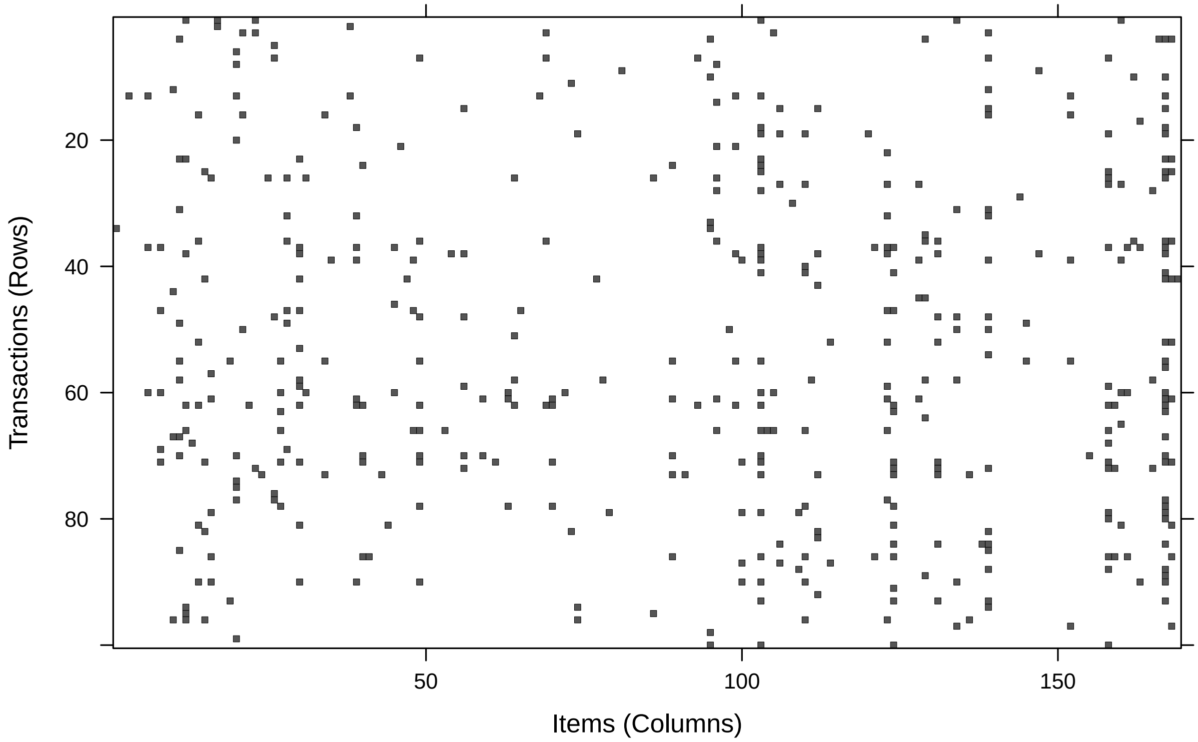

```r
# Generate a Pareto bar graph of the top 20 items with the most support
itemFrequencyPlot(groceries, topN = 20)
```




```
# Visualize the sparse data matrix top 10
image(groceries[1:10])
```




```r
# Useful for determining if there is some repetitive value (that's not a
# transaction item) in the dataset that shouldn't be there. Also useful for
# noticing seasonal patterns if the data is sorted by timeseries.  Visualize 100
# randomly selected rows
image(sample(groceries, 100))  # Vertical patterns (popular items) are notably easier to spot.
```




```r
# Skipping the error run of apriori because thresholds are set at default and
# don't return any rules.
support(groceries, transactions = groceries) %>% quantile
```

```
##               0%              25%              50%              75%
## 0.0001016776817 0.0001016776817 0.0004067107270 0.0130147432639
##             100%
## 0.2555160142349
```

```r
# ----------------------- Tue Mar 27 15:41:40 2018 ------------------------# It
# looks like our third quantile of support begins at .013, so we will set our
# support threshold for the apriori there
sup.trshld <- support(groceries, transactions = groceries) %>% quantile %>% .[4]
# We will stick with the example exercise value for the confidence threshold
# minlen = 2 makes it such that at least 2 items must appear in a rule
groceryrules <- apriori(groceries, parameter = list(support = sup.trshld, confidence = 0.25,
    minlen = 2))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime       support
##        0.25    0.1    1 none FALSE            TRUE       5 0.01301474326
##  minlen maxlen target   ext
##       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: 128
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [76 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [101 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
groceryrules %>% summary  # 101 rules
```

```
## set of 101 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3
## 74 27
##
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 2.000000 2.000000 2.000000 2.267327 3.000000 3.000000
##
## summary of quality measures:
##     support             confidence            lift
##  Min.   :0.01301474   Min.   :0.2517361   Min.   :0.9932367
##  1st Qu.:0.01514997   1st Qu.:0.2975543   1st Qu.:1.5047187
##  Median :0.01972547   Median :0.3614130   Median :1.7597542
##  Mean   :0.02294493   Mean   :0.3652410   Mean   :1.7882018
##  3rd Qu.:0.02613116   3rd Qu.:0.4170616   3rd Qu.:2.0004746
##  Max.   :0.07483477   Max.   :0.5629921   Max.   :3.0403668
##      count
##  Min.   :128.0000
##  1st Qu.:149.0000
##  Median :194.0000
##  Mean   :225.6634
##  3rd Qu.:257.0000
##  Max.   :736.0000
##
## mining info:
##       data ntransactions       support confidence
##  groceries          9835 0.01301474326       0.25
```

```
# Noteable is that with the higher threshold here, the rules that qualified
# contained at max 3 items.  ----------------------- Tue Mar 27 16:01:23 2018
# ------------------------# Inspect the rules with the top 10% of lift, sorted by
# lift

inspect(sort(groceryrules[groceryrules@quality[["lift"]] > qnorm(0.9, mean(groceryrules@quality[["lift"]
    sd(groceryrules@quality[["lift"]]))], by = "lift", decreasing = T))
```

```
##      lhs                      rhs                      support    confidence        lift count
## [1]  {beef}                => {root vegetables}  0.01738688358 0.3313953488 3.040366843   171
## [2]  {other vegetables,
##       whole milk}          => {root vegetables}  0.02318251144 0.3097826087 2.842082049   228
```

```
## [3]  {whole milk,
##       yogurt}              => {tropical fruit}   0.01514997458 0.2704174229 2.577088521   149
## [4]  {pip fruit}           => {tropical fruit}   0.02043721403 0.2701612903 2.574647568   201
## [5]  {tropical fruit,
##       whole milk}          => {yogurt}           0.01514997458 0.3581730769 2.567516189   149
## [6]  {root vegetables,
##       whole milk}          => {other vegetables} 0.02318251144 0.4740124740 2.449770195   228
## [7]  {whole milk,
##       yogurt}              => {root vegetables}  0.01453990849 0.2595281307 2.381025341   143
## [8]  {onions}              => {other vegetables} 0.01423487544 0.4590163934 2.372268119   140
## [9]  {whipped/sour cream,
##       whole milk}          => {other vegetables} 0.01464158617 0.4542586751 2.347679490   144
## [10] {curd}                => {yogurt}           0.01728520590 0.3244274809 2.325615361   170
## [11] {pip fruit,
##       whole milk}          => {other vegetables} 0.01352313167 0.4493243243 2.322177998   133
```

```
# These are likely to be actionable Let's see what gets bought with vegetables or
# fruit
(vegfrurules <- subset(groceryrules, items %pin% c("fruit") | items %pin% c("vegetables")))
```

```
## set of 62 rules
```

```
inspect(sort(vegfrurules[vegfrurules@quality[["lift"]] > qnorm(0.9, mean(vegfrurules@quality[["lift"]])
    sd(vegfrurules@quality[["lift"]]))], by = "lift", decreasing = T))
```

```
##     lhs                               rhs                support
## [1] {beef}                         => {root vegetables}  0.01738688358
## [2] {other vegetables,whole milk} => {root vegetables}  0.02318251144
## [3] {whole milk,yogurt}           => {tropical fruit}   0.01514997458
## [4] {pip fruit}                   => {tropical fruit}   0.02043721403
## [5] {tropical fruit,whole milk}   => {yogurt}           0.01514997458
## [6] {root vegetables,whole milk}  => {other vegetables} 0.02318251144
##     confidence   lift        count
## [1] 0.3313953488 3.040366843 171
## [2] 0.3097826087 2.842082049 228
## [3] 0.2704174229 2.577088521 149
## [4] 0.2701612903 2.574647568 201
## [5] 0.3581730769 2.567516189 149
## [6] 0.4740124740 2.449770195 228
```

```
# Convert to dataframe
vegfrurules %<>% as(Class = "data.frame")
```

It's interesting to learn about the formula behind the "Customers who bought this item also bought" feed on many online retailers. With regards to usage, subsetting takes some experimentation to achieve the expected results. I can see how association rules might also have usefulness in text-analysis showing words that are often associated in the same sentence with one another.