Association Rules

Rob Carver

We can use a modified version of the Framingham Heart Study dataset to demonstrate association analysis (also known as 'market basket' analysis). Unlike the clustering methods we recently studied, association rules work exclusively with discrete categorical data. Hence, early in our process we need to convert all continuous variables into discrete ordinal factors.

As usual, we load packages. For association rules, our main packages are `arules` and `arulesViz`. The former discovers the rules, and the latter provides several options for visualizing rules.

```
# First load all packages

library(arules)
library(arulesViz)
library(readr)
```

As we did in an earlier script, we read in the data, redefine the ANYCHD.2 variable as a factor. Before partitioning the data into training and test sets, we'll create factors based on the numeric variables. As a first approach, we'll transform each continuous numeric into a 5-level ordinal factor.

```
fram <- read.csv("C:/Users/Rob/Box Sync/My R Work/BUS212/Data/frmgham1.csv")

# Target variable is in final column "ANYCHD.2"
# ANYCHD.2 reads in as an integer, so make it a factor

fram$`ANYCHD.2` <- factor(fram$ANYCHD.2)
print("ANYCHD.2 in Full Framingham data table")

## [1] "ANYCHD.2 in Full Framingham data table"

table(fram$ANYCHD.2)  # dummy, 1 = has heart disease

##
##    0    1
## 2859 1071

# This dataset has a small number of missing values # for a few variables.
For this demo, we'll drop cases
# with missing
framcols <- c(2:18, 39)  # subset main data frame,
framsub <- na.omit(fram[,framcols])
print("ANYCHD.2 Subset with missing removed")
```

```
## [1] "ANYCHD.2 Subset with missing removed"
```

```
table(framsub$ANYCHD.2)
```

```
## 
##    0    1 
## 2473  920
```

The dataframe `framsub` now has 18 numeric columns.This chunk of code will create a new dataframe of all categorical variables . The dichotomous will remain 2-level factors, and the others will become a 5-level factor.

We use the `cut` function from base R. According to R Documnentation, "`cut` divides the range of x into intervals, and codes the values in x according to which interval they fall." With `cut` we can either specify a number of levels or specify a list of break-points. For this analysis, we'll ask for 5 levels and ordinal data.

```
attach(framsub)  # reduce typing here
framc <- data.frame(ANYCHD.2)
framc$sex <- factor(SEX, labels=c("Male", "Female"))
framc$chol <- cut(TOTCHOL, 5, ordered_result = T)
framc$age <- cut(AGE,5, ordered_result = T)
framc$sysbp <- cut(SYSBP,5, ordered_result = T)
framc$diabp <- cut(DIABP,5, ordered_result = T)
framc$cursmoke <- factor(CURSMOKE)
framc$cigpday <- cut(CIGPDAY, 5, ordered_result = TRUE)
framc$bmi <- cut(BMI, 5, ordered_result = TRUE)
framc$diabetes <- factor(DIABETES)
framc$bpmeds <- factor(BPMEDS)
framc$heart <- cut(HEARTRTE, 5, ordered_result = TRUE)
framc$glucse <- cut(GLUCOSE, 5, ordered_result = TRUE)
framc$educ <- factor(educ, labels = c("< HS", "HS", "Some college","College
or more"))
framc$prevchd <- factor(PREVCHD)
framc$prevap <- factor(PREVAP)
framc$prevmi <- factor(PREVMI)
framc$prevstrk <- factor(PREVSTRK)

##
### show results of these commands
head(framc)
```

```
##   ANYCHD.2    sex      chol         age      sysbp       diabp cursmoke
## 1        0 Female (183,253] (39.6,47.2] (83.3,126] (70.1,88.2]        0
## 2        0   Male (183,253] (47.2,54.8]  (126,168] (70.1,88.2]        1
## 3        0 Female (183,253] (54.8,62.4]  (126,168]  (88.2,106]        1
## 4        0 Female (253,324] (39.6,47.2]  (126,168] (70.1,88.2]        1
## 5        1 Female (183,253] (39.6,47.2]  (168,210]   (106,124]        0
## 6        1 Female (183,253]   (62.4,70]  (126,168] (70.1,88.2]        0
##      cigpday         bmi diabetes bpmeds       heart     glucse
## 1 (-0.07,14]   (23.8,32]        0      0  (83.6,103] (39.6,111]
## 2    (14,28]   (23.8,32]        0      0 (63.8,83.6] (39.6,111]
## 3    (28,42]   (23.8,32]        0      0 (63.8,83.6] (39.6,111]
## 4    (14,28] (15.5,23.8]        0      0  (83.6,103] (39.6,111]
## 5 (-0.07,14]   (23.8,32]        0      0 (63.8,83.6] (39.6,111]
## 6 (-0.07,14]   (32,40.3]        0      0 (43.9,63.8] (39.6,111]
##           educ prevchd prevap prevmi prevstrk
## 1           HS       0      0      0        0
## 2         < HS       0      0      0        0
## 3 Some college       0      0      0        0
## 4 Some college       0      0      0        0
## 5           HS       0      0      0        0
## 6         < HS       0      0      0        0

detach(framsub)
```

As usual, we partition the data into train and test sets and investigate the training data for association rules

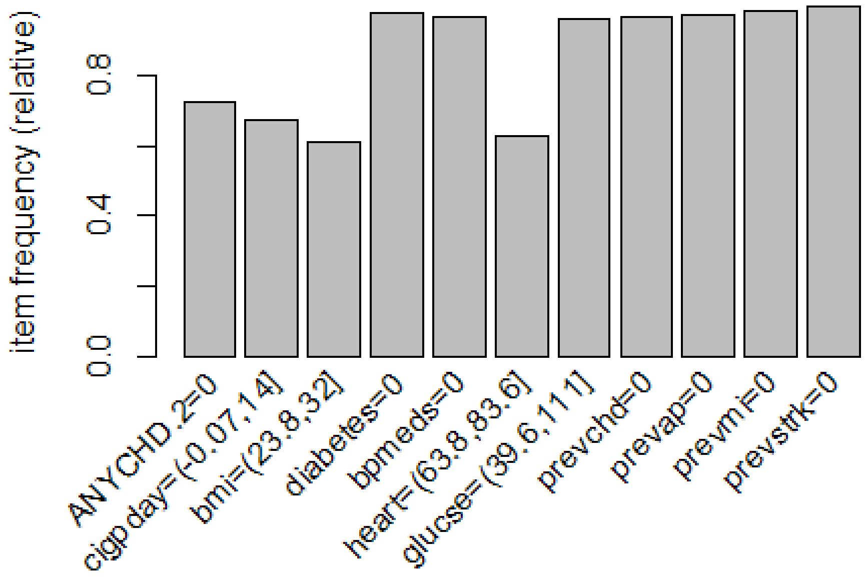
```
# Train and test sets
n <- nrow(framc)
set.seed(7752)
test_idx <- sample.int(n, size= round(0.3 * n))  # 30% test set
train <- framc[-test_idx,]  # train has all rows except the index
test <- framc[test_idx,]
```

Our `train` set is a dataframe of 2,375 adults. The `arules` package needs a *transaction* object rather than a dataframe; in a transaction object, each line of data is understood as a wide list of items contained a single transaction, as explained in class and readings. We can easily convert the dataframe into a formal transaction object with one command:

```
train.trans <- as(train, "transactions")
summary(train.trans)

## transactions as itemMatrix in sparse format with
##  2375 rows (elements/itemsets/transactions) and
##  62 columns (items) and a density of 0.2903226
##
## most frequent items:
## prevstrk=0   prevmi=0 diabetes=0   prevap=0   bpmeds=0    (Other)
##       2363       2344       2324       2311       2304      31104
##
## element (itemset/transaction) length distribution:
## sizes
##   18
## 2375
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      18      18      18      18      18      18
##
## includes extended item information - examples:
##       labels variables levels
## 1 ANYCHD.2=0  ANYCHD.2      0
## 2 ANYCHD.2=1  ANYCHD.2      1
## 3   sex=Male       sex   Male
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
## 2             2
## 3             4

itemFrequencyPlot(train.trans, support=.6)
```



Now let's start looking for rules using the a priori method. Typically this is an iterative process in which we use `paramter=list()` control to specify a minimum support level (frequency of occurrence) and minimum confidence (conditional probabilility of RHS | LHS). The goal is to find a rule set that is manageable in length and applicability. Remember that only about 27% of the patients do have any chd on the 2nd visit, so we should set support below 27% if we want to capture those patients.

```
chd_rules <- apriori(train.trans, parameter=list(support=0.05, conf = 0.50,
          target="rules"))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.05      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: 118
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[62 item(s), 2375 transaction(s)] done [0.00s].
## sorting and recoding items ... [40 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10

## Warning in apriori(train.trans, parameter = list(support = 0.05, conf =
## 0.5, : Mining stopped (maxlen reached). Only patterns up to a length of 10
## returned!

##  done [2.26s].
## writing ... [2452539 rule(s)] done [0.77s].
## creating S4 object  ... done [2.63s].
```

This first set of criteria created more than 2.4 million rules! That's far too many to analyze. Let's raise the minimums substantially (in fact I tried several other combinations before settling on these):

```
chd_rules <- apriori(train.trans, parameter=list(support=0.2, conf = 0.90,
          target="rules"))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.9    0.1    1 none FALSE            TRUE       5     0.2      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: 475
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[62 item(s), 2375 transaction(s)] done [0.00s].
## sorting and recoding items ... [30 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(train.trans, parameter = list(support = 0.2, conf =
## 0.9, : Mining stopped (maxlen reached). Only patterns up to a length of 10
## returned!
```

```
##  done [0.17s].
## writing ... [70408 rule(s)] done [0.02s].
## creating S4 object  ... done [0.03s].
```

Now we have 70,000+ rules, many of which will surely be redundant and not involve our target variable. At this point, though, let's inspects a few of the rules:

```
options(digits = 3)
summary(chd_rules)
```

```
## set of 70408 rules
##
## rule length distribution (lhs + rhs):sizes
##     1     2     3     4     5     6     7     8     9    10
##     7   200  1569  6024 13438 18711 16809  9670  3364   616
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    5.00    6.00    6.25    7.00   10.00
##
## summary of quality measures:
##     support        confidence         lift
##  Min.   :0.200   Min.   :0.900   Min.   :0.925
##  1st Qu.:0.228   1st Qu.:0.979   1st Qu.:1.003
##  Median :0.260   Median :0.993   Median :1.013
##  Mean   :0.295   Mean   :0.988   Mean   :1.027
##  3rd Qu.:0.320   3rd Qu.:1.000   3rd Qu.:1.024
##  Max.   :0.995   Max.   :1.000   Max.   :2.128
##
## mining info:
##         data ntransactions support confidence
##  train.trans          2375     0.2        0.9
```

```
inspect(subset(chd_rules, lift > 2)[1:5,])
```

```
##     lhs                                   rhs          support confidence
## [1] {cigpday=(14,28]}                  => {cursmoke=1} 0.240   1
## [2] {cigpday=(14,28],glucse=(39.6,111]} => {cursmoke=1} 0.229   1
## [3] {cigpday=(14,28],prevchd=0}        => {cursmoke=1} 0.235   1
## [4] {cigpday=(14,28],bpmeds=0}         => {cursmoke=1} 0.236   1
## [5] {cigpday=(14,28],prevap=0}         => {cursmoke=1} 0.237   1
##     lift
## [1] 2.13
## [2] 2.13
## [3] 2.13
## [4] 2.13
## [5] 2.13
```

These few rules are obvious , but they do illustrate how association rules operate. The rules above are *unsupervised* -- they simply identify frequent itemsets that co-occur.

We can also conduct supervised learning but selecting a right-hand side of interest to us. For example, let's find the rules that point to the presence of any coronary heart disease on the second visit.

```
anychd_rules <- apriori(train.trans,
     parameter=list(support=0.05, conf = 0.35, target="rules"),
     appearance = list(rhs=c("ANYCHD.2=1"),
     default = "lhs"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.35    0.1    1 none FALSE            TRUE       5    0.05      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: 118
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[62 item(s), 2375 transaction(s)] done [0.00s].
## sorting and recoding items ... [40 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
```

```
## Warning in apriori(train.trans, parameter = list(support = 0.05, conf =
## 0.35, : Mining stopped (maxlen reached). Only patterns up to a length of
10
## returned!
```

```
##  done [1.89s].
## writing ... [564 rule(s)] done [0.03s].
## creating S4 object  ... done [0.02s].
```

```
rules.sorted <- sort(anychd_rules, by="lift")
inspect(subset(rules.sorted, lift > 1.4)[1:5,])
```

```
##     lhs                     rhs          support confidence lift
## [1] {sex=Male,
##      sysbp=(126,168],
##      bmi=(23.8,32],
##      glucse=(39.6,111],
##      prevstrk=0}        => {ANYCHD.2=1}  0.0712      0.403 1.46
## [2] {sex=Male,
##       chol=(183,253],
##       sysbp=(126,168],
##       diabetes=0,
##       bpmeds=0,
##       prevstrk=0}        => {ANYCHD.2=1}  0.0509      0.403 1.46
## [3] {sex=Male,
##       chol=(183,253],
##       sysbp=(126,168],
##       diabetes=0,
##       prevstrk=0}        => {ANYCHD.2=1}  0.0514      0.403 1.46
## [4] {sex=Male,
##       chol=(183,253],
##       sysbp=(126,168],
##       bpmeds=0,
##       prevstrk=0}        => {ANYCHD.2=1}  0.0522      0.403 1.46
## [5] {sex=Male,
##       sysbp=(126,168],
##       bmi=(23.8,32],
##       glucse=(39.6,111]} => {ANYCHD.2=1}  0.0712      0.402 1.45
```

Notice that some of these rules are redundant. We can "prune" redundant rules as follows:

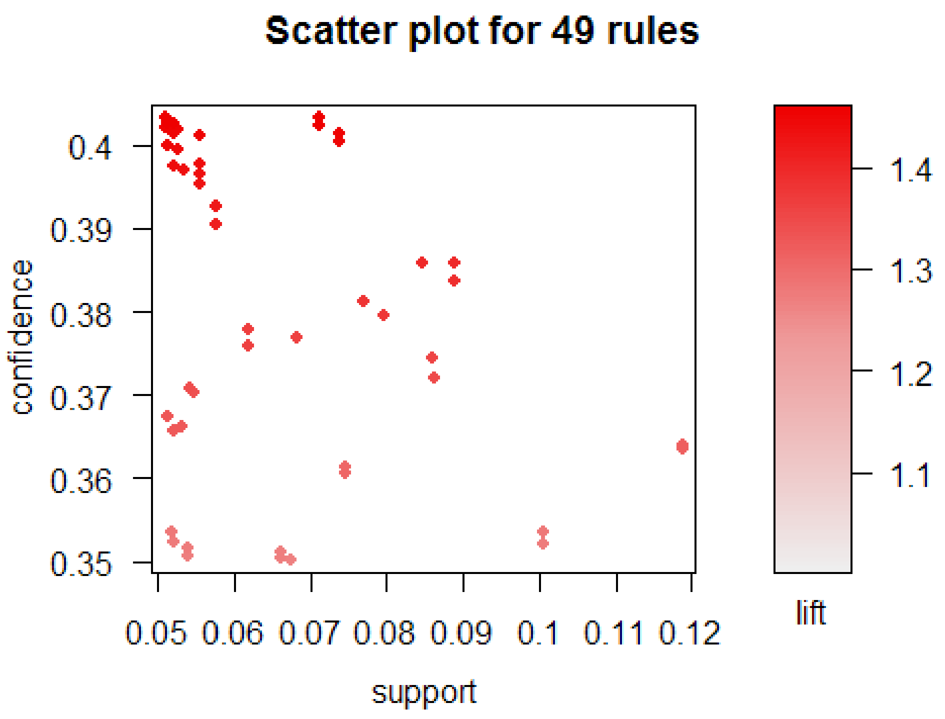
```
# first find redundant rules
subset.matrix <- is.subset(rules.sorted, rules.sorted)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1

# remove redundant rules
rules.pruned <- rules.sorted[!redundant]
inspect(subset(rules.pruned[1:5,]))
```

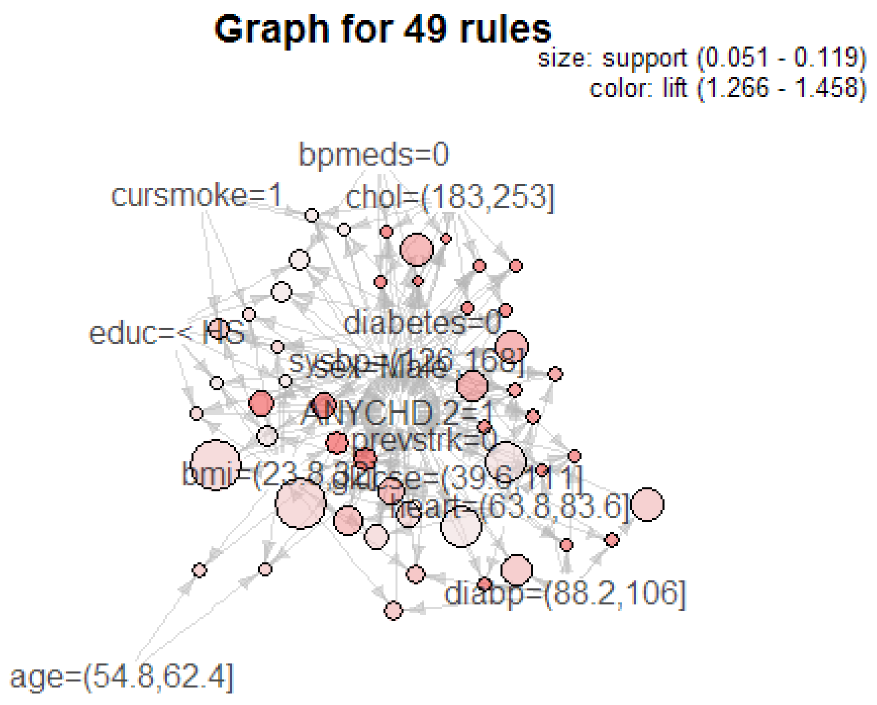
```
##     lhs                        rhs          support confidence lift
## [1] {sex=Male,
##       sysbp=(126,168],
##       bmi=(23.8,32],
##       glucse=(39.6,111],
##       prevstrk=0}        => {ANYCHD.2=1}  0.0712      0.403 1.46
## [2] {sex=Male,
##       chol=(183,253],
##       sysbp=(126,168],
##       diabetes=0,
##       bpmeds=0,
##       prevstrk=0}        => {ANYCHD.2=1}  0.0509      0.403 1.46
## [3] {sex=Male,
##       chol=(183,253],
##       sysbp=(126,168],
##       diabetes=0,
##       prevstrk=0}        => {ANYCHD.2=1}  0.0514      0.403 1.46
## [4] {sex=Male,
##       chol=(183,253],
##       sysbp=(126,168],
##       bpmeds=0,
##       prevstrk=0}        => {ANYCHD.2=1}  0.0522      0.403 1.46
## [5] {sex=Male,
##       sysbp=(126,168],
##       bmi=(23.8,32],
##       glucse=(39.6,111]} => {ANYCHD.2=1}  0.0712      0.402 1.45
```

Now that we have a set of rules, let's take a look at them. Package `arulesViz` provides several helpful ways to visualize the rules.

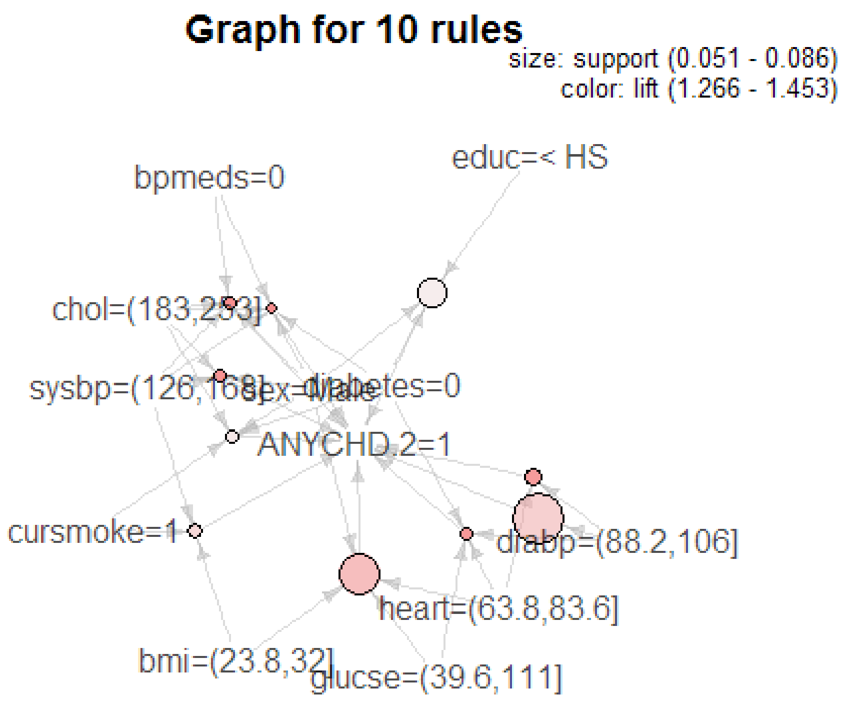
```
plot(rules.pruned)
```



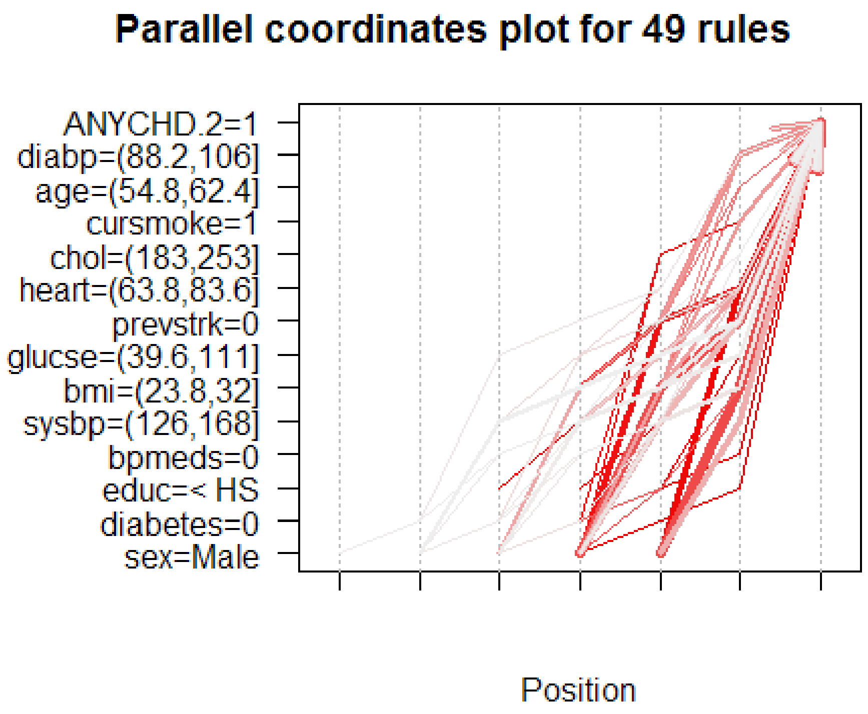
```
plot(rules.pruned, method="graph", control = list(type="items"))
```



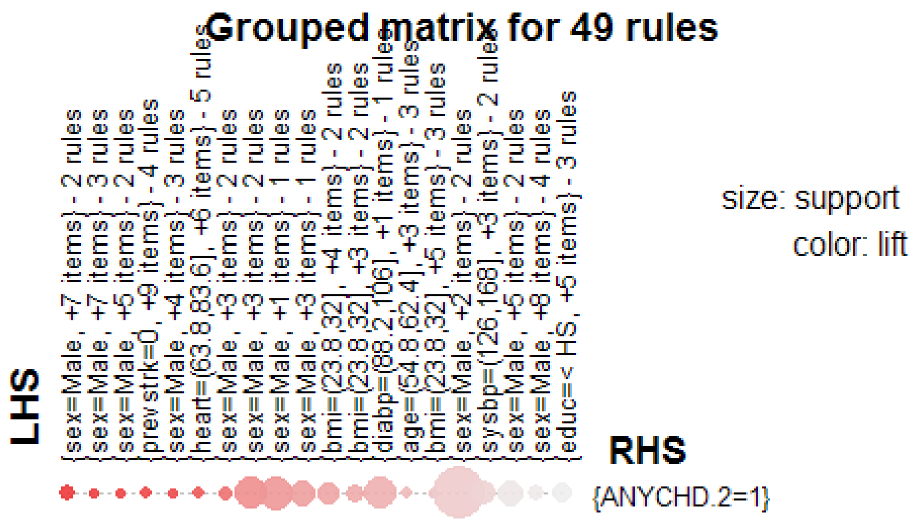
```
subrules <- sample(rules.pruned, 10)
plot(subrules, method="graph")
```



```r
plot(rules.pruned, method="paracoord")
```



```r
plot(rules.pruned, method="grouped")
```


Grouped matrix for 49 rules
LHS
{sex=Male, +7 items} - 2 rules
{sex=Male, +7 items} - 3 rules
{sex=Male, +5 items} - 2 rules
{prevstrk=0, +9 items} - 4 rules
{sex=Male, +4 items} - 3 rules
{heart=(63.8,83.6], +6 items} - 5 rules
{sex=Male, +3 items} - 2 rules
{sex=Male, +3 items} - 2 rules
{sex=Male, +1 items} - 1 rules
{sex=Male, +3 items} - 1 rules
{bmi=(23.8,32], +4 items} - 2 rules
{bmi=(23.8,32], +3 items} - 2 rules
{diabp=(88.2,106], +1 items} - 1 rules
{age=(54.8,62.4], +3 items} - 3 rules
{bmi=(23.8,32], +5 items} - 3 rules
{sex=Male, +2 items} - 2 rules
{sysbp=(126,168], +3 items} - 2 rules
{sex=Male, +5 items} - 2 rules
{sex=Male, +8 items} - 4 rules
{educ=< HS, +5 items} - 3 rules
RHS
{ANYCHD.2=1}
size: support
color: lift