PDXDataSciReccomender

Charles Howard

November 14, 2017

Overview

The goal is to build a recommendation engine for games. The R package arules is used to mine associations between lists of items. The arulesViz package has plot methods to visualize relationships between items.

I started with the original set of 834415 rows and 3 columns. The arules package requires nominal variables be converted to factors and continuous variables to be discretized. I followed examples given in the following webpage: http://michael.hahsler.net/research/arules_RUG_2015/demo/

R code follows:

```r
library(arules)
library(arulesViz)
#library(Matrix)  if needed
datdir<-"C:/Users/Charles/Documents/PDXDataSciRecommender/"
setwd(datdir)
dat<-read.csv(paste(datdir,"boardgame-ratings.csv",sep=""))
# sorting data by 1.) UserId, then 2.) gameID
dat<-dat[order(dat$UserID,dat$gameID),]
# determining groupings by UserID
usergrping<-grouping(dat$UserID)
userid.ends<-attr(usergrping,"ends")
userid.starts<-c(1,userid.ends[1:(length(userid.ends)-1)]+1)
userid.counts<-diff(userid.starts)
# convert to factors
dat[,"UserID"]<-factor(dat[,"UserID"])
dat[,"gameID"]<-factor(dat[,"gameID"])
# discretize ratings
dat[,"rating"]<-discretize(dat$rating,method="interval",categories=5)
# for first attempt, I create a list of gameID's by UserID
translist<-lapply(1:length(userid.ends),function(n){
  rws<-userid.starts[n]:userid.ends[n]
  x<-dat$gameID[rws]
})
# the transaction class is the primary one used for arules
datrans<-as(translist,"transactions")
```

Each list in translist is a "transaction". In this instance, a list of gameID's for UserID "1".

```r
print(translist[[1]])
```

```
[1] 13     3076   31260  36218  40692  68448  129622 148228
27 Levels: 11 13 103 478 822 1927 2163 2651 3076 9209 14996 ... 197376
```

Summary of the datrans transactions object.

```r
summary(datrans)
```

```
transactions as itemMatrix in sparse format with
 154655 rows (elements/itemsets/transactions) and
 27 columns (items) and a density of 0.1998271

most frequent items:
     13     822   30549   36218   68448 (Other)
  57284   57092   54279   47936   45617  572207

element (itemset/transaction) length distribution:
sizes
    1     2     3     4     5     6     7     8     9    10    11    12
44648 14747 12740 10951  9544  8222 13791  5547  4890  4515  4059  3550
   13    14    15    16    17    18    19    20    21    22    23
 6081  2624  2136  1848  1601  1276  1539   235    92    17     2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   4.000   5.395   8.000  23.000

includes extended item information - examples:
  labels
1     11
2     13
3    103
```

Some standard measures for item lists are support and confidence. Support is the proportion of a given item list in the data. Confidence is a conditional probability type measure. The confidence of item set A => item set B is: support(item set A) support(item set B)/support(item set A)
I arbitrarily chose a target of 1000 to arrive at a support value.

```r
# find a support level
sup<-1000/nrow(datrans)
print(paste("support is ",sup,sep=""))
```

```
[1] "support is 0.00646600497882383"
```

```r
itemsets <- apriori(datrans, parameter = list(target = "frequent",
                                              supp=sup, minlen = 3))
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime     support
         NA    0.1    1 none FALSE            TRUE       5 0.006466005
 minlen maxlen            target   ext
      3     10 frequent itemsets FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 1000

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[27 item(s), 154655 transaction(s)] done [0.06s].
sorting and recoding items ... [26 item(s)] done [0.02s].
creating transaction tree ... done [0.10s].
checking subsets of size 1 2 3 4 5 6 7 8 done [5.08s].
writing ... [175295 set(s)] done [0.07s].
```

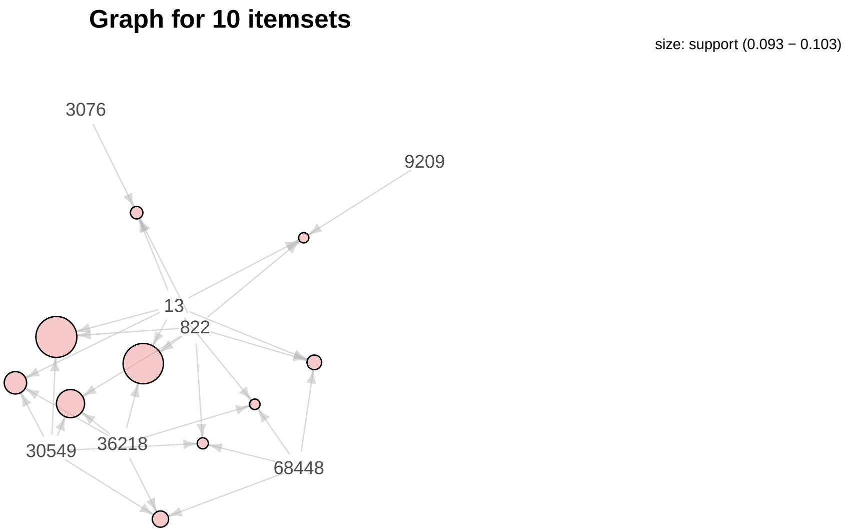
```
creating S4 object  ... done [0.15s].
```

```
inspect(head(sort(itemsets), n=10))
```

```
     items               support    count
[1]  {13,822,30549}      0.10280948 15900
[2]  {13,822,36218}      0.10259610 15867
[3]  {822,30549,36218}   0.09851605 15236
[4]  {13,30549,36218}    0.09659565 14939
[5]  {30549,36218,68448} 0.09453946 14621
[6]  {13,822,68448}      0.09398985 14536
[7]  {13,822,3076}       0.09328505 14427
[8]  {822,30549,68448}   0.09281304 14354
[9]  {13,36218,68448}    0.09260612 14322
[10] {13,822,9209}       0.09252853 14310
```

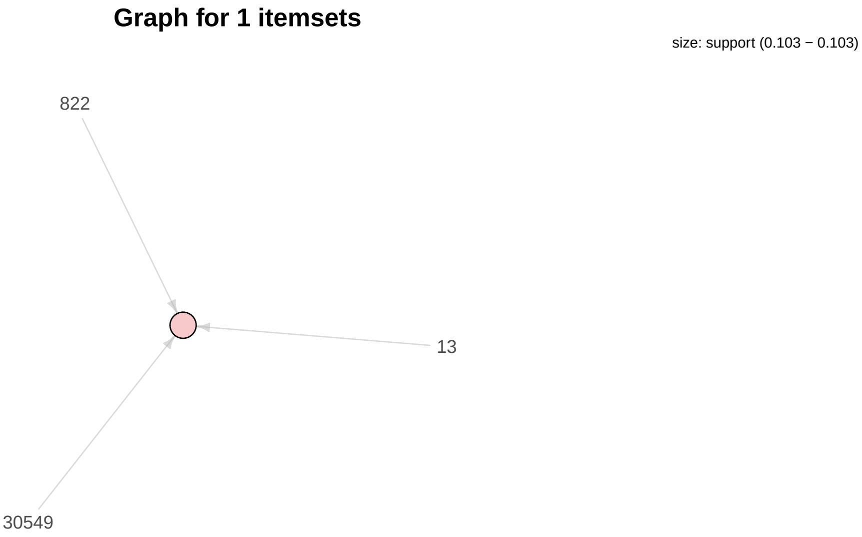
There's a cool graph method.

```
plot(head(sort(itemsets, by = "support"), n=10), method = "graph", control=list(cex=.8))
```



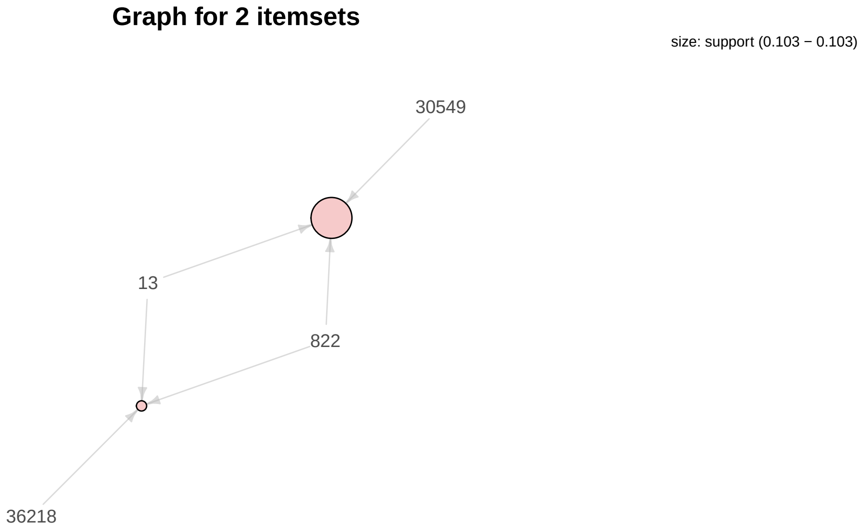
The first grouping in the table above.

```
plot(head(sort(itemsets, by = "support"), n=1), method = "graph", control=list(cex=.8))
```



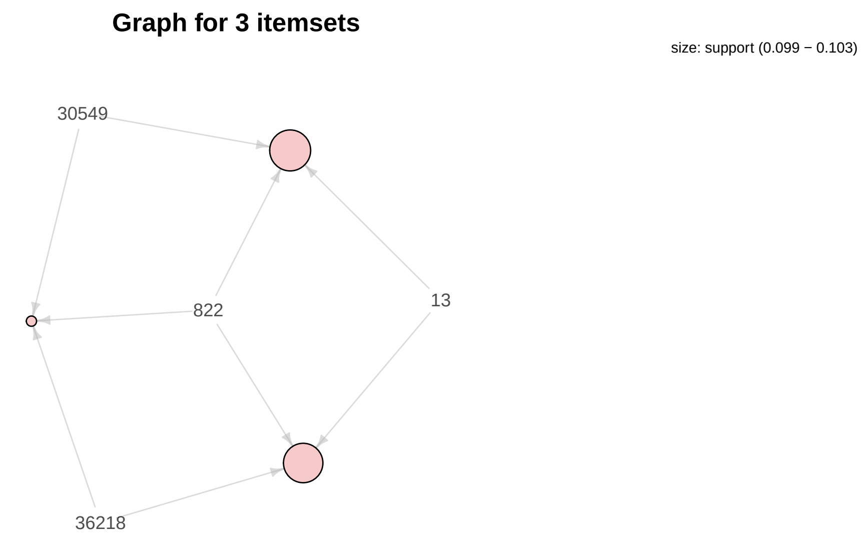
The first and second groupings in the table above.

```r
plot(head(sort(itemsets, by = "support"), n=2), method = "graph", control=list(cex=.8))
```



. . . and so on. . .

```r
plot(head(sort(itemsets, by = "support"), n=3), method = "graph", control=list(cex=.8))
```


Graph for 3 itemsets
size: support (0.099 − 0.103)
30549
822
13
36218