DA420\_project3\_Grahn

Jason Grahn

1/25/2019

# Part 1

Code is below. Output will be attached separately.

# Traditional Conjoint Analysis (R)  
  
print.digits <- 2 # set number of digits on print and spine chart  
library(support.CEs) # package for survey construction   
  
# generate a balanced set of product profiles for survey  
provider.survey <- Lma.design(attribute.names =   
 list(brand = c("AT&T","T-Mobile","US Cellular","Verizon"),   
 startup = c("$100","$200","$300","$400"),   
 monthly = c("$100","$200","$300","$400"),  
 service = c("4G NO","4G YES"),   
 retail = c("Retail NO","Retail YES"),  
 apple = c("Apple NO","Apple YES"),   
 samsung = c("Samsung NO","Samsung YES"),   
 google = c("Nexus NO","Nexus YES")), nalternatives = 1, nblocks=1, seed=9999)  
  
#sink(here::here("project3/questions\_for\_survey.txt") # send survey to external text file  
#questionnaire(provider.survey)  
#sink() # send output back to the screen  
  
# user-defined function for plotting descriptive attribute names   
effect.name.map <- function(effect.name) {   
 if(effect.name=="brand") return("Mobile Service Provider")  
 if(effect.name=="startup") return("Start-up Cost")  
 if(effect.name=="monthly") return("Monthly Cost")  
 if(effect.name=="service") return("Offers 4G Service")  
 if(effect.name=="retail") return("Has Nearby Retail Store")  
 if(effect.name=="apple") return("Sells Apple Products")  
 if(effect.name=="samsung") return("Sells Samsung Products")  
 if(effect.name=="google") return("Sells Google/Nexus Products")  
 }   
  
# read in conjoint survey profiles with respondent ranks  
conjoint.data.frame <- readr::read\_csv(here::here("project3/mobile.csv"))  
  
#building a randomizer to change the rankings in the mobile data.  
set.seed(42)  
ranklist <- c(1:16) #build a vector of 1 through 16  
samplist <- sample(ranklist, #sample that vector and make it a list  
 size=16)  
samplist <- tibble::tibble(samplist) #turn that list into it's own table  
  
#using that table to redo the ranking list.   
conjoint.data.frame <- add\_column(conjoint.data.frame, sample(ranklist,  
 size=16)) %>%   
 rename(ranking2 = "sample(ranklist, size = 16)")   
conjoint.data.frame

## # A tibble: 16 x 10  
## brand startup monthly service retail apple samsung google ranking  
## <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <dbl>  
## 1 AT&T $100 $100 4G NO Retai… APPL… Samsun… Nexus… 11  
## 2 Veri… $300 $100 4G NO Retai… APPL… Samsun… Nexus… 12  
## 3 US C… $400 $200 4G NO Retai… APPL… Samsun… Nexus… 9  
## 4 Veri… $400 $400 4G YES Retai… APPL… Samsun… Nexus… 2  
## 5 Veri… $200 $300 4G NO Retai… APPL… Samsun… Nexus… 8  
## 6 Veri… $100 $200 4G YES Retai… APPL… Samsun… Nexus… 13  
## 7 US C… $300 $300 4G YES Retai… APPL… Samsun… Nexus… 7  
## 8 AT&T $400 $300 4G NO Retai… APPL… Samsun… Nexus… 4  
## 9 AT&T $200 $400 4G YES Retai… APPL… Samsun… Nexus… 5  
## 10 T-mo… $400 $100 4G YES Retai… APPL… Samsun… Nexus… 16  
## 11 US C… $100 $400 4G NO Retai… APPL… Samsun… Nexus… 3  
## 12 T-mo… $200 $200 4G NO Retai… APPL… Samsun… Nexus… 6  
## 13 T-mo… $100 $300 4G YES Retai… APPL… Samsun… Nexus… 10  
## 14 US C… $200 $100 4G YES Retai… APPL… Samsun… Nexus… 15  
## 15 T-mo… $300 $400 4G NO Retai… APPL… Samsun… Nexus… 1  
## 16 AT&T $300 $200 4G YES Retai… APPL… Samsun… Nexus… 14  
## # … with 1 more variable: ranking2 <int>

# set up sum contrasts for effects coding as needed for conjoint analysis  
options(contrasts=c("contr.sum","contr.poly"))  
  
# main effects model specification  
main.effects.model <- {ranking2 ~ brand + startup + monthly + service +   
 retail + apple + samsung + google}  
  
# fit linear regression model using main effects only (no interaction terms)  
main.effects.model.fit <- lm(main.effects.model, data=conjoint.data.frame)  
print(summary(main.effects.model.fit))

##   
## Call:  
## lm.default(formula = main.effects.model, data = conjoint.data.frame)  
##   
## Residuals:  
## 1 2 3 4 5 6 7 8 9 10 11 12   
## 0.25 -0.25 -0.25 0.25 0.25 -0.25 0.25 -0.25 -0.25 0.25 0.25 0.25   
## 13 14 15 16   
## -0.25 -0.25 -0.25 0.25   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 8.500 0.250 34.000 0.0187 \*  
## brand1 1.250 0.433 2.887 0.2123   
## brand2 0.250 0.433 0.577 0.6667   
## brand3 -2.000 0.433 -4.619 0.1357   
## startup1 3.000 0.433 6.928 0.0913 .  
## startup2 -2.750 0.433 -6.351 0.0994 .  
## startup3 1.250 0.433 2.887 0.2123   
## monthly1 -1.500 0.433 -3.464 0.1789   
## monthly2 1.500 0.433 3.464 0.1789   
## monthly3 2.000 0.433 4.619 0.1357   
## service1 -0.125 0.250 -0.500 0.7048   
## retail1 1.250 0.250 5.000 0.1257   
## apple1 2.375 0.250 9.500 0.0668 .  
## samsung1 1.875 0.250 7.500 0.0844 .  
## google1 -0.875 0.250 -3.500 0.1772   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1 on 1 degrees of freedom  
## Multiple R-squared: 0.9971, Adjusted R-squared: 0.9559   
## F-statistic: 24.21 on 14 and 1 DF, p-value: 0.1581

# save key list elements of the fitted model as needed for conjoint measures  
conjoint.results <-   
 main.effects.model.fit[c("contrasts","xlevels","coefficients")]  
  
conjoint.results$attributes <- names(conjoint.results$contrasts)  
  
# compute and store part-worths in the conjoint.results list structure  
part.worths <- conjoint.results$xlevels # list of same structure as xlevels  
end.index.for.coefficient <- 1 # intitialize skipping the intercept  
part.worth.vector <- NULL # used for accumulation of part worths  
for(index.for.attribute in seq(along=conjoint.results$contrasts)) {  
 nlevels <- length(unlist(conjoint.results$xlevels[index.for.attribute]))  
 begin.index.for.coefficient <- end.index.for.coefficient + 1  
 end.index.for.coefficient <- begin.index.for.coefficient + nlevels -2  
 last.part.worth <- -sum(conjoint.results$coefficients[  
 begin.index.for.coefficient:end.index.for.coefficient])  
 part.worths[index.for.attribute] <-   
 list(as.numeric(c(conjoint.results$coefficients[  
 begin.index.for.coefficient:end.index.for.coefficient],  
 last.part.worth)))  
 part.worth.vector <-   
 c(part.worth.vector,unlist(part.worths[index.for.attribute]))   
 }   
conjoint.results$part.worths <- part.worths  
  
# compute standardized part-worths  
standardize <- function(x) {(x - mean(x)) / sd(x)}  
conjoint.results$standardized.part.worths <-   
 lapply(conjoint.results$part.worths,standardize)  
   
# compute and store part-worth ranges for each attribute   
part.worth.ranges <- conjoint.results$contrasts  
for(index.for.attribute in seq(along=conjoint.results$contrasts))   
 part.worth.ranges[index.for.attribute] <-   
 dist(range(conjoint.results$part.worths[index.for.attribute]))  
conjoint.results$part.worth.ranges <- part.worth.ranges  
  
sum.part.worth.ranges <- sum(as.numeric(conjoint.results$part.worth.ranges))  
  
# compute and store importance values for each attribute   
attribute.importance <- conjoint.results$contrasts  
for(index.for.attribute in seq(along=conjoint.results$contrasts))   
 attribute.importance[index.for.attribute] <-   
 (dist(range(conjoint.results$part.worths[index.for.attribute]))/  
 sum.part.worth.ranges) \* 100  
conjoint.results$attribute.importance <- attribute.importance  
   
# data frame for ordering attribute names  
attribute.name <- names(conjoint.results$contrasts)  
attribute.importance <- as.numeric(attribute.importance)  
temp.frame <- data.frame(attribute.name,attribute.importance)  
conjoint.results$ordered.attributes <-   
 as.character(temp.frame[sort.list(  
 temp.frame$attribute.importance,decreasing = TRUE),"attribute.name"])  
  
# respondent internal consistency added to list structure  
conjoint.results$internal.consistency <- summary(main.effects.model.fit)$r.squared   
   
# user-defined function for printing conjoint measures  
if (print.digits == 2)   
 pretty.print <- function(x) {sprintf("%1.2f",round(x,digits = 2))}   
if (print.digits == 3)   
 pretty.print <- function(x) {sprintf("%1.3f",round(x,digits = 3))}   
   
# report conjoint measures to console   
# use pretty.print to provide nicely formated output  
for(k in seq(along=conjoint.results$ordered.attributes)) {  
 cat("\n","\n")  
 cat(conjoint.results$ordered.attributes[k],"Levels: ",  
 unlist(conjoint.results$xlevels[conjoint.results$ordered.attributes[k]]))  
   
 cat("\n"," Part-Worths: ")  
 cat(pretty.print(unlist(conjoint.results$part.worths  
 [conjoint.results$ordered.attributes[k]])))  
   
 cat("\n"," Standardized Part-Worths: ")  
 cat(pretty.print(unlist(conjoint.results$standardized.part.worths  
 [conjoint.results$ordered.attributes[k]])))   
   
 cat("\n"," Attribute Importance: ")  
 cat(pretty.print(unlist(conjoint.results$attribute.importance  
 [conjoint.results$ordered.attributes[k]])))  
 }

##   
##   
## startup Levels: $100 $200 $300 $400  
## Part-Worths: 3.00 -2.75 1.25 -1.50  
## Standardized Part-Worths: 1.15 -1.06 0.48 -0.58  
## Attribute Importance: 22.12  
##   
## apple Levels: APPLE NO APPLE YES  
## Part-Worths: 2.37 -2.37  
## Standardized Part-Worths: 0.71 -0.71  
## Attribute Importance: 18.27  
##   
## monthly Levels: $100 $200 $300 $400  
## Part-Worths: -1.50 1.50 2.00 -2.00  
## Standardized Part-Worths: -0.73 0.73 0.98 -0.98  
## Attribute Importance: 15.38  
##   
## samsung Levels: Samsung NO Samsung YES  
## Part-Worths: 1.87 -1.87  
## Standardized Part-Worths: 0.71 -0.71  
## Attribute Importance: 14.42  
##   
## brand Levels: AT&T T-mobile US Cellular Verizon  
## Part-Worths: 1.25 0.25 -2.00 0.50  
## Standardized Part-Worths: 0.89 0.18 -1.43 0.36  
## Attribute Importance: 12.50  
##   
## retail Levels: Retail NO Retail YES  
## Part-Worths: 1.25 -1.25  
## Standardized Part-Worths: 0.71 -0.71  
## Attribute Importance: 9.62  
##   
## google Levels: Nexus NO Nexus YES  
## Part-Worths: -0.87 0.87  
## Standardized Part-Worths: -0.71 0.71  
## Attribute Importance: 6.73  
##   
## service Levels: 4G NO 4G YES  
## Part-Worths: -0.12 0.12  
## Standardized Part-Worths: -0.71 0.71  
## Attribute Importance: 0.96

# plotting of spine chart begins here  
# all graphical output is routed to external pdf file  
pdf(file = here::here("project3/fig\_preference\_mobile\_services\_results.pdf"), width=8.5, height=11)  
spine.chart(conjoint.results)  
dev.off() # close the graphics output device

## png   
## 2

# Suggestions for the student:  
# Enter your own rankings for the product profiles and generate  
# conjoint measures of attribute importance and level part-worths.  
# Note that the model fit to the data is a linear main-effects model.  
# See if you can build a model with interaction effects for service  
# provider attributes.

# Part 2

Consider Exhibit 4.1 in page 50-51. Suppose your client is someone other than the local farmer, a meat producer/butcher, dairy, or brewer perhaps. Determine association rules relevant to your client’s products guided by the market basket model. What recommendations would you make about future marketplace actions?

## Analysis

We selected running the market basket analysis for a brewer, and opted to find rules related to “beer”.

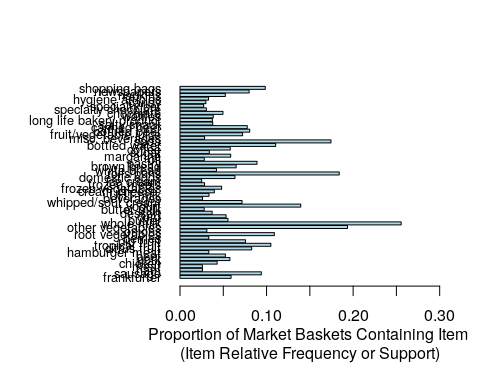
The results included 6 rules, the strongest of which was for non-alcoholic beer. That is to say, when a consumner purchases non-alcoholic beer, they will also purchase beer. The second highest rule is simply beer by itself. The remaining rules in order are bread and baked goods, vegetables, fruit, and dairy produce.

Given these rules, it makes sense that the brewer investigate producing a non-alcoholic beer of their own if they haven’t already. Given the high association of non-alcoholic beer with beer, consumer confidence in a non-alcoholic beer could lead to purchases of the same companies’ alcoholic beer as well, as “birds of a feather.” The remaining rules indicate purchases of items used for cookouts (bread/baked goods, veggies, fruit, and dairy produce). Dairy produce in this case could be cheeses, while bread and baked goods could related to hotdog and hamburger buns.

Additionally, it makes market-sense for the brewer to align their product with a “friendly” supplier of breads and dairy. Perhaps they might offer coupons or discounts to select grocery stores; when customers purchase a these associated products, they receive a discount on beer. As an alternative, purchasing beer of their brand could enable discounts on select breads and baked products.

Given the relationship of beer to these items, we must also inquire toward the seasonality of the data-set. This merits additional study to determine if decisions made now might only be good for a small window of time during the year.

# Association Rules for Market Basket Analysis (R)  
  
library(arules) # association rules  
library(arulesViz) # data visualization of association rules  
library(RColorBrewer) # color palettes for plots  
  
data(Groceries) # grocery transactions object from arules package  
  
# examine frequency for each item with support greater than 0.025  
itemFrequencyPlot(Groceries, support = 0.025, cex.names=0.8, xlim = c(0,0.3),  
 type = "relative", horiz = TRUE, col = "light blue", las = 1,  
 xlab = paste("Proportion of Market Baskets Containing Item",  
 "\n(Item Relative Frequency or Support)"))



# explore possibilities for combining similar items  
print(head(itemInfo(Groceries)))

## labels level2 level1  
## 1 frankfurter sausage meat and sausage  
## 2 sausage sausage meat and sausage  
## 3 liver loaf sausage meat and sausage  
## 4 ham sausage meat and sausage  
## 5 meat sausage meat and sausage  
## 6 finished products sausage meat and sausage

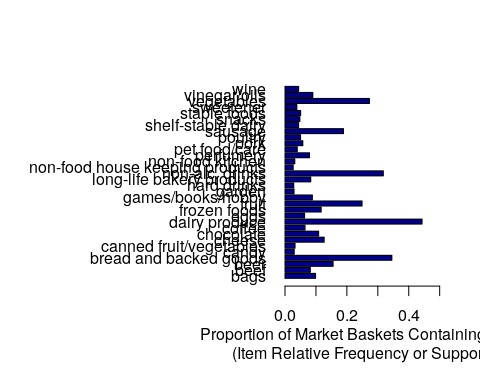
print(levels(itemInfo(Groceries)[["level1"]])) # 10 levels... too few

## [1] "canned food" "detergent" "drinks"   
## [4] "fresh products" "fruit and vegetables" "meat and sausage"   
## [7] "non-food" "perfumery" "processed food"   
## [10] "snacks and candies"

print(levels(itemInfo(Groceries)[["level2"]])) # 55 distinct levels

## [1] "baby food" "bags"   
## [3] "bakery improver" "bathroom cleaner"   
## [5] "beef" "beer"   
## [7] "bread and backed goods" "candy"   
## [9] "canned fish" "canned fruit/vegetables"   
## [11] "cheese" "chewing gum"   
## [13] "chocolate" "cleaner"   
## [15] "coffee" "condiments"   
## [17] "cosmetics" "dairy produce"   
## [19] "delicatessen" "dental care"   
## [21] "detergent/softener" "eggs"   
## [23] "fish" "frozen foods"   
## [25] "fruit" "games/books/hobby"   
## [27] "garden" "hair care"   
## [29] "hard drinks" "health food"   
## [31] "jam/sweet spreads" "long-life bakery products"   
## [33] "meat spreads" "non-alc. drinks"   
## [35] "non-food house keeping products" "non-food kitchen"   
## [37] "packaged fruit/vegetables" "perfumery"   
## [39] "personal hygiene" "pet food/care"   
## [41] "pork" "poultry"   
## [43] "pudding powder" "sausage"   
## [45] "seasonal products" "shelf-stable dairy"   
## [47] "snacks" "soap"   
## [49] "soups/sauces" "staple foods"   
## [51] "sweetener" "tea/cocoa drinks"   
## [53] "vegetables" "vinegar/oils"   
## [55] "wine"

# aggregate items using the 55 level2 levels for food categories  
# to create a more meaningful set of items  
groceries <- aggregate(Groceries, itemInfo(Groceries)[["level2"]])   
  
#print(dim(groceries)[1]) # 9835 market baskets for shopping trips  
#print(dim(groceries)[2]) # 55 final store items (categories)   
itemFrequencyPlot(groceries, support = 0.025, cex.names=1.0, xlim = c(0,0.5),  
 type = "relative", horiz = TRUE, col = "dark blue", las = 1,  
 xlab = paste("Proportion of Market Baskets Containing Item",  
 "\n(Item Relative Frequency or Support)"))



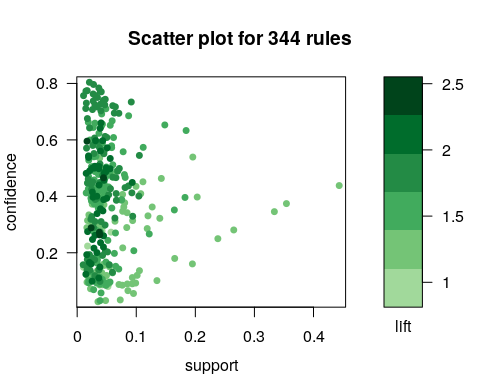
# obtain large set of association rules for items by category and all shoppers  
# this is done by setting very low criteria for support and confidence  
first.rules <- apriori(groceries, parameter = list(support = 0.001, confidence = 0.05))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.05 0.1 1 none FALSE TRUE 5 0.001 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: 9   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[55 item(s), 9835 transaction(s)] done [0.01s].  
## sorting and recoding items ... [54 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 5 6 7 8 done [0.02s].  
## writing ... [69921 rule(s)] done [0.01s].  
## creating S4 object ... done [0.01s].

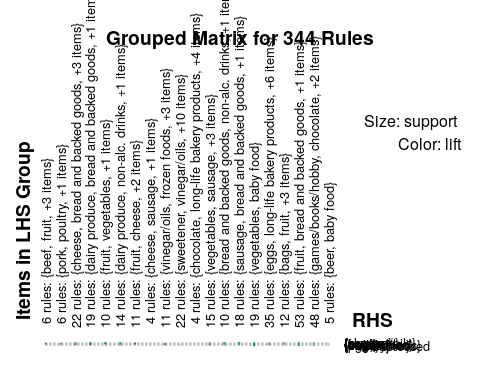
# select association rules using thresholds for support and confidence   
second.rules <- apriori(groceries, parameter = list(support = 0.025, confidence = 0.05))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.05 0.1 1 none FALSE TRUE 5 0.025 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: 245   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[55 item(s), 9835 transaction(s)] done [0.00s].  
## sorting and recoding items ... [32 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 done [0.00s].  
## writing ... [344 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

# data visualization of association rules in scatter plot  
plot(second.rules,   
 control=list(jitter=2,   
 col = rev(brewer.pal(9, "Greens")[4:9])),  
 shading = "lift")



# grouped matrix of rules is too clustered to show anything  
plot(second.rules, method="grouped",   
 control=list(col = rev(brewer.pal(9, "Greens")[4:9])))



# select rules with vegetables in consequent (right-hand-side) item subsets  
beer.rules <- subset(second.rules, subset = rhs %pin% "beer")  
inspect(beer.rules) # 6 rules

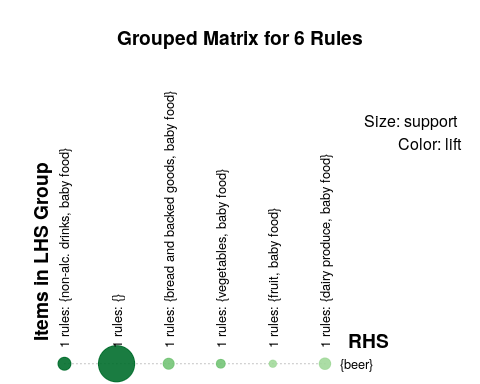
## lhs rhs support confidence lift   
## [1] {} => {beer} 0.15556685 0.1555669 1.0000000  
## [2] {fruit} => {beer} 0.02724962 0.1093878 0.7031559  
## [3] {non-alc. drinks} => {beer} 0.05236401 0.1646946 1.0586741  
## [4] {vegetables} => {beer} 0.03406202 0.1247672 0.8020168  
## [5] {bread and backed goods} => {beer} 0.04372140 0.1265450 0.8134447  
## [6] {dairy produce} => {beer} 0.04595831 0.1037411 0.6668587  
## count  
## [1] 1530   
## [2] 268   
## [3] 515   
## [4] 335   
## [5] 430   
## [6] 452

# sort by lift and identify the top 10 rules  
top.beer.rules <- head(sort(beer.rules, decreasing = TRUE, by = "lift"), 10)  
inspect(top.beer.rules)

## lhs rhs support confidence lift   
## [1] {non-alc. drinks} => {beer} 0.05236401 0.1646946 1.0586741  
## [2] {} => {beer} 0.15556685 0.1555669 1.0000000  
## [3] {bread and backed goods} => {beer} 0.04372140 0.1265450 0.8134447  
## [4] {vegetables} => {beer} 0.03406202 0.1247672 0.8020168  
## [5] {fruit} => {beer} 0.02724962 0.1093878 0.7031559  
## [6] {dairy produce} => {beer} 0.04595831 0.1037411 0.6668587  
## count  
## [1] 515   
## [2] 1530   
## [3] 430   
## [4] 335   
## [5] 268   
## [6] 452

Or visually:

plot(top.beer.rules, method="grouped",   
 control=list(col = rev(brewer.pal(9, "Greens")[4:9])))



plot(top.beer.rules, method="graph",   
 control=list(type="items"),   
 shading = "lift")

## Warning: Unknown control parameters: type

## Available control parameters (with default values):  
## main = Graph for 6 rules  
## nodeColors = c("#66CC6680", "#9999CC80")  
## nodeCol = c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF", "#EE0F0FFF", "#EE1212FF", "#EE1515FF", "#EE1818FF", "#EE1B1BFF", "#EE1E1EFF", "#EE2222FF", "#EE2525FF", "#EE2828FF", "#EE2B2BFF", "#EE2E2EFF", "#EE3131FF", "#EE3434FF", "#EE3737FF", "#EE3A3AFF", "#EE3D3DFF", "#EE4040FF", "#EE4444FF", "#EE4747FF", "#EE4A4AFF", "#EE4D4DFF", "#EE5050FF", "#EE5353FF", "#EE5656FF", "#EE5959FF", "#EE5C5CFF", "#EE5F5FFF", "#EE6262FF", "#EE6666FF", "#EE6969FF", "#EE6C6CFF", "#EE6F6FFF", "#EE7272FF", "#EE7575FF", "#EE7878FF", "#EE7B7BFF", "#EE7E7EFF", "#EE8181FF", "#EE8484FF", "#EE8888FF", "#EE8B8BFF", "#EE8E8EFF", "#EE9191FF", "#EE9494FF", "#EE9797FF", "#EE9999FF", "#EE9B9BFF", "#EE9D9DFF", "#EE9F9FFF", "#EEA0A0FF", "#EEA2A2FF", "#EEA4A4FF", "#EEA5A5FF", "#EEA7A7FF", "#EEA9A9FF", "#EEABABFF", "#EEACACFF", "#EEAEAEFF", "#EEB0B0FF", "#EEB1B1FF", "#EEB3B3FF", "#EEB5B5FF", "#EEB7B7FF", "#EEB8B8FF", "#EEBABAFF", "#EEBCBCFF", "#EEBDBDFF", "#EEBFBFFF", "#EEC1C1FF", "#EEC3C3FF", "#EEC4C4FF", "#EEC6C6FF", "#EEC8C8FF", "#EEC9C9FF", "#EECBCBFF", "#EECDCDFF", "#EECFCFFF", "#EED0D0FF", "#EED2D2FF", "#EED4D4FF", "#EED5D5FF", "#EED7D7FF", "#EED9D9FF", "#EEDBDBFF", "#EEDCDCFF", "#EEDEDEFF", "#EEE0E0FF", "#EEE1E1FF", "#EEE3E3FF", "#EEE5E5FF", "#EEE7E7FF", "#EEE8E8FF", "#EEEAEAFF", "#EEECECFF", "#EEEEEEFF")  
## edgeCol = c("#474747FF", "#494949FF", "#4B4B4BFF", "#4D4D4DFF", "#4F4F4FFF", "#515151FF", "#535353FF", "#555555FF", "#575757FF", "#595959FF", "#5B5B5BFF", "#5E5E5EFF", "#606060FF", "#626262FF", "#646464FF", "#666666FF", "#686868FF", "#6A6A6AFF", "#6C6C6CFF", "#6E6E6EFF", "#707070FF", "#727272FF", "#747474FF", "#767676FF", "#787878FF", "#7A7A7AFF", "#7C7C7CFF", "#7E7E7EFF", "#808080FF", "#828282FF", "#848484FF", "#868686FF", "#888888FF", "#8A8A8AFF", "#8C8C8CFF", "#8D8D8DFF", "#8F8F8FFF", "#919191FF", "#939393FF", "#959595FF", "#979797FF", "#999999FF", "#9A9A9AFF", "#9C9C9CFF", "#9E9E9EFF", "#A0A0A0FF", "#A2A2A2FF", "#A3A3A3FF", "#A5A5A5FF", "#A7A7A7FF", "#A9A9A9FF", "#AAAAAAFF", "#ACACACFF", "#AEAEAEFF", "#AFAFAFFF", "#B1B1B1FF", "#B3B3B3FF", "#B4B4B4FF", "#B6B6B6FF", "#B7B7B7FF", "#B9B9B9FF", "#BBBBBBFF", "#BCBCBCFF", "#BEBEBEFF", "#BFBFBFFF", "#C1C1C1FF", "#C2C2C2FF", "#C3C3C4FF", "#C5C5C5FF", "#C6C6C6FF", "#C8C8C8FF", "#C9C9C9FF", "#CACACAFF", "#CCCCCCFF", "#CDCDCDFF", "#CECECEFF", "#CFCFCFFF", "#D1D1D1FF", "#D2D2D2FF", "#D3D3D3FF", "#D4D4D4FF", "#D5D5D5FF", "#D6D6D6FF", "#D7D7D7FF", "#D8D8D8FF", "#D9D9D9FF", "#DADADAFF", "#DBDBDBFF", "#DCDCDCFF", "#DDDDDDFF", "#DEDEDEFF", "#DEDEDEFF", "#DFDFDFFF", "#E0E0E0FF", "#E0E0E0FF", "#E1E1E1FF", "#E1E1E1FF", "#E2E2E2FF", "#E2E2E2FF", "#E2E2E2FF")  
## alpha = 0.5  
## cex = 1  
## itemLabels = TRUE  
## labelCol = #000000B3  
## measureLabels = FALSE  
## precision = 3  
## layout = NULL  
## layoutParams = list()  
## arrowSize = 0.5  
## engine = igraph  
## plot = TRUE  
## plot\_options = list()  
## max = 100  
## verbose = FALSE

