WEEK 13: FSQCA IN R

THOMAS ELLIOTT

This week we'll see how to run qualitative comparative analysis (QCA) in R. While Charles Ragin provides a program on his website for running QCA, it is not able to do so algorithmically, meaning you have to manually run each analysis. QCA in R is not as easy to do as with Ragin's software, but it does allow for running QCA algorithmically (which can be incredibly useful when trying out various calibrations and cut-offs as robustness checks).

Note that I am assuming some basic knowledge of QCA throughout this hand out.

QCA is an R package for running various types of QCA. To install, type:

```
install.packages("QCA")
```

Once installed, type the following to load the package into memory:

```
library(QCA)
```

You only have to install the package once, but you will need to load the package into memory each time you start R new. QCA comes with several functions, but we will focus on three this week.

1. Calibrating Data

The first function we will use is the `calibrate` function. This allows us to take variables and convert them to membership scores for causal and outcome sets. We will use the `mtcars` data set for our examples.

```
> data("mtcars")
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Let's create a fuzzy set based on the cars' mpg:

```
mtcars$mpg.fuzzy<-calibrate(mtcars$mpg,type="fuzzy",thresholds = c(10,20,30))
> head(mtcars[,c("mpg","mpg.fuzzy")])
                   mpg mpg.fuzzy
Mazda RX4         21.0 0.5730837
Mazda RX4 Wag     21.0 0.5730837
Datsun 710        22.8 0.6951786
Hornet 4 Drive    21.4 0.6016204
```

Date: April 11, 2016.

```
Hornet Sportabout 18.7 0.4054573
Valiant           18.1 0.3636763
```

Above we pass three arguments to the calibrate function. The first is a vector of values to be transformed, the second is the type of set (fuzzy or crisp) we are creating. The third is the thresholds we are using to transform the data. Thresholds should be passed in the order of lower, cross-over, upper. The calibrate function returns a vector of the same length as the one we passed, but with values transformed to membership scores in the set. The `calibrate` function will accept many more arguments to customize how it calculates membership scores, but the above example is probably the most common use of the calibrate function.

Note that the `calibrate` function included in the QCA package does not exactly reproduce the values from a direct transformation based on the logistic function as described in Ragin's Redesigning Social Inquiry, though it gets very close (within 3-4 decimal points) and likely will not produce substantively different results. If you would like a function that reproduces the direct transformation method, you can install the `myQCA` package I wrote, which contains the `fsCalibrate` function. Assuming you have the `devtools` package installed (which comes standard with RStudio):

```
devtools::install_github("thom82/myQCA")
```

2. The Truth Table

QCA includes the `truthTable` function for creating truth tables. The truthTable contains a lot of optional arguments to customize the type of QCA we want to do. For the following examples, we will use the data set used in Cress and Snow's 2000 AJS article about homeless mobilization:

```
> data(d.homeless)
> head(d.homeless)
     VI DT SA CS DF PF REP RES RIG REL
PUH   1  1  1  1  1  1   1   1   1   1
AOS   1  0  1  1  1  1   1   1   1   1
OUH   1  1  1  0  1  1   1   1   1   1
TUH   1  1  1  0  1  1   1   0   1   1
PUEJ  1  1  1  1  1  1   0   0   1   1
DTUH  1  1  1  0  1  1   1   0   1   0
```

The `truthTable` function takes a few different arguments. First, a data frame containing the data for your analysis. Second, the name of the column containing the outcome set. Third, a character vector containing the column names of the condition (or causal) sets. You can leave the conditions argument out and by default the function will use all columns other than the outcome as condition sets. Fourth, `incl.cut1` sets the consistency cut-off for inclusion. You can also set `incl.cut0` for the consistency cut-off for exclusion, and rows in between will be coded as contradictions. By default, the function sets `incl.cut0` to the same value as `incl.cut1` so there are no contradictory rows. Finally, I like to have the truth table sorted by the rows' consistency scores, so I pass the `sort.by="incl"` argument to do this. Below is the truth table for the homeless data with representation as the outcome:

```
> truth<-truthTable(d.homeless,outcome="REP",
+                   conditions=c("VI","DT","SA","CS","DF","PF"),
+                   incl.cut1=0.8,
+                   sort.by="incl")
> truth
```

```
  OUT: outcome value
    n: number of cases in configuration
 incl: sufficiency inclusion score

    VI DT SA CS DF PF OUT n  incl  PRI
40  1  0  0  1  1  1   1  1  1.000 1.000
48  1  0  1  1  1  1   1  1  1.000 1.000
60  1  1  1  0  1  1   1  3  1.000 1.000
64  1  1  1  1  1  1   0  2  0.500 0.500
 1  0  0  0  0  0  0   0  2  0.000 0.000
 2  0  0  0  0  0  1   0  2  0.000 0.000
13  0  0  1  1  0  0   0  1  0.000 0.000
18  0  1  0  0  0  1   0  1  0.000 0.000
21  0  1  0  1  0  0   0  1  0.000 0.000
22  0  1  0  1  0  1   0  1  0.000 0.000
```

One other argument that we don't include here, but that will be useful for larger data sets is the `n.cut` argument, which by default is set to 1, meaning rows with less than 1 case will be coded as remainder rows. You can set this higher to code rows with few cases as remainders.

3. Computing Solutions

The minimizing function in the QCA package is called `eqmcc`. We could pass it the data frame, along with the same parameters we included in the `truthTable` function, but as a short cut we can also just pass the truth table object.

```
> eqmcc(truth,details=TRUE)

n OUT = 1/0/C: 5/10/0
  Total      : 15

M1: VI*dt*CS*DF*PF + VI*DT*SA*cs*DF*PF <=> REP

                      incl   PRI    cov.r  cov.u
-----------------------------------------------
1  VI*dt*CS*DF*PF     1.000  1.000  0.333  0.333
2  VI*DT*SA*cs*DF*PF  1.000  1.000  0.500  0.500
-----------------------------------------------
   M1                 1.000  1.000  0.833
```

You have to include the `details=TRUE` parameter to get the coverage and consistency scores for the solution. This is the complex solution, meaning no simplifying assumptions are being made to produce the solution. The `incl` column contains the consistency score for the row, the `cov.r` contains the raw row coverage and `cov.u` contains the unique row coverage. The last line contains the total consistency and coverage scores for the solution.

We can get the parsimonious solution of the above data. The `include` parameter tells the function to include additional rows based on the coded outcome. The question mark tells the function to include the remainder rows.

```
> eqmcc(truth,details=TRUE,include="?")
```

```
n OUT = 1/0/C: 5/10/0
  Total      : 15

M1: cs*DF + dt*DF <=> REP
M2: cs*DF + VI*dt <=> REP
M3: cs*DF + dt*CS*PF <=> REP
M4: dt*DF + SA*cs <=> REP
M5: dt*DF + VI*cs <=> REP
M6: SA*cs + VI*dt <=> REP
M7: SA*cs + dt*CS*PF <=> REP
M8: VI*cs + VI*dt <=> REP
M9: VI*cs + dt*CS*PF <=> REP
```

		incl	PRI	cov.r	cov.u	(M1)	(M2)	(M3)	(M4)	(M5)	(M6)	(M7)	(M8)	(M9)
1	cs*DF	1.000	1.000	0.500	0.000	0.500	0.500	0.500						
2	dt*DF	1.000	1.000	0.333	0.000	0.333			0.333	0.333				
3	SA*cs	1.000	1.000	0.500	0.000				0.500		0.500	0.500		
4	VI*cs	1.000	1.000	0.500	0.000					0.500			0.500	0.500
5	VI*dt	1.000	1.000	0.333	0.000		0.333				0.333		0.333	
6	dt*CS*PF	1.000	1.000	0.333	0.000			0.333				0.333		0.333
	M1	1.000	1.000	0.833										
	M2	1.000	1.000	0.833										
	M3	1.000	1.000	0.833										
	M4	1.000	1.000	0.833										
	M5	1.000	1.000	0.833										
	M6	1.000	1.000	0.833										
	M7	1.000	1.000	0.833										
	M8	1.000	1.000	0.833										
	M9	1.000	1.000	0.833										

Above illustrates a drawback of the QCA package in R: when there are multiple, redundant prime implicants, where in Charles' software you would be asked to choose which ones to include, the QCA package in R displays all possible results in a way that is not very intuitive. In the above table, to choose a solution look at the various M columns - each one represents a possible solution, containing the individual recipes for which coverage scores are supplied. For example, solution M1 contains the first and second recipes, while solution M2 contains the first and fifth recipes.

To get the intermediate solution, we include remainder rows, but also supply directional expectations. The argument `dir.exp` takes a vector of the same length as `conditions`, where each element can be either 0, 1, or "-". 0 indicates the absence of the condition is expected to lead to the outcome, 1 indicates the presence of the condition is expected to lead to the outcome, and "-" indicates either the presence or absence is expected to lead to the outcome.

```
> eqmcc(truth,details=TRUE,include="?",dir.exp=c(1,1,1,1,1,1))

n OUT = 1/0/C: 5/10/0
  Total      : 15

p.sol: cs*DF + dt*DF

M1:    VI*dt*CS*DF*PF + VI*DT*SA*cs*DF*PF <=> REP
```

```
                      incl   PRI    cov.r  cov.u
-------------------------------------------------
1  VI*dt*CS*DF*PF     1.000  1.000  0.333  0.333
2  VI*DT*SA*cs*DF*PF  1.000  1.000  0.500  0.500
-------------------------------------------------
   M1                 1.000  1.000  0.833
```

Note that R outputs the same solution multiple times. This is another area where multiple redundant prime implicants produces weird output. All are valid solutions.