On the usage of the `gRim` package

Søren Højsgaard

`gRim` version 0.2.1 as of 2020-02-21

Contents

1   Introduction

The `gRim` package is an `R` package for *gRaphical interaction models* (hence the name). `gRim` implements 1) graphical log–linear models for discrete data, that is for contingency tables and 2) Gaussian graphical models for continuous data (multivariate normal data) and 3) mixed homogeneous interaction models for mixed data (data consisiting of both discrete and continuous variables).

The package is at an early stage of development and so is this document.

2   Introductory examples

The main functions for creating models of the various types are:

- Discrete data: The *dmod()* function creates a hierarchical log–linear model.

- Continuous data: The *cmod()* function creates a Gaussian graphical model.

- Mixed data: The *mmod()* function creates a mixed interaction model.

The arguments to the model functions are:

```
> args(dmod)

function (formula, data, marginal = NULL, interactions = NULL,
    fit = TRUE, details = 0, ...)
NULL

> args(cmod)

function (formula, data, marginal = NULL, fit = TRUE, details = 0)
NULL

> args(mmod)

function (formula, data, marginal = NULL, fit = TRUE, details = 0)
NULL
```

The model objects created by these functions are of the respective classes `dModel`, `cModel` and `mModel`. All models are also of the class `iModel`. We focus the presentation on models for discrete data, but most of the topics we discuss apply to all types of models.

2.1   A Discrete Model

The *reinis* data from `gRbase` is a 2^6 contingency table.

```
> data(reinis)
> str(reinis)

 'table' num [1:2, 1:2, 1:2, 1:2, 1:2, 1:2] 44 40 112 67 129 145 12 23 35 12 ...
 - attr(*, "dimnames")=List of 6
  ..$ smoke  : chr [1:2] "y" "n"
  ..$ mental : chr [1:2] "y" "n"
  ..$ phys   : chr [1:2] "y" "n"
  ..$ systol : chr [1:2] "y" "n"
  ..$ protein: chr [1:2] "y" "n"
  ..$ family : chr [1:2] "y" "n"
```

Models are specified as generating classes. A generating class can be a list or a right–hand–sided formula. In addition, various model specification shortcuts are available. Some of these are described in Section 2.2.

The following two specifications of a log–linear model are equivalent:

```
> data(reinis)
> dm1<-dmod(list(c("smoke","systol"),c("smoke","mental","phys")), data=reinis)
> dm1<-dmod(~smoke:systol + smoke:mental:phys, data=reinis)
> dm1

$call
dmod(formula = ~smoke:systol + smoke:mental:phys, data = reinis)

$glist
$glist[[1]]
[1] "smoke"  "systol"

$glist[[2]]
[1] "smoke"  "mental" "phys"


$varNames
[1] "smoke"  "systol" "mental" "phys"

$datainfo
$datainfo$data
, , mental = y, phys = y

     systol
smoke   y    n
    y  79   67
    n  82   40

, , mental = n, phys = y

     systol
smoke   y    n
    y 217  177
    n 156  109

, , mental = y, phys = n

     systol
smoke   y    n
    y 197  179
    n 258  161

, , mental = n, phys = n

     systol
smoke   y    n
    y  22   23
    n  43   31
```

4

The output reads as follows: `-2logL` is minus twice the maximized log–likelihood and `mdim` is the number of parameters in the model (no adjustments have been made for sparsity of data). The `ideviance` and `idf` gives the deviance and degrees of freedom between the model and the independence model for the same variables and `deviance` and `df` is the deviance and degrees of freedom between the model and the saturated model for the same variables.

Section 8.1 describes model objects in more detail. Here we just notice that the generating class of the model is contained in the slot `glist`:

Notice that the generating class does not appear directly. However the generating class can be retrieved using *formula()* and *terms()*:

```
> formula(dm1)

~smoke * systol + smoke * mental * phys

> str(terms(dm1))

List of 2
 $ : chr [1:2] "smoke" "systol"
 $ : chr [1:3] "smoke" "mental" "phys"
```

A summary of a model is provided by the *summary()* function:

```
> summary(dm1)

           Length Class  Mode
call       3      -none- call
glist      2      -none- list
varNames   4      -none- character
datainfo   1      -none- list
fitinfo    9      -none- list
isFitted   1      -none- logical
glistNUM   2      -none- list
properties 2      -none- logical
```

1

2.2 Model specification shortcuts

Below we illustrate various other ways of specifying log–linear models.

---

[1]The summary() method leaves a bit to be desired...

- A saturated model can be specified using `~.^.` whereas `~.^2` specifies the model with all–two–factor interactions. Using `~.^1` specifies the independence model.

- If we want, say, at most two–factor interactions in the model we can use the `interactions` argument.

- Attention can be restricted to a subset of the variables using the `marginal` argument.

- Variable names can be abbreviated.

The following models illustrate these abbreviations:

```
> dm2 <- dmod(~.^2, margin=c("smo","men","phy","sys"),
+             data=reinis)
> formula(dm2)

~smoke * mental + smoke * phys + mental * phys + smoke * systol +
    mental * systol + phys * systol
```

```
> dm3 <- dmod(list(c("smoke", "systol"), c("smoke", "mental", "phys")),
+             data=reinis, interactions=2)
> formula(dm3)

~smoke * systol + smoke * mental + smoke * phys + mental * phys
```

2.3   Plotting models

There are two methods for plotting the dependence graph of a model: Using *iplot()* and *plot()*. The convention for both methods is that discrete variables are drawn as grey dots and continuous variables as white dots. 1) *iplot()* creates an `igraph` object and plots this. 2) 2) *plot()* creates a `graphNEL` object and plots this.

```
> iplot(dm1)
```
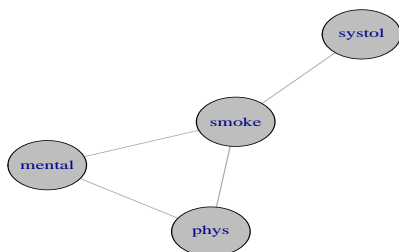
2.4   A Continuous Model

For Gaussian models there are at most second order interactions. Hence we may specify the saturated model in different ways:

```
> data(carcass)
> cm1 <- cmod(~Fat11:Fat12:Fat13, data=carcass)
> cm1 <- cmod(~Fat11:Fat12 + Fat12:Fat13 + Fat11:Fat13, data=carcass)
> cm1

$glist
$glist[[1]]
[1] "Fat11" "Fat12" "Fat13"


$varNames
[1] "Fat11" "Fat12" "Fat13"

$datainfo
$datainfo$S
       Fat11 Fat12 Fat13
Fat11 11.343 8.421 7.664
Fat12  8.421 8.911 6.844
Fat13  7.664 6.844 7.623

$datainfo$n.obs
[1] 344

$datainfo$data
    Fat11 Meat11 Fat12 Meat12 Fat13 Meat13 LeanMeat
1      17     51    12     51    12     61    56.52
2      17     49    15     48    15     54    57.58
3      14     38    11     34    11     40    55.89
4      17     58    12     58    11     58    61.82
5      14     51    12     48    13     54    62.96
6      20     40    14     40    14     45    54.58
7      12     55     8     56     7     65    67.09
8      22     57    19     52    16     56    55.00
9      16     52    12     54    14     59    60.49
10     18     62    16     61    13     64    58.80
11     13     53    13     51    11     56    61.53
12     14     61    13     57    14     60    62.82
13     12     41    14     40    12     47    63.09
14     15     65    14     73    11     79    58.33
15     13     55    12     57     9     62    61.77
16     15     43    15     45    11     52    58.79
17     16     40    14     32    14     37    56.30
18     12     52    10     52    10     56    62.16
19     18     58    16     60    14     63    60.46
20     14     49    13     44    13     50    62.09
21     13     49    11     49    10     54    62.14
22     20     49    19     51    16     56    49.97
23     16     62    13     65    13    8 69    65.40
24     21     47    18     47    15     52    57.16
25     18     49    14     47    14     52    53.30
```
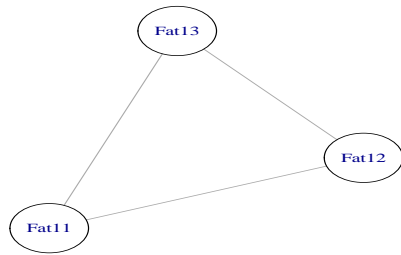
```
> iplot(cm1)
```

---

[2] Harmonize cmod() output with that of dmod()

2.5 A Mixed Model

```
> data(milkcomp1)
> mm1 <- mmod(~.^., data=milkcomp1)
> mm1

$glist
$glist[[1]]
[1] "treat"   "fat"     "protein" "dm"      "lactose"


$varNames
[1] "treat"   "fat"     "protein" "dm"      "lactose"

$datainfo
$datainfo$data
   treat   fat protein    dm lactose
1      d  6.16    6.65 18.55    5.06
2      c  4.06    5.44 18.32    5.23
3      f  9.25    5.67 20.68    5.15
4      b  5.82    5.62 17.57    5.74
5      a  4.98    5.37 16.38    5.55
6      b  9.06    5.08 20.21    5.29
7      e  5.11    5.16 16.32    5.53
8      a  7.00    5.33 18.82    5.45
9      f  6.90    5.73 18.81    5.02
10     f  7.77    5.38 19.51    5.27
11     a  5.96    5.57 17.71    5.62
12     e  8.17    4.85 19.48    5.47
13     c  8.32    5.41 19.85    5.35
14     g  7.42    6.01 19.47    5.26
15     e  7.16    4.85 18.31    5.30
16     d  8.33    5.98 20.85    5.42
17     d  5.97    5.34 17.29    5.42
18     a  9.22    5.73 21.52    5.47
19     a  8.74    5.97 20.85    5.31
20     b  9.52    4.99 20.57    5.28
21     e  5.45    5.07 16.78    5.25
22     g  6.92    5.40 18.36    5.54
23     c  5.22    5.31 16.56    5.55
24     a  4.91    4.92 16.40    5.64
25     g  5.92    5.25 17.13    5.63
26     d  5.37    5.06 16.53    5.76
27     g  8.49    6.15 20.58    5.23
28     g  5.80    5.24 17.10    5.54
29     c  8.07    5.24 19.57    5.65
30     c  6.46    5.74 18.45    5.72
31     g  8.61    5.49 20.32    5.63    10
32     c  8.77    5.64 20.31    5.22
33     b  7.59    5.13 18.80    5.52
```

```
> iplot(mm1)
```
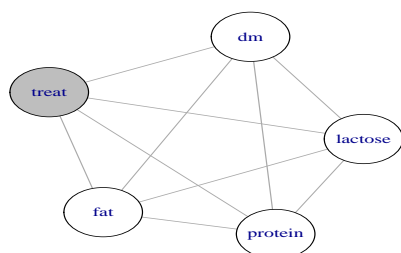

3 Model editing - `update()`

The [update()]() function enables `dModel` objects to be modified by the addition or deletion of interaction terms or edges, using the arguments `aterm`, `dterm`, `aedge` or `dedge`. Some examples follow:

- Set a marginal saturated model:

```
> ms <- dmod(~.^., marginal=c("phys","mental","systol","family"), data=reinis)
> formula(ms)

~phys * mental * systol * family
```

- Delete one edge:

```
> ms1 <- update(ms, list(dedge=~phys:mental))
> formula(ms1)

~phys * systol * family + mental * systol * family
```

- Delete two edges:

```
> ms2<- update(ms, list(dedge=~phys:mental+systol:family))
> formula(ms2)

~phys * systol + phys * family + mental * systol + mental * family
```

---

[3]Harmonize mmod() output with that of dmod()

- Delete all edges in a set:

```
> ms3 <- update(ms, list(dedge=~phys:mental:systol))
> formula(ms3)

~phys * family + mental * family + systol * family
```

- Delete an interaction term

```
> ms4 <- update(ms, list(dterm=~phys:mental:systol) )
> formula(ms4)

~phys * mental * family + phys * systol * family + mental * systol *
    family
```

- Add three interaction terms:

```
> ms5 <- update(ms, list(aterm=~phys:mental+phys:systol+mental:systol) )
> formula(ms5)

~phys * mental * systol * family
```

- Add two edges:

```
> ms6 <- update(ms, list(aedge=~phys:mental+systol:family))
> formula(ms6)

~phys * mental * systol * family
```

A brief explanation of these operations may be helpful. To obtain a hierarchical model when we delete a term from a model, we must delete any higher-order relatives to the term. Similarly, when we add an interaction term we must also add all lower-order relatives that were not already present. Deletion of an edge is equivalent to deleting the corresponding two-factor term. Let $m - e$ be the result of deleting edge e from a model m. Then the result of adding e is defined as the maximal model m^* for which $m^* - e = m$.

4   Testing for conditional independence

Tests of general conditional independence hypotheses of the form $u \perp\!\!\!\perp v \,|\, W$ can be performed using the _ciTest()_ function.

```
> cit <- ciTest(reinis, set=c("systol","smoke","family","phys"))

List of 4
 $ t.uR: num [1:2, 1:2, 1:2] 454 339 80 54 459 329 61 65
  ..- attr(*, "dimnames")=List of 3
  .. ..$ systol: chr [1:2] "y" "n"
  .. ..$ family: chr [1:2] "y" "n"
  .. ..$ phys  : chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2, 1:2] 461 332 79 55 372 416 49 77
  ..- attr(*, "dimnames")=List of 3
  .. ..$ smoke : chr [1:2] "y" "n"
  .. ..$ family: chr [1:2] "y" "n"
  .. ..$ phys  : chr [1:2] "y" "n"
 $ R   : chr [1:2] "family" "phys"
 $ vn  : chr [1:4] "systol" "smoke" "family" "phys"
, , family = y, phys = y

      smoke
systol     y n
     y 263.9 0
     n 141.9 0

, , family = n, phys = y

      smoke
systol     y n
     y 47.16 0
     n 22.16 0

, , family = y, phys = n

      smoke
systol     y n
     y 216.7 0
     n 173.7 0

, , family = n, phys = n

      smoke
systol     y n
     y 23.72 0
     n 39.72 0


Testing systol _|_ smoke | family phys
Statistic (DEV):  104.540 df: 4 p-value: 0.0000 method: CHISQ
Slice information:
  statistic   p.value df family phys
1    141.15 0.000e+00  1      y    y   13
2     23.18 1.478e-06  1      n    y
3    -35.08 1.000e+00  1      y    n
```

The general syntax of the `set` argument is of the form (u, v, W) where u and v are variables and W is a set of variables. The `set` argument can also be given as a right–hand sided formula.

In model terms, the test performed by *ciTest()* corresponds to the test for removing the edge $\{u, v\}$ from the saturated model with variables $\{u, v\} \cup W$. If we (conceptually) form a factor S by crossing the factors in W, we see that the test can be formulated as a test of the conditional independence $u \perp\!\!\!\perp v \mid S$ in a three way table. The deviance decomposes into independent contributions from each stratum:

$$
\begin{aligned}
D &= 2 \sum_{ijs} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} \\
&= \sum_s 2 \sum_{ij} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} = \sum_s D_s
\end{aligned}
$$

where the contribution D_s from the sth slice is the deviance for the independence model of u and v in that slice. For example,

```
> cit$slice

  statistic   p.value df family phys
1    141.15 0.000e+00  1      y    y
2     23.18 1.478e-06  1      n    y
3    -35.08 1.000e+00  1      y    n
4    -24.70 1.000e+00  1      n    n
```

The sth slice is a `u` $\times$ `v` table $\{n_{ijs}\}_{i=1...\mathtt{u}, j=1...\mathtt{v}}$. The number of degrees of freedom corresponding to the test for independence in this slice is

$$
df_s = (\#\{i : n_{i\cdot s} > 0\} - 1)(\#\{j : n_{\cdot js} > 0\} - 1)
$$

where $n_{i\cdot s}$ and $n_{\cdot js}$ are the marginal totals.

An alternative to the asymptotic χ^2 test is to determine the reference distribution using Monte Carlo methods. The marginal totals are sufficient statistics under the null hypothesis, and in a conditional test the test statistic is evaluated in the conditional distribution given the sufficient statistics. Hence one can generate all possible tables with those given margins, calculate the desired test statistic for each of these tables and then see how extreme the observed test statistic is relative to those of the calculated tables. A Monte Carlo approximation to this procedure is to randomly generate large number of tables with the given margins, evaluate the statistic for each simulated table and then see how extreme the observed test statistic is in this distribution. This is called a *Monte Carlo exact test* and it provides a *Monte Carlo p–value*:

```
> ciTest(reinis, set=c("systol","smoke","family","phys"), method='MC')

Testing systol _|_ smoke | family phys
Statistic (DEV):   13.045 df: NA p-value: 0.0110 method: MC
Slice information:
  statistic n.extreme p.value df family phys
1  4.734420          67  0.0335 NA       y    y
2  0.003456        1678  0.8390 NA       n    y
3  7.314160          16  0.0080 NA       y    n
4  0.993337         534  0.2670 NA       n    n
```

5 Fundamental methods for inference

This section describes some fundamental methods for inference in `gRim`. As basis for the description consider the following model shown in Fig. 1:

```
> dm5 <- dmod(~ment:phys:systol+ment:systol:family+phys:systol:smoke,
+            data=reinis)

$call
dmod(formula = ~ment:phys:systol + ment:systol:family + phys:systol:smoke,
    data = reinis)

$glist
$glist[[1]]
[1] "mental" "phys"   "systol"

$glist[[2]]
[1] "mental" "systol" "family"

$glist[[3]]
[1] "phys"   "systol" "smoke"


$varNames
[1] "mental" "phys"   "systol" "family" "smoke"

$datainfo
$datainfo$data
, , systol = y, family = y, smoke = y

       phys
mental    y    n
     y   67  179
     n  182   19

, , systol = n, family = y, smoke = y

       phys
mental    y    n
     y   59  160
     n  153   14

, , systol = y, family = n, smoke = y

       phys
mental    y    n
     y   12   18
     n   35    3

, , systol = n, family = n, smoke = y

       phys
mental    y    n
     y    8   19
     n   24    9
```
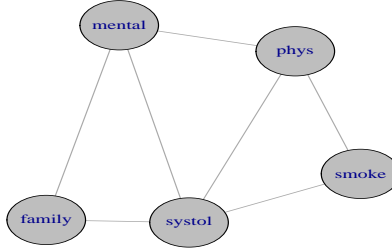16

Figure 1: A decomposable graphical model for the `reinis` data.

5.1 Testing for addition and deletion of edges

Let $\mathcal{M}_0$ be a model and let $e = \{u, v\}$ be an edge in $\mathcal{M}_0$. The candidate model formed by deleting e from $\mathcal{M}_0$ is $\mathcal{M}_1$. The `testdelete()` function can be used to test for deletion of an edge from a model:

```
> testdelete(dm5, ~smoke:systol)

List of 4
 $ t.uR: num [1:2, 1:2] 540 387 421 493
  ..- attr(*, "dimnames")=List of 2
  .. ..$ smoke: chr [1:2] "y" "n"
  .. ..$ phys : chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2] 534 393 520 394
  ..- attr(*, "dimnames")=List of 2
  .. ..$ systol: chr [1:2] "y" "n"
  .. ..$ phys  : chr [1:2] "y" "n"
 $ R   : chr "phys"
 $ vn  : chr [1:3] "smoke" "systol" "phys"
, , phys = y

     systol
smoke     y n
    y 311.1 0
    n 164.1 0

, , phys = n

     systol
smoke     y n
    y 239.5 0
    n 212.5 0

$statistic
[1] 318

$p.value
[1] 0

$df
[1] 2

$statname
[1] "DEV"

$method
[1] "CHISQ"

$adjust.df
[1] TRUE

$varNames
[1] "smoke"  "systol" "phys"
```
```
$slice
NULL
```

In the first case the p–value suggests that the edge can not be deleted. In the second case the p–value suggests that the edge can be deleted. The reported AIC value is the difference in AIC between the candidate model and the original model. A negative value of AIC suggest that the candidate model is to be preferred.

Next, let $\mathcal{M}_0$ be a model and let $e = \{u, v\}$ be an edge not in $\mathcal{M}_0$. The candidate model formed by adding e to $\mathcal{M}_0$ is denoted $\mathcal{M}_1$. The `testadd()` function can be used to test for deletion of an edge from a model:

```
> testadd(dm5, ~smoke:mental)

List of 4
 $ t.uR: num [1:2, 1:2, 1:2] 296 238 244 149 219 301 202 192
  ..- attr(*, "dimnames")=List of 3
  .. ..$ smoke : chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
  .. ..$ phys  : chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2, 1:2] 161 373 107 286 455 65 340 54
  ..- attr(*, "dimnames")=List of 3
  .. ..$ mental: chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
  .. ..$ phys  : chr [1:2] "y" "n"
 $ R    : chr [1:2] "systol" "phys"
 $ vn   : chr [1:4] "smoke" "mental" "systol" "phys"
, , systol = y, phys = y

     mental
smoke      y n
    y  89.24 0
    n 166.24 0

, , systol = n, phys = y

     mental
smoke      y n
    y  66.43 0
    n 108.43 0

, , systol = y, phys = n

     mental
smoke      y n
    y 191.62 0
    n  37.62 0

, , systol = n, phys = n

     mental
smoke      y n
    y 174.31 0
    n  26.31 0

$statistic
[1] 1383

$p.value
[1] 0

$df
```

The p–value suggests that no significant improvedment of the model is obtained by adding the edge. The reported AIC value is the difference in AIC between the candidate model and the original model. A negative value of AIC would have suggested that the candidate model is to be preferred.

4

5.2   Finding edges

The `getInEdges()` function will return a list of all the edges in the dependency graph $\mathcal{G}$ defined by the model. If we set `type='decomposable'` then the edges returned are as follows: An edge $e = \{u, v\}$ is returned if $\mathcal{G}$ minus the edge e is decomposable. In connection with model selection this is convenient because it is thereby possibly to restrict the search to decomposable models.

```
> ed.in <- getInEdges(ugList(dm5$glist), type="decomposable")

      [,1]      [,2]
[1,] "mental" "phys"
[2,] "mental" "family"
[3,] "phys"   "smoke"
[4,] "systol" "family"
[5,] "systol" "smoke"
```

The `getOutEdges()` function will return a list of all the edges which are not in the dependency graph $\mathcal{G}$ defined by the model. If we set `type='decomposable'` then the edges returned are as follows: An edge $e = \{u, v\}$ is returned if $\mathcal{G}$ plus the edge e is decomposable. In connection with model selection this is convenient because it is thereby possibly to restrict the search to decomposable models.

```
> ed.out <- getOutEdges(ugList(dm5$glist), type="decomposable")

      [,1]      [,2]
[1,] "mental" "smoke"
[2,] "phys"   "family"
```

---

[4]A function for testing addition / deletion of more general terms is needed.

5.3   Testing several edges

```
> args(testInEdges)

function (object, edgeMAT = NULL, criterion = "aic", k = 2, alpha = NULL,
    headlong = FALSE, details = 1, ...)
NULL

> args(testOutEdges)

function (object, edgeMAT = NULL, criterion = "aic", k = 2, alpha = NULL,
    headlong = FALSE, details = 1, ...)
NULL
```

The functions `labelInEdges()` and `labelOutEdges()` will test for deletion of edges and addition of edges. The default is to use AIC for evaluating each edge. It is possible to specify the penalty parameter for AIC to being other values than 2 and it is possible to base the evaluation on significance tests instead of AIC. Setting `headlong=TRUE` causes the function to exit once an improvement is found. For example:

```
> testInEdges(dm5, getInEdges(ugList(dm5$glist), type="decomposable"),
+              k=log(sum(reinis)))

List of 4
 $ t.uR: num [1:2, 1:2] 616 438 447 340
  ..- attr(*, "dimnames")=List of 2
  .. ..$ mental: chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2] 534 520 393 394
  ..- attr(*, "dimnames")=List of 2
  .. ..$ phys : chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
 $ R   : chr "systol"
 $ vn  : chr [1:3] "mental" "phys" "systol"
, , systol = y

      phys
mental     y n
     y 312.1 0
     n 216.1 0

, , systol = n

      phys
mental     y n
     y 223.2 0
     n 170.2 0

List of 4
 $ t.uR: num [1:2, 1:2] 616 438 447 340
  ..- attr(*, "dimnames")=List of 2
  .. ..$ mental: chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2] 913 141 668 119
  ..- attr(*, "dimnames")=List of 2
  .. ..$ family: chr [1:2] "y" "n"
  .. ..$ systol: chr [1:2] "y" "n"
 $ R   : chr "systol"
 $ vn  : chr [1:3] "mental" "family" "systol"
, , systol = y

      family
mental      y n
     y 533.59 0
     n  58.59 0

, , systol = n

      family
mental      y n
```

23

6 Stepwise model selection

Two functions are currently available for model selection: `backward()` and `forward()`. These functions employ the functions in Section 5.3)

6.1 Backward search

For example, we start with the saturated model and do a backward search.

```
> dm.sat <- dmod(~.^., data=reinis)
> dm.back <- backward(dm.sat)

List of 4
 $ t.uR: num [1:2, 1:2, 1:2, 1:2, 1:2] 156 107 141 168 115 45 116 76 93 98 ...
  ..- attr(*, "dimnames")=List of 5
  .. ..$ smoke  : chr [1:2] "y" "n"
  .. ..$ phys   : chr [1:2] "y" "n"
  .. ..$ systol : chr [1:2] "y" "n"
  .. ..$ protein: chr [1:2] "y" "n"
  .. ..$ family : chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2, 1:2, 1:2, 1:2] 84 179 274 35 47 113 176 16 55 136 ...
  ..- attr(*, "dimnames")=List of 5
  .. ..$ mental : chr [1:2] "y" "n"
  .. ..$ phys   : chr [1:2] "y" "n"
  .. ..$ systol : chr [1:2] "y" "n"
  .. ..$ protein: chr [1:2] "y" "n"
  .. ..$ family : chr [1:2] "y" "n"
 $ R   : chr [1:4] "phys" "systol" "protein" "family"
 $ vn  : chr [1:6] "smoke" "mental" "phys" "systol" ...
, , phys = y, systol = y, protein = y, family = y


     mental
smoke     y n
    y 49.83 0
    n 72.83 0

, , phys = n, systol = y, protein = y, family = y


     mental
smoke      y n
    y 125.03 0
    n  19.03 0

, , phys = y, systol = n, protein = y, family = y


     mental
smoke     y n
    y 33.78 0
    n 31.78 0

, , phys = n, systol = n, protein = y, family = y


     mental
smoke       y n
    y 106.333 0
    n   6.333 0
                         25
, , phys = y, systol = y, protein = n, family = y
```

Default is to search among decomposable models if the initial model is decomposable. Default is also to label all edges (with AIC values); however setting `search='headlong'` will cause the labelling to stop once an improvement has been found.

6.2 Forward search

Forward search works similarly; for example we start from the independence model:

```
> dm.i    <- dmod(~.^1, data=reinis)
> dm.forw <- forward(dm.i)

List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 1063 778
  ..- attr(*, "dimnames")=List of 1
  .. ..$ mental: chr [1:2] "y" "n"
 $ R   : chr(0)
 $ vn  : chr [1:2] "smoke" "mental"
     mental
smoke     y n
    y 554.9 0
    n 371.9 0
List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 927 914
  ..- attr(*, "dimnames")=List of 1
  .. ..$ phys: chr [1:2] "y" "n"
 $ R   : chr(0)
 $ vn  : chr [1:2] "smoke" "phys"
     phys
smoke     y n
    y 483.9 0
    n 436.9 0
List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 1054 787
  ..- attr(*, "dimnames")=List of 1
  .. ..$ systol: chr [1:2] "y" "n"
 $ R   : chr(0)
 $ vn  : chr [1:2] "smoke" "systol"
     systol
smoke     y n
    y 550.2 0
    n 376.2 0
List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 1061 780
  ..- attr(*, "dimnames")=List of 1
  .. ..$ protein: chr [1:2] "y" "n"
 $ R   : chr(0)
```

27

6.3   Fixing edges/terms in model as part of model selection

The stepwise model selection can be controlled by fixing specific edges. For example we can specify edges which are not to be considered in a bacward selection:

```
> fix <- list(c("smoke","phys","systol"), c("systol","protein"))
> fix <- do.call(rbind, unlist(lapply(fix, names2pairs),recursive=FALSE))
> fix

     [,1]       [,2]
[1,] "phys"     "smoke"
[2,] "smoke"    "systol"
[3,] "phys"     "systol"
[4,] "protein"  "systol"

> dm.s3 <- backward(dm.sat, fixin=fix, details=1)

List of 4
 $ t.uR: num [1:2, 1:2, 1:2, 1:2, 1:2] 156 107 141 168 115 45 116 76 93 98 ...
  ..- attr(*, "dimnames")=List of 5
  .. ..$ smoke  : chr [1:2] "y" "n"
  .. ..$ phys   : chr [1:2] "y" "n"
  .. ..$ systol : chr [1:2] "y" "n"
  .. ..$ protein: chr [1:2] "y" "n"
  .. ..$ family : chr [1:2] "y" "n"
 $ t.wR: num [1:2, 1:2, 1:2, 1:2, 1:2] 84 179 274 35 47 113 176 16 55 136 ...
  ..- attr(*, "dimnames")=List of 5
  .. ..$ mental : chr [1:2] "y" "n"
  .. ..$ phys   : chr [1:2] "y" "n"
  .. ..$ systol : chr [1:2] "y" "n"
  .. ..$ protein: chr [1:2] "y" "n"
  .. ..$ family : chr [1:2] "y" "n"
 $ R   : chr [1:4] "phys" "systol" "protein" "family"
 $ vn  : chr [1:6] "smoke" "mental" "phys" "systol" ...
, , phys = y, systol = y, protein = y, family = y

     mental
smoke     y n
    y 49.83 0
    n 72.83 0

, , phys = n, systol = y, protein = y, family = y

     mental
smoke      y n
    y 125.03 0
    n  19.03 0

, , phys = y, systol = n, protein = y, family = y

     mental
smoke     y n
    y 33.78 0
    n 31.78 0
```

```
   phys = n  systol = n  protein = y  family = y
```

There is an important detail here: The matrix `fix` specifies a set of edges. Submitting these in a call to *`backward`* does not mean that these edges are forced to be in the model. It means that those edges in `fixin` which are in the model will not be removed.

Likewise in forward selection:

```
> dm.i3 <- forward(dm.i, fixout=fix, details=1)

List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 1063 778
  ..- attr(*, "dimnames")=List of 1
  .. ..$ mental: chr [1:2] "y" "n"
 $ R  : chr(0)
 $ vn  : chr [1:2] "smoke" "mental"
      mental
smoke    y n
    y 554.9 0
    n 371.9 0
List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 1061 780
  ..- attr(*, "dimnames")=List of 1
  .. ..$ protein: chr [1:2] "y" "n"
 $ R  : chr(0)
 $ vn  : chr [1:2] "smoke" "protein"
     protein
smoke    y n
    y 553.8 0
    n 372.8 0
List of 4
 $ t.uR: num [1:2(1d)] 961 880
  ..- attr(*, "dimnames")=List of 1
  .. ..$ smoke: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 1581 260
  ..- attr(*, "dimnames")=List of 1
  .. ..$ family: chr [1:2] "y" "n"
 $ R  : chr(0)
 $ vn  : chr [1:2] "smoke" "family"
     family
smoke    y n
    y 825.3 0
    n 124.3 0
List of 4
 $ t.uR: num [1:2(1d)] 1063 778
  ..- attr(*, "dimnames")=List of 1
  .. ..$ mental: chr [1:2] "y" "n"
 $ t.wR: num [1:2(1d)] 927 914
  ..- attr(*, "dimnames")=List of 1
  .. ..$ phys: chr [1:2] "y" "n"          31
 $ R  : chr(0)
 $ vn  : chr [1:2] "mental" "phys"
```

Edges in `fix` will not be added to the model but if they are in the starting model already, they will remain in the final model.

7 Further topics on models for contingency tables

7.1 Adjusting for sparsity

[5]

7.2 Dimension of a log–linear model

The `loglinDim()` is a general function for finding the dimension of a log–linear model. It works on the generating class of a model being represented as a list:

```
> dim_loglin(dm2$glist, reinis)

[1] 10
```

8 Miscellaneous

8.1 The Model Object

It is worth looking at the information in the model object:

```
> dm3 <- dmod(list(c("smoke", "systol"), c("smoke", "mental", "phys")),
+             data=reinis)
> names(dm3)

[1] "call"      "glist"     "varNames"  "datainfo"  "fitinfo"   "isFitted"
[7] "glistNUM"  "properties"
```

- The model, represented as a list of generators, is

```
> str(dm3$glist)

List of 2
 $ : chr [1:2] "smoke" "systol"
 $ : chr [1:3] "smoke" "mental" "phys"
```

---

[5]Comment on adjustment for sparsity in testadd() and testdelete()

```
> str(dm3$glistNUM)

List of 2
 $ : int [1:2] 1 2
 $ : int [1:3] 1 3 4
```

The numeric representation of the generators refers back to

```
> dm3$varNames

[1] "smoke"  "systol" "mental" "phys"
```

Notice the model object does not contain a graph object. Graph objects are generated on the fly when needed.

- Information about the variables etc. is

```
> str(dm3[c("varNames","conNames","conLevels")])

List of 3
 $ varNames: chr [1:4] "smoke" "systol" "mental" "phys"
 $ NA      : NULL
 $ NA      : NULL
```

- Finally `isFitted` is a logical for whether the model is fitted; `data` is the data (as a table) and `fitinfo` consists of fitted values, logL, df etc.