Chapter 2
Log-Linear Models

2.1 Introduction

This chapter gives an account of graphical models for multivariate discrete data. Such data are usually summarized as contingency tables, and Sect. 2.2 describes some general utilities useful when working with such tables. Section 2.3 introduces the theory of log-linear models, illustrating this using `dModel` objects from the **gRim** package. Section 2.5.1 shows how log-linear models can be fit using the `glm` function, and Sect. 2.5.2 describes some aspects of working with `dModel` objects. Some more advanced topics are dealt with in Sect. 2.5.

2.2 Preliminaries

2.2.1 Four Datasets

To introduce contingency table data we consider four examples. All datasets used here are in **gRbase**. The first is shown in Table 2.1. These data originate from Schoener (1968) and are discussed in numerous places, e.g. Edwards (2000) and Whittaker (1990). In a study of lizard behaviour, characteristics of 409 lizards were recorded, namely species (S), perch diameter (D) and perch height (H). The focus of interest is in how the propensities of the lizards to choose perch height and diameter are related, and whether and how these depend on species.

The second dataset we consider is a 2^6 contingency table concerning risk factors for coronary heart disease. The data originated in a prospective study of coronary heart disease carried out in Czechoslovakia (Reiniš et al. 1981). For a sample of 1841 car-workers, the following information was recorded: whether they smoked, whether their work was strenuous mentally, whether their work was strenuous physically, whether their systolic blood pressure was less than 140 mm), whether the ratio of beta to alpha lipoproteins was less than 3, and whether there was a family history of coronary heart disease.

Table 2.1 Perching behaviour of two species of lizards

Species	Perch diameter (inches)	Perch Height (feet)	
		> 4.75	≤ 4.75
Anoli	≤ 4	32	86
	> 4	11	35
Distichus	≤ 4	61	73
	> 4	41	70

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

```
 table [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"
```

The third dataset is a 2^6 contingency table taken from genetics, and analyzed in Edwards (2000). Two isolates of the barley powdery mildew fungus were crossed, and for 70 progeny 6 binary characteristics (genetic markers) were recorded.

```
> data(mildew)
> str(mildew)
```

```
 table [1:2, 1:2, 1:2, 1:2, 1:2, 1:2] 0 0 0 0 3 0 1 0 0 1 ...
 - attr(*, "dimnames")=List of 6
  ..$ la10: chr [1:2] "1" "2"
  ..$ locc: chr [1:2] "1" "2"
  ..$ mp58: chr [1:2] "1" "2"
  ..$ c365: chr [1:2] "1" "2"
  ..$ p53a: chr [1:2] "1" "2"
  ..$ a367: chr [1:2] "1" "2"
```

The fourth dataset is a three-way table containing the results of a study comparing four different surgical operations on patients with duodenal ulcer, carried out in four centres, and described in Grizzle et al. (1969). The four operations were: vagotomy and drainage, vagotomy and antrectomy (removal of 25% of gastric tissue), vagotomy and hemigastrectomy (removal of 50% of gastric tissue), and gastric restriction (removal of 75% of gastric tissue). The response variable is the severity of gastric dumping, an undesirable syndrome associated with gastric surgery.

```
> data(dumping)
> str(dumping)
```

```
 table [1:3, 1:4, 1:4] 23 7 2 23 10 5 20 13 5 24 ...
 - attr(*, "dimnames")=List of 3
  ..$ Symptom  : chr [1:3] "none" "slight" "moderate"
  ..$ Operation: chr [1:4] "Vd" "Va" "Vh" "Gr"
  ..$ Centre   : chr [1:4] "1" "2" "3" "4"
```

The first and second variables are ordinal.

2.2.2 Data Formats

Multivariate discrete data are usually stored in one of three forms, here illustrated with the `lizard` data.

As a Raw Case-List For example, the lizard data could be represented as 409 observations of three discrete variables: species, perch diameter and perch height. This is typically represented in `R` as a dataframe, with the discrete variables represented as factors. For example,

```
> data(lizardRAW)
> head(lizardRAW)

  diam height species
1   >4  >4.75    dist
2   >4  >4.75    dist
3  <=4 <=4.75   anoli
4   >4 <=4.75   anoli
5   >4 <=4.75    dist
6  <=4 <=4.75   anoli
```

As an Aggregated Case-List Sometimes discrete data are represented in aggregated *case-list* form (again typically represented as a `data.frame` in `R`), where one variable (usually called `Freq`) stores the counts for each configuration of variables:

```
> data(lizardAGG)
> lizardAGG

  diam height species Freq
1  <=4  >4.75   anoli   32
2   >4  >4.75   anoli   11
3  <=4 <=4.75   anoli   86
4   >4 <=4.75   anoli   35
5  <=4  >4.75    dist   61
6   >4  >4.75    dist   41
7  <=4 <=4.75    dist   73
8   >4 <=4.75    dist   70
```

As a Contingency Table Another aggregated representation of data is as a *contingency table* (which in `R` is represented as a `table` or as an `array`):

```
> data(lizard)
> lizard

, , species = anoli

     height
diam  >4.75 <=4.75
  <=4    32     86
  >4     11     35

, , species = dist

     height
```

```
diam  >4.75 <=4.75
  <=4    61     73
  >4     41     70
```

Note that the contingency table form is a compact representation of data when these are dense, in the sense that the number of observations is larger than the number of combinations of variable levels. With sparse data, for which the number of combinations of variable levels exceeds the number of observations, the case list format is more compact.

Note that coercion between the different representations can be obtained as follows:

```
> ##
> ## Raw case-list to aggregated case-list:
> as.table(ftable(lizardRAW))
> ##
> ## Raw case-list to table
> xtabs(~., data=lizardRAW)
> ##
> ## Aggregated case-list to table
> xtabs(Freq~., data=lizardAGG)
> ##
> ## Table to aggregated case--list
> as.data.frame(lizard)
```

Note also that the lizard data can be specified as a contingency table using

```
> counts <- c(32, 11, 86, 35, 61, 41, 73, 70)
> dimn <- list(diam=c("<=4", ">4"),
+              height=c(">4.75", "<=4.75"),
+              species=c("anoli", "dist"))
> lizard <- as.table(array(counts, dim=c(2,2,2), dimnames=dimn))
```

2.3 Log-Linear Models

In this section we give a brief account of the theory of log-linear models.

2.3.1 Preliminaries and Notation

Suppose that we have a dataset with N observations of d discrete random variables. For example, the lizard data had $N = 409$ and $d = 3$. We write the collection of discrete variables as $X = (X_v)_{v \in \Delta}$, and we call the possible values a discrete variable may take its *levels*. Write the number of levels of X_v as $|X_v|$. For notational convenience we label the levels $1, \ldots, |X_v|$ though in practice they should be given more meaningful labels. We can then write a generic observation (or *cell*) as $i = (i_1, \ldots, i_d)$, and the set of possible cells as $\mathcal{I}$.

We assume that the observations are independent and are interested in modelling the probabilities $p(i) = \Pr(X = i)$ for $i \in \mathcal{I}$. The joint probability of the observations represented as a case list $(i^\nu,\ \nu = 1, \ldots, N)$ is then

$$p(i^\nu, \nu = 1, \ldots, N) = \prod_{\nu=1}^{N} p(i^\nu) = \prod_{i \in \mathcal{I}} p(i)^{n(i)} \tag{2.1}$$

where we have formed an aggregated case list or, equivalently, the contingency table $\{n(i)\}_{i \in \mathcal{I}}$, where $n(i)$ is the number of cases i^ν with $i^\nu = i$. The joint probability of the observed contingency table is

$$p(\{n(i)\}_{i \in \mathcal{I}}) = \frac{N!}{\prod_{i \in \mathcal{I}} n(i)!} \prod_{i \in \mathcal{I}} p(i)^{n(i)} \tag{2.2}$$

which differs from (2.1) by a multinomial coefficient which does not affect the likelihood as the latter is only determined up to a constant factor.

$$L(p) \propto \prod_{i \in \mathcal{I}} p(i)^{n(i)}. \tag{2.3}$$

If we do not restrict the probabilities in any way (except requiring that they are non-negative and sum to unity), then it is easily shown that the maximum likelihood estimates are given by $\hat{p}(i) = n(i)/N$ for $i \in \mathcal{I}$. The unrestricted model is known as the *saturated model*. In most substantive contexts it is of interest to restrict the probabilities further to obtain parsimony and to identify or exploit structural information, see further in Sect. 2.3.2 below.

We need a little more notation. The expected cell counts are written $m(i) = Np(i)$ for $i \in \mathcal{I}$, and the fitted values as $\hat{m}(i) = N\hat{p}(i)$. We need to work with marginal tables and to do this must first define marginal cells. Recall that Δ contains d variables and so a generic cell i is a d-tuple, that is $i = (i_1, \ldots, i_d)$. For a subset $A \subseteq \Delta$, the corresponding marginal cell is written i_A, and contains the indices $(i_v, v \in A)$. The corresponding marginal counts and probabilities are written $n(i_A)$ and $p(i_A)$. So, for example, we have that $n(i_A) = \sum_{j \in \mathcal{I}: j_A = i_A} n(j)$.

2.3.2 Hierarchical Log-Linear Models

Log-linear models are defined by constraining the logarithms of the probabilities to follow ANOVA-like factorial expansions. For example, for a three-dimensional table (such as Table 2.1) we write a generic cell as $i = (j, k, l)$, the variables as $\Delta = \{a, b, c\}$, and the saturated model as

$$\log p(i) = u + u^a_j + u^b_k + u^c_l + u^{ab}_{jk} + u^{ac}_{jl} + u^{bc}_{kl} + u^{abc}_{jkl}. \tag{2.4}$$

Here the u's are unknown parameters—usually called *interaction terms*. To estimate these uniquely we would need to constrain them further in some way, but we do not need to bother about this now.

The expansion (2.4) is only valid when $p(i) > 0$. We obtain higher generality by letting $\tilde{u} = \exp u$ and writing the expansion in product form

$$p(i) = \tilde{u} \cdot \tilde{u}^a_j \cdot \tilde{u}^b_k \cdot \tilde{u}^c_l \cdot \tilde{u}^{ab}_{jk} \cdot \tilde{u}^{ac}_{jl} \cdot \tilde{u}^{bc}_{kl} \cdot \tilde{u}^{abc}_{jkl} \tag{2.5}$$

as this enables us to deal with cells i with $p(i) = 0$. In general we have to includes limits of distributions satisfying (2.4) or (2.5), see further discussion in Sects. 2.3.4 and 2.5.1 below.

In log-linear models, certain interaction terms are set to zero. For example, we could set all two- and three-factor interaction terms equal to zero, by positing that

$$\log p(i) = u + u^a_j + u^b_k + u^c_l.$$

This is called the *main effect* model.

Usually only hierarchical log-linear models are of interest. The term *hierarchical* means that if a term is set to zero, all its higher-order relatives are also set to zero. Alternatively expressed, if a term is allowed in the expansion, so are other terms of lower order involving the relevant variables. For example, if we set $u^{ab}_{jk} = 0$ for all j, k, then we must also set $u^{abc}_{jkl} = 0$ for all j, k, l and if $u^{ab}_{jk} \neq 0$ is allowed, we must allow $u^a_j \neq 0$ and $u^b_k \neq 0$.

Hierarchical models can be specified in terms of the maximal interaction terms permitted: these are called the *generators* of the model. For example, the generators of the model

$$\log p(i) = u + u^a_j + u^b_k + u^c_l + u^{ab}_{jk} + u^{ac}_{jl} \tag{2.6}$$

are $\{a, b\}$ and $\{a, c\}$.

The **gRim** package has a function `dmod()` to define and fit hierarchical log-linear models. The models can be specified using a model formula or list of character vectors representing the generators. For example,

```
> m1 <- dmod(~species*height+species*diam, data=lizard)
> m2 <- dmod(list(c("species","height"),c("species", "diam")),
             data=lizard)
```

specify the same model. The first form is most useful when specifying small models by hand.

Under (2.6) specified as `m1` or `m2` above the probabilities can be factored into

$$p(i) = (\tilde{u} \cdot \tilde{u}^a_j \cdot \tilde{u}^b_k \cdot \tilde{u}^{ab}_{jk})(\tilde{u}^c_l \cdot \tilde{u}^{ac}_{jl}),$$

i.e. into two factors, the first not involving c and the second not involving b. It then follows from the factorization criterion (1.1) that $b \perp\!\!\!\perp c \mid a$. More generally this reasoning implies that under any hierarchical model, two factors are conditionally independent given the rest if and only if the corresponding two-factor interaction term is set to zero or, equivalently, if no generator contains both factors.

Thus, this model implies that perching diameter and height are independent given species. In other words, for each species considered separately, perching diameter and height are independent.

The *dependence graph* of a hierarchical model is an undirected graph with edges present whenever the corresponding two-factor interaction is allowed. We can display the graph of a `dModel` object using `plot` (see Fig. 2.1).

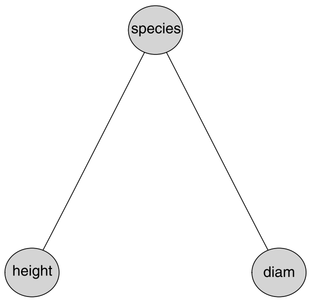


Fig. 2.1 Conditional independence of `diam` and `height` given `species`

From the global Markov property (Sect. 1.3) we can find out which conditional independences hold under a model:

```
> separates("height","diam","species", as(m1,"graphNEL"))
[1] TRUE
```

In the present case the property is evident from the graph, but the facility is useful for higher-dimensional models.

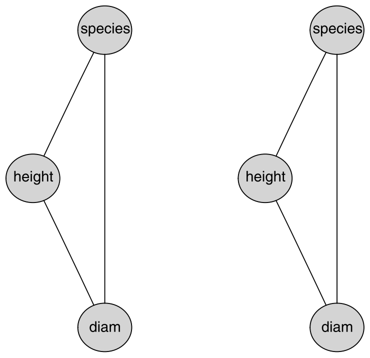
2.3.3 Graphical and Decomposable Log-Linear Models

Suppose that we are given an undirected graph $\mathcal{G} = (\Delta, E)$, and consider the hierarchical log-linear model $\mathcal{M}$ for Δ whose generators are identical to the cliques of the graph. A model that can be specified in this way is called a *graphical* model. Since the two-factor interaction terms that are set to zero in the model correspond to edges that are not present in $\mathcal{G}$, we see that $\mathcal{G}$ is the dependence graph of $\mathcal{M}$. By the hierarchical principle, any higher-order interaction term containing such a 'zero' two-factor term is also set to zero. And any higher-order term that does not contain a 'zero' two-factor term is contained in a generator and so is not set to zero. So one characterization of a graphical log-linear model is that the two-factor interaction terms present in the model completely determine which higher-order interactions are also present. Log-linear models that are not graphical set higher order interactions to zero even though all the corresponding two-factor interactions are not set to zero. The simplest *non-graphical* model is the no three-factor interaction model for a three-way table:

```
> no3f <- dmod(~species:height + species:diam + height:diam,
               data=lizard)
```

Note that this model has the same dependence graph as the saturated model:

```
> par(mfcol=c(1,2))
> sat <- dmod(~species:height:diam, data=lizard)
> plot(no3f, main='no 3-factor interaction')
> plot(sat, main='saturated model')
```



The attractive feature of graphical models is that they can be interpreted solely in terms of patterns of conditional independences, which can be displayed in terms of a graph.

We can also obtain the graphical model corresponding to a given undirected graph, as in

```
> g <- ug(~la10:locc:mp58 + locc:mp58:c365 + mp58:c365:p53a +
+          c365:p53a:a367)
> mg <- dmod(g, data=mildew)
```

In general, to obtain maximum likelihood estimates for log-linear models, iterative methods must be used. But for an important subclass of log-linear models, the *decomposable* models, closed-form expressions are available; see Sect. 2.3.4 for details.

To get information about properties of a model, the `summary()` method may be used:

```
> summary(no3f)

is graphical=FALSE; is decomposable=FALSE
generators (glist):
  :"species" "height"
  :"species" "diam"
  :"height" "diam"
```

2.3.4 Estimation, Likelihood, and Model Fitting

Decomposable models are characterized as graphical models whose graphs are triangulated. For decomposable models, closed-form expressions exist for the maximum likelihood estimate. The closed-form expressions are closely related to RIP-orderings of the cliques; see Sect. 1.4.1 for further details.

Let $\mathcal{C} = (C_1, \dots, C_k)$ be such an ordering and $\mathcal{S} = (S_1, \dots, S_k)$ the corresponding separators. Then the ML estimator is given by

$$\hat{m}(i) = \frac{\prod_{j=1\dots k} n(i_{C_j})}{\prod_{j=1\dots k} n(i_{S_j})}.$$

For non-decomposable models we need another way to find the maximum likelihood estimates. Most commonly the IPS (iterative proportional scaling) algorithm is used. This is a simple and robust algorithm, which works by storing and iteratively updating a table of fitted values $\{m(i)\}_{i\in\mathcal{I}}$.

Let $\mathcal{C} = \{a_1, \dots, a_Q\}$ be the generators of a hierarchical log-linear model. The corresponding marginal tables $n(i_{a_k})$, $k = 1, \dots, Q$, are a set of sufficient statistics. The maximum likelihood estimate is obtained by equating the sufficient statistics with their expectations $m(i_{a_k})$.

Initially the $m(i)$ are set to some constant, say $m(i) = 1$ for all $i \in \mathcal{I}$. One iteration consists of updating for each $k = 1, \dots, Q$

$$m(i) \leftarrow m(i)\frac{n(i_{a_k})}{m(i_{a_k})} \quad \forall i \in \mathcal{I}. \tag{2.7}$$

Iteration continues until convergence which happens when $m(i_{a_k}) = n(i_{a_k})$. The algorithm is always theoretically convergent with the limiting value being the maximum likelihood estimate under the model $\{\hat{m}(i)\}_{i\in\mathcal{I}}$, although these may not all have $\hat{m}(i) > 0$ for all cells i and thus may not admit a logarithmic expansion.

In `R`, the IPS algorithm is implemented in the `loglin()` function: the function `dmod()` in the **gRim** package provides an interface to this.

Notice that if the cliques of a decomposable model are given such that they follow a RIP-ordering then the IPS algorithm will converge after one iteration. If the cliques do not follow a RIP-ordering then IPS will converge after two iterations.

A disadvantage of IPS is that for high-dimensional problems it can be computationally expensive to store and update the whole table, as the iteration (2.7) passes through all possible values of i. It is possible to avoid this using message passing techniques based on the factorization (2.5), similar to those implemented in **gRain** and described in Chap. 3.

Another algorithm is that of iteratively reweighted least squares which is used for generalized linear models. This alternative is attractive when there is interest in the log-linear parameters (u-terms) themselves, since as a byproduct it provides estimates and standard errors of these. However, this approach can be problematic for other reasons; see Sect. 2.5.1 for an example and further discussion.

2.3.5 Hypothesis Testing

The maximized log-likelihood of a model m is given, up to an arbitrary additive constant, by

$$\ell = \sum_{i\in\mathcal{I}} n(i) \log \hat{p}(i)$$

where $\hat{p}(i)$ are the maximum likelihood estimates.

The *deviance* of a model $\mathcal{M}$ is twice the log-likelihood ratio of $\mathcal{M}$ versus the saturated model, i.e.,

$$D = \texttt{dev} = 2(\hat{\ell}_s - \hat{\ell}_m),$$

where $\hat{\ell}_s$ and $\hat{\ell}_m$ are the maximized log-likelihoods under the saturated model and $\mathcal{M}$, respectively. In this case we obtain

$$D = \texttt{dev} = G^2 = 2\sum_{i\in\mathcal{I}} n(i)\log\frac{n(i)}{\hat{m}(i)}.$$

Under $\mathcal{M}$, D is asymptotically $\chi^2(k)$ where the degrees of freedom k is the difference in dimension (number of free parameters) between the saturated model and m. So the deviance provides a goodness-of-fit test for the model. For example the following model fits rather well:

```
> m1 <- dmod(~species:height+species:diam, data=lizard)
> m1
```

```
Model: A dModel with 3 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :         1604.43 mdim :    5 aic :      1614.43
 ideviance :           23.01 idf  :    2 bic :      1634.49
 deviance  :            2.03 df   :    2
```

An alternative to the deviance is Pearson's goodness-of-fit test, defined by

$$X^2 = \sum_{i\in\mathcal{I}} \frac{\{n(i) - \hat{m}(i)\}^2}{\hat{m}(i)}$$

which has the same asymptotic distribution under the null hypothesis. This can be obtained using

```
> m1$fitinfo$pearson
```

```
[1] 2.017
```

Notice that it follows from the general definition of deviance given above that to calculate the deviance, it must be possible to fit the saturated model. This can always be done for log-linear models, but may not be possible in general, for example for Gaussian models; see Chap. 4. When working with sparse graphical models it is therefore often simpler to consider the *ideviance* (or independence deviance) which we define as twice the log-likelihood ratio between the model in question and the model of complete independence, corresponding to a graph with no edges, i.e. in this case

$$iD = \texttt{idev} = 2\sum_{i\in\mathcal{I}} n(i)\log\frac{\hat{m}(i)}{\prod_{v\in V} n(i_v)}.$$

The deviance or ideviance *difference* between two nested models makes always sense, provided both can be fitted, and it is in both cases equal to twice the log-likelihood ratio.

A related issue is that the dimension of a model depends, strictly speaking, on the sampling scheme employed, whereas the difference in dimension between two nested models (i.e. the degrees of freedom) does not. As we have described it, data have been assumed to be collected as a fixed number of independent units, referred to as the *multinomial sampling scheme*. If we instead assume that the total number of observations follows a Poisson distribution with unknown parameter $\lambda > 0$, the counts $N(i)$ become independent with parameters $\mathrm{e}\{N(i)\} = m(i)$. This is the *Poisson sampling scheme*.

It can be shown that the maximum likelihood estimate of λ is then equal to $\hat{\lambda} = n$ and the likelihood function for $\lambda = \hat{\lambda}$ is proportional to the likelihood function in the multinomial sampling scheme, thus not affecting deviances nor maximum likelihood estimates. This is known as the *Poisson trick*.

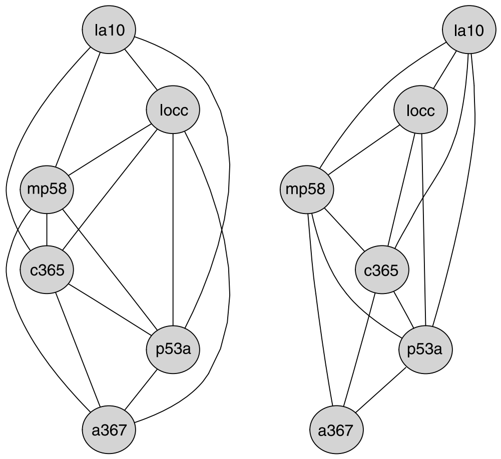
In general, it is simplest to calculate dimensions of models for the Poisson sampling scheme and therefore all dimensions refer to this scheme.

The calculation of the model dimension by `dmod()` assumes that $\hat{m}(i) > 0$ for all cells i, which will be the case when the data are dense, for example when all the cell counts are positive. When the data are sparse, as is usually the case for moderate to high-dimensional problems, some cells may have $\hat{m}(i) = 0$ and so the degrees of freedom shown need adjustment. Calculation of the appropriate degrees of freedom when the data are sparse is a hard problem, and we are aware of no software that does this correctly in all cases. In any case the asymptotic χ^2 approximation may be poor if $\hat{m}(i)$ is small.

A viable approach to analysis is to focus on comparisons of nested decomposable models, for which the correct adjustment to the degrees of freedom can be calculated, and for which it is straightforward to calculate exact conditional tests. A key result is that if $\mathcal{M}_0 \subset \mathcal{M}_1$ are decomposable log-linear models differing by one edge $e = \{u, v\}$ only, then e is contained in one clique C of $\mathcal{M}_1$ only, and the likelihood ratio test for $\mathcal{M}_0$ versus $\mathcal{M}_1$ can be performed in the marginal C-table as a test of $u \perp\!\!\!\perp v \mid C \setminus \{u, v\}$. The point being partly that the marginal table on C may not be sparse, but more importantly, that it is straightforward to adjust the degrees of freedom for a pure test of conditional independence like this, as we describe shortly.

For example, suppose that we specify a decomposable model `m3` for the mildew data, and delete the edge {`locc`, `a367`} from `m3` using the `update()` function, obtaining a model `m4`. This function is described below in Sect. 2.5.2. The edge deleted is contained in one clique only of `m3`, so `m4` is also decomposable.

```
> m3 <- dmod(~la10*locc*mp58*c365*p53a+locc*mp58*c365*p53a*a367,
             data=mildew)
> m4 <- update(m3, list(dedge=~locc*a367))
> oldpar<-par(mfrow=c(1,2))
> plot(m3, "neato")
> plot(m4, "neato")
> par(oldpar)
```



A direct comparison of m3 and m4 using the following function gives an incorrect value for the degrees of freedom

```
>   comparemodels <- function(m1,m2) {
+   lrt <- m2$fitinfo$dev - m1$fitinfo$dev
+   dfdiff <- m1$fitinfo$dimension[1] - m2$fitinfo$dimension[1]
+   c('lrt'=lrt, 'df'=dfdiff)
+ }
> m3

Model: A dModel with 6 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :         366.16 mdim :   47 aic :       460.16
 ideviance :         209.32 idf  :   41 bic :       565.83
 deviance  :           0.40 df   :   16
Notice: Table is sparse
  Asymptotic chi2 distribution may be questionable.
  Degrees of freedom can not be trusted.
  Model dimension adjusted for sparsity : 21

> m4

Model: A dModel with 6 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :         370.73 mdim :   39 aic :       448.73
 ideviance :         204.74 idf  :   33 bic :       536.42
 deviance  :           4.98 df   :   24
Notice: Table is sparse
  Asymptotic chi2 distribution may be questionable.
  Degrees of freedom can not be trusted.
  Model dimension adjusted for sparsity : 22

> comparemodels(m3,m4)

       lrt df.mod.dim
     4.573      8.000
```

The correct test may be obtained using the `testdelete()` function:

```
> testdelete(m3, edge=c("locc","a367"))
```

```
dev:   4.573 df:  3 p.value: 0.20585 AIC(k=2.0):   -1.4 edge: locc:a367
host:  locc mp58 c365 p53a a367
Notice: Test performed in saturated marginal model
```

This function identifies that `m3` is decomposable and that the edge {`locc`, `a367`} is in one clique C only. The test is then performed as a test of $u \perp\!\!\!\perp v \mid C \setminus \{u, v\}$. Note that the test statistic matches with that of `comparemodels()` and the degrees of freedom have been correctly adjusted.

Tests of general conditional independence hypotheses of the form $u \perp\!\!\!\perp v \mid W$ can be performed using the `ciTest_table()` function.

```
> cit <- ciTest_table(mildew, set=c("locc","a367","mp58","c365",
                                    "p53a"))
```

```
Testing locc _|_ a367 | mp58 c365 p53a
Statistic (DEV):    4.573 df: 3 p-value: 0.2059 method: CHISQ
```

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.

Notice that in this case the results are identical to those given by the `test-delete()` function, since we have specified the correct conditioning set. If we had conditioned on more variables

```
> cit2 <- ciTest_table(mildew, set=c("locc","a367","mp58","c365",
+                                    "p53a","la10"))
```

```
Testing locc _|_ a367 | mp58 c365 p53a la10
Statistic (DEV):    4.553 df: 3 p-value: 0.2076 method: CHISQ
```

different results would be obtained.

In model terms, the test performed by `ciTest_table()` 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 mp58 c365 p53a
1    0.0000 1.00000  0    1    1    1
2    0.5053 0.47716  1    2    1    1
3    1.2953 0.25508  1    1    2    1
4    2.7726 0.09589  1    2    2    1
5    0.0000 1.00000  0    1    1    2
6    0.0000 1.00000  0    2    1    2
```

```
7     0.0000 1.00000  0     1     2     2
8     0.0000 1.00000  0     2     2     2
```

The sth slice is a $|u| \times |v|$ table $\{n_{ijs}\}_{i=1\ldots|u|,j=1\ldots|v|}$. The output shows the degrees of freedom corresponding to the test for independence in each slice, given by

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

where $n_{i\cdot s}$ and $n_{\cdot js}$ are the marginal totals. So the correct number of degrees of freedom for the test in the present example is not 8 but 3, as calculated by the `ciTest_table()` and `testdelete()` functions.

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 a 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*. In the present example we get a Monte Carlo p-value which is considerably larger than the asymptotic one:

```
> ciTest_table(mildew, set=c("locc","a367","mp58","c365","p53a"),
+ method='MC')

Testing locc _|_ a367 | mp58 c365 p53a
Statistic (DEV):    4.573 df: NA p-value: 0.5550 method: MC
```

An advantage of the Monte Carlo method is that any test statistic can be used, so statistics that are sensitive to specific forms of deviation from independence can be used. In particular, when one or both of u and v are ordinal, more powerful tests of $u \perp\!\!\!\perp v \mid S$ can be applied. The `ciTest_ordinal()` function supports this approach for three rank tests: the Wilcoxon, Kruskal-Wallis and Jonckheere-Terpstra tests. The Wilcoxon test is applicable when u is binary and v ordinal; the Kruskal-Wallis test when u is nominal and v is ordinal; and the Jonckheere-Terpstra test when both u and v are ordinal. We illustrate use of the function using the dumping syndrome data described above in Sect. 2.2.1. Recall that the three variables are `Symptom`, `Operation` and `Centre`. The first two are ordinal and the third is nominal.

```
> ciTest_ordinal(dumping,c(2,1,3),"jt", N=1000)

$JT
[1] 9566

$EJT
[1] 8705
```

```
$P
[1] 0.009804

$montecarlo.P
[1] 0.005

$set
[1] "Operation"  "Symptom"   "Centre"

> ciTest_ordinal(dumping,c(2,1,3),"deviance", N=1000)

$deviance
[1] 23.54

$df
[1] 24

$P
[1] 0.4883

$montecarlo.P
[1] 0.585

$set
[1] "Operation"  "Symptom"   "Centre"
```

The second argument is a vector of column numbers (if a dataframe is supplied) or dimension numbers (if a table is supplied, as here) of $\{u, v, S\}$. The corresponding names may also be given. The function calculates the Monte Carlo p-value based on N random samples, together with the asymptotic p-value. If $N = 0$, only the latter is calculated. We see that the ordinal test strongly rejects the hypothesis that `Symptom` is independent of `Operation` given `Centre`, whereas the non-ordinal test finds no evidence against this. In this example, the Monte Carlo p-values are similar to the asymptotic ones. To examine whether the conditional distribution of `Symptom` given `Operation` is homogeneous over the centres, the Kruskal-Wallis test may be used:

```
> ciTest_ordinal(dumping, c(3,1,2),"kruskal", N=1000)

$KW
[1] 10.02

$df
[1] 12

$P
[1] 0.6143

$montecarlo.P
[1] 0.615

$set
[1] "Centre"    "Symptom"   "Operation"
```

The distributions appear to be homogeneous.

2.4 Model Selection

Using graphs to represent models has the effect of shifting the emphasis from estimation of parameters for a given model towards estimation of the model structure, that is, selecting an appropriate model. Model selection is challenging, not least because the number of possible models is huge. For example, the number of undirected graphs with 30 nodes is $2^{30\times 29/2} = 2^{435} > 10^{80}$, the estimated number of atoms in the observable universe.

Many different methods to select graphical models have been proposed, but generally they fall into three categories:

- Use of low-order conditional independence tests to infer the structure of the joint model. An example is the PC algorithm (Sect. 4.6.1).
- Heuristic search to optimize some criterion. Often local search around a current model is used to find a local optimum, possibly with combined with a stochastic search method. An example is the hill-climbing algorithm described in Sect. 4.6.2.1.
- Bayesian methods, often involving Markov chain Monte Carlo methods. We do not discuss Bayesian approaches to model selection further, but in Chap. 6 we describe aspects of graphical models from a Bayesian perspective.

Sometimes the first type of methods are used in a preliminary phase and then combined with others for refinement.

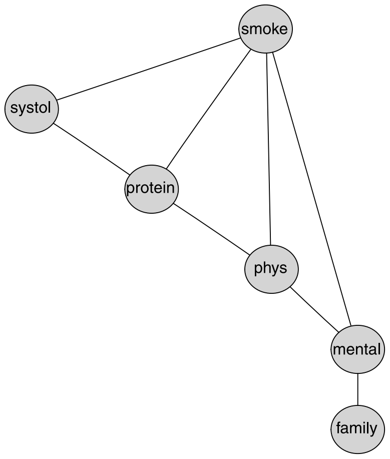
The **gRim** package implements a popular variant of the second type using well-known model selection criteria of *penalized likelihood* type. Consider a set of models $\mathcal{M}(j)$ for $j = 0, 1, \ldots, R$. We select the model $\mathcal{M}(j)$ which minimizes $-2\log L(j) + kp(j)$, where $p(j)$ is the number of free parameters in model $\mathcal{M}(j)$ and k is a penalty parameter.

Akaike's Information Criterion or *AIC* (Akaike 1974) uses $k = 2$. A popular alternative is the *Bayesian Information Criterion* or *BIC* (Schwarz 1978), which sets k to the logarithm of the number of observations. Use of a larger k penalizes complex models more heavily, and so tends to select simpler models. Other values of k can be chosen. It is standard usage in R to call the criterion AIC, even though strictly speaking only the value $k = 2$ gives the "genuine AIC".

The `stepwise()` function searches by default incrementally from an initial model, adding or deleting the edge that gives the largest decrease in the AIC. If there is none the process stops. The search is directional: either forward (adding edges) or backward (deleting edges). Alternatively, significance tests can be used to judge the relative adequacy of the models compared.

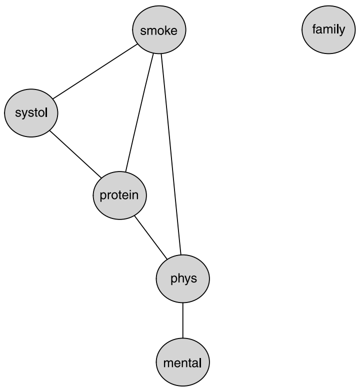
The following code selects a model for the `reinis` dataset. The initial model is set to be the saturated model, using a model specification shortcut described in Sect. 2.5.2.

```
> m.init <- dmod(~.^., data=reinis)
> m.reinis <- stepwise(m.init)
> plot(m.reinis)
```



The penalty term k is by default 2, but this can be changed using the argument of the same name. For example, the BIC criterion uses the logarithm of the number of observations as the penalty term:

```
> m.reinis.2  <- stepwise(m.init,k=log(sum(reinis)))
> plot(m.reinis.2)
```



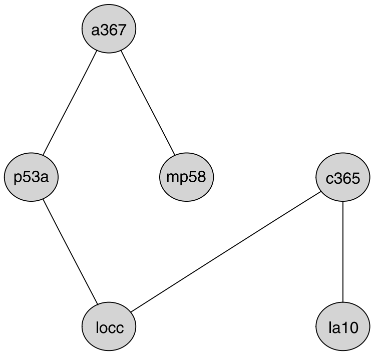
The choice of k is usually argued on the basis of asymptotic considerations. The motivation for AIC is that, under suitable assumptions, it is an approximate measure of the expected Kullback–Leibler distance between the true distribution and the estimated. The BIC difference between two models is the logarithm of a Laplace approximation to the associated Bayes factor for large number of observations n, but a term of lower order of magnitude than $\log n$ is ignored. Under reasonable assumptions the BIC is consistent in the sense that for n large it will choose the simplest model consistent with the data. This will typically only be true for the AIC if that is the saturated model. For a more general discussion of the issues involved, see Ripley (1996, Sect. 2.6).

The default direction is backward but may be changed to forward; notice that we set `details=1` to obtain some output from the model selection process:

```
> mildew.init <- dmod(~.^1, data=mildew)
> m.mildew  <- stepwise(mildew.init, k=log(sum(mildew)),
+                        direction="forward", details=1)
```

```
STEPWISE:
 criterion: aic ( k = 4.25 )
 direction: forward
 type     : decomposable
 search   : all
 steps    : 1000
. FORWARD: type=decomposable search=all, criterion=aic(4.25),
                                          alpha=0.00
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC  -59.0762 Edge added: a367 p53a
  change.AIC  -55.3386 Edge added: c365 la10
  change.AIC  -48.3388 Edge added: a367 mp58
  change.AIC   -6.3085 Edge added: c365 locc
  change.AIC   -2.1590 Edge added: locc p53a
```

```
> plot(m.mildew, "twopi")
```



The expression `~.^1` is a shortcut for the main effects model (see Sect. 2.5.2). The selected model shows the order of the markers on the chromosome: see Edwards (2000).

Other variants are possible. Setting `headlong=TRUE` results in *headlong search*: instead of adding or deleting the edge that gives the greatest decrease in the AIC, the edges at random are examined in random order and the first one found that decreases the AIC is added or deleted. This is generally faster for high-dimensional models.

Output can be suppressed using `details=0` whereas setting `details=2` will print test statistics for all edges, providings an indication of the strength of evidence for the edges present and the weakness of evidence for the absent edges. When searching among decomposable models (obtained by setting `type="decomposable"` as opposed to `type="unrestricted"`), the degrees of freedom are adjusted for sparsity.

```
> mildew.init.2 <- dmod(~.^., data=mildew)
> m.mildew.2  <- stepwise(mildew.init.2, crit="test", alpha=0.05,
+                          details=0)
> m.mildew.2
```

```
Model: A dModel with 6 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :         383.01 mdim :   11 aic :       405.01
 ideviance :         192.46 idf  :    5 bic :       429.74
 deviance  :          17.26 df   :   52
Notice: Table is sparse
  Asymptotic chi2 distribution may be questionable.
  Degrees of freedom can not be trusted.
  Model dimension adjusted for sparsity : 10
```

giving the same model as before.

2.5 Further Topics

2.5.1 Fitting Log-Linear Models with `glm()`

As we described in Sect. 2.3.5, we could just as well have assumed that cell counts $\{n(i)\}_{i\in\mathcal{I}}$ are independent realisations of Poisson distributions with means $\{\lambda(i)\}_{i\in\mathcal{I}}$. It follows that we can fit log-linear models as generalized linear models by means of the `glm()` function, using the Poisson distribution and (default) log-link. The estimation method is then Fisher Scoring (which requires inversion of a potentially large matrix).

It is worth mentioning that there may be computational problems with this approach: if the data are sparse and there are only few observations relative to the complexity of the model then the `glm()` estimation algorithm may fail, as it implicitly assumes that $\hat{m}(i) > 0$ for all $i \in \mathcal{I}$. The IPS algorithm, on the other hand, always works.

The data need to be in aggregrated case list form, as described in Sect. 2.2.2. In the present case we use

```
> lizardAGG
```

```
  diam height species Freq
1  <=4  >4.75   anoli   32
2   >4  >4.75   anoli   11
3  <=4 <=4.75   anoli   86
4   >4 <=4.75   anoli   35
5  <=4  >4.75    dist   61
6   >4  >4.75    dist   41
7  <=4 <=4.75    dist   73
8   >4 <=4.75    dist   70
```

We use the `Freq` variable as response variable. Note that it is important that all cells, also any empty ones, are present in the data. To fit the model shown in (2.1) we can use the code:

```
> m1glm <- glm(Freq~-1+diam:species+height:species,family=poisson,
+              data=lizardAGG)
> summary(m1glm)

Call:
glm(formula = Freq ~ -1 + diam:species + height:species,
    family = poisson, data = lizardAGG)

Deviance Residuals:
     1       2       3       4       5       6       7       8
 0.190  -0.310  -0.114   0.181   0.687  -0.782  -0.596   0.639

Coefficients:
                         Estimate Std. Error z value Pr(>|z|)
diam<=4:speciesanoli       4.467      0.103   43.30  < 2e-16 ***
diam>4:speciesanoli        3.525      0.155   22.80  < 2e-16 ***
diam<=4:speciesdist        4.359      0.102   42.80  < 2e-16 ***
diam>4:speciesdist         4.171      0.109   38.20  < 2e-16 ***
speciesanoli:height>4.75  -1.035      0.178   -5.83  5.6e-09 ***
speciesdist:height>4.75   -0.338      0.130   -2.61   0.0091 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2514.8188  on 8  degrees of freedom
Residual deviance:    2.0256  on 2  degrees of freedom
AIC: 59

Number of Fisher Scoring iterations: 4
```

By using `glm()` we automatically get the asymptotic standard errors of the parameter estimates and also these are not affected by the sampling scheme and hence are valid under both the Poission and multinomial sampling schemes.

By including `-1` in the right-hand side of the model formula we set the intercept to zero. This only affects the parametrisation of the model. The residual deviance gives the likelihood ratio test against the saturated model.

```
> msat  <- glm(Freq ~ -1 + diam*height*species, family=poisson,
+              data=lizardAGG)
> mno3f <- glm(Freq ~ -1 + diam*height + diam*species + species*height,
+              family=poisson, data=lizardAGG)
> anova(msat, mno3f, m1glm, test="Chisq")

Analysis of Deviance Table

Model 1: Freq ~ -1 + diam * height * species
Model 2: Freq ~ -1 + diam * height + diam * species + species * height
Model 3: Freq ~ -1 + diam:species + height:species
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1         0      0.000
2         1      0.149 -1   -0.149      0.70
3         2      2.026 -1   -1.876      0.17
```

Omission of empty cells from the input data corresponds to treating them as structural zeroes. This allows exotic hypotheses such as quasi-independence to be examined (Bishop et al. 1975). But for sparse tables, the `glm()` approach runs into problems, and IPS is to be preferred. For example,

```
> glm(Freq ~.^3, ,family=poisson, data=as.data.frame(mildew))
```

fails to converge but

```
> dmod(~.^3, data=mildew)
```

is unproblematic.

2.5.2 *Working with* `dModel` *Objects*

The `dmod()` function supports some useful shortcut expressions for model formulae. For example, `~.^.` is the saturated model, `~.^1` is the main effect model and `~.^p` is the model with all p-factor interactions. Furthermore, to specify marginal models (that is, not including all the variables in the table), the `marginal` argument can be used. Lastly, it is possible to abbreviate variable names. For example,

```
> m <- dmod(~.^2, marginal=c("smo", "prot", "sys","fam"),
+           data=reinis)
```

```
Model: A dModel with 4 variables
 graphical : FALSE  decomposable : FALSE
 -2logL    :        9021.61 mdim :   10 aic :      9041.61
 ideviance :          48.67 idf  :    6 bic :      9096.79
 deviance  :           9.24 df   :    5
```

The generating class of the model as a list and as a right-hand sided formula can be retrieved using `terms()` and `formula()`:

```
> str(terms(m))
```

```
List of 6
 $ : chr [1:2] "smoke" "protein"
 $ : chr [1:2] "smoke" "systol"
 $ : chr [1:2] "smoke" "family"
 $ : chr [1:2] "protein" "systol"
 $ : chr [1:2] "protein" "family"
 $ : chr [1:2] "systol" "family"
```

```
> formula(m)
```

```
~smoke * protein + smoke * systol + smoke * family + protein *
    systol + protein * family + systol * family
```

The dependence graph and adjacency matrix of a model object can be obtained using the `as()` function:

```
> as(m, "graphNEL")
```

```
A graphNEL graph with undirected edges
Number of Nodes = 4
Number of Edges = 6
```

```
> as(m, "matrix")
```

```
        smoke protein systol family
smoke       0       1      1      1
protein     1       0      1      1
systol      1       1      0      1
family      1       1      1      0
```

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`. No prize to work out which does which. 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
```

- 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 $m^* - e = m$.

2.6 Various

Other R packages which support discrete graphical models include **CoCo** (Badsberg 1991) and **gRapHD**, see Chap. 7. The packages **SIN**, **pcalg** and **bnlearn** support algorithms to select discrete graphical models: Sects. 4.4.4, 4.6.1, 4.6.2 and the following sections, illustrate their use in a Gaussian context. Chapter 3 describes the use of discrete, directed graphical models and Sect. 3.4 illustrates the selection of such a model.