5 Generalized Linear Models

Generalized linear models are just as easy to fit in R as ordinary linear model. In fact, they require only an additional parameter to specify the variance and link functions.

5.1 Variance and Link Families

The basic tool for fitting generalized linear models is the `glm` function, which has the folllowing general structure:

```
> glm(formula, family, data, weights, subset, ...)
```

where `...` stands for more esoteric options. The only parameter that we have not encountered before is `family`, which is a simple way of specifying a choice of variance and link functions. There are six choices of family:

Family	Variance	Link
gaussian	gaussian	identity
binomial	binomial	logit, probit or cloglog
poisson	poisson	log, identity or sqrt
Gamma	Gamma	inverse, identity or log
inverse.gaussian	inverse.gaussian	1/mu^2
quasi	user-defined	user-defined

As can be seen, each of the first five choices has an associated variance function (for binomial the binomial variance $\mu(1-\mu)$), and one or more choices of link functions (for binomial the logit, probit or complementary log-log).

As long as you want the default link, all you have to specify is the family name. If you want an alternative link, you must add a `link` argument. For example to do probits you use

```
> glm( formula, family=binomial(link=probit))
```

The last family on the list, `quasi`, is there to allow fitting user-defined models by maximum quasi-likelihood.

5.2 Logistic Regression

We will illustrate fitting logistic regression models using the contraceptive use data shown below:

```
   age education wantsMore notUsing using
   <25       low       yes       53     6
   <25       low        no       10     4
   <25      high       yes      212    52
   <25      high        no       50    10
 25-29       low       yes       60    14
 25-29       low        no       19    10
 25-29      high       yes      155    54
 25-29      high        no       65    27
 30-39       low       yes      112    33
 30-39       low        no       77    80
 30-39      high       yes      118    46
 30-39      high        no       68    78
 40-49       low       yes       35     6
 40-49       low        no       46    48
 40-49      high       yes        8     8
 40-49      high        no       12    31
```

The data are available from the datasets section of the website for my generalized linear models course. Visit http://data.princeton.edu/wws509/datasets (http://data.princeton.edu/wws509/datasets) to read a short description and follow the link to cuse.dat (http://data.princeton.edu/wws509/datasets/cuse.dat).

Of course the data can be downloaded directly from R:

```
> cuse <- read.table("http://data.princeton.edu/wws509/datasets/cuse.dat",
+     header=TRUE)
> cuse
     age education wantsMore notUsing using
1    <25       low       yes       53     6
2    <25       low        no       10     4
3    <25      high       yes      212    52
4    <25      high        no       50    10
5  25-29       low       yes       60    14
        ... output edited ...
16   40-49      high        no       12    31
```

I specified the `header` parameter as `TRUE`, because otherwise it would not have been obvious that the first line in the file has the variable names. There are no row names specified, so the rows will be numbered from 1 to 16. Print `cuse` to make sure you got the data in alright. Then make it your default dataset:

```
> attach(cuse)
```

Let us first try a simple additive model where contraceptive use depends on age, education and wantsMore:

```
> lrfit <- glm( cbind(using, notUsing) ~
+  age + education + wantsMore , family = binomial)
```

There are a few things to explain here. First, the function is called `glm` and I have assigned its value to an object called `lrfit` (for logistic regression fit). The first argument of the function is a model formula, which defines the response and linear predictor.

With binomial data the response can be either a vector or a matrix with two columns.

- If the response is a vector it can be numeric with 0 for failure and 1 for success, or a factor with the first level representing "failure" and all others representing "success". In these cases R generates a vector of ones to represent the binomial denominators.
- Alternatively, the response can be a matrix where the first column is the number of "successes" and the second column is the number of "failures". In this case R adds the two columns together to produce the correct binomial denominator.

Because the latter approach is clearly the right one for us I used the function `cbind` to create a matrix by binding the column vectors containing the numbers using and not using contraception.

Following the special symbol `~` that separates the response from the predictors, we have a standard Wilkinson-Rogers model formula. In this case we are specifying main effects of age, education and wantsMore. Because all three predictors are categorical variables, they are treated automatically as factors, as you can see by inspecting the results:

```
> lrfit

Call:  glm(formula = cbind(using, notUsing) ~ age + education + wantsMore,
      family = binomial)

Coefficients:
 (Intercept)      age25-29      age30-39      age40-49  educationlow
     -0.8082        0.3894        0.9086        1.1892       -0.3250
wantsMoreyes
     -0.8330

Degrees of Freedom: 15 Total (i.e. Null);  10 Residual
Null Deviance:      165.8
Residual Deviance: 29.92        AIC: 113.4
```

Recall that R sorts the levels of a factor in alphabetical order. Because <25 comes before 25-29, 30-39, and 40-49, it has been picked as the reference cell for `age`. Similarly, high is the reference cell for `education` because high comes before low! Finally, R picked no as the base for `wantsMore`.

If you are unhappy about these choices you can (1) use `relevel` to change the base category, or (2) define your own indicator variables. I will use the latter approach by defining indicators for women with high education and women who want no more children:

```
> noMore <- wantsMore == "no"
> hiEduc <- education == "high"
```

Now try the model again:

```
> glm( cbind(using,notUsing) ~ age + hiEduc + noMore, family=binomial)

Call:  glm(formula = cbind(using, notUsing) ~ age + hiEduc + noMore,
     family = binomial)

Coefficients:

(Intercept)     age25-29     age30-39     age40-49       hiEduc       noMore
    -1.9662       0.3894       0.9086       1.1892       0.3250       0.8330

Degrees of Freedom: 15 Total (i.e. Null);  10 Residual
Null Deviance:      165.8
Residual Deviance: 29.92        AIC: 113.4
```

The residual deviance of 29.92 on 10 d.f. is highly significant:

```
> 1-pchisq(29.92,10)
[1] 0.0008828339
```

so we need a better model. One of my favorites introduces an interaction between age and desire for no more children:

```
> lrfit <- glm( cbind(using,notUsing) ~ age * noMore + hiEduc , family=binomial)
> lrfit

Call:  glm(formula = cbind(using, notUsing) ~ age * noMore + hiEduc,
    family = binomial)

Coefficients:
    (Intercept)         age25-29         age30-39         age40-49
       -1.80317          0.39460          0.54666          0.57952
         noMore           hiEduc  age25-29:noMore  age30-39:noMore
        0.06622          0.34065          0.25918          1.11266
age40-49:noMore
        1.36167

Degrees of Freedom: 15 Total (i.e. Null);  7 Residual
Null Deviance:      165.8
Residual Deviance: 12.63        AIC: 102.1
```

Note how R built the interaction terms automatically, and even came up with sensible labels for them. The model's deviance of 12.63 on 7 d.f. is not significant at the conventional five per cent level, so we have no evidence against this model.

To obtain more detailed information about this fit try the `summary` function:

```
> summary(lrfit)

Call:
glm(formula = cbind(using, notUsing) ~ age * noMore + hiEduc,
    family = binomial)


Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.30027  -0.66163  -0.03286   0.81945   1.73851

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -1.80317    0.18018 -10.008  < 2e-16 ***
age25-29        0.39460    0.20145   1.959  0.05013 .
age30-39        0.54666    0.19842   2.755  0.00587 **
age40-49        0.57952    0.34733   1.669  0.09522 .
noMore          0.06622    0.33064   0.200  0.84126
hiEduc          0.34065    0.12576   2.709  0.00676 **
age25-29:noMore 0.25918    0.40970   0.633  0.52699
age30-39:noMore 1.11266    0.37398   2.975  0.00293 **
age40-49:noMore 1.36167    0.48422   2.812  0.00492 **
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 165.772  on 15  degrees of freedom
Residual deviance:  12.630  on  7  degrees of freedom
AIC: 102.14

Number of Fisher Scoring iterations: 3
```

R follows the popular custom of flagging significant coefficients with one, two or three stars depending on their p-values. Try `plot(lrfit)`. You get the same plots as in a linear model, but adapted to a generalized linear model; for example the residuals plotted are deviance residuals (the square root of the contribution of an observation to the deviance, with the same sign as the raw residual).

The functions that can be used to extract results from the fit include

- `residuals` or `resid`, for the deviance residuals
- `fitted` or `fitted.values`, for the fitted values (estimated probabilities)
- `predict`, for the linear predictor (estimated logits)
- `coef` or `coefficients`, for the coefficients, and
- `deviance`, for the deviance.

Some of these functions have optional arguments; for example, you can extract five different types of residuals, called "deviance", "pearson", "response" (response - fitted value), "working" (the working dependent variable in the IRLS algorithm - linear predictor), and "partial" (a matrix of working residuals formed by omitting each term in the model). You specify the one you want using the `type` argument, for example `residuals(lrfit,type="pearson")`.

5.3 Updating Models

If you want to modify a model you may consider using the special function `update`. For example to drop the `age:noMore` interaction in our model one could use

```
> lrfit0 <- update(lrfit, ~ . - age:noMore)
```

The first argument is the result of a fit, and the second an updating formula. The place holder `~` separates the response from the predictors and the dot `.` refers to the right hand side of the original formula, so here we simply remove `age:noMore`. Alternatively, one can give a new formula as the second argument.

The update function can be used to fit the same model to different datasets, using the argument `data` to specify a new data frame. Another useful argument is `subset`, to fit the model to a different subsample. This function works with linear models as well as generalized linear models.

If you plan to fit a sequence of models you will find the `anova` function useful. Given a series of *nested* models, it will calculate the change in deviance between them. Try

```
> anova(lrfit0,lrfit)
Analysis of Deviance Table

Model 1: cbind(using, notUsing) ~ age + noMore + hiEduc
Model 2: cbind(using, notUsing) ~ age + noMore + hiEduc + age:noMore
  Resid. Df Resid. Dev Df Deviance
1        10     29.917
2         7     12.630  3   17.288
```

Adding the interaction has reduced the deviance by 17.288 at the expense of 3 d.f.

If the argument to `anova` is a single model, the function will show the change in deviance obtained by adding each of the terms in the order listed in the model formula, just as it did for linear models. Because this requires fitting as many models as there are terms in the formula, the function may take a while to complete its calculations.

The anova function lets you specify an optional test. The usual choices will be "F" for linear models and "Chisq" for generalized linear models. Adding the parameter `test="Chisq"` adds p-values next to the deviances. In our case

```
> anova(lrfit,test="Chisq")

Analysis of Deviance Table
Model: binomial, link: logit
Response: cbind(using, notUsing)
Terms added sequentially (first to last)

           Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                         15    165.772
age         3   79.192        12     86.581 4.575e-17
noMore      1   49.693        11     36.888 1.798e-12
hiEduc      1    6.971        10     29.917     0.008
age:noMore  3   17.288         7     12.630     0.001
```

We can see that all terms were highly significant when they were introduced into the model.

5.4 Model Selection

A very powerful tool in R is a function for stepwise regression that has three remarkable features:

1. It works with generalized linear models, so it will do stepwise logistic regression, or stepwise Poisson regression,
2. It understand about hierarchical models, so it will only consider adding interactions only after including the corresponding main effects in the models, and
3. It understands terms involving more than one degree of freedom, so it it will keep together dummy variables representing the effects of a factor.

The basic idea of the procedure is to start from a given model (which could well be the null model) and take a series of steps by either deleting a term already in the model or adding a term from a list of candidates for inclusion, called the *scope* of the search and defined, of course, by a model formula.

Selection of terms for deletion or inclusion is based on Akaike's information criterion (AIC). R defines AIC as

$$-2 \text{ maximized log-likelihood} + 2 \text{ number of parameters}$$

(S-Plus defines it as the deviance minus twice the number of parameters in the model. The two definitions differ by a constant, so differences in AIC are the same in the two environments.) The procedure stops when the AIC criterion cannot be improved.

In R all of this work is done by calling a couple of functions, `add1` and `drop1`, that consider adding or dropping a term from a model. These functions can be very useful in model selection, and both of them accept a `test` argument just like `anova`.

Consider first `drop1`. For our logistic regression model,

```
> drop1(lrfit, test="Chisq")
Single term deletions

Model:
cbind(using, notUsing) ~ age + noMore + hiEduc + age:noMore
           Df Deviance     AIC     LRT   Pr(Chi)
          12.630 102.137
hiEduc      1   20.099 107.607   7.469 0.0062755 **
age:noMore  3   29.917 113.425  17.288 0.0006167 ***


---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1
```

Obviously we can't drop any of these terms. Note that R considered dropping the main effect of education and the age by want no more interaction, but did not examine the main effects of age or want no more, because one would not drop these main effects while retaining the interaction.

The sister function `add1` requires a scope to define the additional terms to be considered. In our example we will consider all possible two-factor interactions:

```
> add1(lrfit, ~.^2,test="Chisq")
Single term additions

Model:
cbind(using, notUsing) ~ age + noMore + hiEduc + age:noMore
              Df Deviance     AIC     LRT Pr(Chi)

             12.630 102.137
age:hiEduc     3    5.798 101.306   6.831 0.07747 .
noMore:hiEduc  1   10.824 102.332   1.806 0.17905
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1
```

We see that neither of the missing two-factor interactions is significant by itself at the conventional five percent level. (However, they happen to be jointly significant.) Note that the model with the age by education interaction has a lower AIC than our starting model.

The `step` function will do an automatic search. Here we let it search in a scope defined by all two-factor interactions:

```
> search <- step(additive, ~.^2)
  ... trace output supressed ...
```

The `step` function produces detailed trace output that we have supressed. The returned object, however, includes an `anova` component that summarizes the search:

```
> search$anova
              Step Df   Deviance Resid. Df Resid. Dev      AIC
1                 NA         NA        10  29.917222 113.4251
2    + age:noMore -3 -17.287669         7  12.629553 102.1375
3    + age:hiEduc -3  -6.831288         4   5.798265 101.3062
4 + hiEduc:noMore -1  -3.356777         3   2.441488  99.9494
```

As you can see, the automated procedure introduced, one by one, all three remaining two-factor interactions, to yield a final AIC of 99.9. This is an example where AIC, by requiring a deviance improvement of only 2 per parameter, may have led to overfitting the data.

Some analysts prefer a higher penalty per parameter. In particular, using log(n) instead of 2 as a multiplier yields BIC, the Bayesian Information Criterion. In our example log(1607) = 7.38, so we would require a deviance reduction of 7.38 per additional parameter. The `step` function accepts `k` as an argument, with default 2. You may verify that specifying `k=log(1607)` leads to a much simpler model; not only are no new interactions introduced, but the main effect of education is dropped (even though it is significant).

Continue with Conclusion and References (conclusion)