Regression Functions Supported by the effects Package
And How to Support Other Classes of Regression Models

John Fox and Sanford Weisberg

2022-07-07

1 Introduction

Effect plots, as implemented in the effects package, represent the “effects” (in the not necessarily causal sense of “partial relationship”) of one or more predictors on a response variable, in regression models in which the response depends on a linear predictor—a linear combination of main effects and interactions among the predictors (Fox and Weisberg 2019, Sec. 4.6.3). Effect() is the basic generic function in the effects package; Effect() is called directly or indirectly by several other functions in the package, such as predictorEffects() and allEffects().

Table 1 provides a list of regression modeling functions in R that can be used with the effects package. This list, which is almost surely incomplete, includes functions that are directly supported by Effect() methods supplied by the effects package, by Effect() methods supplied by other CRAN packages, or by the default Effect() method, which works with many classes of regression models.

The most basic type of model for which Effect() is appropriate is a standard linear model fit by the lm() function; for example:

```r
library("effects")
Prestige$type <- factor(Prestige$type, c("bc", "wc", "prof"))  # reorder levels
g1 <- lm(prestige ~ education + type + education:type, data = Prestige)  # equivalent to lm(prestige ~ education*type, data = Prestige)
plot(predictorEffects(g1), lines=list(multiline=TRUE))
```

Table 1: Regression Modeling Functions in R that Can Be Used with the effects Package.
Table 1: R regression functions known to be compatible with the `Effect()` function. The name before the double-colon is the package that includes the function; for example `stats::lm()` means that `lm()` is in the `stats` package. In some cases, `Effect()` may support only a subset of regression models fit by a particular function. Effects for mixed-effects models represent the fixed-effects part of the model.

<table>
<thead>
<tr>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>glm-type models</strong></td>
<td></td>
</tr>
<tr>
<td><code>stats::lm()</code></td>
<td>Standard linear regression models fit by least-squares or weighted least-squares. A multivariate response, generating a multivariate linear model, is permitted, and in this case effects are computed for each response separately.</td>
</tr>
<tr>
<td><code>stats::glm()</code></td>
<td>Generalized linear models.</td>
</tr>
<tr>
<td><code>nlme::lme()</code></td>
<td>Linear mixed-effects models.</td>
</tr>
<tr>
<td><code>nlme::gls()</code></td>
<td>Linear models fit by generalized least squares.</td>
</tr>
<tr>
<td><code>lmer::lmer()</code></td>
<td>Linear mixed-effects models.</td>
</tr>
<tr>
<td><code>lmer::glmer()</code></td>
<td>Generalized linear mixed-effects models.</td>
</tr>
<tr>
<td><code>survey::svyglm()</code></td>
<td>Generalized linear models for complex survey designs.</td>
</tr>
<tr>
<td><code>MASS::rlm()</code></td>
<td>Linear regression models estimated by robust M or MM regression.</td>
</tr>
<tr>
<td><code>MASS::glmmPQL()</code></td>
<td>Generalized linear mixed-effects models via partial quadratic likelihood.</td>
</tr>
<tr>
<td><code>robustlmm::rlmer()</code></td>
<td>Robust linear mixed-effects models.</td>
</tr>
<tr>
<td><code>betareg::betareg()</code></td>
<td>Beta regression models for rates and proportions.</td>
</tr>
<tr>
<td><code>ivreg::ivreg()</code></td>
<td>Linear regression models estimated by instrumental variables (2SLS regression).</td>
</tr>
<tr>
<td><code>glmmTMB::glmmTMB()</code></td>
<td>Generalized linear mixed-effects regression models (similar to <code>lmer::glmer()</code> but accommodating a broader selection of models).</td>
</tr>
<tr>
<td><strong>multinom-type models</strong></td>
<td></td>
</tr>
<tr>
<td><code>nnet::multinom()</code></td>
<td>Multinomial logistic-regression models. If the response has $K$ categories, the response for <code>nnet::multinom()</code> can be a factor with $K$ levels or a matrix with $K$ columns, which will be interpreted as counts for each of $K$ categories. Effects plots require the response to be a factor, not a matrix.</td>
</tr>
<tr>
<td><code>poLCA::poLCA()</code></td>
<td>Latent class analysis regression models for polytomous outcomes. Latent class analysis has a similar structure to multinomial regression, except that class membership of observations is unobserved but estimated in the analysis.</td>
</tr>
<tr>
<td><strong>polr-type models</strong></td>
<td></td>
</tr>
<tr>
<td><code>MASS::polr()</code></td>
<td>Ordinal logistic (proportional-odds) and probit regression models.</td>
</tr>
<tr>
<td><code>ordinal::clm()</code></td>
<td>Cumulative-link regression models (similar to, but more extensive than, <code>polr()</code>).</td>
</tr>
<tr>
<td><code>ordinal::clm2()</code></td>
<td>Updated version of <code>ordinal::clm()</code>.</td>
</tr>
<tr>
<td><code>ordinal::clmm()</code></td>
<td>Cumulative-link regression models with random effects.</td>
</tr>
</tbody>
</table>
In this example the response prestige is modeled as a linear function of years of education, the factor type, with levels blue collar ("bc"), white collar ("wc"), and professional ("prof"), and their interaction. Because of the interaction, the estimated partial relationship of prestige to education (depicted in the predictor effect plot for education, at the left) is different for each level of type, and the partial relationship of prestige to type (depicted in the predictor effect plot for type, at the right) varies with the value education.

A linear mixed-effects model is a more complicated regression model, fit, for example, by the \texttt{lmer()} function in the \texttt{lme4} package \cite{Bates2015}:

\begin{verbatim}
data(Orthodont, package="nlme")
g2 <- lme4::lmer(distance ~ age + Sex + (1 | Subject), data = Orthodont)
summary(g2)
\end{verbatim}

Linear mixed model fit by REML ['lmerMod']
Formula: distance ~ age + Sex + (1 | Subject)
Data: Orthodont

REML criterion at convergence: 437.5

Scaled residuals:
\[
\begin{array}{cccccc}
\text{Min} & 1\text{Q} & \text{Median} & 3\text{Q} & \text{Max} \\
-3.7489 & -0.5503 & -0.0252 & 0.4534 & 3.6575 \\
\end{array}
\]

Random effects:
\[
\begin{array}{llll}
\text{Groups} & \text{Name} & \text{Variance} & \text{Std.Dev.} \\
\text{Subject} & (\text{Intercept}) & 3.267 & 1.807 \\
\text{Residual} & & 2.049 & 1.432 \\
\end{array}
\]

Number of obs: 108, groups: Subject, 27

Fixed effects:
\[
\begin{array}{llll}
\text{Estimate} & \text{Std. Error} & \text{t value} \\
(\text{Intercept}) & 17.70671 & 0.83392 & 21.233 \\
\text{age} & 0.66019 & 0.06161 & 10.716 \\
\text{SexFemale} & -2.32102 & 0.76142 & -3.048 \\
\end{array}
\]

Correlation of Fixed Effects:
\[
\begin{array}{ll}
\text{(Intr) age} & \text{age}  \\
& -0.813 \\
\text{SexFemale} & -0.372 \\
\end{array}
\]

This model has a fixed effect part, with response distance and predictors age and Sex. The random intercept (represented by 1) varies by Subject. Effect plots for mixed-effects models are based only on the estimated fixed-effects in the model:

\begin{verbatim}
plot(predictorEffects(g2))
\end{verbatim}
2 Basic Types of Regression Models in the effects Package

The Effects() function supports three basic types of regression models:

- The preceding examples that use the lm() and lmer() functions are examples of glm-type models, which express, via a link function, the dependence of a discrete or continuous numeric response or of a binary response on a set of main effects and interactions among fixed-effect predictors comprising a linear predictor. The glm() function is the prototype for this kind of model. As shown in Table I, most of the regression functions currently supported by the effects package are of this type.

- multinom-type models are multinomial regression models that arise when the response is an unordered multi-category variable, also modeled, via a suitable multivariate link function, as a linear function of fixed-effect main effects and interactions. The prototype for multinom-type models is the multinom() function in the nnet package [Venables and Ripley 2002].

- polr-type models (i.e., ordinal regression models) are used for an ordered polytomous response variable. The prototype for polr-type models is the polr() function in the MASS package [Venables and Ripley 2002].

3 Supporting Specific Regression Functions

To support a specific class of regression models, say of class "foo" produced by the function foo(), one could write a method Effect.foo() for the S3 generic Effect() function. That approach is generally undesirable, for two reasons: (1) writing an Effect() method from scratch is a complicated endeavor; (2) the resulting object may not work properly with other functions in the effects package, such as plot() methods.

The effects package defines and exports several methods for the Effect() function, including a default method, and three specific methods corresponding to the three types of regression models introduced in the preceding section: Effect.lm() (which is also inherited by models of class "glm"), Effect.multinom(), and Effect.polr(). Moreover, Effect.default() works by setting up a call
to one of the three specific `Effect()` methods.\footnote{There are, as well, two additional specific `Effect()` methods provided by the `effects` package: `Effect.merMod()` for models produced by the `lmer()` and `glmer()` functions in the `lme4` package; and `Effect.svyglm()` for models produced by the `svyglm()` function in the `survey` package \cite{Lumley2004}. To see the code for these methods, enter the commands `getAnywhere("Effect.merMod")` and `getAnywhere("Effect.svyglm")`, after loading the `effects` package.}

The three basic `Effect()` methods collect information from the regression model of interest via a suitable method for the generic `effects::effSources()` function, and then use that information to compute effects and their standard errors. The required information is summarized in Table 2.

The default `effSources()` method simply returns `NULL`, which corresponds to selecting all of the defaults in Table 2. If that doesn’t work, it usually suffices to provide a suitable `effSources()` method. We illustrate by a few examples.

### 3.1 Examples

The following examples, with the exception of the last, are drawn directly from the `effects` package.

#### 3.1.1 `glmmPQL()`

Objects of class "glmmPQL", produced by MASS::glmmPQL() do not respond to the generic `family()` function, but the name of the family can be obtained from the call; thus:

```r
effSources.glmmPQL <- function(mod) {
  list(family = mod$family)
}
```

#### 3.1.2 `gls()`

The `weights` argument has different meaning for `gls()` in the `nlme` package \cite{Pinheiro2018} and for the standard R `glm()` function, and consequently the call must be modified to set `weights` to `NULL`:

```r
effSources.gls <- function(mod){
    cl <- mod$call
    cl$weights <- NULL
    list(call = cl)
}
```

#### 3.1.3 `betareg()`

The `betareg` function in the `betareg` package \cite{Grun2012} fits response data similar to a binomial regression but with beta errors. Adapting these models for use with `Effect()` is considerably more complex than the two previous examples:

```r
effSources.gls <- function(mod){
    coef <- mod$coefficients$mean
    vco <- vcov(mod)[1:length(coef), 1:length(coef)]
    # betareg uses beta errors with mean link given in mod$link$mean.
    # Construct a family based on the binomial() family
    fam <- binomial(link=mod$link$mean)
    list(family = fam)
}
```
Table 2: Values supplied by `effSources()` methods. In the table, the regression model object is called \textit{m}. For functions cited in the `insight` package see [Lüdecke et al.] (2019).

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{type}</td>
<td>The type of the regression model: one of &quot;glm&quot; (the default if \textit{type} isn't supplied), &quot;multinom&quot;, or &quot;polr&quot;.</td>
</tr>
<tr>
<td>\textit{call}</td>
<td>The call that created the regression model, which is generally returned by either \texttt{m$call} or \texttt{m@call} or \texttt{insight::get_call(m)}. The call is used to find the usual \texttt{data} and \texttt{subset} arguments that \texttt{Effect()} needs to perform the computation. See the discussion of \texttt{nlme::gls()} below for an example where the \texttt{call} must be modified.</td>
</tr>
<tr>
<td>\textit{formula}</td>
<td>The formula for the fixed-effects linear predictor, which is often returned by \texttt{stats::formula(m)} or \texttt{insight::find_formula(m)$conditional}.</td>
</tr>
<tr>
<td>\textit{family}</td>
<td>Many \texttt{glm}-type models include a family, with an error distribution and a link function. These are often returned by the default \texttt{stats::family(m)} or \texttt{insight::get_family(m)}.</td>
</tr>
<tr>
<td>\textit{coefficients}</td>
<td>The vector of fixed-effect parameter estimates, often returned by \texttt{coef(m)}. Alternatively \texttt{b &lt;- insight::get_parameters(m)} returns the coefficient estimates as a two-column matrix with parameter names in the first column, so \texttt{stats:setNames(b[,2], b[,1])} returns the estimates as a vector. For a \texttt{polr}-type model, coefficients should return the regression coefficients excluding the thresholds. The estimated covariance matrix of the fixed-effect estimates, often given by \texttt{stats::vcov(m)} or \texttt{insight::get_varcov(m)}. For a \texttt{polr}-type model, the covariance matrix should include both the regression coefficients and the thresholds, with the regression coefficients \textit{preceding} the thresholds.</td>
</tr>
<tr>
<td>\textit{vcov}</td>
<td></td>
</tr>
<tr>
<td>\textit{zeta}</td>
<td>The vector of estimated thresholds for a \texttt{polr}-type model, one fewer than the number of levels of the response. The default for a \texttt{polr}-type model is \texttt{zeta = m$zeta}.</td>
</tr>
<tr>
<td>\textit{method}</td>
<td>For a \texttt{polr}-type model, the name of a link supported by the \texttt{MASS::polr()} function: one of &quot;logistic&quot;, &quot;probit&quot;, &quot;loglog&quot;, &quot;cloglog&quot;, or &quot;cauchit&quot;. The default for a \texttt{polr}-type model is \texttt{method = &quot;logistic&quot;}.</td>
</tr>
</tbody>
</table>
# adjust the variance function to account for beta variance
fam$variance <- function(mu)
    f0 <- function(mu, eta) (1-mu)*mu/(1+eta)
do.call("f0", list(mu, mod$coefficient$precision))

# adjust initialize
fam$initialize <- expression(mustart <- y)

# collect arguments
args <- list(
call = mod$call,
formula = formula(mod),
family = fam,
coefficients = coef,
vcov = vco)
args

3.1.4 clm2()
The clm2() function in the ordinal package (Christensen, 2015) fits ordinal regression models, and so the aim is to create polr-type effects:

effSources.clm2 <- function(mod){
    if (!requireNamespace("MASS", quietly=TRUE))
        stop("MASS package is required")
polr.methods <- c("logistic", "probit", "loglog", "cloglog", "cauchit")
method <- mod$link
if(!(method %in% polr.methods))
    stop("'link' must be a 'method' supported by polr; see help(polr)")
if(is.null(mod$Hessian)){
    message("Re-fitting to get Hessian")
    mod <- update(mod, Hess=TRUE)
}
if(mod$threshold != "flexible")
    stop("Effects only supports the flexible threshold")
numTheta <- length(mod$Theta)
numBeta <- length(mod$beta)
or <- c( (numTheta+1):(numTheta + numBeta), 1:(numTheta))
list(
    type = "polr",
    formula = mod$call$location,
    coefficients = mod$beta,
zeta = mod$Theta,
    method = method,
    vcov = as.matrix(vcov(mod)[or, or]))
}
3.1.5 ivreg::ivreg()

Sometimes it doesn’t suffice to define an appropriate effSources() method, but it is still possible to avoid writing a detailed Effect() method. We use the ivreg() function (for instrumental-variables regression) in the ivreg package (Fox et al. [2021]) as an example; that package defines the following Effect.ivreg() method:

```r
Effect.ivreg <- function (focal.predictors, mod, ...) {
  mod$contrasts <- mod$contrasts$regressors
  NextMethod()
}
```

Here it is sufficient to set the contrasts element of the model object to conform to the way it is defined in "lm" objects. That works because "ivreg" objects inherit from class lm, and thus Effect.lm() is called by NextMethod().

References


