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:

```
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))
```

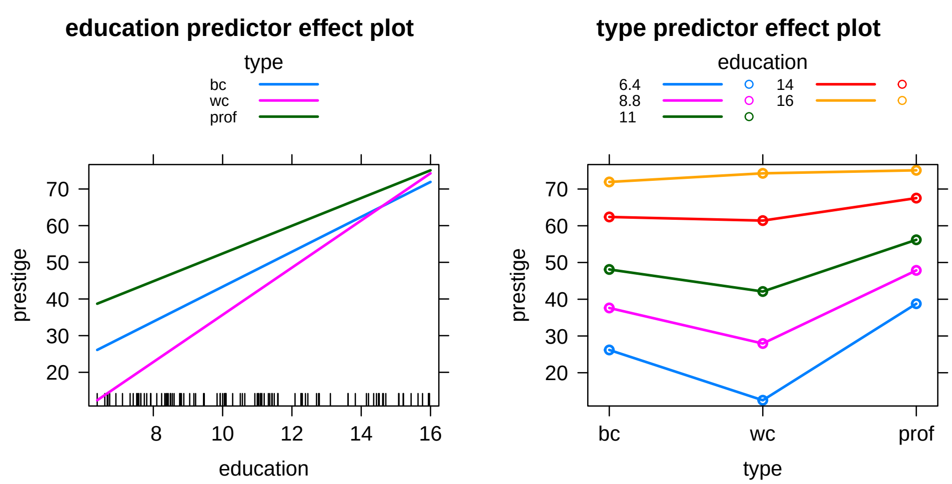


Table 1: R regression functions known to be compatible with the `Effect()` function. The name before the double-colon is the package that includes the function; for example `stats::lm()` means that `lm()` is in the **stats** package. In some cases, `Effect()` may support only a subset of regression models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

Function	Comments
`glm`-type models	
`stats::lm()`	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.
`stats::glm()`	Generalized linear models.
`nlme::lme()`	Linear mixed-effects models.
`nlme::gls()`	Linear models fit by generalized least squares.
`lmer::lmer()`	Linear mixed-effects models.
`lmer::glmer()`	Generalized linear mixed-effects models.
`survey::svyglm()`	Generalized linear models for complex survey designs.
`MASS::rlm()`	Linear regression models estimated by robust M or MM regression.
`MASS::glmmPQL()`	Generalized linear mixed-effects models via partial quadratic likelihood.
`robustlmm::rlmer()`	Robust linear mixed-effects models.
`betareg::betareg()`	Beta regression models for rates and proportions.
`ivreg::ivreg()`	Linear regression models estimated by instrumental variables (2SLS regression).
`glmmTMB::glmmTMB()`	Generalized linear mixed-effects regression models (similar to `lmer::glmer()` but accommodating a broader selection of models).
`multinom`-type models	
`nnet::multinom()`	Multinomial logistic-regression models. If the response has K categories, the response for `nnet::multinom()` 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.
`poLCA::poLCA()`	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.
`polr`-type models	
`MASS:polr()`	Ordinal logistic (proportional-odds) and probit regression models.
`ordinal::clm()`	Cumulative-link regression models (similar to, but more extensive than, `polr()`).
`ordinal::clm2()`	Updated version of `ordinal::clm()`.
`ordinal::clmm()`	Cumulative-link regression models with random effects.

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`.

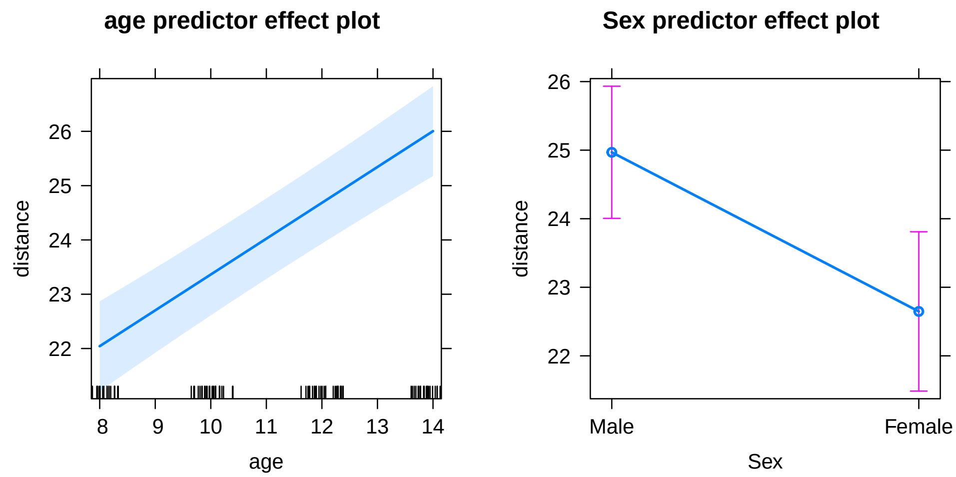
A linear mixed-effects model is a more complicated regression model, fit, for example, by the `lmer()` function in the **lme4** package (Bates et al., 2015):

```
data(Orthodont, package="nlme")
g2 <- lme4::lmer(distance ~ age + Sex + (1 | Subject), data = Orthodont)
summary(g2)
```

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

REML criterion at convergence: 437.5

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.7489 -0.5503 -0.0252  0.4534  3.6575

Random effects:
 Groups   Name        Variance Std.Dev.
 Subject  (Intercept) 3.267    1.807
 Residual             2.049    1.432
Number of obs: 108, groups:  Subject, 27

Fixed effects:
            Estimate Std. Error t value
(Intercept) 17.70671    0.83392  21.233
age          0.66019    0.06161  10.716
SexFemale   -2.32102    0.76142  -3.048

Correlation of Fixed Effects:
          (Intr) age
age       -0.813
SexFemale -0.372  0.000
```

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:

```
plot(predictorEffects(g2))
```



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 1, 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.[1]

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:

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

3.1.2 `gls()`

The `weights` argument has different meaning for `gls()` in the **nlme** package (Pinheiro et al., 2018) and for the standard R `glm()` function, and consequently the `call` must be modified to set `weights` to `NULL`:

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

3.1.3 `betareg()`

The `betareg` function in the **betareg** package (Grün et al., 2012) 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:

```
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)
```

[1] 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 (Lumley, 2004). To see the code for these methods, enter the commands `getAnywhere("Effect.merMod")` and `getAnywhere("Effect.svyglm")`, after loading the **effects** package.

Table 2: Values supplied by `effSources()` methods. In the table, the regression model object is called `m`. For functions cited in the **insight** package see Lüdecke et al. (2019).

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

```
# 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:

```
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

Bates, D., M. Mächler, B. Bolker, and S. Walker (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software 67*(1), 1–48.

Christensen, R. H. B. (2015). ***ordinal***—*Regression Models for Ordinal Data.* R package version 2015.6-28.

Fox, J., C. Kleiber, and A. Zeileis (2021). *ivreg: Instrumental-Variables Regression by '2SLS', '2SM', or '2SMM', with Diagnostics.* R package version 0.6-1.

Fox, J. and S. Weisberg (2019). *An* R *Companion to Applied Regression* (3nd ed.). Thousand Oaks CA: Sage.

Grün, B., I. Kosmidis, and A. Zeileis (2012). Extended beta regression in R: Shaken, stirred, mixed, and partitioned. *Journal of Statistical Software 48*(11), 1–25.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software 9*(1), 1–19. R package version 2.2.

Lüdecke, D., P. Waggoner, and D. Makowski (2019). insight: A unified interface to access information from model objects in R. *Journal of Open Source Software 4*(38), 1412.

Pinheiro, J., D. Bates, S. DebRoy, D. Sarkar, and R Core Team (2018). *nlme: Linear and Nonlinear Mixed Effects Models.* R package version 3.1-137.

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S* (4th ed.). New York: Springer-Verlag.