Predictor Effects Gallery

John Fox and Sanford Weisberg

October 17, 2018

Contents

Abstract

Predictor effect displays visualize the response surface of complex regression models by averaging and conditioning, producing a sequence of 2D line graphs, one graph or set of graphs for each predictor in the regression problem (Fox and Weisberg, 2019b,a). In this vignette we give examples of the use of the `effects` package in R for drawing effects plots, including many of the optional arguments.

1 Introduction

Predictor effect plots (Fox and Weisberg, 2019b) provide graphical summaries for fitted models with a linear predictor, including linear models, generalized linear models, linear mixed models and many others. They provide an alternative to tables of fitted coefficients that are can be much harder to interpret than are effects plots. These plots are implemented in R in the `effects` package documented in Fox and Weisberg (2019a). This vignette provides many examples of variations on these graphical displays that an be obtained with the `effects` package. Many of the details, and more complete description of the data sets used as examples, are provided in the references cited at the end of the vignette.

1.1 Effects and Predictor Effect Plots

We begin with an example of a multiple linear regression, using a data set in the `carData` package.

```
R> require(car) # also loads the carData package for the data set
R> lm1 <- lm(prestige ~ education + poly(women, 2) +
+                          log(income) * type, data=Prestige)
```

The model `lm1` is a linear model with response `prestige`, and continuous predictors `income`, `education`, `women`, and a factor predictor `type` with three levels. The predictor `education` represents itself in the linear model, and so it is both a predictor and a *regressor*, as defined in Fox and Weisberg (2019a, Sec. 4.1). The predictor `income` is represented by the regressor log(`income`). The variable `women`, a percentage between 0 and 100, is represented by regressors that define a polynomial of degree two using R's default orthogonal polynomials. The variable `type` is a factor with three levels so it is represented by two dummy variables defined using R's default. Finally the

formula includes an interaction between `income` and `type` defined by multiplying each of the regressors that represent `income` by each of the regressors that represent `type`.

A usual numeric summary of the fit of `lm1` is a table of estimated coefficients, using the `S` function in the `car` package,

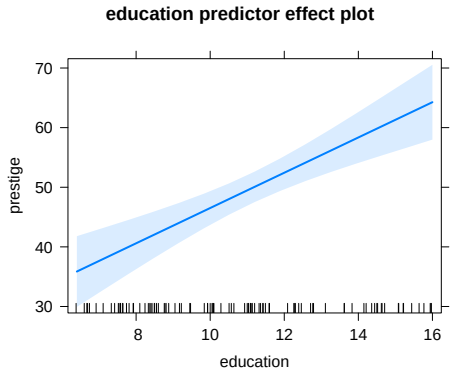
```
R> S(lm1)

Call: lm(formula = prestige ~ education + poly(women, 2) + log(income) * type,
         data = Prestige)

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          -137.500     23.522   -5.85  8.2e-08
education               2.959      0.582    5.09  2.0e-06
poly(women, 2)1        28.339     10.190    2.78   0.0066
poly(women, 2)2        12.566      7.095    1.77   0.0800
log(income)            17.514      2.916    6.01  4.1e-08
typeprof               74.276     30.736    2.42   0.0177
typewc                  0.969     39.495    0.02   0.9805
log(income):typeprof   -7.698      3.451   -2.23   0.0282
log(income):typewc     -0.466      4.620   -0.10   0.9199

Residual standard deviation: 6.2 on 89 degrees of freedom
  (4 observations deleted due to missingness)
Multiple R-squared: 0.879
F-statistic: 81.1 on 8 and 89 DF,  p-value: <2e-16
   AIC    BIC
646.26 672.11
```

Interpretation of this output is straightforward only for the predictor `education`, where an increase of `education` by one year holding other predictors fixed corresponds to an estimated expected increase in the response of 2.959 units. Even ignoring the interaction, the logarithm complicates the interpretation of the effect of `income`. The predictor `women` is represented by two regressors, so the effect of `women` requires examining two coefficient estimates that are interpretable only by experts, and even if raw polynomials were used, via `poly(women, 2, raw=TRUE)` in place of `poly(women, 2)`, interpretation of the effect of `women` is complicated. Understanding the coefficients for the main effect of `type` depends on the contrasts used to define the effect. The contrasts can be changed by the user, and the default contrasts in R are different from the default contrasts used by `SAS` or other programs, so the coefficients cannot be reliably interpreted without information not present in the summary shown. Finally, the interaction further complicates the interpretation of the effect of either `income` or `type` because the interaction coefficients need to be interpreted along with the main effect coefficients. In other words, summarization of the effects of predictors using tables of coefficient estimates is often incomplete. Effects, and particularly plots of

effects, can in many instances visualize effects clearly. This conclusion is even clearer with linear predictors including interactions, as illustrated later in this *Gallery*.

Predictor effects summarize the role of a selected *focal* predictor in a fitted regression model. The `predictorEffect` function is used to compute and summary of the regression that is needed, and then the `plot` method is used to draw the plot.

```
R> require(effects)
R> e1.lm1 <- predictorEffect("education", lm1)
R> plot(e1.lm1)
```



This plot visualizes the partial slope for `education`, that for each year increase in `education`, the fitted `prestige` increases by 2.959 points, when the other predictors are held fixed. The intercept of the line is determined by the choices for averaging over the fixed variables, but for any choice of averaging method the slope of the line would be the same. The shaded area is a pointwise confidence band for the fitted values computed using the standard errors.

The information that is needed to draw the plot is computed by the `predictorEffect` method. The minimal syntax for `predictorEffect` is the quoted name of a predictor followed by the name of the fitted model. The essential purpose of this function is to compute fitted values from the model with `education` varying and all other predictors fixed at some typical value (Fox and Weisberg, 2019a, Sec. 4.3). The command below displays the values of the regressors for which fitted values were computed, including the column of ones for the intercept:

```
R> e1.lm1$model.matrix
```

```
  (Intercept) education poly(women, 2)1 poly(women, 2)2 log(income) typeprof
1           1       6.4     0.000020998        -0.13496      8.8449  0.31633
2           1       8.8     0.000020998        -0.13496      8.8449  0.31633
3           1      11.0     0.000020998        -0.13496      8.8449  0.31633
4           1      14.0     0.000020998        -0.13496      8.8449  0.31633
5           1      16.0     0.000020998        -0.13496      8.8449  0.31633
   typewc log(income):typeprof log(income):typewc
1 0.23469               2.7979            2.0758
2 0.23469               2.7979            2.0758
3 0.23469               2.7979            2.0758
```

```
4 0.23469                2.7979              2.0758
5 0.23469                2.7979              2.0758
attr(,"assign")
[1] 0 1 2 2 3 4 4 5 5
attr(,"contrasts")
attr(,"contrasts")$type
[1] "contr.treatment"
```

Only five fitted values were computed, each with a different value of the focal predictor `education`. The remaining regressors have the same value for each fitted value. The value for log(`income`) is the logarithm of the sample mean `income`, the values for the regressors for `women` are computed at the average of `women` in the data, and the fixed value for the regressors for `type` are effectively taking a weighted average of the fitted values for the three levels of `type`, with weights proportional to the number of observations in each level of the factor. Differences in the fitted values are due to `education` alone because all the other predictors, and their corresponding regressors, are fixed. Thus the output gives the marginal effect of `education` with all other predictors fixed.

The computed fitted values can be viewed using

```
R> as.data.frame(e1.lm1)
```

```
  education    fit     se  lower  upper
1       6.4 35.864 2.9865 29.930 41.798
2       8.8 42.965 1.8275 39.334 46.596
3      11.0 49.474 1.2842 46.923 52.026
4      14.0 58.351 2.1500 54.079 62.623
5      16.0 64.268 3.1604 57.989 70.548
```

The values in the column `education` are the values of `education`. The remaining columns are the fitted values, their standard errors, and lower and upper end points of confidence intervals for the fitted values. The predictor effects plot is simply a plot of the fitted values on the vertical axis versus the focal predictor on the horizontal axis. When the focal predictor is continuous, as it is here, a line is drawn between the fitted values that represents the fitted `prestige` for any value of `education`.

We turn next to the predictor effects plot for `income`. According to the fitted model, the effect of `income` may depend on `type` due to the interaction between them, so simply averaging over `type` would be incorrect. Rather, we must allow both `income` and `type` to vary, fixing the other predictors at their mean or other typical value.

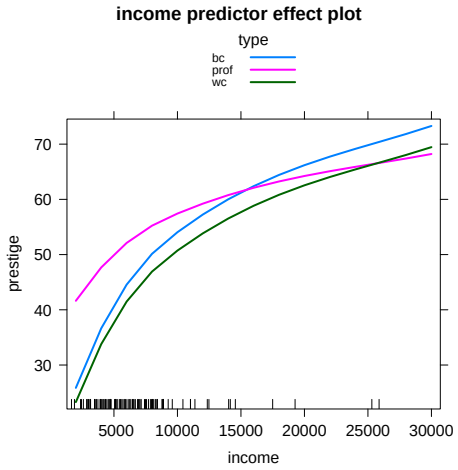
```
R> e2.lm1 <- predictorEffect("income", lm1)
R> as.data.frame(e2.lm1)
```

```
   income type    fit     se  lower  upper
1    2000   bc 25.863 3.3037 19.299 32.428
```

```
2    8000   bc 50.142 2.3737 45.426 54.859
3   10000   bc 54.050 2.7996 48.487 59.613
4   20000   bc 66.190 4.4814 57.285 75.094
5   30000   bc 73.291 5.5708 62.222 84.360
6    2000 prof 41.630 4.4812 32.726 50.534
7    8000 prof 55.237 2.3316 50.605 59.870
8   10000 prof 57.428 2.4552 52.549 62.306
9   20000 prof 64.231 3.6170 57.045 71.418
10  30000 prof 68.211 4.5680 59.135 77.288
11   2000   wc 23.290 4.5674 14.214 32.365
12   8000   wc 46.922 2.3106 42.331 51.513
13  10000   wc 50.726 3.0575 44.651 56.802
14  20000   wc 62.543 5.7716 51.075 74.011
15  30000   wc 69.455 7.4432 54.665 84.244
```

The fitted values are now evaluated at 15 points, five selected values of `income` for each level of `type`. The predictor effects plot is

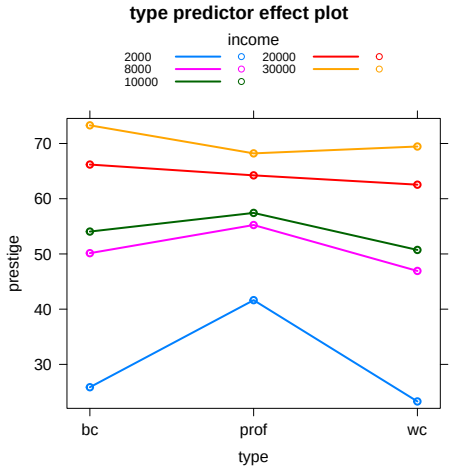
```
R> plot(e2.lm1, lines=list(multiline=TRUE))
```



The focal predictor is displayed on the horizontal axis. There is a separate line fit to the fitted values for each level of `type`. The lines are curves rather than straight lines because `income` is used in the model in log-scale, but displayed in the predictor effects plot in arithmetic scale. The lines are not parallel because of the interaction. For `type=prof` the fitted values of `prestige` are relatively high for lower values of `income`, and are relatively unaffected by increasing values of `income`.

The predictor effects plot for `type` uses the same fitted values as the plot for `income`, but we now get five lines, one for each of the values of `income` selected by the program.

```
R> plot(predictorEffect("type", lm1),
+         lines=list(multiline=TRUE))
```



We applied both the `predictorEffect` and `plot` functions in the same command. Since the horizontal axis is now a factor, the fitted values are displayed explicitly, and the lines that join them are merely a visual aid. Fitted `prestige` increases with `income` for all levels of `type`, but as we found before when `type=prof` the fitted `prestige` is relatively high for lower `income`.

These first plots used only defaults to computing `predictorEffects`, and, apart from the `multiline` argument to put all the lines in the same graph, used defaults for drawing the plots. We further elaborate the plots in this *Gallery*.

1.2 General Outline of Using Predictor Effects Plots

Using the `effects` package to draw plots usually follows this outline:

1. Fit a regression model with a linear predictor. The package supports models created with `lm`, `glm`, `lmer`, `lme`, `glmer`, and many others.

2. The regression model created at the first step is then used as input to either `predictorEffect`, to get the effects for one predictor, or `predictorEffects`, to get effects for one or more predictors, to do the averaging needed to get the values that will ultimately be plotted. There are many arguments for customizing the computation of the effects.

3. Use the generic `plot` function to draw the graph or graphs based on the object created in Step 2.

In Section 1.3 we discuss the functions described at step 2 above, and many of the arguments for their use. In Section **??** we discuss and provide examples of the many arguments to the `plot` method for effects objects. Adding residuals to effects plots is discussed in Section 4 Finally in Section 7 we discuss the connection between the `effects` package and the `emmeans` package (Lenth, 2018) that provides tests that can complement predictor effects plots.

1.3 How `predictorEffect` Chooses Conditioning Predictors

Suppose you select one focal predictor for which you want to draw a predictor effects plot. The function `predictorEffects` divides the predictors in a formula into three groups:

1. The focal predictor
2. The *conditioning group*, consisting of all predictors with at least one interaction in common with the focal predictor.
3. The *fixed group*, consisting of all predictors with no interactions in common with the focal predictor.

The predictors in the fixed group are all evaluated at a typical value, usually a mean, effectively averaging out the influence of these variables on fitted value. Fitted values are computed for all combinations of levels of the predictors in the conditioning group, with continuous predictors in the conditioning group replaced a discrete version with a few levels spanning the range of the continuous predictor, for example replacing years of `education` by a discrete variable with levels 8, 12, 16 years.

Suppose we have a fitted model with R formula

```
y ~ x1 + x2 + x3 + x4 + x2:x3 + x2:x4
```
(1)

There are four possible predictor effects plots, one for each predictor selected as the focal predictor:

Focal Predictor	Conditioning Group	Fixed Group
`x1`	none	`x2`, `x3`, `x4`
`x2`	`x3`, `x4`	`x1`
`x3`	`x2`	`x1`, `x4`
`x4`	`x2`	`x1` `x3`

The predictor `x1` does not interact with anything, so its conditioning set is empty and all the remaining predictors are averaged over. `x2` interacts with both `x3` and `x4`, `x3` interacts only with `x2` and `x4` interacts with `x2`.

1.4 The `Effect` Function

Prior to late 2018 the primary function in `effects` for computing and displaying effects was the `Effect` function. Whereas the `predictorEffect` function automatically determines the condition group and the fixed group, the `Effect` function puts that burden on the user. Each call to `predictorEffect` is equivalent to a specific call the the `Effect` function as follows. Suppose the `m` is the name of a fitted model

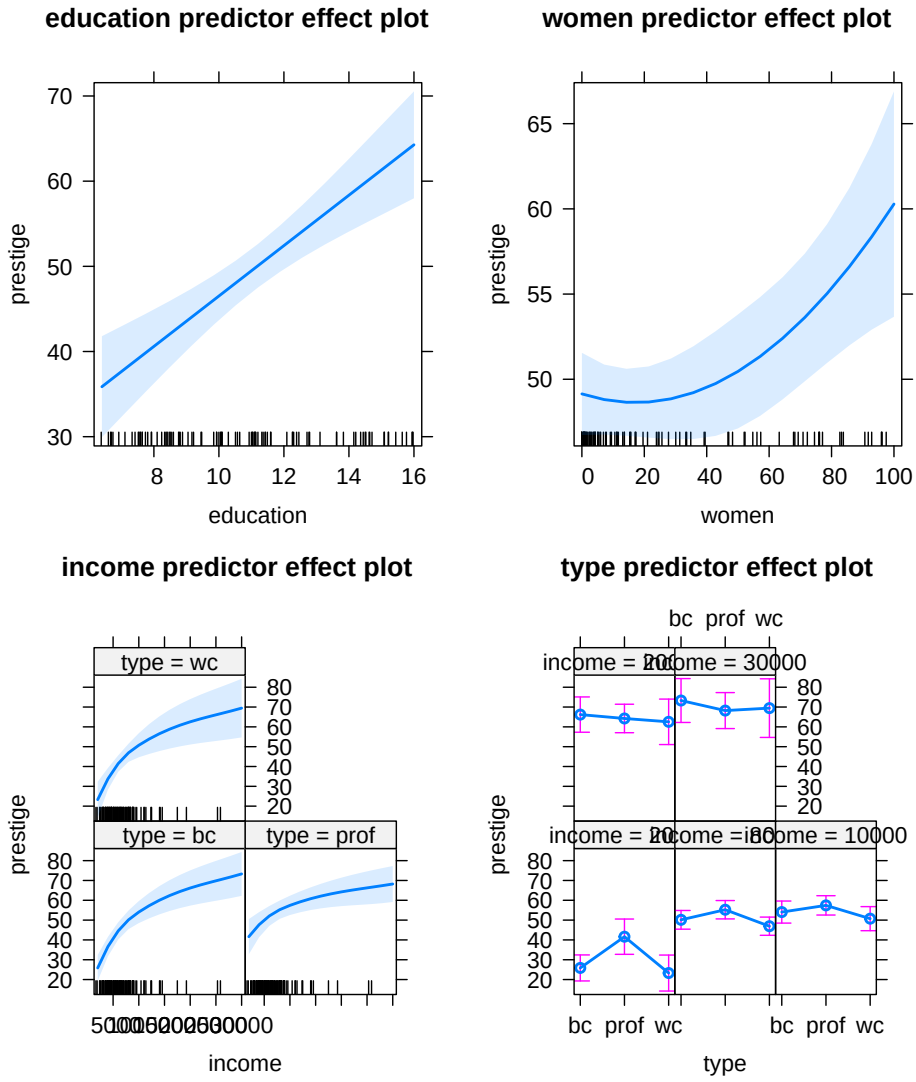
```
R> predictorEffect("x1", m) # equivalent to Effect("x1", m)
R> predictorEffect("x2", m) # equivalent to Effect(c("x2", "x3", "x4"), m)
R> predictorEffect("x3", m) # equivalent to Effect(c("x3", "x2"), m)
R> predictorEffect("x4", m) # equivalent to Effect(c("x4", "x2"), m)
```

The `predictorEffect` function determines the correct call to `Effect` based on the choice of focal predictor and on the formula of main effects and interactions. It then uses the `Effect` function to do the computing. As a result most of the option for using `predictorEffect` are shown on the help page for `help("Effect")` rather than the help page for `predictorEffect`.

1.5 The `predictorEffects` Function

This function, ending with an "s", computes the values needed for one or more predictor effects plots. For example,

```
R> eall.lm1 <- predictorEffects(lm1)
R> plot(eall.lm1)
```



This presents all predictor effects plots for this model in an array of plots. The plots for `income` and `type` have a separate plot for each level of the conditioning variable

because the default option `lines=list(multiline=FALSE)` was used. Confidence bounds are shown for all the variables by default when `multilines=FALSE`.

The result `eall.lm1` is a list with four elements, so `eall.lm1[[1]]` is the summary needed for the first predictor effects plot, `eall.lm1[[2]]` for the second plot, and so on. The following equivalent commands will draw an array of predictor effects plots:

```
R> plot(eall.lm1)
R> plot(predictorEffects(lm1))
R> plot(predictorEffects(lm1, ~ income + education + women + type))
```

If you want only predictor effects plots for `type` and `education`, in that order, you could enter

```
R> plot(predictorEffects(lm1, ~ type + education))
```

The commands

```
R> plot(predictorEffects(lm1, ~ women))
R> plot(predictorEffects(lm1)[[3]])
R> plot(predictorEffect("women",lm1))
```

would all produce the same plot.

Predictor effects plots in an array can be a useful shortcut for drawing many plots quickly, but can lead to problems with the displayed graphs. For example, the horizontal axis labels for the plot for `income` are overprinted, and the labels at the top of the plots for `type` with conditioning variable `income` are larger than the available space. These problems can often be fixed using the arguments we describe later in this *Gallery*.

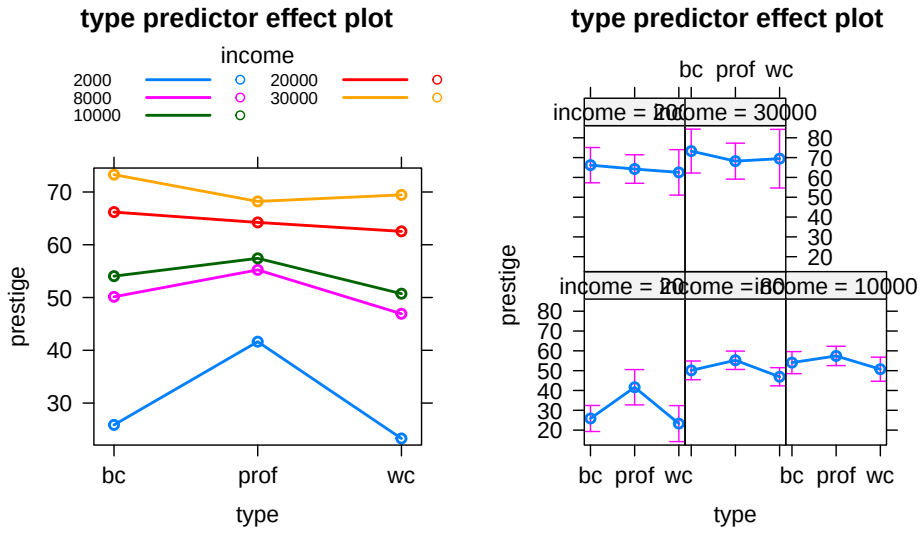
2 Options for the `predictorEffect` Function

The help page `help("Effect")` describes the options that are available for the `Effect` function. The following is a list of the arguments available to modify the behavior of the `predictorEffect` and `Effect` functions. The help file `help("Effect")` is somewhat more comprehensive, for example giving exceptions for use with `svyglm` objects, or when plotting residuals.

2.1 Options for Predictors in the Conditioning Group: `xlevels`

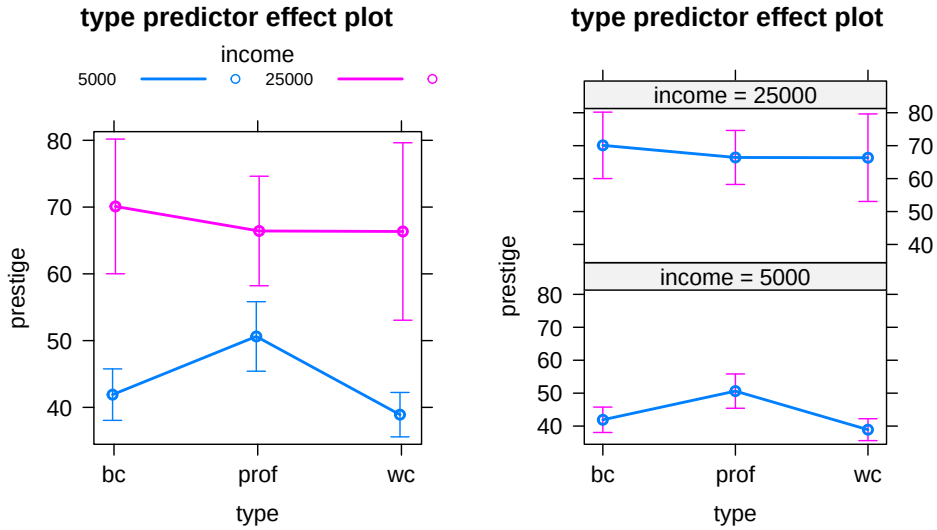
Numeric predictors in the conditioning group need to be discretized to draw the predictor effects plot. For example the predictor effects plot for `type` will consist of a separate line, or separate graph, for each discrete value of `income`,

```
R> e3.lm1 <- predictorEffect("type", lm1)
R> plot(e3.lm1, lines=list(multiline=TRUE))

R> plot(e3.lm1, lines=list(multiline=FALSE))
```



The numeric conditioning predictor `income` was evaluated by default at five equally spaced values `income`, and then rounded to "nice" numbers. In this example using two values of 5000 and 25000 might display the graph more clearly,

```
R> e3.lm1 <- predictorEffect("type", lm1,
+                            xlevels=list(income=c(5000, 25000)))
R> plot(e3.lm1, lines=list(multiline=TRUE),
+         confint=list(style="bars"))

R> plot(e3.lm1,
+         lines=list(multiline=FALSE))
```



The argument `xlevels` is a list of sub-arguments that control how numeric predictors are discretized when used in the conditioning group. For example, `xlevels=list(x1=c(2, 4, 7), x2=6)` would use the values 2, 4 and 7 for the levels of a predictor named `x1`, use six equally spaced levels for the levels of a predictor named `x2`, and use the

default for any other numeric predictors. Numeric predictors in the fixed group are not affected by the `xlevels` argument.

The number of points at which the focal predictor will be evaluated is also controlled by the `xlevels` argument. Increasing the number of points may make plotted lines smoother. See `help("plot.eff")` for information on the `quantiles` argument that provides an alterntive method of setting `xlevels`.

2.2 Predictors in the Fixed Group

2.2.1 Factor Predictors

Predictors in the fixed group are replaced by a fixed "typical" value. Fitted values are then computed using the fixed values for the fixed group, and varying the values in the conditioning group and for the focal predictor. The user can control how the fixed values are determined using the `fixed.predictors` argument. This argument has a list of sub-arguments to allow for controlling each predictor in the fixed group, with different rules for factors and numeric predictors.

For a fixed factor, imagine computing the fitted values evaluting the fixed factor at each of its levels. The fitted value that is used is the weighed average of this within-level fitted values, with weights proportional to the number of observations at each level of the factor. This is an appropriate notion of "typical" if the data at hand are viewed as a random sample from a population, and so the sample fraction at each level estimates the population fraction. A second approach is to average with equal weighting at each level. This may be particularly appropriate in designed experiments in which the levels of a factor are assigned by an investigator. This latter method obtained by setting `fixed.predictors=list(given.values="equal")`.

You can construct other weighting schemes for averaging over the levels of a factor, as described on the help page for `Effect`.

2.2.2 Numeric Predictors

For numeric predictors in the fixed group the default method of selecting a typical value is the `mean` function. The specification `fixed.predictors = list(typical=median)` would use the median; in general `typical` can be any function that evaluates a vector and return a single number.

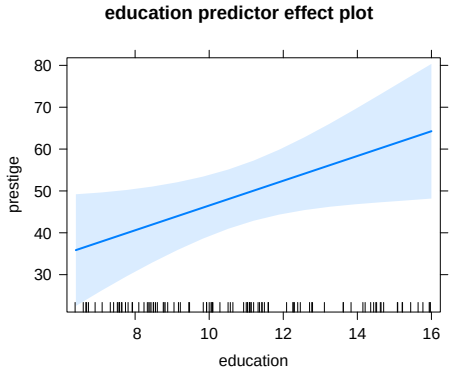
Other sub-arguments to `fixed.predictors` apply to the use of offsets, and to the `survey` package; see the help file for `Effect`.

2.3 Standard Errors and Confidence Intervals

Standard errors and confidence intervals for fitted values are are computed by default. The default corresponds to setting the argument `se = list(compute=TRUE, type="pointwise", level=.95)`. Setting `se=FALSE` omits standard errors, `type= "scheffe"` uses wider Scheffé intervals that adjust for multiple testing, `level=.8` gives 80% intervals.

Standard errors are based on the "usual" estimated variance matrix of the regression coefficient estimates. You can replace this with some other estimate of variance, such as from a bootstrap or a sandwich estimator, by setting the argument `vcov.` to equal either the name of a function that returns a sample covariance matrix such as `hccm` for linear models, or equal to a matrix of the correct size.

```
R> require(effects)
R> e4.lm1 <- predictorEffect("education", lm1,
+                            se=list(type="scheffe", level=.99),
+                            vcov. = hccm)
R> plot(e4.lm1)
```



This plot displays 99% Scheffé intervals based on the robust covariance matrix computed with the sandwich method; see `help("hccm")`.

2.4 Residuals

The argument `residuals`, or `partial.residuals`< is used to add partial to effects plots; see Section 4.

3 Arguments for Plotting Predictor Effects

The arguments described in Section 2 are for the `predictorEffect` function for changing the computations done by the `Effect` function, such as methods for averaging and fixing predictors, and computing standard errors. Arguments for the `plot` method are described in this section, and these change the appearance of the plot or the quantities plotted. The help page for these options at `help("plot.eff")` is more comprehensive than the examples we provide here.

In late 2018 we reorganized the `plot` method by combining arguments into five major groups of like arguments, with the goal of simplifying using the many arguments that are available. For example, the `lines` group of arguments is a list of subarguments for determining line type, color and width, whether or not multiple lines should be drawn on the same graph, and whether lines should be smoothed before plotting. The defaults for the arguments are the choices we generally find the most useful, but they will not be the best choices in all circumstances. The cost of the

reorganization is the necessity of getting all the parentheses right, since the major arguments all require a list, and some of the sub-arguments are lists as well.

In addition to the five argument groups the plot method accepts the arguments `main` for the main title of a plot and `id` for identifying points in plots that includes residuals, Section 4.

Finally, the `plot` method has a number of "legacy" arguments shown in the help file. They have been kept so existing scripts using effects would not break, but they are all duplicated as sub-arguments in the argument groups. The legacy arguments work, but they may not be supported forever, so learners should use the arguments and sub-arguments.

3.1 `axes` Group: Change Axis Characteristics

The `axes` argument group has two major sub-arguments, `x` for the horizontal axis, `y` for the vertical axis, and two minor sub-arguments, `grid` argument adds a background grid to the plot, and `alternating`, for changing the placement of axis labels in some plots.

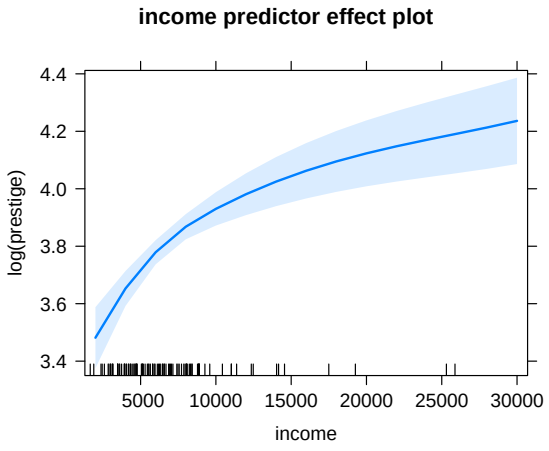
3.1.1 `x`: Horizontal Axis Modification

We introduce a another linear model as an example:

```
R> lm2 <- lm(log(prestige) ~ log(income) + education + type, Prestige)
```

The default predictor effects plot for `income` is

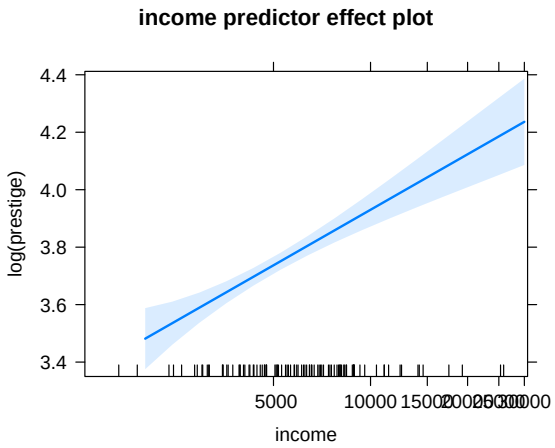
```
R> plot(predictorEffects(lm2, ~ income))
```



The plot is a curve because the predictor `income` is represented by its logarithm in the formula, but the default predictor effects plot uses the predictor, not the regressor, on the horizontal axis. The `x` sub-argument can be used transform the horizontal axis, for example to replace `income` by log(`income`):

```
R> plot(predictorEffects(lm2, ~ income),
+     axes=list(
```

```
+       x=list(income=list(transform=list(trans=log, inverse=exp)))
+       ))
```



The transformation changed the scale on the horizontal axis to log-scale, but left the tick labels in arithmetic scale. The graph is now a straight line because of the change to log-scale. This plot has several undesirable features with regard to range on the horizontal axis and over-printing of tick marks that the user can modify using additional arguments.

A more elaborate version that illustrates all the sub-arguments to `x` is

```
R> plot(predictorEffects(lm2, ~ income),
+     main="Transformed Plot",
+     axes=list(
+        grid=TRUE,
+        x=list(rotate=30,
+               rug=FALSE,
+               income=list(transform=list(trans=log, inverse=exp),
+                           lab="income, log-scale",
+                           ticks=list(at=c(1000, 2000, 5000, 10000, 20000)),
+                           lim=c(2000, 20000))
+        )))
```

We used the top-level argument `main="Transformed Plot"` to set the title of the plot. The `axes` argument is a list with two sub-arguments, `grid` to turn on the background grid, and `x` to modify the horizontal axis.

The `x` sub-argument is itself a list with three elements. The sub-arguments `rotate` and `rug` set the rotation angle for the tick labels and suppress the rug plot, respectively. The additional sub-argument is a list called `income`, and its name is the same as the name of the focal predictor. If you were plotting many predictor effects plots you would supply one list named after each of the focal predictors. All of the sub-arguments for `income` are displayed in the example code above. The sub-argument `transform=list(trans=log, inverse=exp)` specifies how to transform the x-axis, using the `inverse` specification. The `ticks` and `lim` arguments set the tick marks and range for the plot.

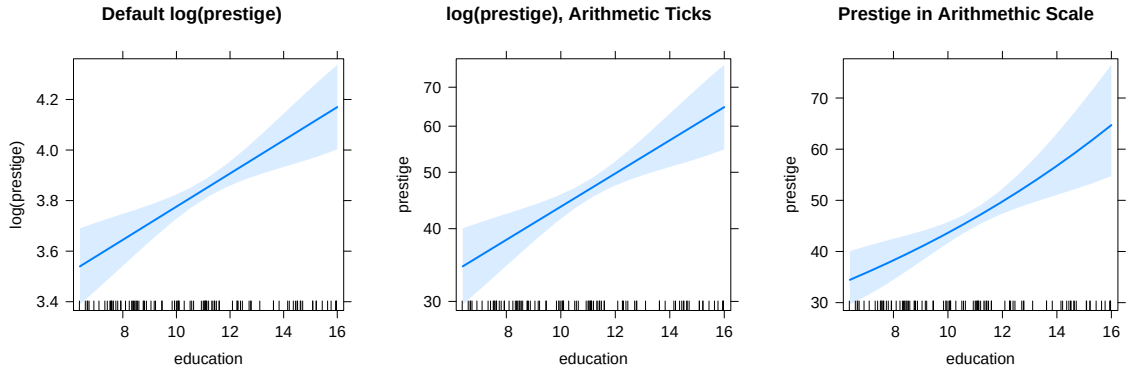
This is a *very complex command* that allows you to fine-tune a graph to look the way you want. You will likely have problems getting the parentheses in the right places because many of the arguments are lists of lists. Be patient.

3.1.2 `y`: Change the Vertical Axis, Linear Models

The model `lm2` has a transformed response log(`prestige`), and "untransforming" the response to arithmetic scale may be desirable. This can be accomplished with the `y` sub-argument that has two sub-arguments named `transform` and `type`. There are three options for drawing the predictor effects plot for a predictor like `education`.

```
R> # default:
R> plot(predictorEffects(lm2, ~ education),
+         main="Default log(prestige)")

R> # Change only tick-mark labels to arithmetic
R> plot(predictorEffects(lm2, ~ education),
+         main="log(prestige), Arithmetic Ticks",
```

```
+         axes=list(y=list(transform=list(trans=log, inverse=exp),
+                          lab="prestige", type="rescale")))

R> # Replace log(presige) by prestige
R> plot(predictorEffects(lm2, ~ education),
+         main="Prestige in Arithmethic Scale",
+         axes=list(y=list(transform=exp, lab="prestige")))
```



The first plot is the default plot with a log-response. In the second plot the `transform` sub-argument specifies the transformation and its inverse, and the code `type="rescale"` changes the tick marks to arithmetic scale. In the third version with `transform=exp, lab="prestige"` the vertical axis now is in arithmetic scale, not log scale, although that may not be completely obvious in the example because $\log(x)$ is approximately linear over the range of 30 to 80 for `prestige` values in this data set. The help page gives a somewhat more detailed explanation.

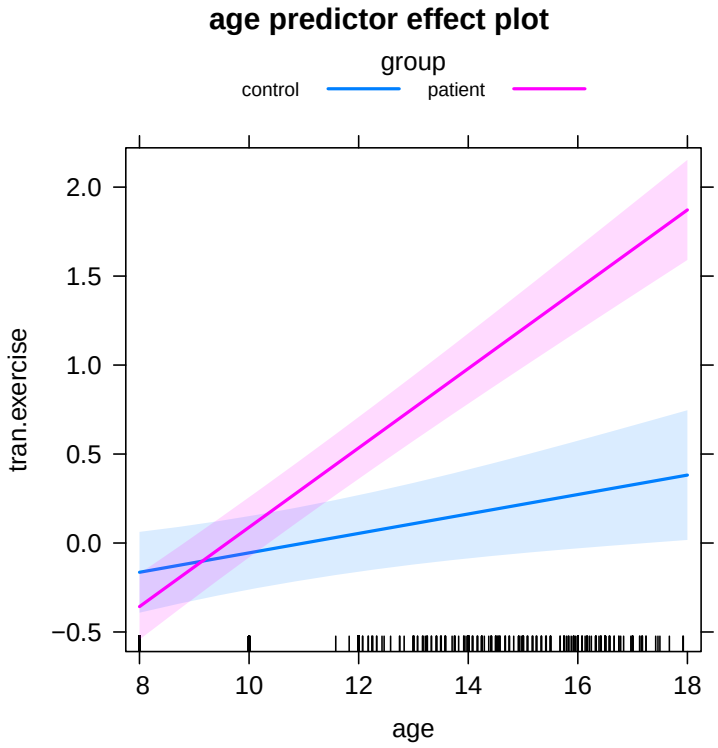
As a second example we will reconstruct Figure 7.10 in (Fox and Weisberg, 2019a, Sec. 7.2). In that section we fit a linear mixed-effects model using a data frame called `Blackmore` in the `carData` package. We selected a response for the regression using the Box-Cox procedure to transform a response, and since the response variable in this example, hours of `exercise`, has zero values we used a family of transformations called `bcnPower`, a modification of the Box-Cox power family that allows for zero or negative responses, summarized briefly in Fox and Weisberg (2019a, Sec. 3.4) and more thoroughly in Hawkins and Weisberg (2017). The fitted model we used was

```
R> require(lme4) # for the lmer function

Loading required package: lme4

Loading required package: Matrix

R> Blackmore$tran.exercise <-
+      bcnPower(Blackmore$exercise, gamma=0.25, 0.1)
R> mm1 <- lmer(tran.exercise ~ I(age-8)*group +
+                  (I(age - 8) | subject), data=Blackmore)
```

This model has predictors `age` and `group`. The model fit is a linear mixed model with random slope for `age` for each subject. The response variable is a transformation of `exercise` similar to the fourth root with adjustment for zero-values; see `help("bcnPower")`.

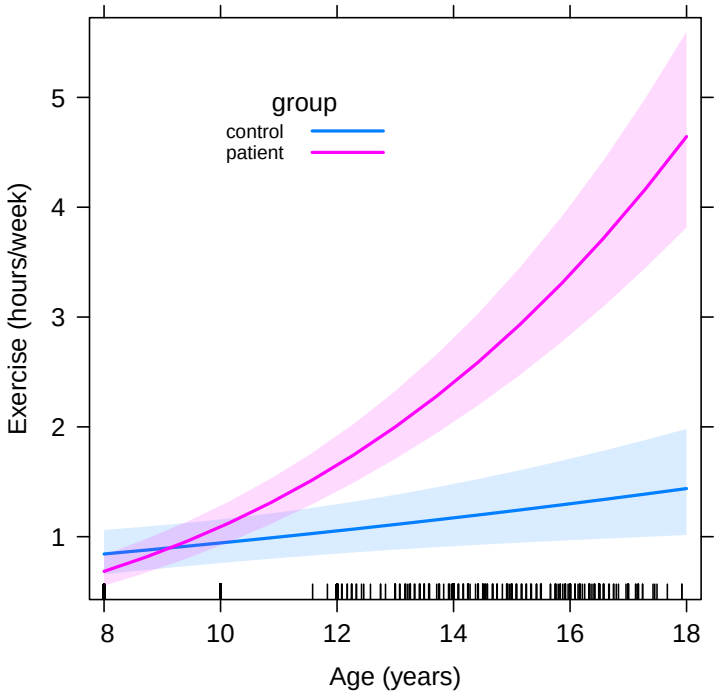
The predictor effects plot for the fixed effect of `age` is

```
R> e1.mm1 <- predictorEffect("age", mm1)
R> plot(e1.mm1, lines=list(multiline=TRUE), confint=list(style="auto"))
```



The plot clearly shows the difference between the `control` and `patient` groups, with the fitted response having larger slope. The graph is misleading because the vertical axis is more or less in the scale of the fourth-root of hours, so untransforming may be more informative. Because the `bcnPower` transformation is complex the `car` package includes a function `bcnPowerInverse` to reverse the transformation:

```
R> f.trans <- function(x) bcnPower(x, lambda=0.25, gamma=0.1)
R> f.inverse <- function(x) bcnPowerInverse(x, lambda=0.25, gamma=0.1)
R> plot(e1.mm1, lines=list(multiline=TRUE),
+         confint=list(style="auto"),
+         axes=list(y=list(transform=list(trans=f.trans,
+                                         inverse=f.inverse),
+                          type="response")),
+         lattice=list(key.args=list(x=.20, y=.75, corner=c(0, 0),
+                                    padding.text=1.25)),
+         xlab="Age (years)", ylab="Exercise (hours/week)", main=""
+    )
```



The response scale is now in hours per week, and we see that the fitted values increase more quickly in the patient group for older subjects. We used several additional arguments in this plot to match Fox and Weisberg (2019a, Fig. 7.10) including moving the key, Section 3.4.1, and changing the axis labels and removing the main title to the plot using standard commands for most `plot` methods. The code shown for the plot in Fox and Weisberg (2019a) is somewhat different because it uses legacy arguments. Either code will work and produce the same plot.

3.1.3 `y`: Change the Vertical Axis, Generalized Linear Models

Transforming the vertical axis for generalized linear models also uses the `y` sub-argument. You do not need to specify the `transform` argument because the program obtains the right functions from the regression model's `family` argument. The `type` sub-argument has the same three possible values as for linear models, but their interpretion is somewhat different:

1. Predictor effects plots in `type="link"` or linear predictor scale, in which the horizontal axis of each plot is a predictor and the vertical axis is in the scale of the linear predictor. For logistic regression, the vertical axis is the log-odds scale. For Poisson regression with log-link the vertical axis is the log-mean scale.

2. Predictor effects plots in `type="response"` or mean scale are obtained by "un-transforming" the y axis using the inverse of the link function. For the log-link, this corresponds to transforming the y axis and plotting $\exp(y)$. For logistic regression, since if $y = \log(p/(1-p))$, then solving for p gives $p =$

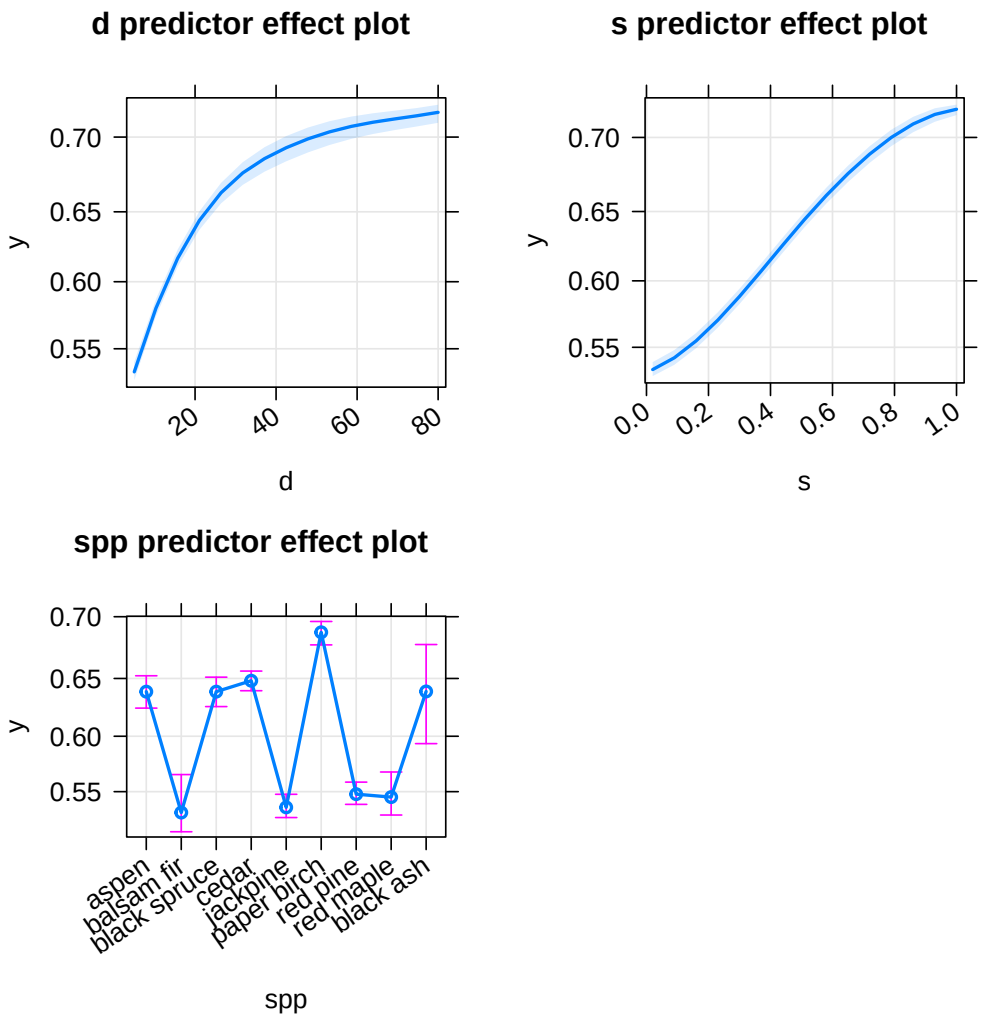
$\exp(y)/(1+\exp(y))$, so the plot in mean scale uses $\exp(y)/(1+\exp(y))$ on the vertical axis.

3. We also define a third version called `type="rescale"` that plots in linear predictor scale, but labels the tick marks on the vertical axis in mean scale.

The default is `"rescale"`, although it is the hardest to explain.

We use the `Blowdown` data from the `alr4` package to supply example plots. These data model the probability of blowdown, a tree being uprooted, as a function of diameter `d` of the tree, local severity `s` of the storm and species `spp` of the tree as the result of a major straight-line wind storm in the Boundary Waters Canoe Area Wilderness in 1999. We fit a main-effects model and then display all three predictor effects plots.

```
R> require(alr4) # to get the data

Loading required package: alr4

R> gm1 <- glm(y ~ log(d) + s + spp, binomial, Blowdown)

R> plot(predictorEffects(gm1),
+         axes=list(grid=TRUE, x=list(rug=FALSE,rotate=35)))
```



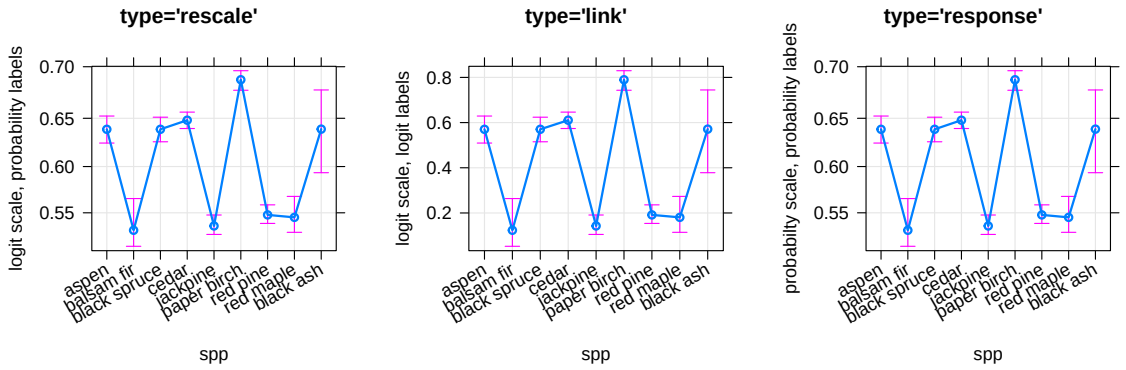
The `rug` sub-argument to `x` suppressed the rug plot, and the `grid` sub-argument to `axes` adds background grids. The `rotate` argument prints the horizontal tick labels at an angle to avoid overprinting.

Interpretation of glm predictor effects plots in link scale is similar to predictor effects plots for linear models, and all the modifications previously describe can be used for these plots. Since the default is `type="rescale"`, the vertical axis is in linear predictor scale, but the vertical axis labels are in probability scale, so the tick-marks are not equally spaced.

The next three plots show the possible values of `type`.

```
R> e1.gm1 <- predictorEffect("spp", gm1)
R> plot(e1.gm1, main="type='rescale'",
+         axes=list(y=list(type="rescale",
+                          lab="logit scale, probability labels"),
+                   x=list(rotate=30),
+                   grid=TRUE))

R> plot(e1.gm1, main="type='link'",
+         axes=list(y=list(type="link",
```

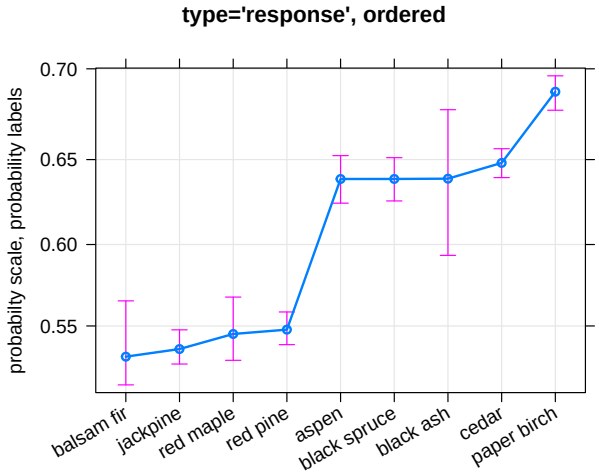
```
+                         lab="logit scale, logit labels"),
+                   x=list(rotate=30),
+                   grid=TRUE))

R> plot(e1.gm1, main="type='response'",
+        axes=list(y=list(type="rescale", grid=TRUE,
+                         lab="probabilty scale, probability labels"),
+                   x=list(rotate=30),
+                   grid=TRUE))
```



The first two plots show the same graph, but in the first the tick-marks are unequally spaced and are in probability scale, and in the second the tick-marks are equally spaced and are in log-odds scale. In the third plot the graph as been transformed to probability scale, and the corresponding tick-marks are now equally spaced.

The plot for species would be more helpful if the levels of the factor were ordered according to the estimated log-odds of blowdown. First, we need to recover the fitted values in link scale, which are log-odds of blowdown for a logistic model. The fitted log-odds are stored in `as.data.frame(e1.gm1)$fit` using the `e1.gm1` object previously computed.

```
R> or <- order(as.data.frame(e1.gm1)$fit) # order smallest to largest
R> Blowdown$spp1 <- factor(Blowdown$spp,
+                          levels=levels(Blowdown$spp)[or])
R> gm2 <- update(gm1, ~ . - spp + spp1)
R> plot(predictorEffects(gm2, ~ spp1), main="type='response', ordered",
+        axes=list(y=list(type="rescale",
+                         lab="probabilty scale, probability labels"),
+                   x=list(rotate=30, spp=list(lab="Species")),
+                   grid=TRUE))
```



In this order the separation of species into two groups of low probability species and high probability species is reasonably clear, with paper birch much more susceptible to blowdown and possibly in a group by itself.

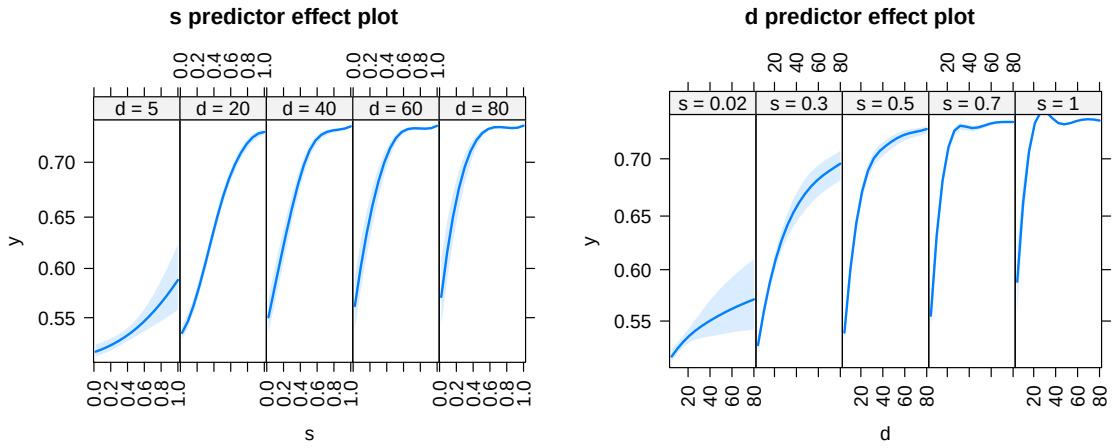
3.2 `lines` Group: Plotted lines

The `lines` argument group allows changing the color, type, thickness and smoothness of lines. This could be useful if the colors used by `effects` by default are for some reason unacceptable; for example, if only black or gray-scale lines are permitted. The most common use of this argument group is to allow more than one line to be plotted on the graph graph using the `multiline` sub-argument.

3.2.1 `multiline` and `z.var`: Multiple Lines in a Plot

Default predictor effects plots with conditioning variables generate a separate plot for each level of the conditioning variable(s). For an example, we add the `log(d):s` interaction to the model `gm1`, and plot the predictor effects plots for `s` and for `d`.

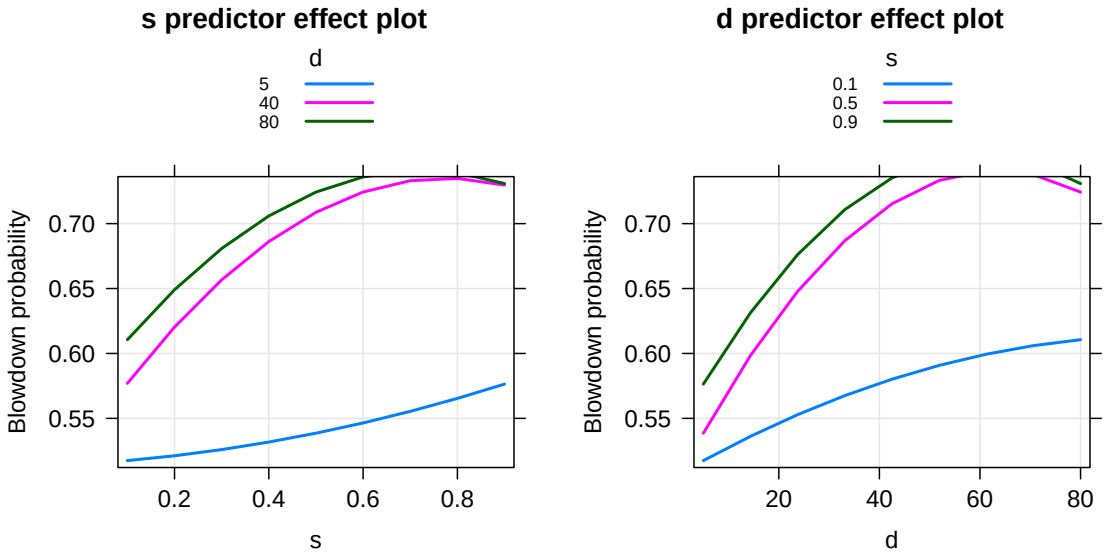
```
R> gm3 <- update(gm2, ~ . + s:log(d)) # add an interaction
R> plot(predictorEffects(gm3, ~ s + d),
+         axes=list(x=list(rug=FALSE, rotate=90)))
```



The predictor effects plot for `s` conditions on the level of `d`, and displays the plot of the fitted values for `y` versus `s` in a separate graph for each value of `d`. Similarly, the predictor effects plot for `d` displays a separate graph for each level of `s`. Confidence bands are displayed by default around each fitted line. These two graphs are based on the same fitted values, with the interaction between `s` and `d` varying, and or fixing, a value for `spp` at a typical value, as described in Section 2.2.1. Concentrating on `s`, when `s` is small the probability of blowdown is estimated to be in the range of about .53 to .57 for any value of `d`, but for larger values of `s` the probability of blowdown increases rapidly with `d`. Similar comments can be made concerning the predictor effects of `s`.

Setting `multiline=TRUE` will superimpose all the groups into a single graph. In the example below we will also reduce the number of levels of `s` and `d` to three each to get simpler graphs, although this is not required.

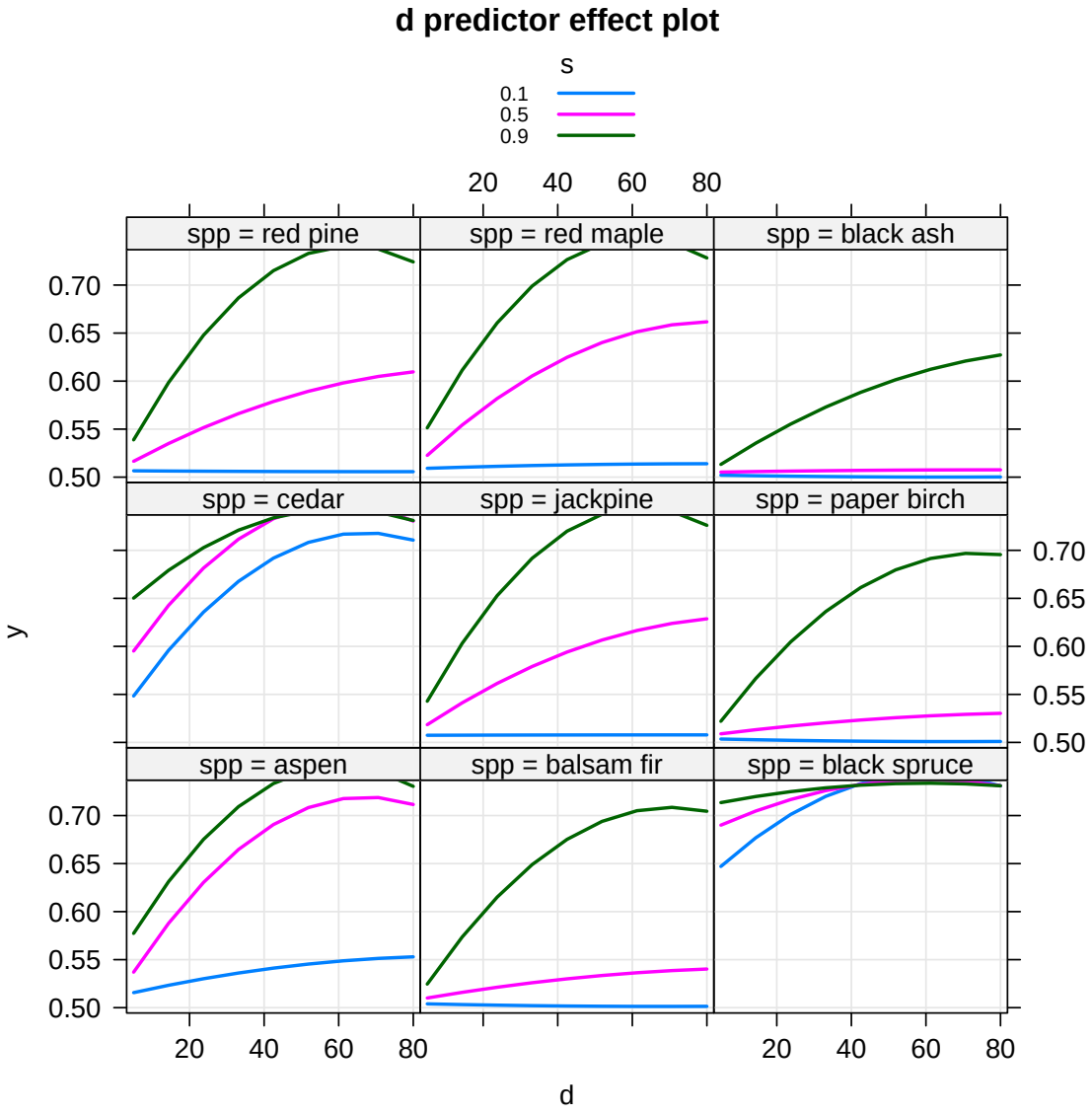
```
R> plot(predictorEffects(gm3, ~ s + d,
+                          xlevels=list(s=c(.1, .5, .9), d=c(5,40,80))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE),
+                   y=list(type="response", lab="Blowdown probability")),
+         lines=list(multiline=TRUE))
```



We kept, more or less, the lowest, middle, and highest values of the two predictors in the interaction. We added a grid, removed rotating axis labels, and converted to response scale. Multiline plots by default omit confidence bands or intervals, but these can be restored using the `confint` argument, Section 3.3. By default different levels of the conditioning factor are distinguished by color, and a key is provided. The placement an appearance of the key are controlled with a sub-argument in the `lattice` group, Section 3.4.

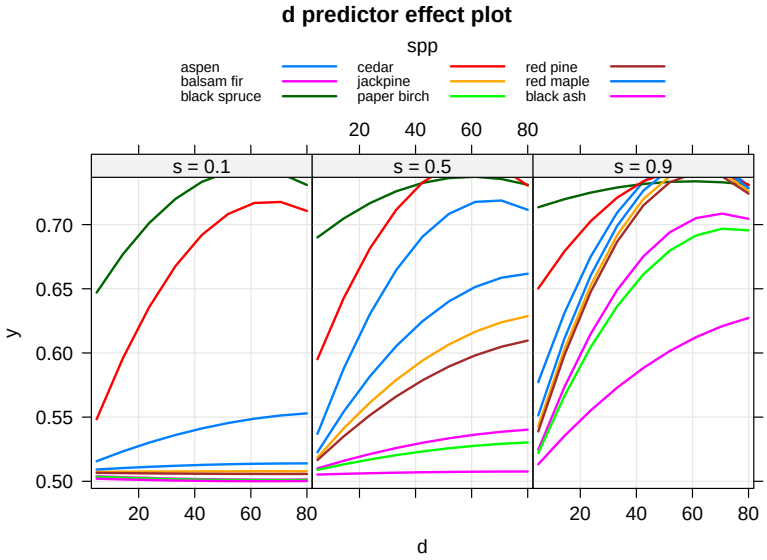
When the conditioning group includes two or more predictors beyond the focal predictor, multiline plots are almost always required because the array of subplots becomes too large to be useful. Suppose we add a `spp:log(d)` interaction to the model. The predictor effects plot for `d` includes both `s` and `spp` in the conditioning set because `d` interacts with both of them.

```
R> gm4 <- update(gm3, ~ . + spp:log(d))
R> plot(predictorEffects(gm4, ~ d,
+                         xlevels=list(s=c(.1, .5, .9),
+                                      d=c(5,40,80))),
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rug=FALSE)),
+         lines=list(multiline=TRUE))
```



This plot now combines the lines for all levels of `s` for each levels of `spp` separately. Compare to

```
R> plot(predictorEffects(gm4, ~ d, xlevels=list(s=c(.1, .5, .9),
+                                                d=c(5,40,80))),
+          rug=FALSE,
+          axes=list(grid=TRUE, y=list(type="response")),
+          lines=list(multiline=TRUE, z.var="spp"),
+          lattice=list(layout=c(3, 1)))
```



The `z.var` sub-argument selects the predictor that determines the lines within a graph and the remaining predictors are between graphs. The default that the program choose is usually, but not always, appropriate. We also used the `lattice` argument to set the array of graphs to have one row and three columns.

3.2.2 Line Color, Type, Width, Smoothness

Different lines in the same plot are differentiated by default using color. This can be modified by the sub-arguments `lty`, `lwd` and `col` to set line types, widths, and colors, respectively. For example, in the last graph shown you can get all black lines of different line types using `lines=list(multiline=TRUE, col="black", lty=1:9)`, or using a gray scale, `lines=list(multiline=TRUE, col=gray((1:9)/10))`.

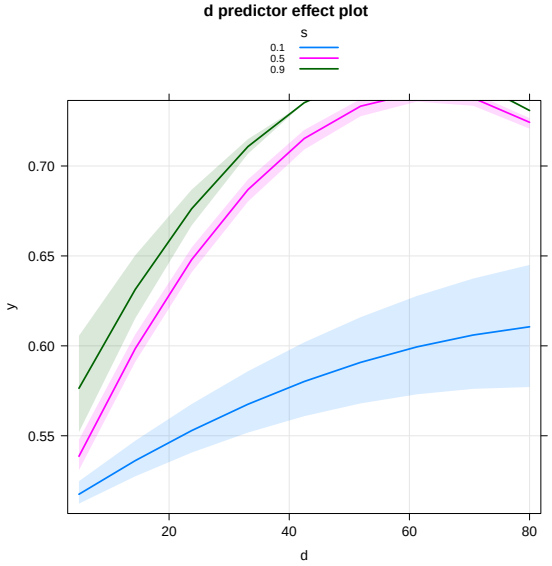
The plot method uses smoothing splines to interpolate between plotted points. This can be turned off with `splines=FALSE` in the `lines` argument, but we rarely expect this to be a good idea.

3.3 `confit`: Confidence Interval Style or Exclusion

The `confint` group controls addition or removal of confidence intervals and regions. This argument has three sub-arguments. The `style` argument is either `"bars"`, for bars typically around the adjusted mean for a factor, `"bands"` for shaded confidence bands for numeric focal predictors, `"auto"` to let the program automatically choose between `"bars"` and `"bands"`, `"lines"` to draw only the edges of confidence bands with no shading, or `"none"` for no confidence intervals. The default is `"auto"` when `multiline=FALSE` and `"none"` when multiline is true.

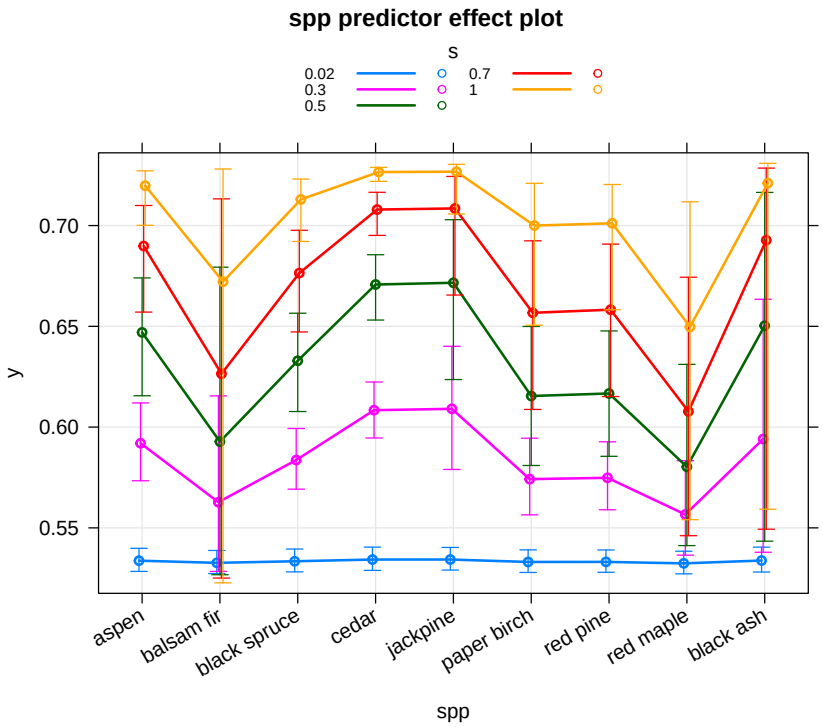
```
R> plot(predictorEffects(gm3, ~ d, xlevels=list(s=c(.1, .5, .9),
+                                                d=c(5,40,80))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE),
```

```
+                     y=list(type="response")),
+          lines=list(multiline=TRUE),
+          confint=list(style="auto"))
```



In this example the confidence bands are well separated, as so the inclusion of them causes no problem, but overlapping confidence lead to an artistic, but often uninterpretable, mess. With a factor focal predictor,

```
R> gm5 <- update(gm2, ~ . + spp:s)
R> plot(predictorEffects(gm5, ~ spp),
+          axes=list(grid=TRUE,
+                    y=list(type="response"),
+                    x=list(rug=FALSE, rotate=30)),
+          lines=list(multiline=TRUE),
+          confint=list(style="auto"))
```



The lines for the various levels of `s` are slightly staggered to avoid over plotting of the error bars. The use of `lattice` to modify the key is discussed in Section 3.4.1.

Two additional arguments `col` and `alpha` control the color of confidence bars and regions and the transparency of confidence regions. Users are unlikely to use these options. Finally, the type of confidence interval, either pointwise or Scheffé corrected for multiple comparisons is controlled by the `se` argument in Section 2.3.

3.4 `lattice`: Pass Arguments to the `lattice` Package

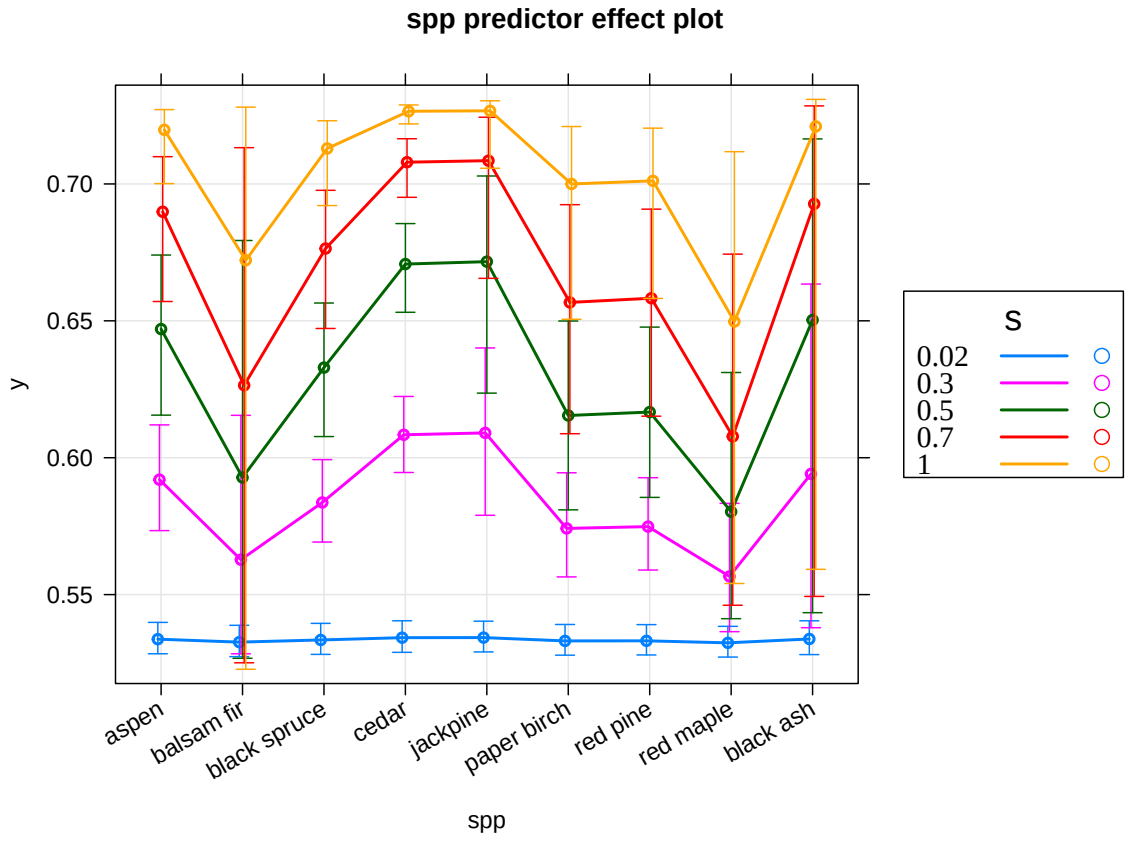
The `effects` packages uses the `lattice` package to draw scatterplots and lattice graphics, which are rectangular arrays of scatterplots (Sarkar, 2008). The `lattice` group of arguments modify plotting options available from `lattice` functions. In particular, you can change the number of rows and columns when plots are in an array; modify the key, and change the contents of the "strip", the shaded region of text above each plot in the lattice array. The `array` argument for advanced users allows control of printing of the layout of multiple predictor effects plots.

3.4.1 Modifying the key with `key.args`

A user can modify the placement and appearance of the key with sub-arguments of the `key.args` argument. For example

```
R> plot(predictorEffects(gm5, ~ spp),
+         rug=FALSE,
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
```

```
+                     x=list(rotate=30)),
+          lines=list(multiline=TRUE),
+          confint=list(style="auto"),
+          lattice=list(key.args=list(space="right",
+                                     columns=1,
+                                     border=TRUE,
+                                     fontfamily="serif",
+                                     cex=1.25,
+                                     cex.title=1.5)))
```



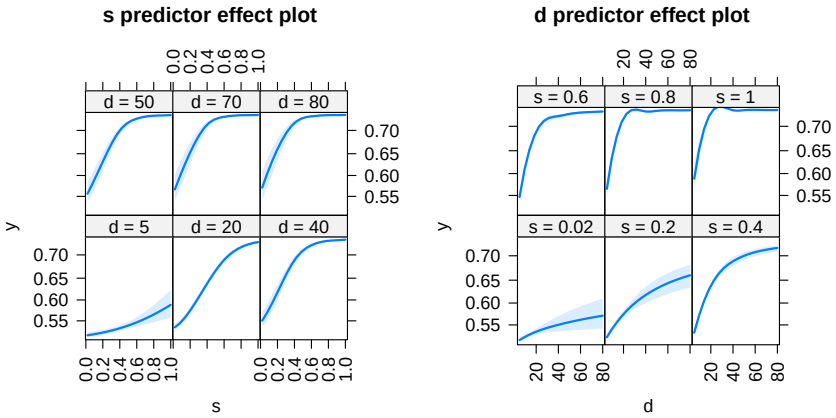
The sub-argument `space="right"` moved the key to the right of the graph, overriding the default `space="top"`. Alternatively the key can be placed on the graph using the `x`, `y` and `corner` arguments as illustrated in the graph on page 19. The choices for `fontfamily` are `c("sans", "serif")` affect only the key; the rest of the plot uses `"sans"`. The arguments `cex` and `cex.title` are the relative sizes of the key entries and the key title, respectively. Finally any argument documented at `help("xyplot")` in the `key` section can be set with this argument.

If you use the default `space="top"` you may wish to adjust the number of columns, particularly if the level names are long.

3.4.2 `layout`

The `layout` argument allows a user to select a layout of the multiple plots in a lattice graph, for example,

```
R> plot(predictorEffects(gm3, ~ s + d, xlevels=list(s=6, d=6)),
+         axes=list(x=list(rug=FALSE, rotate=90)),
+         lattice=list(layout=c(3, 2)))
```



The `layout` sub-argument to `lattice` specifies an array of three columns and two rows for each of the predictor effects plots.

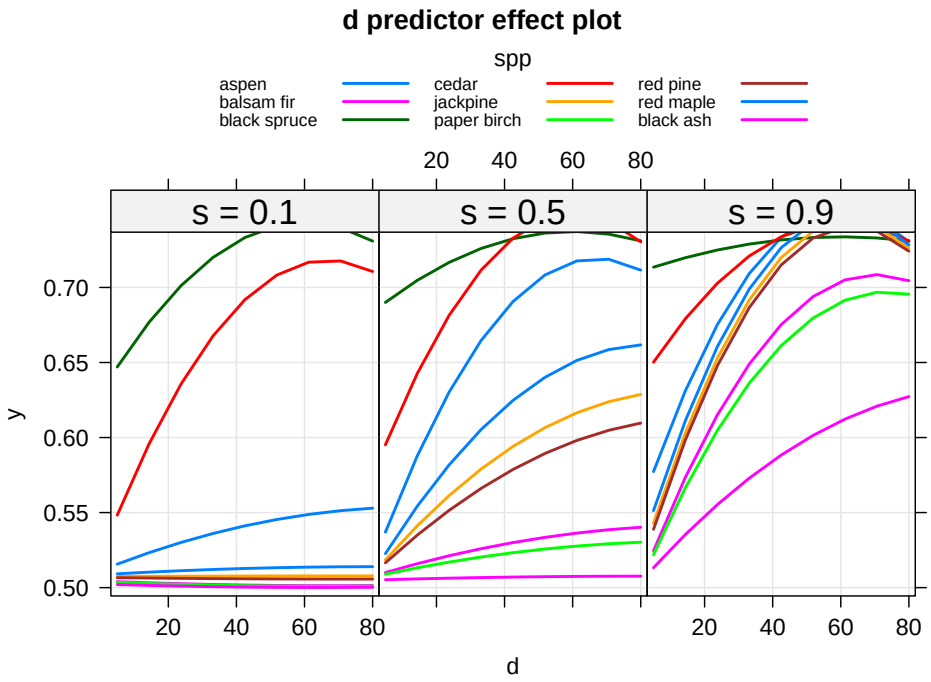
3.4.3 `array` of multiple plots

If you create several predictor effect objects with the function `predictorEffects`, the program uses the `array` argument to divide your plotting region into parts so the predictor effects plots can be drawn without overlapping. The user can also use this argument to make a custom array of predictor effects plots; see `help("plot.eff")` for the syntax.

3.4.4 `strip`

Lattice graphics with more than one plot typically provide a title for each graph above the graph in an area called a *strip*. The default title in the strip is the names of the fixed variable or variables in that strip and the values at which they are fixed.

```
R> plot(predictorEffects(gm4, ~ d, xlevels=list(s=c(.1, .5, .9),
+                                                d=c(5,40,80))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE),
+                   y=list(type="response")),
+         lines=list(multiline=TRUE, z.var="spp"),
+         lattice=list(layout=c(3, 1),
+                      strip=list(factor.names=TRUE,
+                                 values=TRUE,
+                                 cex=1.5)))
```

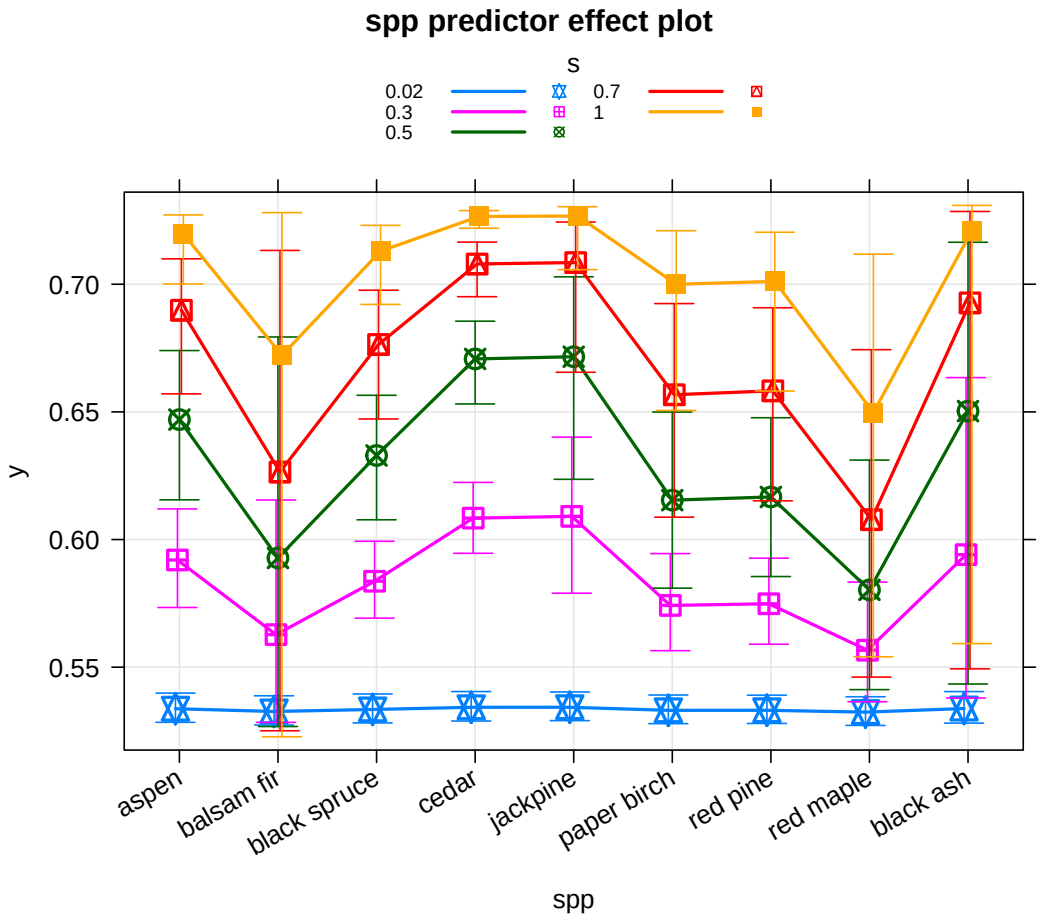


Setting `factor.names=FALSE` displays only the value, not the name of the conditioning predictor, usually desirable only if the name is too long to fit. Setting `values=FALSE` replaces the conditioning value with a symbolic representation. The most useful sub-argument is `cex` that allows you to reduce or expand the relative size of the text in the strip, in this case increasing the size to 150% of the nominal size.

3.5 `symbols`: Plotted symbols

Symbols are used to represent adjusted means when the horizontal axis is a factor. You can control the symbol used and its relative size:

```
R> gm5 <- update(gm2, ~ . + spp:s)
R> plot(predictorEffects(gm5, ~ spp),
+         rug=FALSE,
+         symbols=list(pch=11:15, cex=1.5),
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rotate=30)),
+         lines=list(multiline=TRUE),
+         confint=list(style="auto"))
```



We used the `pch` sub-argument to set the symbol number for plotted symbols; you can type the command `plot(1:25, pch=1:25, cex=.5)` to see the 25 plotting symbols in R. The argument to `pch` can also be character strings, such as `letters[1:10]`. We set `cex=1.5` to multiply the default size of symbol by 1.5. Since only one value is given, it is recycled and used for all groups. These change the size of the symbol in the plot but not in the key.

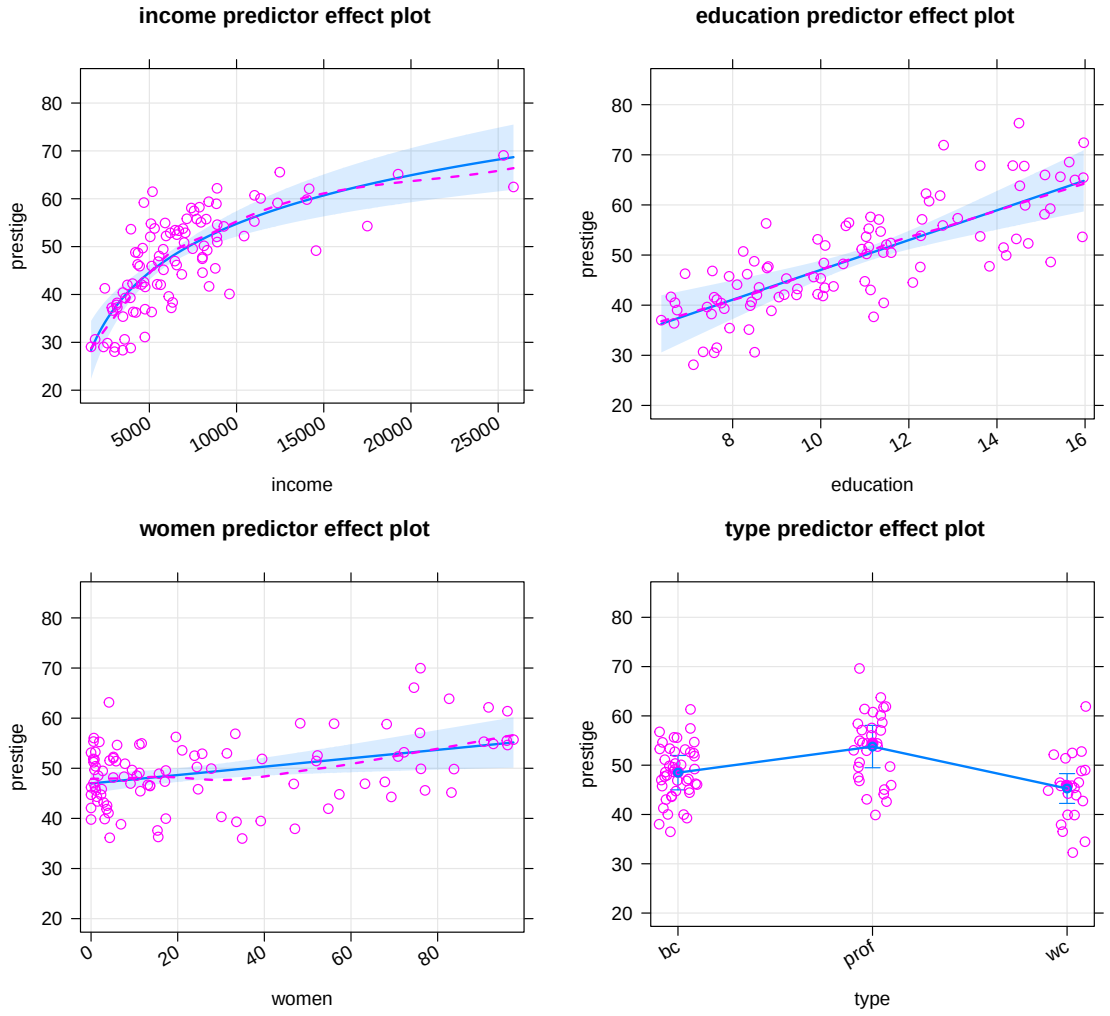
4 Displaying Residuals in Predictor Effects Plots

Fox and Weisberg (2019b) introduced methodology for adding partial residuals to a predictor effects plot. This can be desirable to display variation in data around fitted partial regression line, or to diagnose possible lack of fit, as the resulting plots are similar to component-plus-residual plots (Fox and Weisberg, 2019a, Sec. 8.4).

The predictor effect plot for a focal predictor that does not interact with other predictors is equivalent to a component-plus-residual plot,

```
R> lm5 <- lm(prestige ~ log(income) + education + women + type,
+             Prestige)
R> plot(predictorEffects(lm5, residuals=TRUE),
+         axes=list(grid=TRUE,
```

```
+                     x=list(rotate=30)),
+          partial.residuals=list(smooth=TRUE,
+                                 lty="dashed"))
```

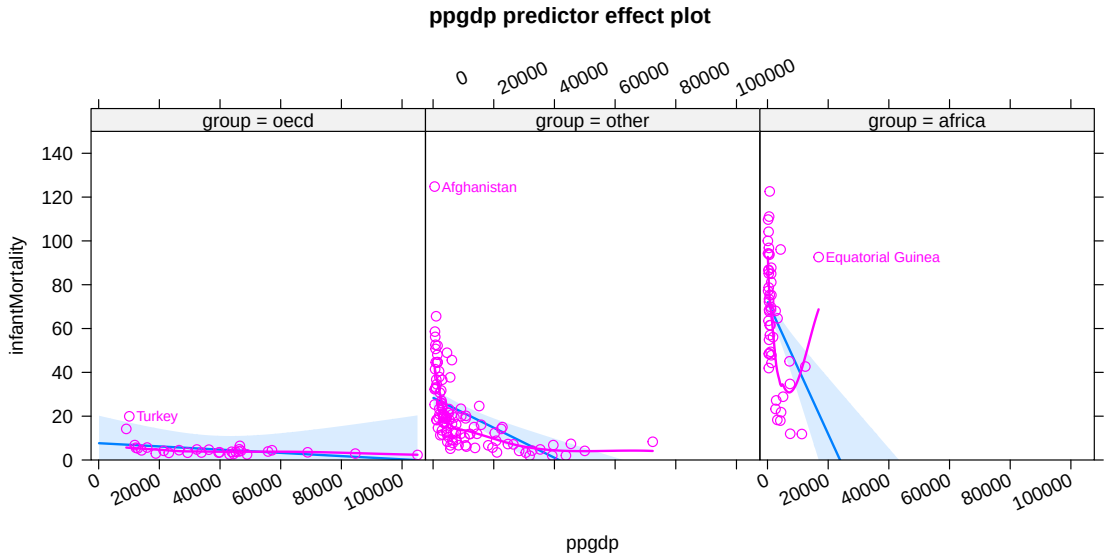


The partial residuals to be plotted are computed using the `residuals` argument to the `predictorEffect` or `predictorEffects` package. For the numeric predictors `income`, `education` and `women`, the plotted points are equal to a point on the fitted (blue) line from plus the corresponding residual. For `income` the fitted line in curved because of the log-function, but it is a straight line for the other two numeric predictors. The dashed line in the same color as the plotted points on the graph is a nonparametric smooth of the points shown in the graph. If the model matches the data then the dashed line should match the line fitted from the fit of the model. For the factor `type` the points have been jittered before plotting because the only possible values are at the factor levels. Smooths are not fit to factors.

The `plot` method has an argument `partial.residuals` with several sub-arguments. In the above plot we used the sub-argument `smooth` to add the smoother, and `lty` to change the line type from the default of a solid line to a dashed line. All the arguments are described at `help("plot.eff")`.

For a second example we turn to a linear model with an interaction, modelling `infantMortality` rate as a function of `ppgdp`, per person GDP, and country `group` in the data frame `UN` in the `carData` package, using data collected by the UN.

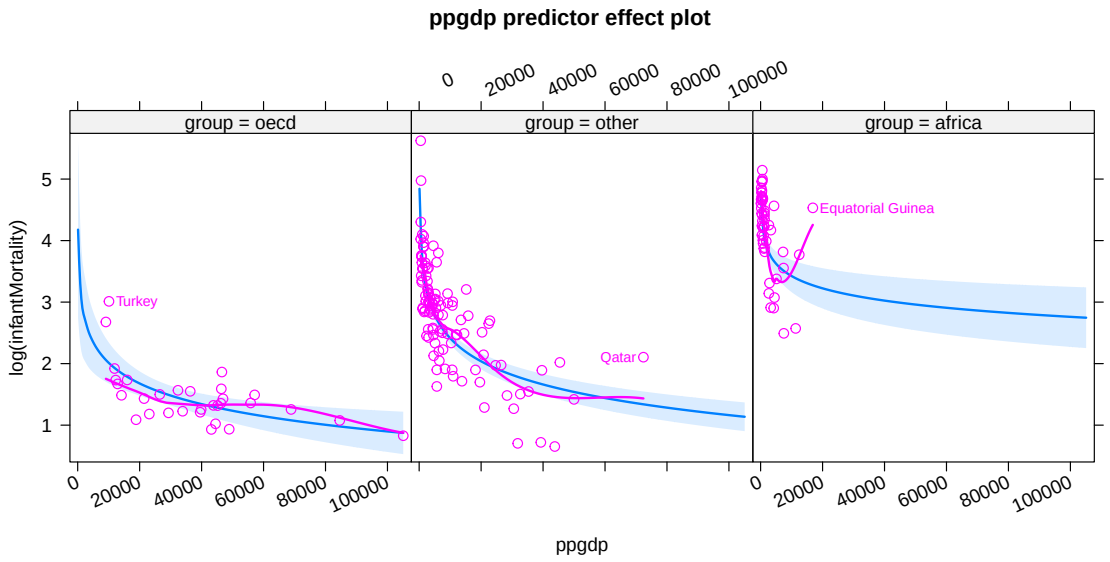
```
R> options(scipen=10000)
R> lm6 <- lm(infantMortality ~ group * ppgdp, data = UN)
R> plot(predictorEffects(lm6, ~ ppgdp,
+                         partial.residuals = TRUE),
+             axes = list(x = list(rotate = 25),
+                         y = list(lim = c(0, 150))),
+             id = list(n = 1))
```



This plot conditions on the factor `group` because of the interaction. Several problems are apparent in these plots. the `id` argument was used to identify the most unusual point, as described in the details at `help("plot.eff")`. Turkey had higher than predicted infant mortality for the `OECD` group, Equatorial Guinea is clearly unusual for the `Other` group, and Afghanistan had infant mortality much higher than predicted. In addition the points do not match the fitted line. We used the command `options(scipen=1000)` to suppress the annoying use of scientific notation in the axis labels.

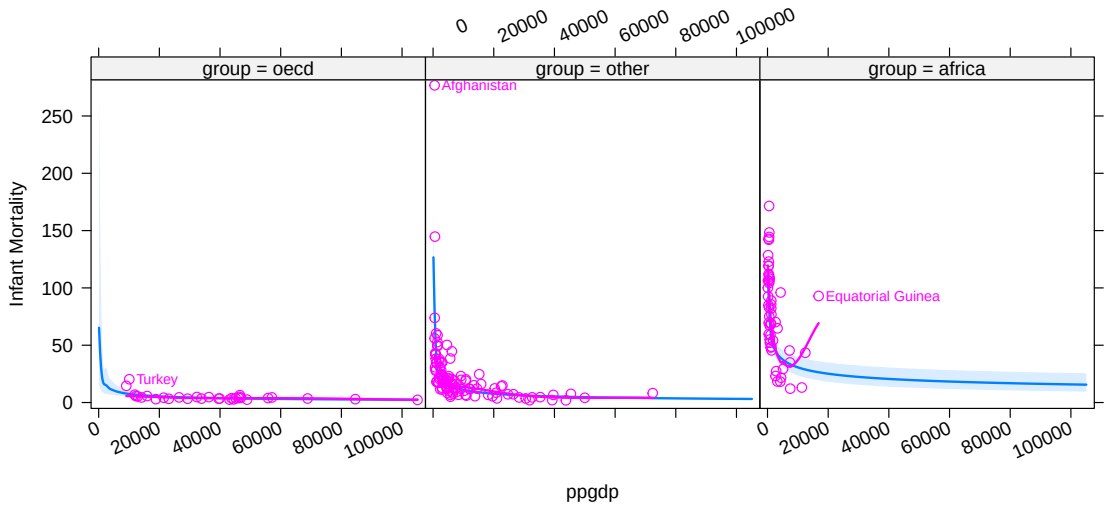
Using the regressor log(`ppgdp`) and the response log(`infantMortality`),

```
R> lm7 <- lm(log(infantMortality) ~ group * log(ppgdp), data = UN)
R> plot(predictorEffects(lm7, ~ ppgdp,
+                         partial.residuals = TRUE),
+         axes = list(x = list(rotate = 25)),
+         id = list(n = 1))
```



The fit is much better in log-scale, although Equatorial Guinea is still anomalous. Rescaling the plot to arithmetic scale gives a slightly different, but possibly useful, picture,

```
R> plot(predictorEffects(lm7, ~ ppgdp,
+                        partial.residuals = TRUE),
+        axes = list(x = list(rotate = 25),
+                    y = list(transform=list(trans= log,
+                                            inverse=exp),
+                             type="response",
+                             lab="Infant Mortality")),
+        id = list(n = 1))
```



Partial residual plots can be added to linear or linear mixed models or to generalized linear or generalized linear mixed models in the default link scale.

5 Predictor Effects Plots with Multivariate Responses

5.1 Multivariate Regression

Not written. Please add.

5.2 Multi-category Responses

The `effects` package includes special graphics types for used with a discrete response with multiple categories. In an ordinal regression the response is an ordered categorical variable. For example, in a study of labor force participation the response could be either not working, working part time or working full time. The proportional odds model (Fox and Weisberg, 2019a, Sec. 6.9) estimates the probability of a response equal to one of these three categories given a linear combination of regressors defined by a set of predictors, assuming a logit link function. Using the `Womenlf` data set in the `carData` package, and the `polr` function in the `MASS` package,

```
R> require("MASS") # to get the polr function for fitting

Loading required package: MASS

R> Womenlf$partic <- factor(Womenlf$partic,
+        levels=c("not.work", "parttime", "fulltime"))
R> or1 <- polr(partic ~ log(hincome) + children, Womenlf)
R> S(or1)


Re-fitting to get Hessian

Call: polr(formula = partic ~ log(hincome) + children, data = Womenlf)

 Coefficients:
                Estimate Std. Error z value      Pr(>|z|)
log(hincome)      -0.666      0.233   -2.86        0.0042
childrenpresent   -1.948      0.287   -6.80 0.00000000001

 Intercepts (Thresholds):
                   Estimate Std. Error z value Pr(>|z|)
not.work|parttime    -2.747      0.654   -4.20 0.000027
parttime|fulltime    -1.837      0.640   -2.87   0.0041

Residual Deviance: 441.12

 logLik      df     AIC     BIC
-220.56       4  449.12  463.40
```

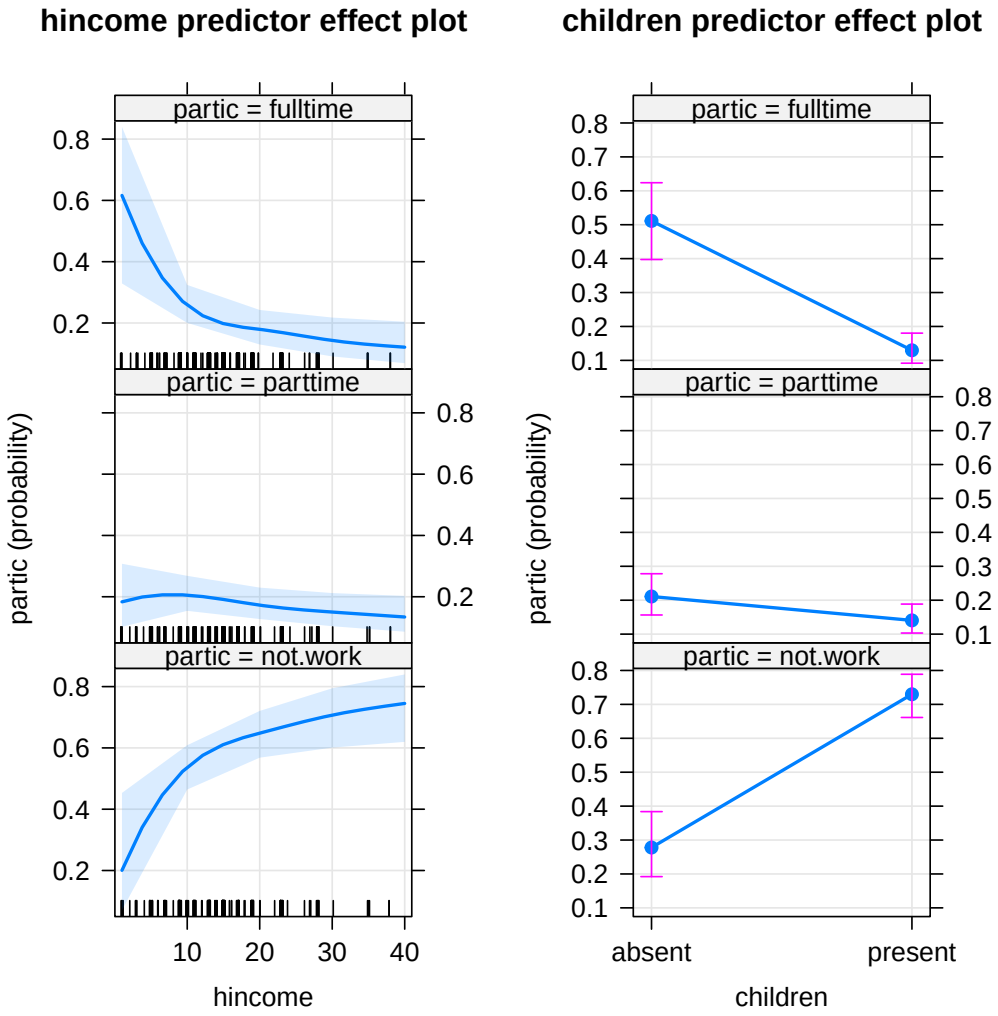
The response variable `partic` had its levels in alphabetical order, which does not correspond to the natural ordering of the levels. We start by reordering the levels to increase from no work, to part time work to full time work. The printed summary is is fairly complex, and is described in (Fox and Weisberg, 2019a, Sec. 6.9).

The predictor effects plots greatly simplify the summary:

```
R> plot(predictorEffects(or1),
+         axes=list(grid=TRUE),
+         lattice=list(key.args=list(columns=1)))
```

```
Re-fitting to get Hessian
```

```
Re-fitting to get Hessian
```



Unlike predictor effects plots for generalized linear models, the default scaling for the vertical axis is in probability scale, equivalent to `axes=list(y=list(type="response"))`

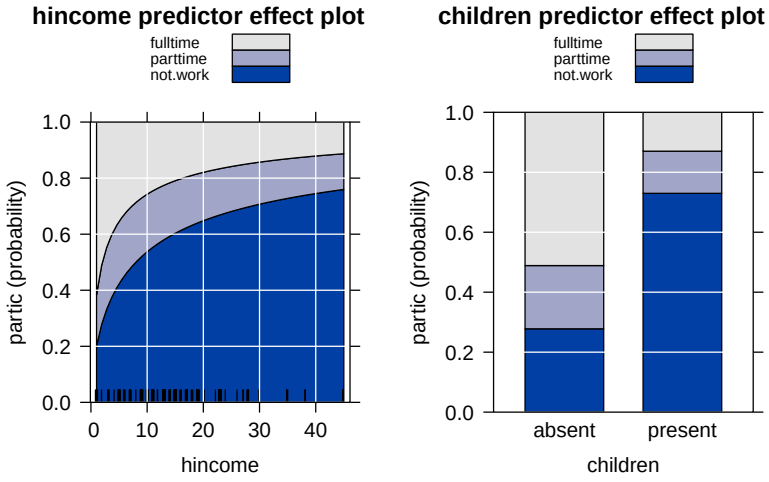
for a glm, and the alternative is `axes=list(y=list(type="logit"))`, which is analogous to `type="link"` for a glm. Also confidence bands are present, unless turned off with the argument `confint=list(style="none")`. The plot for `hincome` suggests high probability of full time work if the husband's income is low, with the probability of full time work sharply decreasing to about $15,000 and then leveling off at about .1 to .2. The probability of not working and of part time work rapidly increase with husband's income. A similar pattern is present for children present in the home, with full time work much less prevalent with children than without, and not working or part time work more like with children present.

Stacked plots are sometimes more useful for examining these models:

```
R> plot(predictorEffects(or1, xlevels=list(hincome=50)),
+         axes=list(grid=TRUE, y=list(style="stacked")),
+         lattice=list(key.args=list(columns=1)))
```

```
Re-fitting to get Hessian
```

```
Re-fitting to get Hessian
```



For each fixed value on the horizontal axis, the vertical axis "stacks" the probabilities in each of the groups for the response. For example with children absent from the household, about 25% of women did not work, 25% worked part time and the rest full time. The `xlevels` argument was used to evaluate `hincome` over a fine grid to get smoother curves in the plot.

Some models with with the functions `clm`, `clm2`, `clmm`in the `ordinal` package can be used with predictor effects. To work with these functions you will also need to load the `MASS` library.

Similar graphs are possible with the more general multinomial response model, in which the categorical response has unordered categories (Fox and Weisberg, 2019a, Sec. 6.7). The details of the model, the parameters and the assumptions are different

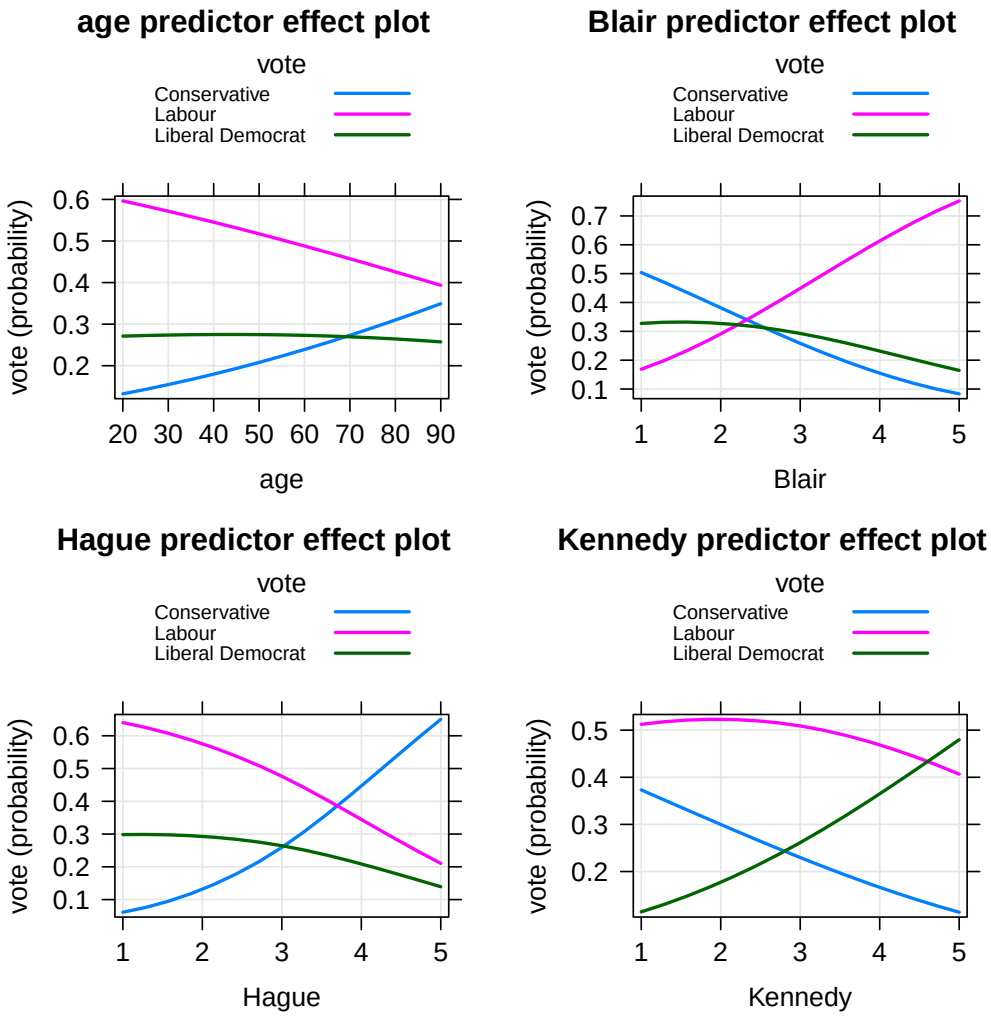
from the proportional response model, but the summarization by predictor effects plots is similar.

As an example we use the `BEPS` data in the `carData` package, consisting of about 1,500 observations from the 997-2001 British Election Panel Study. The response variable was party choice, one of liberal democrat, labour or conservative. There were numerous predictors, and we consider fitting the model

```
R> require("nnet")

Loading required package: nnet

R> mr1 <- multinom(vote ~ age + gender + economic.cond.national +
+                            economic.cond.household + Blair + Hague + Kennedy +
+                            Europe*political.knowledge, data=BEPS)

# weights:  36 (22 variable)
initial  value 1675.383740
iter  10 value 1240.047788
iter  20 value 1163.199642
iter  30 value 1116.519687
final  value 1116.519666
converged
```

The response `vote` is a factor giving the party identification. There are 9 predictors, 7 of which are scales with values between 0 and 5 about respondent attitudes that enter the model as main effects. The remaining 2 are scales between 0 and 3 for `political.knowledge` and 1 and 11 for `Europe` that enter the model with a two-factor interaction. Drawing all 9 predictor effects plots simultaneously is not productive because the overhead of the plots, space between the plots, room for a title, a key and for axis and tick labels will consume most of the available screen space. We use a strategy of drawing only a few of these plots at a time.

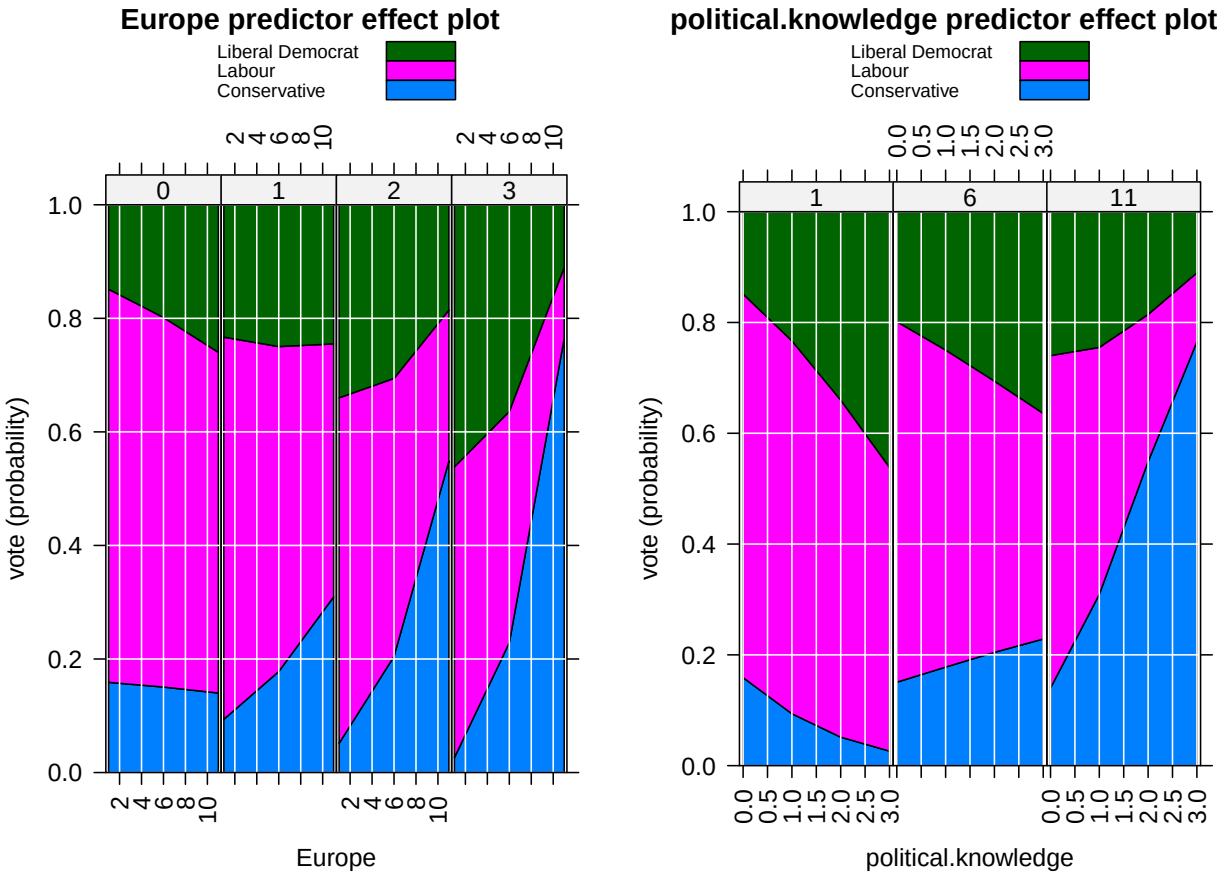
```
R> plot(predictorEffects(mr1, ~ age + Blair + Hague + Kennedy),
+         axes=list(grid=TRUE, x=list(rug=FALSE)),
+         lattice=list(key.args=list(columns=1)),
+         lines=list(multiline=TRUE))
```



We used optional arguments to get a multiline plot with a grid and no rug plot and to modify the key. Interpreting these plots is challenging. The probability of preferring labour decreases with age, increases with attitude toward the labour leader Blair, strongly decreases with attitude toward the conservative leader Hague, and is relatively unaffected by attitude toward the liberal democrat's leader Kennedy. In general a positive attitude toward a party leader increases the probability of favoring that leader's party.

We next turn to examining the interaction between `Europe` and `political.knowledge`, this time using a stacked display. We also use the `xlevels` argument to control the number of individual plots that are drawn.

```
R> plot(predictorEffects(mr1, ~ Europe + political.knowledge,
+                          xlevels=list(political.knowledge=0:3,
+                                       Europe=c(1, 6, 11))),
+         axes=list(grid=TRUE, x=list(rug=FALSE, rotate=90),
+                   y=list(style="stacked")),
+         lattice=list(key.args=list(columns=1),
+                      strip=list(factor.names=FALSE)))
```



Both plots are of the same fitted values, in the first graph with `Europe` varying and `political.knowledge` fixed, and in the second with `political.knowledge` varying `Europe` fixed. Concentrating on the first graph we see that preference for the conservative party increase with a positive attitude toward Europe with respondents with high political knowledge, but not low political knowledge. Preference for labour is lowest among respondents that are high on both variables.

6 The Lattice Theme for Predictor Effects

Most of the graphics in the `effects` package use the standard `lattice` package Sarkar (2008) to draw the graphs. The `lattice` package has many options for customizing the appearance of graphs that are collected into a *lattice theme*. We have created a custom theme for use with the `effects` that will automatically replace the default theme, *unless the* `lattice` package has been previously loaded. You can load the theme manually with the command

```
R> effectsTheme()
```

You can change the theme; see `help("effectsTheme")`.

7 Tests with Predictor Effects

Not written. This will make connections to Anvoa and emmeans.

References

Fox, J. and S. Weisberg (2019a). *An R companion to applied regression*. Sage.

Fox, J. and S. Weisberg (2019b). Visualizing fit and lack of fit in complex regression models with predictor effect plots and partial residuals. *Journal of Statistical Software*.

Hawkins, D. M. and S. Weisberg (2017). Combining the box-cox power and generalised log transformations to accommodate negative responses in linear and mixed-effects linear models. *South African Statistics Journal 51*, pp. 317–328.

Lenth, R. (2018). *emmeans: Estimated Marginal Means, aka Least-Squares Means*. R package version 1.2.1.

Sarkar, D. (2008). *Lattice: multivariate data visualization with R*. Springer Science & Business Media.