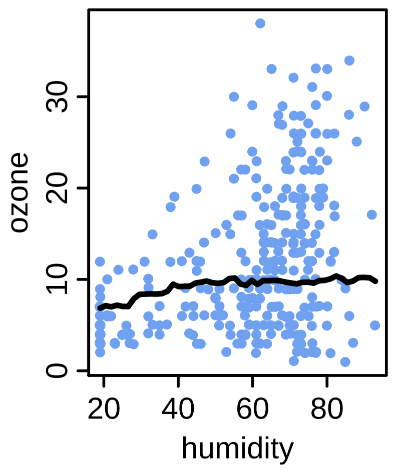
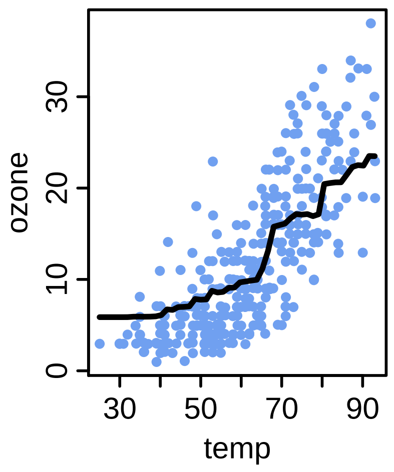
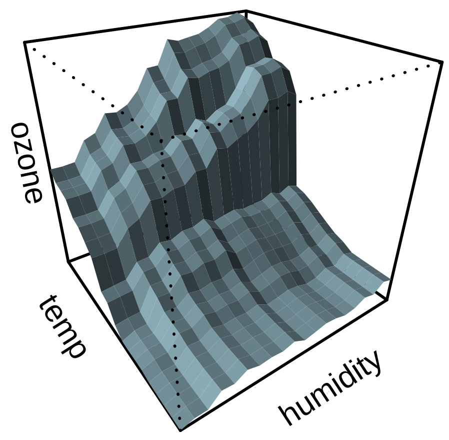
Plotting regression surfaces with `plotmo`







Stephen Milborrow

September 7, 2016

Contents

1 Introduction

The `plotmo` function in the `plotmo` R package [13] makes it easy to plot regression surfaces for a model. These plots can be useful for understanding the model.

The plots on the title page of this document are examples—those plots are for a random forest, but plotmo can be used on a wide variety of R models.

Plotmo automatically creates a separate plot for each variable. Each such *degree1* plot is generated by plotting the predicted response as the variable changes. The top two plots on the title page are examples. The variables that don't appear in a plot are the *background* variables (or simply the "other" variables). In each plot the background variables are held fixed at their median values (the medians are calculated from the training data).

Plotmo can also show interactions between pairs of variables. A *degree2* plot is generated by plotting the predicted response as two variables are changed (once again with all other variables held at their median values). The bottom plot on the title page is an example.

2 Examples

Here are some examples which illustrate plotmo on various models. Figure 1 illustrates these examples.

```
# use a small set of variables for illustration
library(earth) # for ozone1 data
data(ozone1)
oz <- ozone1[, c("O3", "humidity", "temp", "ibt")]

lm.mod <- lm(O3 ~ humidity + temp*ibt, data=oz)             ## linear model
plotmo(lm.mod)

library(rpart)                                              ## rpart
rpart.mod <- rpart(O3 ~ ., data=oz)
plotmo(rpart.mod)

library(randomForest)                                       ## randomForest
rf.mod <- randomForest(O3 ~ ., data=oz)
plotmo(rf.mod)
# partialPlot(rf.mod, oz, temp)       # compare to partial-dependence plot

library(gbm)                                                ## gbm
gbm.mod <- gbm(O3 ~ ., data=oz, dist="gaussian", inter=2, n.trees=1000)
plotmo(gbm.mod)
# plot(gbm.mod, i.var=2)              # compare to partial-dependence plots
# plot(gbm.mod, i.var=c(2,3))
```

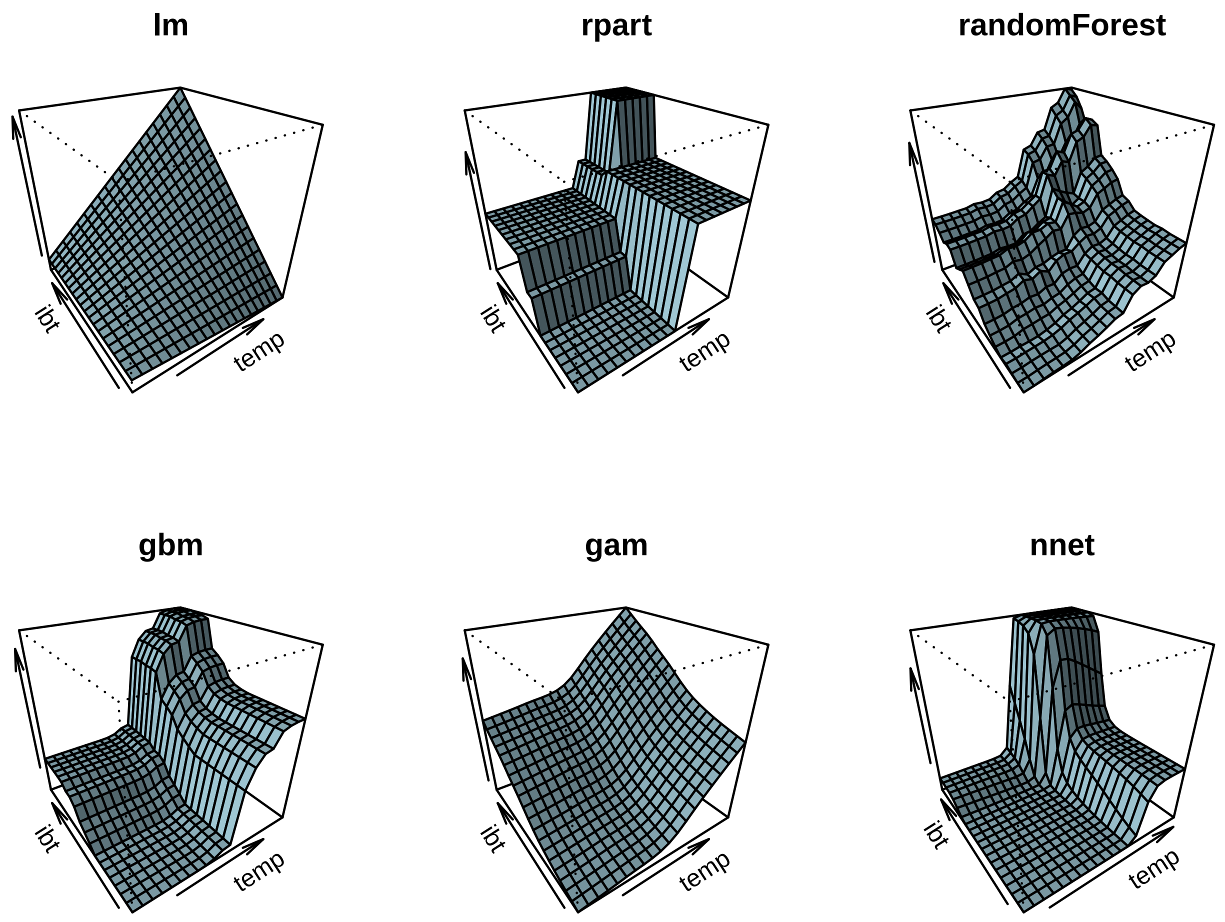


Figure 1: *Plotmo graphs on various models, generated by the code in the text. (A single degree2 plot for each model is illustrated here, but by default plotmo displays a set of plots on the same page for each model. See Section 6 "Which variables get plotted?")*

```
library(gam)                                               ## gam
gam.mod <- gam(O3 ~ s(humidity) + s(temp) + s(ibt), data=oz)
plotmo(gam.mod, all2=TRUE)            # all2=TRUE to show interaction plots

library(nnet)                                               ## nnet
set.seed(4)
nnet.mod <- nnet(O3 ~ ., data=scale(oz), size=2, decay=0.01, trace=FALSE)
plotmo(nnet.mod, type="raw", all2=T)  # type="raw" gets passed to predict
```

This is by no means an exhaustive list of models supported by plotmo. The packages used in the above code are [7, 9, 16, 17, 19].

3 Limitations

There are inherent limitations because the plots give only a partial view of the model. There are also practical limitations to do with the way some models are built in R. This section discusses these limitations in turn.

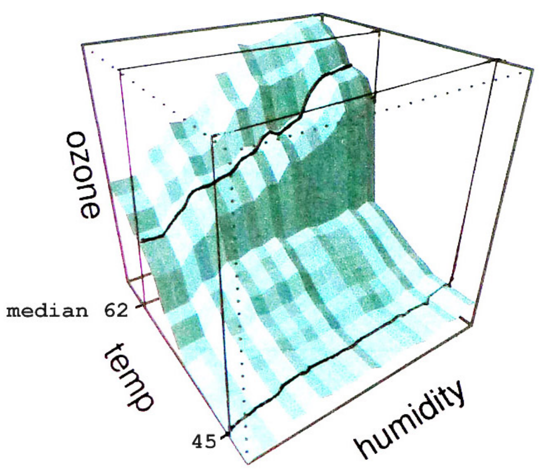


Figure 2: *The shape of the* `ozone` *response curve as* `humidity` *varies is quite different for slices at different values of* `temp`.

These image illustrates 2D slices through a 3D space. With a typical multivariate model the plotmo slices are through a much higher-dimensional space.

3.1 Inherent limitations

Each plots shows only a thin slice[1] of the data, because the background variables are pegged at fixed values. For example, in Figure 2 the response curve as `humidity` varies is quite different when `temp` is say 45 than when it's pegged at the median. Please be aware of this loss of information when interpreting the graphs. Over-interpretation is a temptation.

For a one variable model the regression surface is fully described by a degree1 plot, and for a two variable model by a degree2 plot. For additive models (no variable interactions), the regression surface is fully described by the set of degree1 plots.

More generally, models with many variables must be viewed in a piecemeal fashion by looking at the action of one or two predictors at a time. The plots are most informative when the variables being plotted do not have strong interactions with the other variables. Chapter 10 in the vignette for the rpart.plot package has a short discussion on these topics.

3.2 Practical limitations

Plotmo needs to access the data used when building the model so it can pass the correct data to `predict`. For some models this isn't possible (some models don't save the `call` or any references to the data). Section 11 has details.

For the model to work with plotmo, it's best to keep the variable names and formula in the original call to the model-building function simple. Use temporary variables or `attach` rather than using `$` and similar in formulas. Error messages may be issued if there are NAs in the data (it depends on the model). Section 10 has more details.

[1]Each plot is a lower-dimensional "slice" through a higher-dimensional space, like a slice of bread is a 2D plane through a 3D loaf.

4 Alternatives

There are many ways of condensing a multi-dimensional model onto the two dimensions of a page. The technique used by plotmo is one of them. There is no silver bullet; large amounts of information are necessarily discarded when the complexities of a model must be plotted on a page.

Arguably the most important of these plots, although often overlooked, is the humble *residuals-vs-fitted* plot. The `plotres` function (also in the `plotmo` package) is an easy way to make various residual plots for "any" model. See the plotres vignette. Sometimes it's also worthwhile plotting the residuals against the variables or the model basis functions.

The `termplot` function in the standard `stats` package can be helpful, but it's supported by only a few models (the `predict` method for the model must support `type="terms"`), and it doesn't generate degree2 plots.

`Partial-dependence` plots are a well-known technique for plotting regression surfaces (e.g. Hastie et al. [6] Section 10.13.2). Plotmo sets the background variables to their median values, whereas in a partial-dependence plot at each plotted point the effect of the background variables is averaged. Computing this can take a long time. But for decision trees the effect of averaging can be determined without actual brute force summation, so partial-dependence plots for random forests and `gbm`'s can be generated quite quickly.

In general, partial-dependence plots and plotmo plots will differ, but for additive models (no interaction terms) the *shape* of the curves is identical although the scale may differ. Partial-dependence plots incorporate more overall information than plotmo plots, but it's easier to understand in principle what the graph *doesn't* show with plotmo than with partial-dependence plots (Section 3.1).

Some other possibilities for plotting the response on a per-predictor basis are partial-residual plots, partial-regression variable plots, and marginal-model plots (e.g. `crPlots`, `avPlots`, and `marginalModelPlot` in the `car` package [2]). The `effects` package is also of interest [3]. These packages are orientated towards linear and parametric models, whereas plotmo is mainly for non-parametric models.

Quite a few methods have been invented specifically for random forests. Although each tree in the forest is easy to interpret (a white box), the interaction between the large number of trees in a random forest makes the model as a whole a black box. Techniques such as plotmo thus become useful. See also the discussion on the CrossValidated web page Obtaining-knowledge-from-a-random-forest.

5 Some details

This section covers a few details that are useful to know when using plotmo.

5.1 Page layout

Plotmo puts all the plots on a single page. That can be overridden with the `do.par` argument.

Plotmo has special knowledge of some kinds of model. It uses that knowledge to plot only important plots, to limit crowding on the page. For example, for `earth` models it plots only the variables that are used in the final model, and for `randomForest` models it plots only the most important variables. We can also explicitly specify which variables get plotted by passing arguments to plotmo. Section 6 has details.

5.2 Predict type

Some models can make different kinds of predictions. For example, binomial `glm` models can predict the raw response or predict probabilities. By default, plotmo tries to automatically select a suitable response type for the model (often `type="response"`). Explicitly tell plotmo what kind of prediction to plot using plotmo's `type` argument. This gets passed internally to `predict`.

The predict function for some models returns a *matrix* rather than a vector of predicted values. For example, the predict function may return a two column matrix showing `absent` and `present` probabilities. By default, plotmo tries to automatically select which of these columns to display. Explicitly specify which column to use with plotmo's `ncolumn` argument, which can be a column number or column name.

Plotmo tries to use sensible default arguments for `predict`, but they won't always be correct (plotmo can't know about the predict method for every kind of model). Change the defaults if necessary using plotmo arguments with a `predict.` prefix. Plotmo passes any argument prefixed with `predict.` directly to `predict`, after removing the prefix. Section 4 of the plotres vignette has an example.

5.3 Background variables

As mentioned in the introduction, plotmo holds the background variables at their medians. But if a background variable is a factor, then the first level is used instead of the median.

Change what values are used for the background variables with the `grid.func` and `grid.levels` arguments. For example `grid.func=mean` or `grid.levels=list(sex="male", age=21))`. Use these arguments in a for loop to make a grid of plots conditioned on background variable values.

5.4 The `ylim` and `clip` arguments

Plotmo determines `ylim` for the graphs automatically. If the automatic `ylim` isn't correct for the model, specify a `ylim` when invoking plotmo, or try specifying `clip=FALSE`.

Here are some details. Typically we want all plots on a page to have the same `ylim` (the same vertical axis limits), so we can see the effect of each variable relative to the other variables. The range of predicted values over all the plots is the obvious way for plotmo to auomatically set `ylim`. However, a few wild predictions can make this range very wide, and reduce resolution over all graphs. Therefore when determining the range, plotmo ignores outlying predictions (unless `clip=FALSE`). Predictions that are more than 50% beyond the range of the observed response are considered outlying. In practice such outlying predictions seem quite rare.

6 Which variables get plotted?

The default set of variables plotted for some common models is listed below [9, 15–18].

The default set of plots for the model may leave out some variables that we would like to see. In that case, use `all1=TRUE` and/or `all2=TRUE`. To limit the subset of displayed variables use the `degree1` and `degree2` arguments.

- `earth`
  `degree1` variables in additive (non interaction) terms

  `degree2` variables appearing together in interaction terms

- `rpart`
  `degree1` variables used in the tree

  `degree2` parent-child pairs

- `randomForest` a 4×4 grid of plots (or less if fewer variables) as follows:

  `degree1` ten most important variables (ranked on
  the first column of `model$importance`)

  `degree2` pairs of the four most important variables
  (thus six degree2 plots)

- `gbm` a 4×4 grid of plots (or less if fewer variables) as follows:

  `degree1` variables with `relative.influence` >= 1%,
  up to a maximum of ten variables

  `degree2` pairs of the four variables with the largest
  relative influence (thus six degree2 plots)

- `lm, glm, gam, lda,` etc.

  These are processed using plotmo's default methods (Section 11):

  `degree1` all variables

  `degree2` variables in the formula associated with each other by
  terms like `x1 * x2`, `x1:x2`, and `s(x1,x2)`

7 Notes on miscellaneous packages

This section gives some specifics on how `plotmo` and `plotres` handle some miscellaneous models [4, 5, 8, 10, 16–18, 22].

By default, `predict.gbm` is called with `n.trees = object$n.trees`

By default, `predict.glmnet` is called with `type="response"` and `s = 0`.

By default, `predict.quantregForest` is called with `quantiles = .5`

By default, `predict.cosso` is called with `M = min(ncol(newdata), 2)`

For rpart models, plotres uses the rpart.plot package [11] if it's available, else it uses the plotting routines built into the rpart package.

For models built with the adabag package, plotmo's type argument should be "votes", "prob" (default), or "class" to select the corresponding field in predict.boosting's returned value. Plotmo's nresponse argument will typically also be necessary to select a column in the matrix of predicted values.

The predict methods for rq and rqs models (quantreg package) return multiple columns, and plotmo chooses the column corresponding to tau=0.5. Plotmo will plot prediction intervals if the quantreg model is built with say tau=c(.05, .5, .95) and plotmo is called with the corresponding level argument, in this case level=0.90.

The neuralnet package doesn't provide a predict method, but plotmo provides one internally:

```
predict.nn(object, newdata=NULL, rep="mean", trace=FALSE)
```

where rep can be an integer, "best", or "mean" (default). These last two are equivalent if the model was built with nrep=1. Examples:

```
plotres(nn.model, predict.rep="mean") # resids for mean prediction over all reps
plotres(nn.model)                     # same
plotres(nn.model, predict.rep="best") # resids for prediction from best rep
```

For biglm objects, only the residuals from the first call to biglm are plotted by plotres (the residuals for subsequent calls to update aren't plotted).

The predict methods for qda and lda models (MASS package) are extended internally within plotmo to take a type argument. This can be one of "class" (default), "posterior", or "response". This selects the "class", "posterior", or "x" field in the value returned by predict.lda and predict.qda. Use the nresponse argument to select a column within the selected field. Example (Figure 3):

```
library(MASS)
lcush <- data.frame(Type=as.numeric(Cushings$Type), log(Cushings[,1:2]))[1:21,]
qda.mod <- qda(Type ~ ., data=lcush)

plotmo(qda.mod,                 # figure shown below
       all2=TRUE,               # show all interact plots
       type2="image",           # use image instead of persp for interact plot
       ngrid2=200,              # increase resolution in image plot
       image.col=c("lightpink", "palegreen1", "lightblue"),
       pt.col=as.numeric(Cushings$Type)+1, pt.pch=as.character(Cushings$Type))

for(nresponse in 1:3)           # not shown
    plotmo(qda.mod, type="post", nresponse=nresponse,
           all2=TRUE, persp.border=NA)
           persp.theta=30)      # same theta for all plots so can compare
```

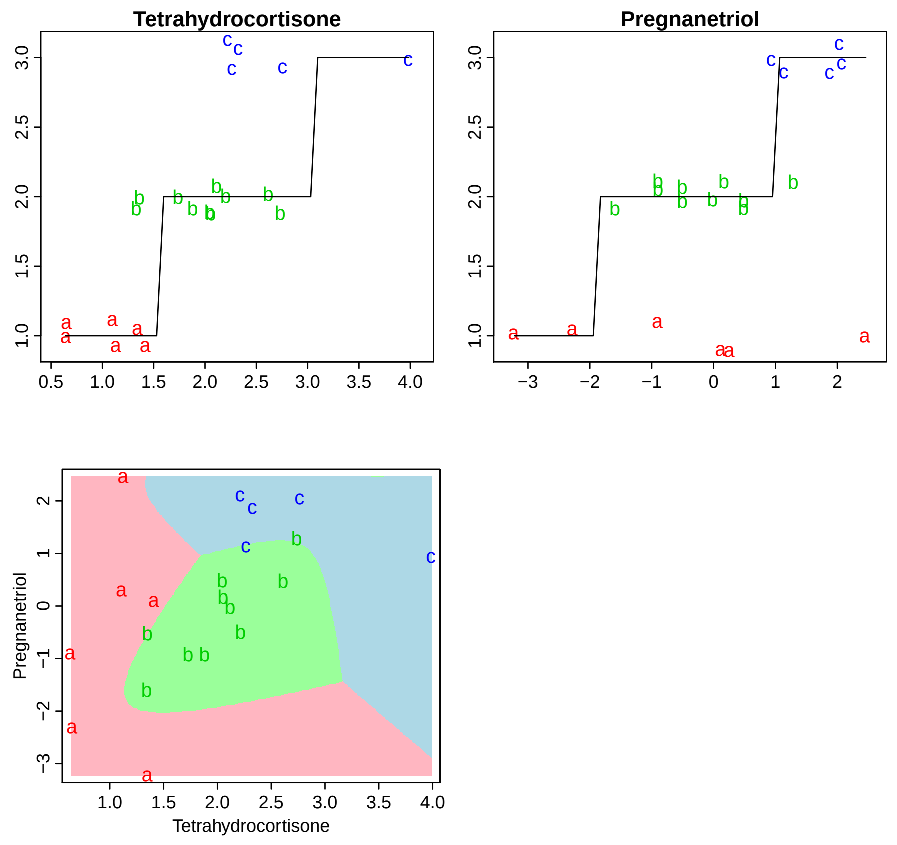


Figure 3: *A* `qda` *model of the log Cushings data.*

The background colors in the interaction plot show the predicted class.

The slightly messy look of the `a,b,c` *labels in the top two plots is caused by* plotmo*'s automatic jittering of factor labels (see the* `jitter` *argument).*

8 Prediction intervals (the `level` argument)

Use plotmo's `level` argument to plot pointwise confidence or prediction intervals. The `predict` method of the model object must support this. Examples (Figure 4):

```
par(mfrow=c(2,3))
log.trees <- log(trees) # make the resids more homoscedastic
                        # (necessary for lm)

                                                         ## lm
lm.model <- lm(Volume~Height, data=log.trees)
plot(lm.model, which=1) # residual vs fitted graph, check homoscedasticity
plotmo(lm.model, level=.90, pt.col=1,
       main="lm\n(conf and pred intervals)", do.par=F)

                                                          ## earth
library(earth)
earth.model <- earth(Volume~Height, data=log.trees,
                     nfold=5, ncross=30, varmod.method="lm")
plotmo(earth.model, level=.90, pt.col=1, main="earth", do.par=F)

                                                          ## quantreg
library(quantreg)
rq.model <- rq(Volume~Height, data=log.trees, tau=c(.05, .5, .95))
plotmo(rq.model, level=.90, pt.col=1, main="rq", do.par=F)
```

```
                                                          ## quantregForest
# quantregForest is a layer on randomForest that allows prediction intervals
library(quantregForest)
x <- data.frame(Height=log.trees$Height)
qrf.model <- quantregForest(x, log.trees$Volume)
plotmo(qrf.model, level=.90, pt.col=1, main="qrf", do.par=F)

                                                          ## gam
library(mgcv)
gam.model <- gam(Volume~s(Height), data=log.trees)
plotmo(gam.model, level=.90, pt.col=1,
       main="gam\n(conf not pred intervals)", do.par=F)
```

The packages used in the above code are [8, 10, 15, 21].

Confidence intervals versus prediction intervals

Be aware of the distinction between the two types of interval:

(i) intervals for the prediction of the mean response (often called *confidence intervals*)
(ii) intervals for the prediction of a future value (often called *prediction intervals*).

The model's `predict` method determines which of these intervals get returned and plotted by plotmo. Currently only `lm` supports both types of interval on new data (see

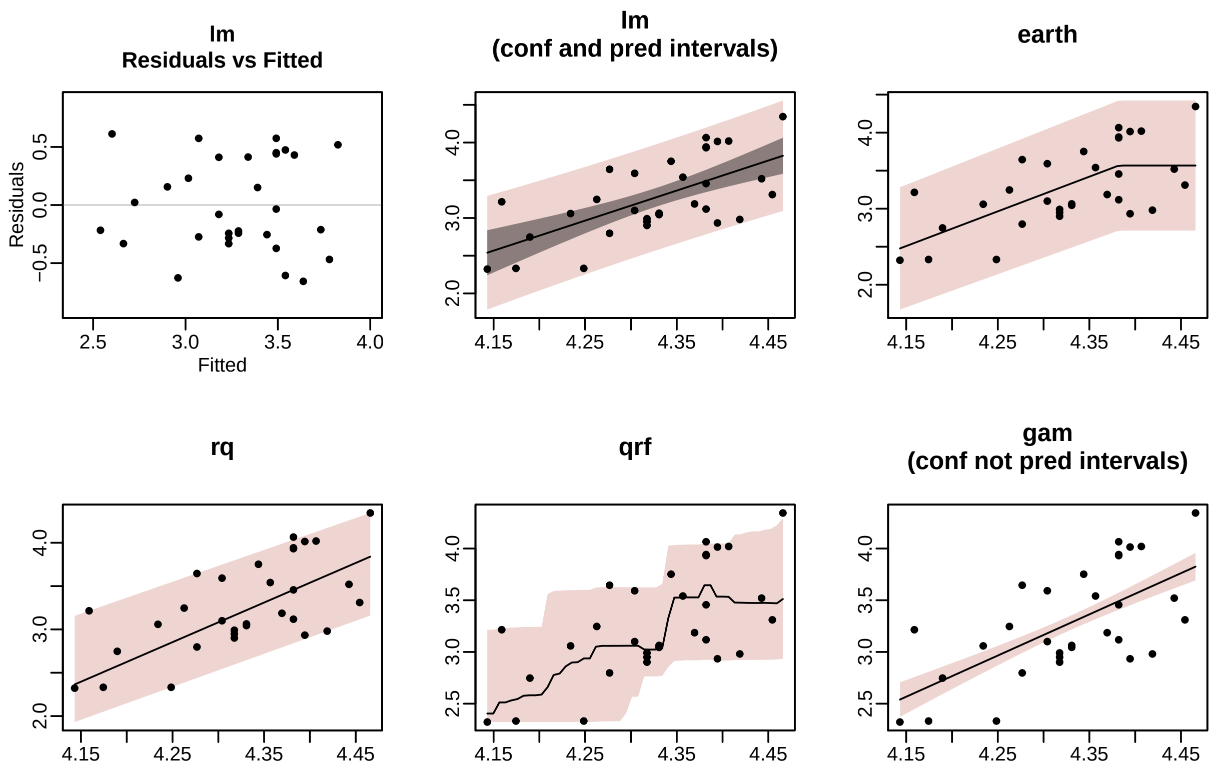


Figure 4: *Prediction intervals with plotmo. These plots were produced by the code on the previous page.*

`predict.lm`'s `interval` argument), and both are plotted by plotmo.

A reference is Section 3.5 of Julian Faraway's online linear regression book [1]. See also the vignette Variance models in `earth` [14], which comes with the `earth` package.

Assumptions for prediction intervals

Just because the intervals are displayed doesn't mean that they can be trusted. Be aware of the assumptions made to estimate the limits. At the very least, the model needs to fit the data adequately. Most models will impose further conditions. For example, linear model residuals must be homoscedastic.

Examination of the "Residual versus Fitted" plot is the standard way of detecting issues. So for example, with linear models use `plot.lm(mod, which=1)` and with earth models use `plot(mod, which=3)`. More generally, for any model use `plotres(mod, which=3)`.

Look at the the distribution of residual points to detect non-homoscedasity. Also look at the smooth line (the lowess line) in the residuals plot to detect non-linearity. If this is highly curved, we can't trust the intervals. One good place for more background on residual analysis is the *Regression Diagnostics: Residuals* section in Weisberg [20].

These are *pointwise* limits. They should only be interpreted in a pointwise fashion. So for non-parametric models they shouldn't be used to infer bumps or dips that are dependent on a range of the curve. For that we need *simultaneous* confidence bands, which none of the above models support.

9 FAQ

I'm not seeing any interaction plots

Use `all2=TRUE` to force the display of interaction plots. By default, degree2 plots are drawn only for some types of model (Section 6). When `all2=TRUE` is used, the `degree1` and `degree2` arguments can be useful to limit the number of plots.

Plotmo always prints messages. How do I make it silent?

Use `trace =- 1`. The grid message printed by default is a reminder that plotmo is displaying just a slice of the data.

The `image` display has blue "holes" in it. What gives?

The holes are areas where the predicted response is out-of-range. Try using `clip=FALSE` (Section 5.4).

I want to add lines or points to a plot created by plotmo. and am having trouble getting my axis scaling right.

Use `do.par=FALSE` or `do.par=2`. With the default `do.par=TRUE`, plotmo restores the `par` parameters and axis scales to their values before plotmo was called.

After plotmo reports an error, traceback() says "No traceback available"

Try using `trace = -1` when invoking plotmo. This will often (but not always) allow traceback at the point of failure.

How to cite plotmo

Stephen Milborrow. *plotmo: Plot a Model's Response and Residuals.*
R Package (2015).

```
@Manual{plotmopackage,
  title  = {plotmo: Plot a Model's Response and Residuals},
  author = {Stephen Milborrow},
  year   = {2015},
  note   = {R package},
  url    = {http://CRAN.R-project.org/package=plotmo }
}
```

10 Common error messages

This section list some common error messages.

○ `Error in match.arg(type): 'arg' should be one of ...`

The message is probably issued by the `predict` method for the model. Set plotmo's `type` argument to a legal value for the model, as described on the help page for the `predict` method for the model.

○ `Error: cannot get the original model predictors`
○ `Error: model does not have a 'call' field or an 'x' field`

These and similar messages mean that plotmo cannot get the data it needs from the model (Section 11).

Try simplifying the way the model function is called. Try using the `x,y` interface instead of the `formula` interface, or vice versa.

Perhaps `keepxy` or similar is needed in the call to the model function, so the data is attached to the model object and available for plotmo.

A workaround is to manually add the `x` and `y` fields to the model object before calling plotmo, like this

```
    model$x <- xdata
    model$y <- ydata
```

where `xdata` and `ydata` are the x and y matrices used to build the model. This workaround often suffices for plotmo to do its job, assuming the model has a standard `predict` method that accepts `data.frames` (some `predict` methods accept only matrices).

○ `Error: predict.lm(xgrid, type="response") returned the wrong length`
○ `Warning: 'newdata' had 100 rows but variable(s) found have 30 rows`
○ `Error: variable 'x' was fitted with type "nmatrix.2" but type "numeric" was supplied`
○ `Error in model.frame: invalid type (list) for variable 'x[,3]'`

These and similar messages usually mean that `predict` is misinterpreting the new data generated by plotmo.

The underlying issue is that many `predict` methods, including `predict.lm`, seem to reject any reasonably constructed new data if the function used to create the model was called in an unconventional way.

The workaround is to simplify the way the model function is called. Use a formula and a data frame, or at least explicitly name the variables rather than passing a matrix on the right hand side of the formula. Use simple variable names (so `x1` rather than `dat$x1`, for example).

If the symptoms persist after changing the way the model is called, it's possible that the model doesn't save the data in a form accessible by plotmo (Section 11).

11 Accessing the model data

This section discusses some of plotmo's internals. Plotmo needs to access the data used to build the model. It does that with the method functions listed below.

As an example, the job of the `plotmo.x` function is to return the `x` matrix used when the model was built. The default function `plotmo.x.default` essentially[2] does the following:

(i) it uses `model$x`

(i) if that doesn't exist, it uses the rhs of the model formula (so if the model was built with a formula, it must have a `terms` field)

(i) if it can't access that, it uses `model$call$x`

(i) if all that fails, it prints an error message.

The default method suffices for models that save the call and data with the model in a standard way (described in detail in the Guidelines for S3 Regression Models [12]). Specific method functions can often be written to handle other situations.

11.1 Method functions

The plotmo method functions are listed below. Use `trace=2` to see plotmo calling these functions.

- `plotmo.x` Return the model x matrix. The default method is described above.
- `plotmo.y` Return the model y matrix. Similar to `plotmo.x`.
- `plotmo.predict` Make predictions on new data. This is invoked for each subplot.

  The default method calls the usual `predict` method for the model. The prediction `newdata` for each subplot is the grid of values for the subplot. The `newdata` is a `data.frame` and not a matrix to allow both numerics and factors in general.

  Model-specific predict methods exist for some model classes, usually because a minor tweak is needed. For example plotmo has an internal one-line function `plotmo.predict.lars`—this converts `newdata` to a `matrix` before passing it to `predict.lars`, because `predict.lars` accepts only matrices.
- `plotmo.type` Select a `type` argument suitable for the current model's `predict` method.
- `plotmo.prolog` Called at the start of plotmo to do any model-specific initialization.
- `plotmo.singles` Figure out which variables should appear in degree1 plots.

[2]There are actually a few more complications. For example, it also tries the `model` field saved with some `lm` models.

- `plotmo.pairs` Figure out which variables should appear in degree2 plots.
- `plotmo.convert.na.nresponse` Convert the default `nresponse` argument to a column number for multiple response models.
- `plotmo.pint` Get the prediction intervals when plotmo's `level` argument is used.

11.2 Environment for the model data

One `x` isn't necessarily the same as another `x`. Plotmo must access the data used to build the model in the correct environment:

- It uses the `.Environment` attribute of `model$terms`. (The `terms` field is standard for models built with a formula.)
- If that isn't available it uses `model$.Environment`. (Most models don't have such a field.)
- If that isn't available it uses `parent.frame()`. This last resort is correct if the model was built in the user's workspace and plotmo is called from the same workspace. But all bets are off if the model was created within a function and plotmo is called from a different function.

Note that the environment isn't actually necessary if the data is saved with the model in the `x` and `y` fields of the model. Some models allow you to save `x` and `y` with a `keepxy` or similar argument (plotmo will use those fields if available).

References

[1] Julian Faraway. *Linear Models With R.* CRC, 2009. Cited on page 11.

[2] John Fox, Sanford Weisberg, et al. *car: Companion to Applied Regression*, 2014. R package, `http://CRAN.R-project.org/package=car`. Cited on page 5.

[3] John Fox, Sanford Weisberg, et al. *effects: Effect Displays for Linear, Generalized Linear, Multinomial-Logit, Proportional-Odds Logit Models and Mixed-Effects Models*, 2014. R package, `http://CRAN.R-project.org/package=effects`. Cited on page 5.

[4] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *Regularization Paths for Generalized Linear Models via Coordinate Descent.* JASS, 2010. Cited on page 7.

[5] Stefan Fritsch and Frauke Guenther; following earlier work by Marc Suling. *neuralnet: Training of neural networks*, 2012. R package, `http://CRAN.R-project.org/package=neuralnet`. Cited on page 7.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd Edition).* Springer, 2009. Downloadable from `http://web.stanford.edu/~hastie/ElemStatLearn`. Cited on page 5.

[7] Trevor Hastie. *gam: Generalized Additive Models*, 2014. R package, `http://CRAN.R-project.org/package=gam`. Cited on page 3.

[8] Roger Koenker. *quantreg: Quantile Regression*, 2014. R package, `http://CRAN.R-project.org/package=quantreg`. Cited on pages 7 and 10.

[9] Andy Liaw, Mathew Weiner; Fortran original by Leo Breiman, and Adele Cutler. *randomForest: Breiman and Cutler's random forests for regression and classification*, 2014. R package, `http://CRAN.R-project.org/package=randomForest`. Cited on pages 3 and 7.

[10] Nicolai Meinshausen. *quantregForest: Quantile Regression Forests*, 2014. R package, `http://CRAN.R-project.org/package=quantregForest`. Cited on pages 7 and 10.

[11] S. Milborrow. *rpart.plot: Plot rpart models. An enhanced version of plot.rpart*, 2011. R package, `http://www.milbo.org/rpart-plot`. Cited on page 8.

[12] S. Milborrow. *Guidelines for S3 regression models*, 2015. Vignette for R package plotmo, `http://www.milbo.org/doc/modguide.pdf`. Cited on page 14.

[13] S. Milborrow. *plotmo: Plot a model's response and residuals*, 2015. R package, `http://CRAN.R-project.org/package=plotmo`. Cited on page 2.

[14] S. Milborrow. *Variance models in earth*, 2015. Vignette for R package earth, `http://www.milbo.org/doc/earth-varmod.pdf`. Cited on page 11.

[15] S. Milborrow. Derived from mda:mars by T. Hastie and R. Tibshirani. *earth: Multivariate Adaptive Regression Splines*, 2011. R package, `http://www.milbo.users.sonic.net/earth`. Cited on pages 7 and 10.

[16] Greg Ridgeway et al. *gbm: Generalized Boosted Regression Models*, 2014. R package, `http://CRAN.R-project.org/package=gbm`. Cited on pages 3 and 7.

[17] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2014. R package, `http://CRAN.R-project.org/package=rpart`. Cited on pages 3 and 7.

[18] W.N. Venables and B.D. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2014. R package, `http://www.stats.ox.ac.uk/pub/MASS4`. Cited on page 7.

[19] W.N. Venables and B.D. Ripley. *nnet: Feed-forward Neural Networks and Multinomial Log-Linear Models*, 2014. R package, `http://CRAN.R-project.org/package=MASS`. Cited on page 3.

[20] Sanford Weisberg. *Applied Linear Regression (4th Edition).* Wiley, 2013. Cited on page 11.

[21] Simon Wood. *mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation*, 2014. R package, `http://CRAN.R-project.org/package=mgcv`. Cited on page 10.

[22] Hao Helen Zhang and Chen-Yen Lin. *cosso: Fit Regularized Nonparametric Regression Models Using COSSO Penalty.*, 2013. R package, `http://CRAN.R-project.org/package=cosso`. Cited on page 7.