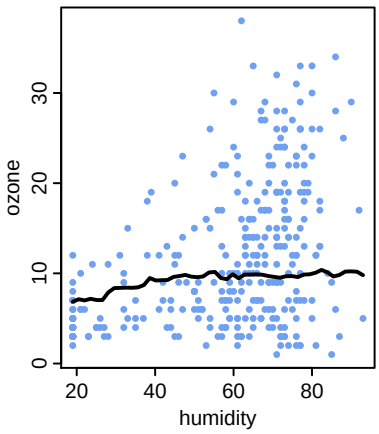
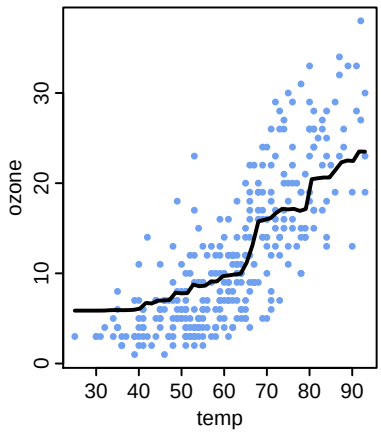
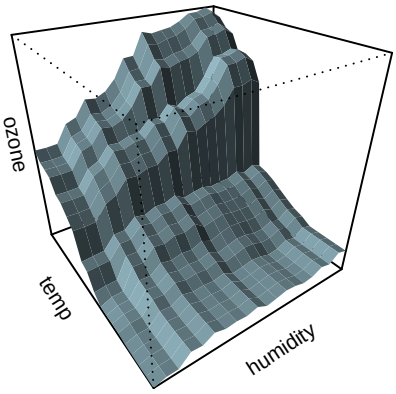
plotmo

Stephen Milborrow

May 27, 2015







Contents

1 Introduction

The `plotmo` function [12] plots regression surfaces. The frontspiece on the title page is an example. That example is for a random forest, but `plotmo` can be used on a wide variety of R models.

It creates a plot for a model variable by plotting the predicted response as the variable is changed, with all other variables held at their median values. For interaction plots, two variables are changed while the others are held at their medians.

The first level is used instead of the median for factors. You can change that with the `grid.func` and `grid.levels` arguments.

Each graph shows only a thin slice of the data, because most variables are fixed. Please be aware of that when interpreting the graphs — over-interpretation is a temptation.

`Plotmo` was originally part of the `earth` package [13] and a few connections to that package still remain.

2 Some examples

Here are some examples which illustrate `plotmo` on various models (Figure 1). The models here are just for illustrating `plotmo` and shouldn't be taken too seriously.

```
# 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))

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
```

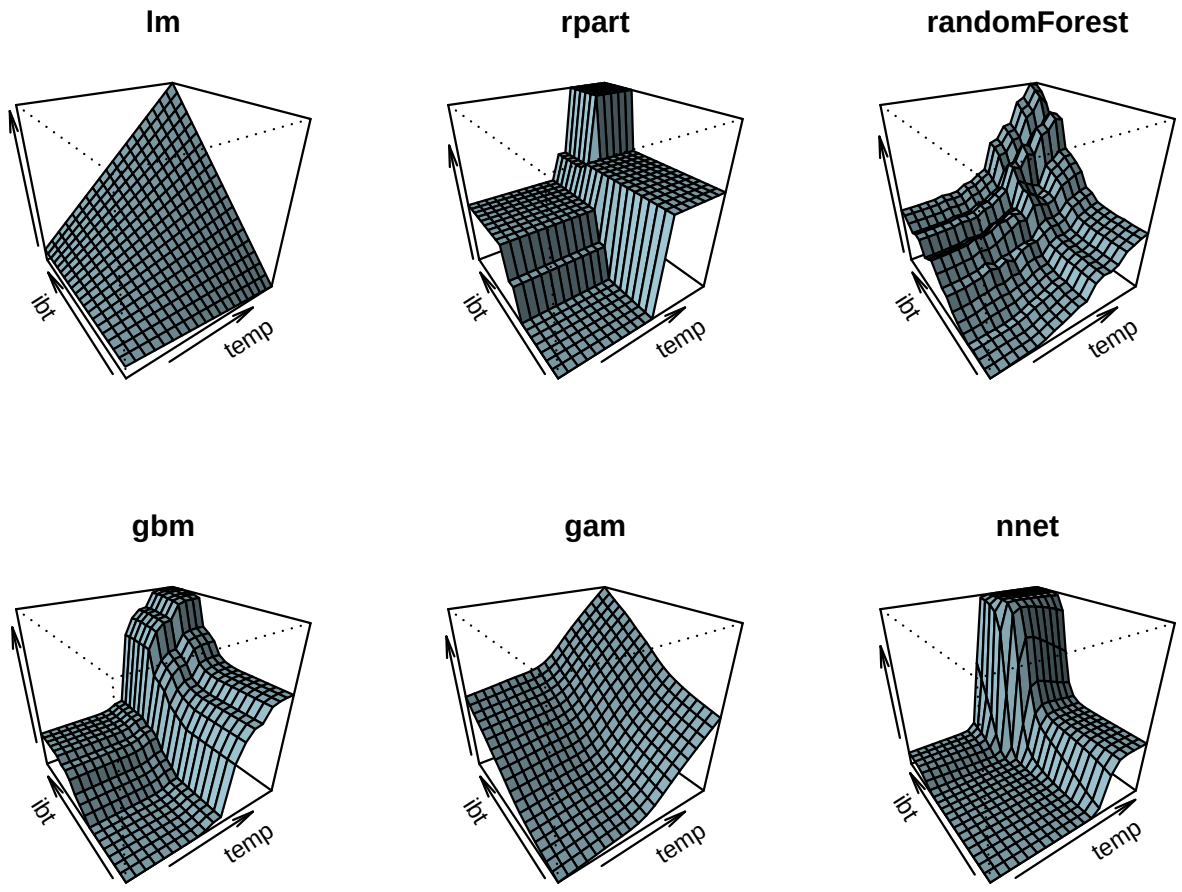


Figure 1: `Plotmo` *on various models.*
These plots were generated with the models on the previous page. A single `degree2` *plot for each model type is illustrated here.*

```
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, 14, 15, 17].

3 Limitations

In general, this function won't work on models that don't save the call and data with the model in a standard way (Section 9).

It's best to keep the variable names and formula in the original call to the model-building function simple. The current version of `plotmo` parses model formulas with `grep`, and is liable to get confused by complex formulas. Use temporary variables or

`attach` rather than using `$` and similar in formulas.

You may get error messages if there are NAs in the data.

4 Alternatives

An alternative approach is to use partial-dependence plots (e.g. Hastie et al. [6] Section 10.13.2). `Plotmo` sets the "other" variables to their median value, whereas in a partial-dependence plot at each plotted point the effect of the other variables is averaged. In general, partial-dependence plots and `plotmo` plots will differ, but for additive models (no interaction terms) the *shape* of the curves is identical.

The `termplot` function in the standard `stats` package is effective but can be used only on models with a `predict` method that supports `type="terms"`, and it doesn't generate degree2 plots.

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.

All of these plots are different ways of condensing a multi-dimensional model onto the two dimensions of a page.

5 The `ylim` and `clip` arguments

Plotmo determines `ylim` for the graphs automatically. If the automatic `ylim` isn't correct for your 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 to 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.

Old versions of `plotmo` used a slightly different algorithm for clipping `ylim`.

6 Which variables are plotted?

The set of variables plotted for some common models is listed below [9, 13–16].

The default behavior for your model may leave out some variables that you would like to see. In that case, use `all1 = TRUE` and `all2 = TRUE`.

- `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`

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

  `degree2` pairs of the four most important variables

- `gbm`

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

  `degree2` pairs of the four variables with the largest relative influence

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

  These are processed using `plotmo`'s default methods (Section 9):

  `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, 14–16, 20].

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

By default, `predict.glmnet` is called with `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 is available, else it uses the plotting routines built into the `rpart` package.

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 2):

```
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
```

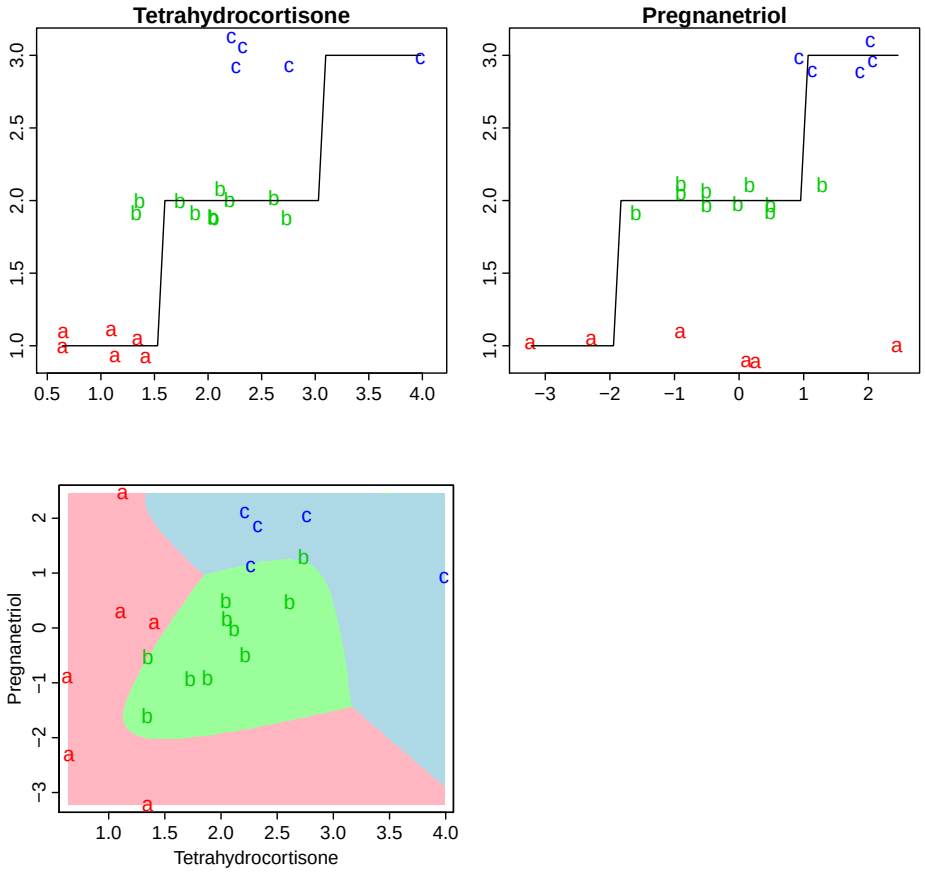


Figure 2: *A* `qda` *model of the log Cushings data. The background colors in the interaction plot show the predicted class.*
The slightly messy look of the `a`,`b`,`c` *labels in the top two plots is caused by* `plotmo`*'s automatic jittering of factor labels (see the* ***`jitter`*** *argument).*

8 Prediction intervals (the `level` argument)

Use the `level` argument to plot pointwise confidence or prediction intervals. The `predict` method of the model object must support this. Examples (Figure 3):

```
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, 13, 19].

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

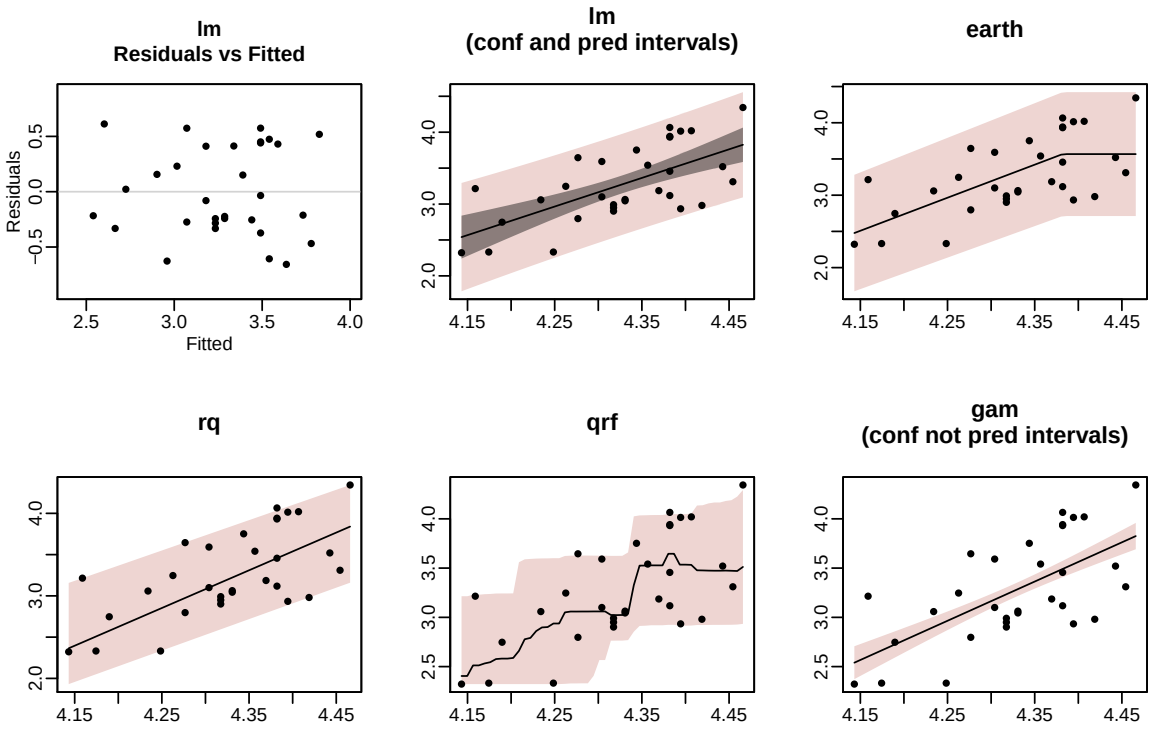


Figure 3: *Prediction intervals with* `plotmo`*. These plots were produced by the code on the previous page.*

plotted by `plotmo`. Currently only `lm` supports both types of interval on new data (see `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] `http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf`.

Assumptions

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(which=1)` and with earth models use `plot.earth(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, you can't trust the intervals. One good place for more background on residual analysis is the *Regression Diagnostics: Residuals* section in Weisberg [18].

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 you need *simultaneous* confidence bands, which none of the models above support.

9 Accessing the model data

`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[1] does the following:

(i) it uses `model$x`

(i) if that doesn't exist, it uses the rhs of the model formula (so the model 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. Specific method functions can often be written to handle other situations.

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`.
- `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.

[1] 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.

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.

10 FAQ

I'm not seeing any interaction plots. How do I change that?

Use `all2=TRUE`. By default, degree2 plots are drawn only for some types of model (Section 6).

`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).

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"

You can try using `trace = -1` when invoking `plotmo`. This will often (but not always) allow traceback at the point of failure. (Actually, error handling in `plotmo` and `plotres` needs some work.)

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 }
}
```

11 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 your 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: model does not have a 'call' field or an 'x' field`

Plotmo cannot get the data it needs from the model (Section 9).

A workaround is to 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.frame`s.

Otherwise contact the maintainer of `plotmo` — it is often a straightforward exercise to add support for a new type of model.

◦ `Error: cannot get the original model predictors`

These and similar messages mean that `plotmo` cannot get the data it needs from the model.

You can try simplifying the way the model function is called.

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

Please see also the discussion for the above error message.

◦ `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 9).

References

[1] Julian Faraway. *Linear Models With R*. CRC, 2009. Cited on page 9.

[2] John Fox, Sanford Weisberg, et al. *car: Companion to Applied Regression*, 2014. R package. Cited on page 4.

[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. Cited on page 4.

[4] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *Regularization Paths for Generalized Linear Models via Coordinate Descent*. JASS, 2010. Cited on page 6.

[5] Stefan Fritsch and Frauke Guenther; following earlier work by Marc Suling. *neuralnet: Training of neural networks*, 2012. R package. Cited on page 6.

[6] Hastie, Tibshirani, and Friedman. *The Elements of Statistical Learning (2nd ed.)*. Springer, 2009. `http://www-stat.stanford.edu/~hastie/pub.htm`. Cited on page 4.

[7] Trevor Hastie. *gam: Generalized Additive Models*, 2014. R package version 1.09.1. Cited on page 3.

[8] Roger Koenker. *quantreg: Quantile Regression*, 2014. R package. Cited on pages 6 and 8.

[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. Cited on pages 3 and 5.

[10] Nicolai Meinshausen. *quantregForest: Quantile Regression Forests*, 2014. R package. Cited on pages 6 and 8.

[11] Stephen Milborrow. *rpart.plot: Plot rpart Models. An Enhanced Version of plot.rpart*, 2011. R package. Cited on page 6.

[12] Stephen Milborrow. *plotmo: Plot a Model's Response and Residuals*, 2015. R package. Cited on page 2.

[13] S. Milborrow. Derived from mda:mars by T. Hastie and R. Tibshirani. *earth: Multivariate Adaptive Regression Splines*, 2011. R package. Cited on pages 2, 5, and 8.

[14] Greg Ridgeway et al. *gbm: Generalized Boosted Regression Models*, 2014. R package. Cited on pages 3, 5, and 6.

[15] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2014. R package. Cited on pages 3, 5, and 6.

[16] W. N. Venables and B. D. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2014. R package. Cited on pages 5 and 6.

[17] W. N. Venables and B. D. Ripley. *nnet: Feed-forward Neural Networks and Multinomial Log-Linear Models*, 2014. R package. Cited on page 3.

[18] Sanford Weisberg. *Applied Linear Regression (4th Edition)*. Wiley, 2013. Cited on page 9.

[19] Simon Wood. *mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation*, 2014. R package. Cited on page 8.

[20] Hao Helen Zhang and Chen-Yen Lin. *cosso: Fit Regularized Nonparametric Regression Models Using COSSO Penalty.*, 2013. R package. Cited on page 6.