Notes on the `plotmo` package

Stephen Milborrow

December 8, 2014

Contents

1 Introduction

This document is a set of notes to accompany the `plotmo` R package [7, 10].

`Plotmo` can be used on a wide variety of regression models. It plots a `degree1` (main effect) plot by calling `predict` to predict the response when changing one variable while holding all other variables at their median values. For `degree2` (interaction) plots, two variables are changed while holding others at their medians.

The first level is used instead of the median for factors. You can change this 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 graph — over-interpretation is a temptation.

There is section on `plotmo` in the vignette for the `rpart.plot` package [8] "Plotting `rpart` trees with `prp`". This vignette is also downloadable from `http://www.milbo.org/rpart-plot/prp.pdf`.

`Plotmo` was originally part of the `earth` package [9] and a few connections to that package still remain.

Citing the package

If you use this package in a published document, please do the right thing and cite it [7]:

> Stephen Milborrow. *plotmo: Plot a model's response while varying the values of the predictors*. R Package (2011).

```
@Manual{plotmopackage,
  author = {Stephen Milborrow},
  title  = {{plotmo: Plot a model's response while varying
            the values of the predictors}},
  year   = {2011},
  note   = {R package},
  url    = {http://CRAN.R-project.org/package=plotmo }
}
```

2 Using plotmo on various models

Here are some examples which illustrate plotmo on various objects (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.model <- lm(O3 ~ humidity + temp*ibt, data=oz)       # linear model
plotmo(lm.model, col.response="gray", nrug=-1)

library(rpart)                                          # rpart
rpart.model <- rpart(O3 ~ ., data=oz)
plotmo(rpart.model, all2=TRUE)

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

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

library(mgcv)                                           # gam
gam.model <- gam(O3 ~ s(humidity)+s(temp)+s(ibt)+s(temp,ibt), data=oz)
plotmo(gam.model, level=.95, all2=TRUE)

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

library(MASS)                                           # qda
lcush <- data.frame(Type=as.numeric(Cushings$Type),log(Cushings[,1:2]))
lcush <- lcush[1:21,]
qda.model <- qda(Type~., data=lcush)
plotmo(qda.model, type="class", all2=TRUE,
       type2="contour", ngrid2=100, nlevels=2, drawlabels=FALSE,
       col.response=as.numeric(lcush$Type)+1,
       pch.response=as.character(lcush$Type))
```

The packages used in the above code are [5, 11, 12, 14, 16].

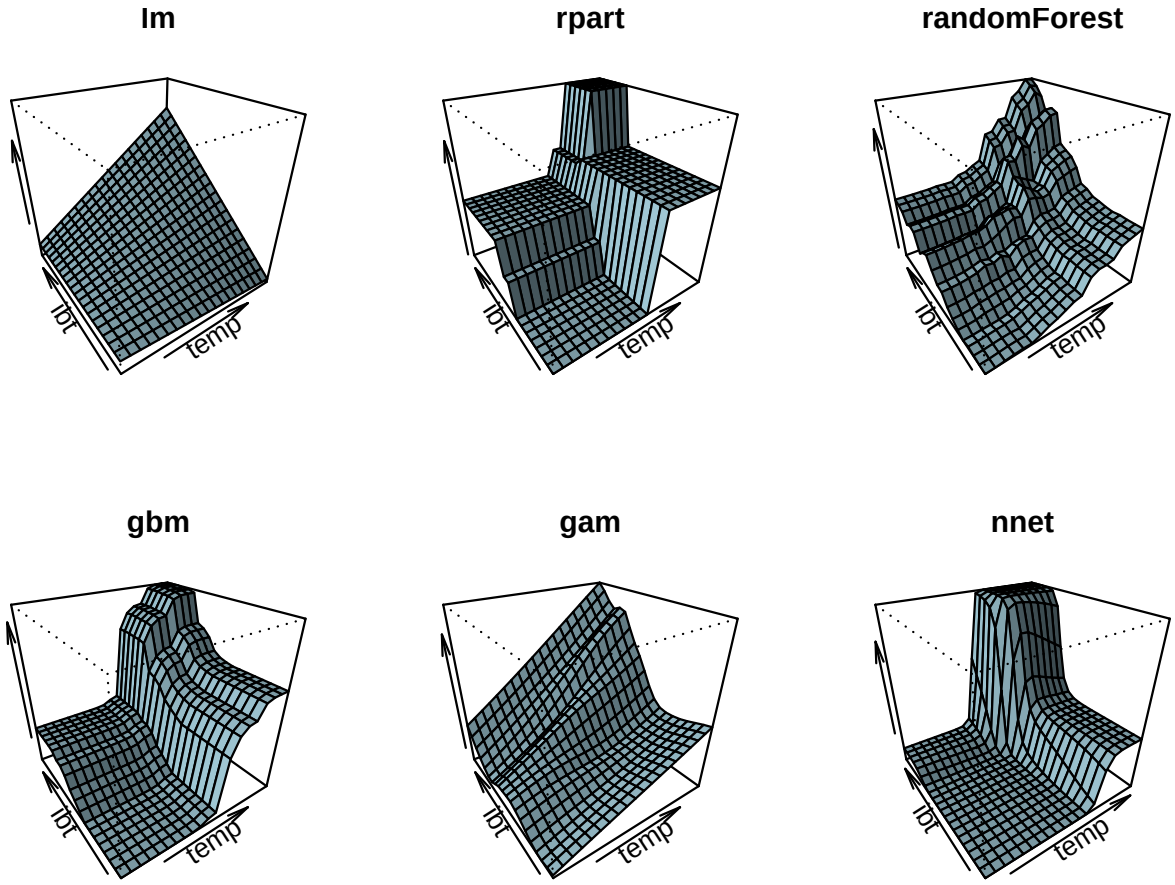

Figure 1: `Plotmo` *on various models.*
These plots were generated with the models on on the previous page. Just one `degree2` *plot for each model type is illustrated here.*

3 Limitations

NAs aren't supported. To prevent confusing error messages from functions called by `plotmo`, it is safest to remove NAs before building your model.

(However, `rpart` models are treated specially by `plotmo`. For these, `plotmo` predicts with `na.pass` so `plotmo` can be used with `rpart`'s default NA handling.)

Keep the variable names in the original model formula simple. Use temporary variables or `attach` rather than using `$` and similar in formulas.

4 Environment for the model data

`Plotmo` evaluates the model data in the R environment used when the model was built, if that environment was saved with the model (typically this is the case if the formula interface was used to the model function). If the environment wasn't saved with the model (typically if the `x,y` interface was used), the model data is evaluated in the environment in which `plotmo` is called.

In other words, `plotmo` uses the environment attribute of `object$terms`, and if that's not available it uses `parent.frame()`.

5 The `clip` argument

With the default `clip=TRUE`, predicted values out of the expected range aren't displayed.

Generally, the "expected range" is the range of the response `y` used when building the model. But that depends on the type of model, and `plotmo` knows about some special cases. For example, it knows that for some models we are predicting a probability, and it scales the axes accordingly, `0` to `1`. However, `plotmo` cannot know about every possible model and prediction `type`, and will sometimes determine the expected response range incorrectly. In that case use `clip=FALSE`.

The default `clip` is `TRUE` because it is a useful sanity check to test that the predicted values are in the expected range. While not necessarily an error, predictions outside the expected range are usually something we want to know about. Also, with `clip=FALSE`, a few errant predictions can expand the entire y-axis, making it difficult to see the shape of the other predictions.

6 Alternatives

An alternative approach is to use partial-dependence plots (e.g. Hastie et al. [3] 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 the *shape* of the curves will match identically. Eventually `plotmo` may be enhanced to draw partial-dependence plots.

The `termplot` function 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 [1]). These plots are orientated towards linear models. The `effects` package is also of interest [2].

7 Which variables are plotted?

The set of variables plotted for some common objects is listed below [5, 9, 11–13].

The default behavior 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` all variables

  `degree2` pairs of the four most important variables (ranked on the first column of `object$importance`)

- `gbm`

  `degree1` variables with `relative.influence` >= 1%

  `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)`

8 Prediction intervals

Use the `level` argument to plot pointwise confidence or prediction intervals. The `predict` method of the object must support this. Examples (Figure 2):

```
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, main="lm") # Residual vs Fitted graph
plotmo(lm.model, level=.90, col.response=1,
       main="lm\n(conf and pred intervals)", do.par=F)

                                                          # earth (requires earth 3.3)
library(earth)
earth.model <- earth(Volume~Height, data=log.trees,
                     nfold=5, ncross=30, varmod.method="lm")
plotmo(earth.model, level=.90, col.response=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, col.response=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, col.response=1, main="qrf", do.par=F)

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

The packages used in the above code are [4, 6, 9, 16].

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*).

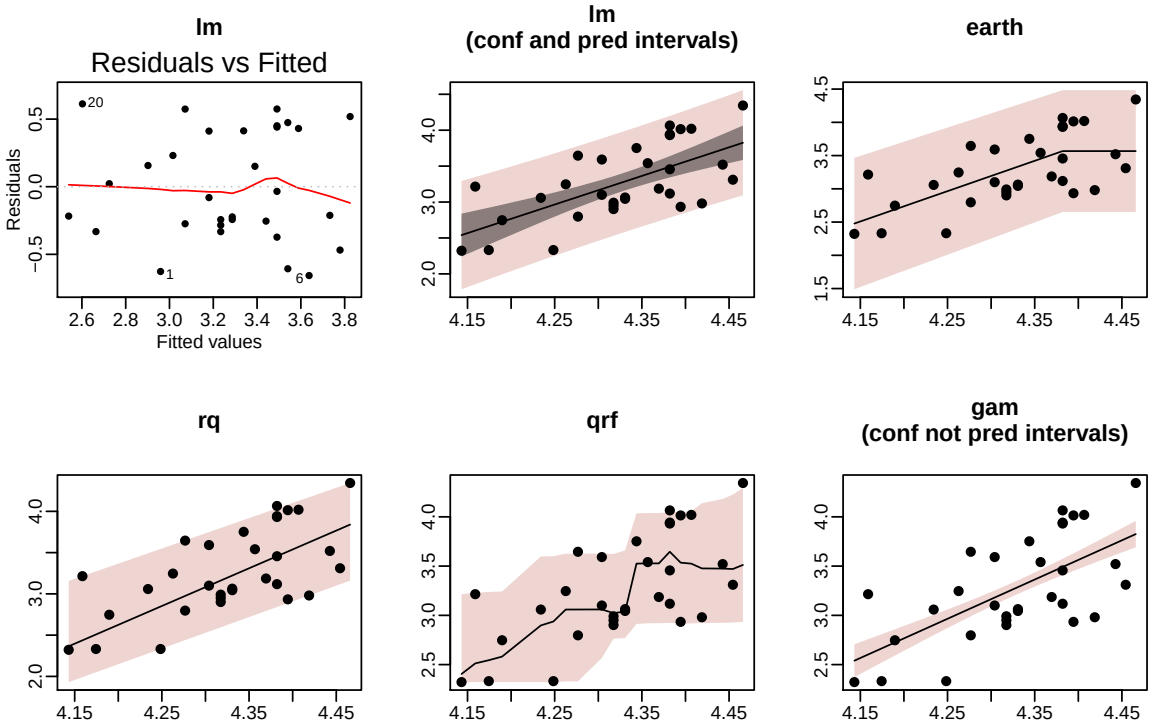

Figure 2: *Prediction intervals with* `plotmo`*. These plots were produced by the code on the previous page.*

A reference is Section 3.5 of Julian Faraway's online linear regression book `http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf`.

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

Assumptions

Be aware of the assumptions made to generate the limits. At the very least, the model needs to fit the data adequately. Most models will impose further conditions. For example, linear models 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.which(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 cannot trust the intervals.

One good place for more background on residual analysis is *Regression Diagnostics: Residuals* in Weisberg [15]. See also the "Variance Models with earth" vignette for the `earth` package.

These are *pointwise* limits. They can 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 Extending plotmo

`Plotmo` needs to access the data used to build the model. It does that with the method functions listed below. The default methods suffice for many objects.

For example, the job of the `get.plotmo.x` function is to return the `x` matrix used when building the given model. The default function `get.plotmo.x.default` essentially does the following:

(i) it uses `model[["x"]]`

(ii) if that doesn't exist, it uses the rhs of the model formula

(iii) if that doesn't exist, it uses `model$call[["x"]]`

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

The default methods will fail if the model function didn't save the data or call with the object in a standard fashion (and `plotmo` will issue an error message). Object-specific methods can usually be written to deal with such issues. For examples, See the source files `plotmo.methods.*.R`.

The methods are:

`plotmo.prolog`

called before plotting begins, sanity check of the `object`

`plotmo.predict`

invokes `predict` for each sub-plot

`get.plotmo.x`

the model matrix `x`

`get.plotmo.y`

the model response `y`

`get.plotmo.default.type`

the value of the `type` argument when not specified by the user

`get.plotmo.singles`

the vector of variables to be plotted in degree1 plots

`get.plotmo.pairs`

the array of pairs to be plotted in degree2 plots

`get.plotmo.ylim`

the value of `ylim` when not specified by the user

`get.plotmo.clip.limits`

the clip range when `clip=TRUE`

`get.plotmo.nresponse`

the correct column when the response has multiple columns

`plotmo.pint`

the prediction intervals for `plotmo`'s `level` argument

10 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 `object`. Set `plotmo`'s `type` argument to a legal value for that object, as described on the help page for the `predict` method for the object.

`Error: predicted values are out of ylim, try clip=FALSE`

Probably `plotmo` has incorrectly determined the expected range of the response, and hence also `ylim`. Re-invoke `plotmo` with `clip=FALSE`. See Section 5 "The `clip` argument".

`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 work-around is to simplify or standardize 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. Use simple variable names (so `x1` rather than `dat$x1`, for example).

If the symptoms persist after changing the way the model is called, and the model isn't one of those listed in Section 7, it is possible that the model class isn't supported by `plotmo`. See Section 9.

`Error: get.plotmo.x.default cannot get the x matrix`

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

You can try simplifying and standardizing the way the model function is called, as described above. Perhaps you need to use `keepxy` or similar in the call to the model function, so the data is attached to the object and available for `plotmo`. Is a variable that was used to build the model no longer available in the environment when `plotmo`

is called?

`Error: this object is not supported by plotmo`

`Plotmo`'s default methods are insufficient for your model object. See Section 9 (and contact the author — this is often easy to fix).

11 FAQ

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

Use `all2=TRUE`. By default, degree2 plots are drawn only for some types of model. See Section 7.

The `persp` display is unnaturally jagged. How can I change that?

Use `clip=FALSE`. The jaggedness is probably an artifact of the way `persp` works at the boundaries. You can also try increasing `ngrid2`.

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`.

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

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.

References

[1] John Fox, Sanford Weisberg, et al. *car: Companion to Applied Regression*, 2014. R package. Cited on page 6.

[2] 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 6.

[3] Hastie, Tibshirani, and Friedman. *The Elements of Statistical Learning (2nd ed.)*. Springer, 2009. `http://www-stat.stanford.edu/~hastie/pub.htm`. Cited on page 5.

[4] Roger Koenker. *quantreg: Quantile Regression*, 2014. R package. Cited on page 8.

[5] 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 7.

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

[7] Stephen Milborrow. *plotmo: Plot a Model's Response while Varying the Values of the Predictors*, 2011. R package. Cited on page 2.

[8] Stephen Milborrow. *rpart.plot: Plot rpart Models. An Enhanced Version of plot.rpart*, 2011. R package. Cited on page 2.

[9] Stephen Milborrow. Derived from mda:mars by Trevor Hastie and Rob Tibshirani. *earth: Multivariate Adaptive Regression Spline Models*, 2009. R package. Cited on pages 2, 7, and 8.

[10] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2014. Cited on page 2.

[11] Greg Ridgeway et al. *gbm: Generalized Boosted Regression Models*, 2014. R package. Cited on pages 3 and 7.

[12] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2014. R package. Cited on pages 3 and 7.

[13] W. N. Venables and B. D. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2014. R package. Cited on page 7.

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

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

[16] Simon Wood. *mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation*, 2014. R package. Cited on pages 3 and 8.