Plotting regression surfaces with `plotmo`
1 Introduction

The plotmo function in the plotmo R package [17] 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 in the model. Each such degree 1 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 degree 2 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.

Plotmo invokes predict internally to generate the predicted response. Which specific predict method gets invoked is determined by the model’s class — thus for say a randomForest model, predict.randomForest is invoked.

Plotmo also supports partial dependence plots (Section 9), as an alternative to its default method of plotting which simply holds the background variables at their medians.

2 Examples

Here are some examples which illustrate plotmo on various models. Figure 1 shows the resulting plots.

```r
# 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
```

[17]
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

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 [9,11,20,21,23].

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?”.

3
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.

3.1 Inherent limitations

The plots can give only a partial view of the model. Each plot shows only a thin slice of the data with the background variables pegged at fixed values.

For example, in Figure 2 the response curve as \( x_1 \) varies depends greatly on the value of the other variable \( x_2 \). Compare the response curve when \( x_2 \) is at 15 (say) to the curve when \( x_2 \) is pegged at its median 72. Please be aware of this loss of information when interpreting the graphs. Over-interpretation is a temptation.

Similar issues arise for partial-dependence plots, although harder to illustrate.

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.

---

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. The slice doesn’t tell you what’s going on at the other end of the loaf.

Figure 2: The shape of the response curve as \( x_1 \) varies is quite different for slices at different values of \( x_2 \).

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

There are practical limitations to do with the way some models are built in R.

Plotmo needs to access the data used when building the model, so it can formulate new data to pass to `predict`. For some models this isn’t possible. Some models don’t save the `call` or any references to the original data. For details see Section 13 “Accessing the model data”.

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). For more details see Section 12 “Common error messages”.
4 Alternatives to plotmo

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 such plots, although often overlooked, is the humble residuals-vs-fitted plot. The residuals plot is helpful for detecting unusual observations and other potential issues with the data. 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. (See e.g. Hastie et al. [8] Section 10.13.2. To my knowledge, partial-dependence plots were first described in Friedman’s gradient boosting paper [5].) 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. For the special case of 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.

Note added Nov 2016: Plotmo now supports partial-dependence plots (Section 9).

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 “Inherent limitations”).

The randomForest and gbm packages have functions for generating partial-dependence plots for their respective models. The pdp [7] package, similar in spirit to plotmo, offers partial dependence plots for a variety of models

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. For details see Section 6 “Which variables get plotted?”.

5.2 The type and nresponse arguments

Some models can make different kinds of predictions. For example, for classification models we can typically predict a response class or predict a probability. When calling predict internally, by default plotmo tries to automatically select a suitable response type for the model (often type="response"; use trace=1 to see what plotmo uses). 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 if columns are named.

Section 8 “Plotting classification models” gives some examples of using the type and nresponse arguments.

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. 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 most common 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

```r
grid.func = mean
```

or

```r
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` (the vertical range) for the graphs automatically. If this automatic `ylim` isn't correct for your model, explicitly use the `ylim` argument when invoking plotmo.

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 obvious way for plotmo to automatically set `ylim` is to use the range of predicted values over all the plots. 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 are quite rare, but that depends on the model.
6 Which variables get plotted?

The default set of variables plotted for some common models is listed below [11,18,20–22].

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 set of displayed variables use the degree1 and degree2 arguments. Sometimes it’s useful to use all those arguments: first expand the set with all1 and all2, then trim that back with degree1 and degree2.

- **earth**
  - degree1 variables in additive (non interaction) terms
  - degree2 variables appearing together in interaction terms

- **rpart**
  - degree1 variables used in the tree
  - degree2 variables which appear in parent-child pairs in the tree

- **randomForest**
  A 4 × 4 grid of plots (or less if fewer variables) as follows:
  - degree1 The ten most important variables.
    How importance is measured depends on whether model was built with importance=TRUE, and whether the model is a regression or classification model. Use trace=1 in the call to plotmo to see which measure of importance is used.
  - 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 The ten most important variables (measured by relative.influence). Variables with relative.influence < 1% are ignored.
  - degree2 Pairs of the four variables with the largest importance (thus six degree2 plots)

- **lm, glm, gam, lda, etc.**
  These are processed using plotmo’s default methods (Section 13):
  - 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, 6, 10, 12, 13, 19–22, 26].

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

By default, predict.glmnet is called with type="response" and s = 0.
Change that by passing say predict.s=.02 to plotmo or plotres.

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

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

By default, predict.svm (e1071 package) is called with decision.values and probability set to FALSE, but as usual we can change that by passing those arguments to plotmo with a predict. prefix, and plotmo will use those values if specified.

For rpart models, plotres uses the rpart.plot package [14] 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).

For C5.0 models (from the C50 package), variable selection works as in gbm models (page 9) but with relative importance measured using C5imp.
7.1 The MASS package and lda and qda

The predict methods for lda and qda 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):

```r
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 Classification models

This section discusses classification models, focusing on models with a two-class response (where the classes simply may be \texttt{TRUE} and \texttt{FALSE}). This section repeats information in other parts of this document, but is geared towards users of classification models.

With classification models we often have to take care to set plotmo’s \texttt{type} and \texttt{nresponse} arguments appropriately (Section 5.2). The scale of the response plotted by plotmo is determined by the \texttt{type} argument and possibly other arguments for \texttt{predict}. (These get passed to \texttt{predict} via plotmo.) For example, for binomial \texttt{glm} models we can predict probabilities (\texttt{type="response"}, which can vary from 0 to 1) or log-odds (\texttt{type="link"}, which can vary from -infinity to +infinity, although in practice the response is restrained to a reasonable range).

For some models, the predict method returns multiple columns, and we need to select the appropriate column using plotmo’s \texttt{nresponse} argument. For example, when predicting probabilities for \texttt{randomForest} two-class models, \texttt{predict.randomForest} returns two columns. We must use plotmo’s \texttt{nresponse} argument to select the column for the class of interest. If we select the other column, the plotted curves will be upside down.

For some classification models, plotmo doesn’t calculate \texttt{ylim} correctly. In that case, explicitly pass \texttt{ylim} to plotmo. We see that being done in the \texttt{svm} example below.

Here are some example models with a two-class response. We use a subset of the \texttt{iris} data for simplicity, and plot the probability of a \texttt{virginica} response. Figure 4 shows the plots. Optionally add \texttt{pmethod="partdep"} to the calls to plotmo below to generate partial-dependence instead of classical plotmo plots.

```r
data(iris)
iris1 <- data.frame(virginica = iris$Species == "virginica",
                   length = iris$Sepal.Length,
                   width = iris$Sepal.Width)

glm.mod <- glm(virginica~., data=iris1, family="binomial") ## glm
plotmo(glm.mod) # default type="response" returns probabilities

library(earth) ## earth
earth.mod <- earth(virginica~., data=iris1, degree=2, glm=list(family=binomial))
plotmo(earth.mod)

library(mgcv) ## mgcv gam
gam.mod <- gam(virginica~ s(length)+s(width), data=iris1, family=binomial())
plotmo(gam.mod)

library(gbm) ## gbm
gbm.mod <- gbm(virginica~., data=iris1, dist="bernoulli", inter=2, n.trees=1000)
plotmo(gbm.mod)
```
library(e1071) ## svm
iris.fac <- iris1 # svm requires a factor for classification (not a logical)
iris.fac$virginica <- factor(ifelse(iris1$virginica,"yes","no"))
svm.mod <- svm(virginica~., data=iris.fac, type="C-classification", probability=TRUE)
# plotmo knows how to handle predict.svm probability=TRUE
plotmo(svm.mod, predict.probability=TRUE, nresponse="yes", ylim=c(0,1))

library(randomForest) ## randomForest
iris.fac <- iris1 # randomForest requires a factor for classification (not a logical)
iris.fac$virginica <- factor(ifelse(iris1$virginica,"yes","no"))
rf.mod <- randomForest(virginica~., data=iris.fac, ntree=100)
plotmo(rf.mod, type="prob", nresponse="yes")
8.1 Multinomial models

For multinomial models we must plot the probabilities for each class one at a time. (Plotmo doesn’t allow us to plot probability curves for more than one class on the same plot.) Typically we select the class of interest by using `nresponse` to select the appropriate column in the matrix returned by `predict`, but exactly how that works depends on the `predict` method for the model in question.
9 Partial-dependence plots (the \texttt{pmethod} argument)

By default plotmo fixes the background variables in each plot at their medians (or most common level for factors). In contrast, in partial-dependence plots the effect of the background variables is averaged. We tell plotmo to generate partial-dependence plots by setting plotmo’s \texttt{pmethod} argument to "\texttt{partdep}". Further discussion of partial-dependence plots can be found in Section 4 “Alternatives to plotmo”.

9.1 An example

This section demonstrates partial-dependence plots using artificial data with two variables \(x_1\) and \(x_2\) and a response \(y\). The data is plotted on the left of Figure 5. There are large interactions between the variables.

The code to generate the data is:

\begin{verbatim}
f <- function(x1, x2) {
  ifelse(x2 > .7, x1, # big x2
        ifelse(x2 > .4, 1 - x1, # medium x2
                 .5 * sin(pi * x1))) # small x2
}
n <- 5000; x1 <- runif(n); x2 <- runif(n) # uniformly distributed
data <- data.frame(x1=x1, x2=x2, y=f(x1, x2))
\end{verbatim}

From this data we generate a random forest model (although any model could be used). We also generate a default plotmo plot and a partial-dependence plot:

\begin{verbatim}
library(randomForest)
mod <- randomForest(y~., data=data)
plotmo(mod) # middle figure, default plotmo plot
plotmo(mod, pmethod="partdep") # right figure, partial-dependence plot
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5}
\caption{A function surface showing strong interaction between \(x_1\) and \(x_2\). A classical plotmo plot showing the predicted response varying as \(x_1\) changes, with \(x_2\) fixed at its median 0.5. In a partial-dependence plot the effect of \(x_2\) is averaged.}
\end{figure}
For simplicity we show results only for the $x_1$ variable (though the above code generates plots for the other variable too).

The middle figure is the default plotmo plot for $x_1$. It shows the downward slope in the left figure when $x_2$ is at its median 0.5. In contrast, in the partial-dependence plot in the right figure the effects of downward and upward slopes cancel, leaving just the effect of the hump at low values of $x_2$. (The small kinks at the extremes of the plotted curves are artifacts of the way the random forest handles the borders of the distribution.)

### 9.2 Approximate partial-dependence plots

Calculating partial dependencies can be slow. At each point in the plot we have to make $n$ predictions (where $n$ is the number of cases in the training data), and then average these predictions. For certain models there are techniques to calculate the plots quickly, but plotmo currently doesn’t avail itself of these techniques. To increase speed, plotmo reduces the number of repeated internal calls to `predict` by accumulating data for each call. This may require quite a lot of temporary memory.

Plotmo can also plot approximate partial-dependence plots\(^2\) (pmethod="apartdep"). These are like partial-dependence plots but the background variables are averaged over a subset of cases, rather than all cases in the training data. Approximate plots are much faster for large datasets. The plots are usually similar to standard partial-dependence plots, but guarantees can’t be made.

How do we choose the subset of cases? We must select cases that are representative of the distribution of the data—for example if the data is concentrated in a banana shape in multi-dimensional space, we should select points along the banana. In general estimating which cases are representative is a very difficult problem, so plotmo makes a compromise estimate: the subset is created by selecting 50 rows at equally spaced intervals in the training matrix, after sorting the rows of the matrix on the response values. The idea here is that the density of the response values gives some indication of the density of the data locations. For responses with a small number of discrete values (classification models) this sorting approach doesn’t really work.

Some details: Ties in the response value are randomly broken (so the subset isn’t dependent on the original order of rows in the training matrix). The number 50 can be changed using the `ngrid1` argument. If `ngrid1` is greater than the number of cases then all cases are used, and "apartdep" is identical to "partdep".

### 9.3 Transforming the response for partial dependencies

When doing the averaging for partial-dependencies, plotmo directly averages the predicted responses. The vertical axis of the plots is thus on the same scale as the values returned by `predict`.

\(^2\)Bear in mind that we in fact are already making an approximation for all empirical partial-distribution plots, because we approximate the distribution of the background variables by using the training sample.
This point is raised because for classification models some partial-dependence functions transform the predicted probabilities before taking their average (for example `plotPartial` in the `randomForest` package). The transform is described by Equation 10.48 in Hastie et al. [8].

There seems to be no compelling reason to implement the transformation, especially for two-class (binomial) models—for most models we can use `predict type="link"` to get the same result, and in any case plotted probabilities are usually easier to work with than link functions.

Finally, it should be mentioned that different implementations of partial-dependence plots give slightly different curves. For example, we have to set `ngrid1=100` for plotmo’s partial-dependence curves to exactly match `gbm` package plots.
10 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 6):

```r
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)q.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 [10, 12, 18, 25].

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
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 [16], 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), making use of the plotres function in the plotmo package.

Look at the distribution of residual points to detect non-homoscedasticity. 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 [24].

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.

Figure 6: Prediction intervals with plotmo. These plots were produced by the code on the previous page.
11 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.

On what scale are the vertical axes of the plotmo plots?

Plotmo calls `predict` internally to generate the plots. So the vertical axis will be plotted in whatever units `predict` returns for your model. Plotmo sets the same vertical axis `ylim` for all plots on the page.

For more details, see Section 5.4 “The ylim and clip arguments” and Section 5.2 “The type and nresponse arguments”.

For further discussion, see the following CrossValidated web page: Interpreting partial dependence plots (marginal effects) using plotmo.

How do I get more detail on the axes of degree2 plots?

Get more information on the axes by invoking `persp` with `ticktype="detailed"`. To do this, pass `persp.ticktype="detailed"` to plotmo. Example:

```
plotmo(model, persp.ticktype="detailed")
```

Any plotmo argument prefixed by `persp.` gets passed on internally to `persp`. In the plotmo help page, see the `type2` argument and the description of the dots argument near the bottom of the help page.

How do I display the image plots in color (instead of black and white)?

Pass a vector of colors to plotmo using `image.col`. Example:

```
library(earth)
earth.mod <- earth(Volume ~ ., data = trees)
plotmo(earth.mod, all2=TRUE, type2="image",
       ngrid2=50,  # increase resolution in image plot
       image.col=terrain.colors(30))  # 30 is arbitrary
```
Any plotmo argument prefixed by `image.` gets passed on internally to R's standard `image` function. In the plotmo help page, see the `type2` argument and the description of the dots argument near the bottom of the help page.

The **image display has blue “holes” in it. What gives?**

The light blue 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 do I pass arguments to predict() called internally by plotmo?**

Plotmo passes any argument prefixed with `predict.` directly to `predict`, after removing the prefix.

For example `plotmo(glmnet.mod, predict.s = .8)` will pass `s = .8` to plotmo's internal calls to `predict.glmnet`.

For more information, please see the description of the “...” argument in the help page of `plotmo`.

**How to cite plotmo**


```r
@Manual{plotmopackage, 
title = {plotmo: Plot a Model's Residuals, Response, and Partial Dependence Plots},
author = {Stephen Milborrow},
year = {2018},
note = {R package},
url = {http://CRAN.R-project.org/package=plotmo } 
}
```
12 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 13).

  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

  ```r
  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).

  Certain types of model built with NAs in the data will cause the above error messages.

- **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 13).
13 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\(^3\) 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 [15]). Specific method functions have been written to handle some other situations. For certain models this isn’t possible—for example xgboost models save an incorrect call and use a custom matrix class from which the data can’t be retrieved using R functions.

13.1 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.

  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.

\(^3\)There are actually a few more nuances. For example, it also tries the `model` field saved with some `lm` models.
○ plotmo.singles Figure out which variables should appear in degree 1 plots.
○ plotmo.pairs Figure out which variables should appear in degree 2 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.

13.2 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, typically in the x and y fields of the model. Some models allow us to save x and y with a keepxy or similar argument (plotmo will use those fields if available).
References


