partial: An R Package for Creating Partial Dependence Plots

by Author One, Author Two and Author Three

Abstract An abstract of less than 150 words.

Outline

- Other packages of interest:

  plotmo allows for up to two variables and fixes other variables at there median values (or the first level for factors). Less accurate than partial dependence, but flexible and fast! For additive models (i.e., no interactions), these plots are identical in shape to partial dependence plots.

  ICEbox can also calculate derivatives, but only allows for one variable at a time (i.e., no multivariate displays) – however, color can be used to diplay information about an additional predictor.

  car contains many functions for constructing partial-residual and marginal-model plots. The **effects** is also of interest. However, these packages are orientated towards parametric models (e.g., linear and generalized linear models), whereas **plotmo**, **ICEbox**, and **partial** are mainly for nonparametric and black box models (though, they can be used for simple parametric models as well).

Introduction

Predictor importance is an important task in any supervised learning problem. However, ranking variables is only part of the story and once a subset of "important" variables is identified it is often necessary to assess the relationship between them and the response. This can be done in many ways, but in machine learning it is often accomplished by constructing *partial dependence plots* (PDPs). PDPs help visualize the relationship between a subset of the predictors and the response while accounting for the average effect of the other independent variables. They are particularly effective with black box models like support vector machines or random forest.

Many techniques exist for visualizing the relationship between a predictor and the response based on a fitted model. For example, the output from complex parametric models are typically augmented with partial residual plots (see J. FOX for details). Most existing techniques arose in applied regression and typically only apply to parametric models. *Partial dependence functions* (Friedman, 2000), on the other hand, offer a low-dimensional graphical rendering of the prediction function $\widehat{f}(\boldsymbol{x})$ for any type of fitted models and are especially useful in visualizing the relationships discovered by complex machine learning algorithms such as a *random forest*.

Targeted implementations of Friedman's partial dependence plots are available in packages **randomForest** (Liaw and Wiener, 2002) and **gbm**, among others. While the **randomForest** implementation will only allow for a single predictor, the **gbm** implementation can deal with any subset of the predictor space. Partial dependence functions are not restricted to tree-based models; they can be applied to any supervised learning algorithm (e.g., neural networks). However, to our knowledge, there is no general package for obtaining partial dependence plots for other types of model fits (e.g., a *conditional inference forest* as implemented by the `cforest` function in the **party** package). The **partial** package tries to close this gap by offering a general framework that can be applied to several types of fitted models – though, package **plotmo** has come a long way in this direction..

Partial dependence plots are very useful but can be misleading in the presence of substantial interactions (Goldstein et al., 2015). To overcome this issue Goldstein et al. (2015) developed the concept of *individual conditional expectation* (ICE) plots – available in the **ICEbox** package.

There are two main functions available in the **partial** package (PKG REF):

- `partial`
- `plotPartial`

The `partial` function evaluates the partial dependence (1) from a fitted model over a grid of predctor vaulues. By default, `partial` returns a data frame with an ddditional class: `"partial"`. Otherwise a

`"lattice"` object (i.e., a plot) is returned. For more advanced plotting, the `plotPartial` functions will take a `"partial"` object and display a customizable lattice plot.

There are two ways to install the **partial** package. Obtaining the latest stable release from CRAN:

```
install.packages("partial")
```

or installing the development version from GitHub:

```
devtools::install_github("bgreenwell/partial")
```

Partial dependence plots

Let $\boldsymbol{x}$ represent the set of predictors in a model whose prediction function is $\widehat{f}(\boldsymbol{x})$ (these are the values returned by `predict` in R). If we partition $\boldsymbol{x}$ into an interest set, $\boldsymbol{z}_s$, and its compliment, $\boldsymbol{z}_c = \boldsymbol{x} \setminus \boldsymbol{z}_s$, then the partial dependence of the response on $\boldsymbol{z}_s$ is defined as

$$\bar{f}_s(\boldsymbol{z}_s) = \frac{1}{n} \sum_{i=1}^{n} \widehat{f}(\boldsymbol{z}_s, \boldsymbol{z}_{i,c}), \tag{1}$$

where $\boldsymbol{z}_{i,c}$ $(i = 1, 2, \ldots, n)$ are the values of $\boldsymbol{z}_c$ that occur in the training sample. This can be quite computationally intensive since Equation (1) involves a pass over the training records for each set of combinations of $\boldsymbol{z}_s$. Fortunately, this problem is *embarrassingly parallel*.

The basic algorithm for computing a partial dependence function is as follows:

1. Fit a model
2. Suppose x_i is the predictor set of interest. For each unique value of x_i, do the following:
   (a) Make a copy of the original training data and replace x_i with the unique value – everything else remains th same.
   (b) Compute the predicted values using the modified copy of the training data.
   (c) Average the predictions over each of the observations.
3. Plot the average prediction against x_i.
4. Repeat for each predictor of interest.

Partial dependence plots in R

For illustration, we will use the (corrected) Boston housing data which are available from the **mlbench** package. We begin by loading the data and omitting unimportant columns.

```
data(BostonHousing2, package = "mlbench")  # load the data
boston <- BostonHousing2[, -c(1, 2, 5)]
```

Next, we fit a traditional random forest (as implemented in the **randomForest** package) to the entire data set.

```
# Load required packages
library(partial)
library(randomForest)

# Fit a random forest
set.seed(101)  # for reproducibility
fit.rf <- randomForest(cmedv ~ ., data = boston, importance = TRUE)
print(fit.rf)  # check model results
varImpPlot(fit.rf, main = "")  # variable importance plot

# Call:
#   randomForest(formula = cmedv ~ ., data = boston, importance = TRUE)
# Type of random forest: regression
# Number of trees: 500
# No. of variables tried at each split: 5
#
# Mean of squared residuals: 9.114248
# % Var explained: 89.17
```

The model fit is reasonable, with an *out-of-bag* R^2 of 0.89. The variable importance scores are displayed in Figure 1. Both plots indicate that the percentage of lower status of the population (`lstat`) and the average number of rooms per dwelling (`rm`) are highly associated with the median value of owner-occupied homes. The question then arises, "What is the nature of these associations?" To help answer this, we can look at the partial dependence of `cmedv` on `lstat` and `rm`, individually and together.

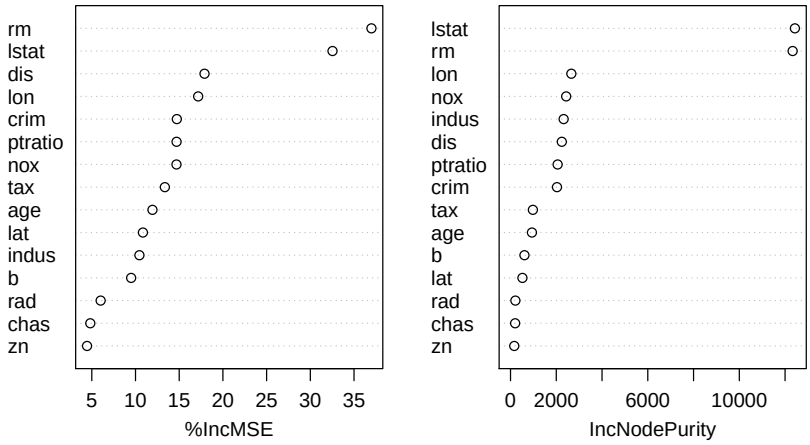

Figure 1: Dotchart of variable importance scores for the Boston housing data based on a random forest with 500 trees.

Univariate partial dependence

As previously mentioned, the `randomForest` package has its own `partialPlot` function for displaying partial dependence of the response on a single predictor. For example, the following snippet of code plots the partial dependence of `cmedv` on `lstat`:

```
partialPlot(fit.rf, pred.data = boston, x.var = "lstat")
```

The same plot (implemented in **lattice**) can be achieved using the `partial` function and setting `plot = TRUE` (the default is `FALSE`):

```
partial(fit.rf, pred.var = "lstat", plot = TRUE)
```

By default, `partial` returns a data frame:

```
head(partial(fit.rf, pred.var = "lstat"))
```

```
#    lstat        y
# 1 1.7300 31.34696
# 2 2.4548 31.34564
# 3 3.1796 31.28286
# 4 3.9044 30.82356
# 5 4.6292 29.12636
# 6 5.3540 26.89093
```

The benefit of using `partial` is threefold: (1) it is a generic function that can be used for other types of model fits (e.g., a *conditional inference forest* as implemented in the **party** package), (2) it will allow for any number of predictors to be used, and (3) it can utilize any of the parallel backends supported by the **foreach** package (more on this later). For example, to assess the joint effect of `lstat` and `rm` on `cmedv` we could use

```
partial(fit.rf, pred.var = c("lstat", "rm"), plot = TRUE)
```

Finer plotting control can be achieved using the accompanying `plotPartial` function. `plotPartial` takes a `"partial"` object and provides a convenient **lattice** display that can be easily tailored. For example, the code chunk

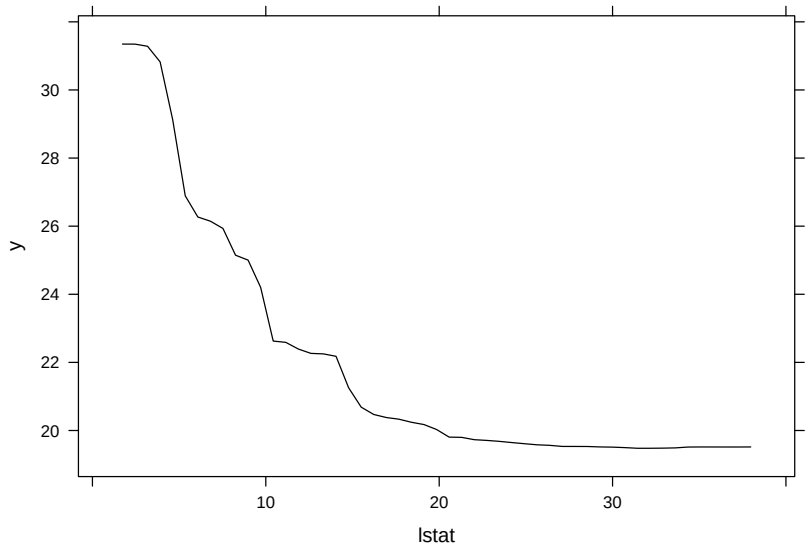

Figure 2: Partial dependence of `cmedv` on `lstat`.

```
pd.lstat.rm <- partial(fit.rf, pred.var = c("lstat", "rm"))
p1 <- plotPartial(pd.lstat.rm)
p2 <- plotPartial(pd.lstat.rm, contour = FALSE, zlab = "cmedv", drape = TRUE,
                  colorkey = TRUE, screen = list(z = -20, x = -60))
print(p1, position = c(0, 0, 0.5, 1), more = TRUE)
print(p2, position = c(0.5, 0, 1, 1))
```

produces the three-dimensional displays in Figure 3.

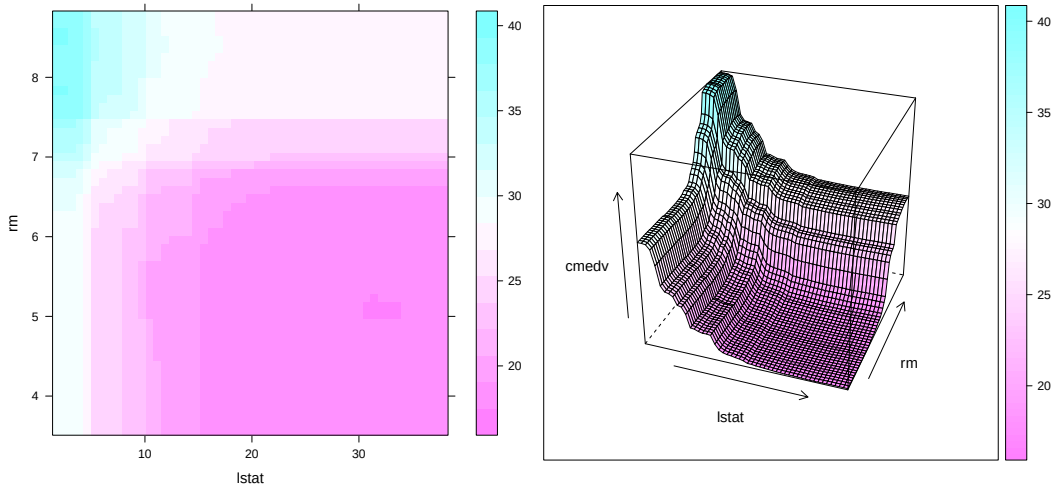

Figure 3: Partial dependence of `cmedv` on `lstat` and `rm`.

Unfortunately, the `cforest` implementation contains no such function. In this case we can use `partial`:

```
crf.lstat <- partial_1d(fit.crf, x.name = "lstat", n = 51)
crf.rm <- partial_1d(fit.crf, x.name = "rm", n = 51)
par(mfrow = c(1, 2))
plot(crf.lstat, type = "l")
plot(crf.rm, type = "l")
```

In some cases it is difficult for `partial` to tell whether the response is continuous (regression) or categorical (classification). For these cases, we added the option `super.type` which accepts a character

string specifying either "regression" or "classification" – this distinction is important because for classification problems, PDPs use something similar to a *logit* scale for response estimation.

By default partial returns a data frame. If called with plot = TRUE, however, a lattice plot is displayed instead

```
partial_1d(fit.crf, x.name = "lstat", n = 51, plot = TRUE)
```

Multivariate partial dependence

Bivariate (and higher order) partial dependence can also be handled

```
# Bivariate partial dependence
rf.rm.lstat <- partial_2d(fit.rf, x1.name = "rm", x2.name = "lstat",
                          n1 = 51, n2 = 51, .progress = "text")
wireframe(y ~ rm * lstat, data = rf.rm.lstat, zlab = "cmedv", drape = TRUE,
          colorkey = TRUE, screen = list(z = 110, x = -60))
```

The variables are plotted in the order they are given; that is, the first variable is plotted along the x-axis, the second on the y-axis, and the rest becom panel variables. Hence, it is best to use numeric variables for the first two, and factors for the remaining. The following snippet of code plots the partial dependence of cmedv on rm, lstat, and chas, but since chas is a binary variable (whether or not the census tract bounds river), a separate panel is displayed for each category.

```
# Trivariate partial dependence
rf.rm.lstat.ptratio <- partial_3d(fit.rf, x1.name = "rm", x2.name = "lstat",
                                  x3.name = "chas", n1 = 51, n2 = 51, n2 = 51,
                                  .progress = "text")
wireframe(y ~ rm * lstat, data = rf.rm.lstat, zlab = "cmedv", drape = TRUE,
          colorkey = TRUE, screen = list(z = 110, x = -60))
```

The Boston Housing Data

In **plotmo** the other variables are held fixed at their median values. This results in a less accurate, but much faster display. To illustrate we fit a multivariate adaptive regression spline (MARS) model to the Boston housing data. MARS models can be fit using the mars function from the **mda** package, or using the earth function from the **earth** package. Here we use the **earth** implementation which conveniently loads the **plotmo** package as well.

```
# Fit a MARS model
fit.earth <- earth(cmedv ~ ., data = boston, degree = 3)

# Partial dependence function
earth.rm <- partial_1d(fit.earth, x.name = "rm")

# Compare partial and plotmo
pdf("plotmo_vs_partial.pdf", width = 7, height = 5)
plotmo(fit.earth, degree1 = "rm", degree2 = FALSE, do.par = FALSE,
       ylim = c(16.79417, 41.87501), main = "", xlab = "rm", ylab = "cmedv")
lines(earth.rm, lty = 2)
legend("topleft", legend = c("plotmo", "partial"),
       lty = 1:2, inset = 0.01)
dev.off()

partial(fit.earth, pred.var = c("rm", "ptratio"), plot = TRUE)
```

partial returns a"data.frame" object with an additioanl class called "partial"; the additional class is recognized by plotPartial for convenient default plotting. To create more elaborate plots, just work with the data frame directly. Also, the convenient plot option in partial returns a basic plot, but is not as customizable as uning plotPartial. The following snipet of code demonstrates:

```
# Basic partial dependence plot
partial(fit.rf, "rm", plot = TRUE)

# Save the partial dependence data for customized plotting
pd.rm <- partial(fit.rf, "rm")
```

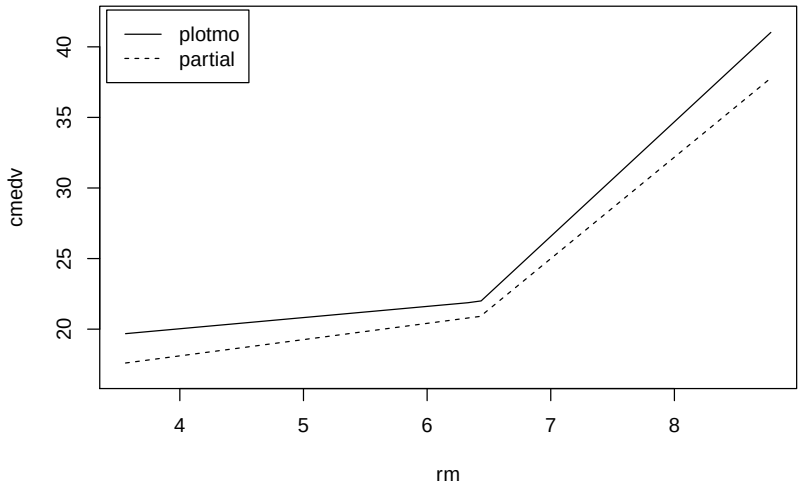

Figure 4: The logo of R.

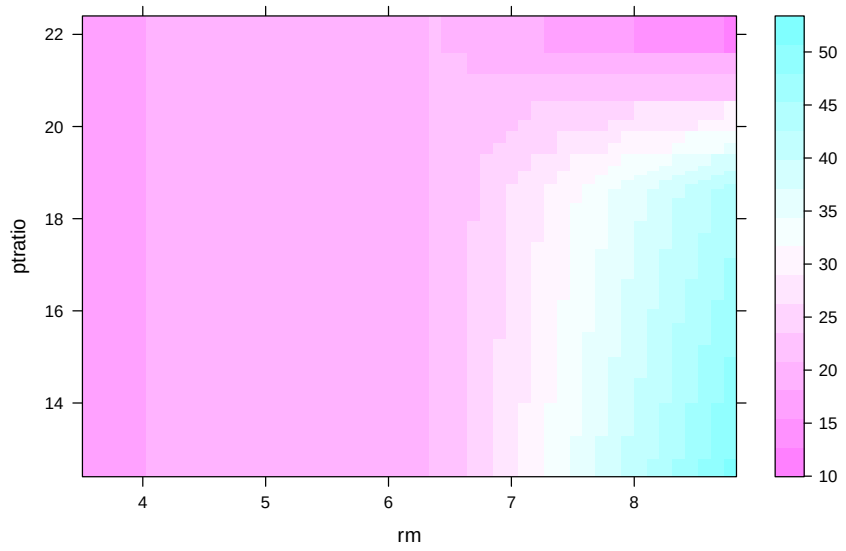

Figure 5: Partial dependence of `cmedv` on `rm` and `ptratio`.

```
partialPlot(pd.rm, xlab = "Average number of rooms", ylab = "Median home value")
plot(pd.rm, type = "l")  # base R graphics

# Try using ggplot2
library(ggplot2)
ggplot(pd.rm, aes(rm, y)) +
  geom_point() +
  geom_line() +
  theme_light()
```

Avoiding extrapolation

There are few ways to mitigate the risk of extrapolation in PDPs: rug displays and convex hulls. Rug displays are one-dimensional plots added to the axes. The `plotPartial` has a `rug` option that will display the deciles of the distribution (as well as the minimum and maximum values) for the predictors on the horizontal or vertical axes. The following snippet of code produces the plot on the left side of Figure 6. Notice that in both examples we had to provide the original training data to `plotPartial` via the `training.data` option.

```
plotPartial(pf.lstat, rug = TRUE, training.data = boston)
```

In higher dimensions, plotting the convex hull is more informative; it outlines the region of the predictor space that the model was trained on. When `convex.hull = TRUE`, the convex hull of the first two dimensions of z_s (i.e., the variables used for the horizontal and vertical axes) is added to the plot. Over interpreting the partial dependence plot outside of this region can be dangerous. The right side of Figure 6 was produced using:

```
plotPartial(pd.lstat.rm, convex.hull = TRUE, training.data = boston)
```

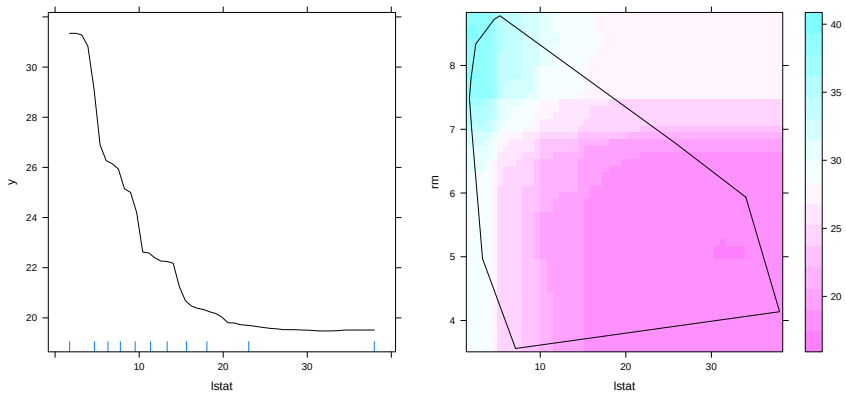

Figure 6: Examples of partial dependence plots with the addition of a rug display (left) and a convex hull (right).

Addressing computational concerns

Additional options are available to ease the computational intensity for large problems. For example, there was no need to compute partial dependence of median home value using each uniqe value of `rm` in the training data (which would require 446 passes over the data!). We could get very reasonable results using a reduced number of points. Current options are to use a grid of equally spaced values in the range of the variable of interest; the number of points is controlled by the option `n.pts`. Alternatively, a specific set of values can be supplied (e.g., quantiles of interest) through the `pred.data` argument. For example, the following snippet of code re-computes the partial dependence of median home value on `rm` using both options; the results are displayed in Figure 7.

```
partial(fit.rf, "rm", grid.resolution = 30, plot = TRUE)
partial(fit.rf, "rm", pred.grid = 3:9, plot = TRUE)
```

The `partial` function relies on the **plyr** package (Wickham, 2011), rather than for loops. This makes it poosible to request progress bars (e.g., `.progress = "text"`) or run `partial` in parallel. In fact, `partial` can use any of the parallel backends supported by the **foreach** package. To use this

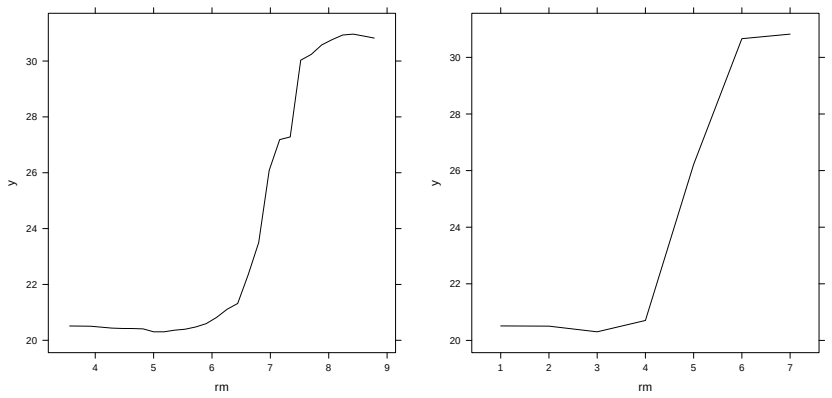

Figure 7: Examples of partial dependence plots using a reduced set of predictor values.

functionality, we must load and register a supported parallel backend (e.g., **doParallel** or **doMC**). The following snippet of code obtains the partial dependence of median home value on `rm`, `lstat`, and `ptratio` in parallel. The result is displayed in Figure 8.

```
library(doParallel)
cl <- makeCluster(2)  # using snow-like functionality
registerDoParallel(cl)  # use cores = 2 for multicore functionality
pd3 <- partial(fit.rf, pred.var = c("lstat", "rm", "ptratio"),
               grid.resolution = 10, .parallel = TRUE)
stopCluster(cl)  # required when using snow-like functionality
plotPartial(pd3, number = 4, overlap = 0.1)
```

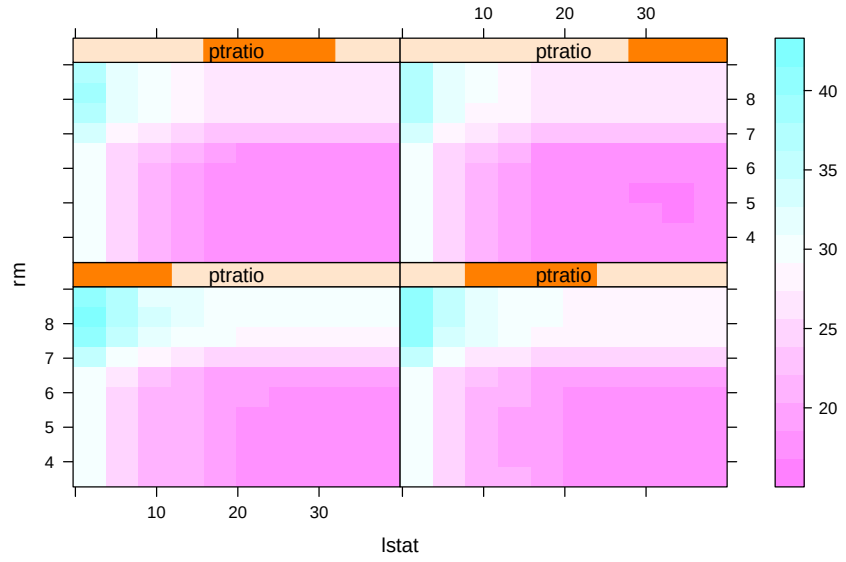

Figure 8: Examples of partial dependence plots using a reduced set of predictor values.

Classification problems

For classification problems, partial dependence functions are on a scale simliar to the logit; in particular, an average logit. Suppose the response is categorical with K levels, then for each class we have

$$f_k(x) = \log\left[p_k(x)\right] - \frac{1}{K}\sum_{i=1}^{K}\log\left[p_k(x)\right], \quad k = 1, 2, \ldots, K, \tag{2}$$

where $p_k(x)$ is the predicted probability for the k-th class. Plotting $f_k(x)$ helps us understand how the log-odds for the k-th cass depends on different subsets of the predictor variables.

To illustrate, consider the Edgar Anderson's iris data from the **datasets** package. We fit a support vector machine with a radial basis function kernel to the data (the parameters were determined using 5-fold cross-validation).

```
library(kernlab)
iris.svm <- ksvm(Species ~ ., data = iris, kernel = "rbfdot", C = 0.25,
                 kpar = list(sigma = 0.75), prob.model = TRUE)
```

Next, we plot the partial dependence of `Species` on both `Petal.Width` and `Petal.Length` for each of the three classes. The result is displayed in Figure 9.

```
pd <- NULL
for (i in 1:3) {
  tmp <- partial(iris.svm, pred.var = c("Petal.Width", "Petal.Length"),
                 which.class = i, training.data = iris)
  pd <- rbind(pd, cbind(tmp, Species = levels(iris$Species)[i]))
}
levelplot(y ~ Petal.Width * Petal.Length | Species, data = pd)
```

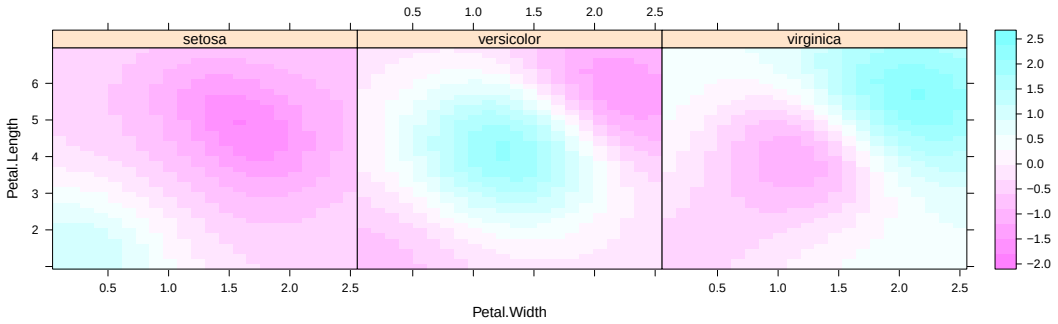

Figure 9: Partial dependence of species on petal width and petal length for the iris data.

Summary

In the future, this package will be expanded to appropriately handle other types of responses as well (e.g., countd data).

Bibliography

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29: 1189–1232, 2000. [p1]

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. [p1]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL http://CRAN.R-project.org/doc/Rnews/. [p1]

H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL http://www.jstatsoft.org/v40/i01/. [p7]

Brandon Greenwell
Infoscitex, a dcs company
4027 Colonel Glenn Highway
Suite 210
Dayton, OH 45431-1672
United States of America
bgreenwell@infoscitex.com