

# partial: An R Package for Creating Partial Dependence Plots

by *Brandon M. Greenwell*

**Abstract** Complex nonparametric models (e.g., neural networks, random forests, and support vector machines) are often used in predictive analytics, especially when dealing with large observational databases that don't adhere to the strict assumptions imposed by traditional statistical techniques (e.g. multiple linear regression). Unfortunately, it can be challenging for the uninitiated to understand the results of such models and explain them to management. Partial dependence plots (PDPs) offer a simple solution. Partial dependence plots are low-dimensional graphical renderings of the prediction function  $\hat{f}(x)$  so that the relationship between the outcome and predictors of interest can be easily understood. These plots are especially useful in explaining the output from complex black box models like support vector machines or neural networks or constructing simpler parsimonious models on newly obtained data.

## 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" features is identified it is often necessary to assess the relationship between them (or subset thereof) and the response. This can be done in many ways, but in machine learning it is often accomplished by constructing *partial dependence plots* (PDPs); see [Friedman \(2000\)](#) for details. PDPs help visualize the relationship between a subset (typically 1-3) of the predictors and the response while accounting for the average effect of the other independent variables in the model. They are particularly effective with black box models like random forest and support vector machines.

Let  $x = \{x_1, x_2, \dots, x_n\}$  represent the predictors in a model whose prediction function is  $\hat{f}(x)$ . If we partition  $x$  into an interest set,  $z_s$ , and its complement,  $z_c = x \setminus z_s$ , then the partial dependence of the response on  $z_s$  is defined as

$$\bar{f}_s(z_s) = \frac{1}{n} \sum_{i=1}^n \hat{f}(z_s, z_{i,c}), \quad (1)$$

where  $z_{i,c}$  ( $i = 1, 2, \dots, n$ ) are the values of  $z_c$  that occur in the training sample; that is, we average out the effects of all the other predictors in the model. Let  $x_1$  be the predictor variable of interest with unique values  $\{x_{11}, x_{12}, \dots, x_{1k}\}$ . The partial dependence of the response on  $x_1$  can be constructed as follows:

1. For  $i \in \{1, 2, \dots, k\}$ :
  - (a) Create a copy of the training data and replace the original values of  $x_1$  with the constant  $x_{1i}$ .
  - (b) Compute the vector of predicted values from the modified copy of the training data.
  - (c) Compute the average prediction to obtain  $\bar{f}_1(x_{1i})$ .
2. Plot the pairs  $[x_{1i}, \bar{f}_1(x_{1i})]$  for  $i = 1, 2, \dots, k$ .

This can be quite computationally intensive since the algorithm involves a pass over the training records for each set of combinations of  $z_s$ . Fortunately, this parallelized quite easily. This algorithm can also be easily extended to larger subsets of 2 or more features as well.

Limited implementations of Friedman's PDPs are available in packages [randomForest](#) ([Liaw and Wiener, 2002](#)) and [gbm](#), among others; these are limited in the sense that they only apply to the models fit using these packages. While the [randomForest](#) implementation will only allow for a single predictor, the [gbm](#) implementation can deal with any subset of the predictor space; however, these implementations only apply to models fit using the respective package. 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 constructing PDPs in R (e.g., PDPs for a *conditional random forest* as implemented by the `cforest` function in the [party](#) and [partykit](#) packages; see [Torsten Hothorn and Zeileis \(2015\)](#) and [Hothorn and Zeileis \(2016\)](#), respectively). The [partial](#) ([Greenwell, 2016](#)) package tries to close this gap by offering a general framework for PDPs that can be applied to several types of fitted models.

The [plotmo](#) package ([Milborrow, 2015](#)) has come a long way in this regard. According to the author, [plotmo](#) constructs "a poor man's partial dependence plot." In particular, it plots a model's

response when varying one or two predictors while holding the other predictors constant (continuous features are fixed at their median value, while factors are held at their first level). These plots allow for up to two variables at a time and are less accurate than PDPs, but are faster to construct! For additive models (i.e., no interactions), these plots are identical in shape to partial dependence plots.

PDPs are very useful but can be misleading in the presence of substantial interactions (Goldstein et al., 2015). To overcome this issue Goldstein, Kapelner, Bleich, and Pitkin developed the concept of *individual conditional expectation* (ICE) plots – available in the **ICEbox** package. **ICEbox** only allows for one variable at a time (i.e., no multivariate displays), though, color can be used effectively to display information about an additional predictor. The ability to construct derivative plots is also available.

Many other techniques exist for visualizing relationships between the predictors and the response based on a fitted model. For example, the **car** package (Fox and Weisberg, 2011) contains many functions for constructing partial-residual and marginal-model plots. The **effects** package (Fox, 2003) is also of interest. However, these methods are orientated towards simpler 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).

## Constructing PDPs in R

As described in the introduction, the **partial** package is useful for constructing PDPs for many types of fitted models in R. These are especially useful for visualizing the relationships discovered by complex machine learning algorithms such as a random forest. The latest stable release is available from CRAN:

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

Currently, only two functions are exported by **partial**:

- `partial`
- `plotPartial`

The `partial` function evaluates the partial dependence (1) from a fitted model over a grid of predictor values. If `plot = FALSE` (the default), `partial` returns a data frame with an additional class: "partial". The columns are labeled in the same order as the features supplied to `pred.var`, and the last column is always labeled `y` and contains the partial dependence function values. If `plot = TRUE`, then `partial` calls `plotPartial` and returns a "lattice" object representing the PDP. For more advanced plotting, the `plotPartial` functions will take a "partial" object and display a more customizable lattice plot. Currently supported models are described in Table 1. The `partial.default` method should be able to

Model	Package	Class
Bagged decision trees	<b>ipred</b> (Peters and Hothorn, 2015) <b>adabag</b> (Alfaro et al., 2013)	"classbagg", "regbagg" "bagging"
Boosted decision trees	<b>gbm</b> <b>adabag</b> (Alfaro et al., 2013)	"gbm" "boosting"
Conditional random forest	<b>party</b>	"RandomForest"
Decision trees	<b>partykit</b> <b>rpart</b> (Therneau et al., 2015) <b>party</b> <b>partykit</b>	"cforest" "rpart" "BinaryTree" "constparty"/"party"
Linear model	<b>stats</b>	"lm"
Multivariate adaptive regression splines (MARS)	<b>earth</b> (Milborrow, 2016)	"earth"
Random forest	<b>randomForest</b>	"randomForest"
Support vector machines	<b>e1071</b> (Meyer et al., 2015) <b>kernlab</b> (Karatzoglou et al., 2004)	"svm" "ksvm"

**Table 1:** Models specifically supported by the **package**.

handle most other types of models not listed in Table 1; for example, neural networks from the **nnet** package (Venables and Ripley, 2002) or projection pursuit regression (Friedman and Stuetzle, 1981) using the `ppr` function in the **stats** package.

For illustration, we will use the (corrected) Boston housing data which are available from the **mlbench** package (Dimitriadou, 2010). These data contain the median value of owner-occupied homes in 506 U.S. census tracts in the Boston area, along with 13 independent variables such as the per capita crime rate by town. We begin by loading the data and omitting unimportant columns.

```
data(BostonHousing2, package = "mlbench") # mlbench must be installed!
boston <- BostonHousing2[, -c(1, 2, 5)]
```

Next, we fit a traditional random forest to the entire data set using the following code snippet:

```
library(randomForest)
set.seed(101) # for reproducibility
boston.rf <- randomForest(cmedv ~ ., data = boston, importance = TRUE)
varImpPlot(boston.rf) # variable importance plot
```

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 (cmedv). 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, both individually and together.



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

### Single predictor PDPs

As previously mentioned, the randomForest package has its own partialPlot function for visualizing the 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(boston.rf, pred.data = boston, x.var = "lstat")
```

The same plot can be achieved using the partial function and setting plot = TRUE (see the left side of Figure 2) as in the following example:

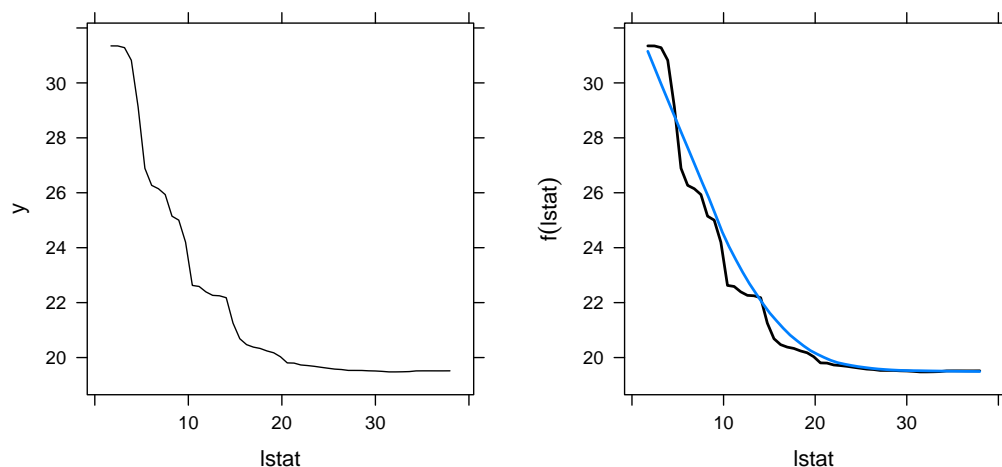
```
library(partial)
partial(boston.rf, pred.var = "lstat", plot = TRUE)
```

The only difference is that **partia** uses **lattice** (Sarkar, 2008) to produce all of its displays.

For a more customizable plot, call **partial** with `plot = FALSE` and use the `plotPartial` function. **Note:** the *pipe* operator provided by the **magrittr** package (Bache and Wickham, 2014) is imported for writing more convenient code, as illustrated in snippet of code below.

```
partial(boston.rf, pred.var = "lstat") %>%
  plotPartial(smooth = TRUE, lwd = 2, ylab = expression(f(lstat)))
```

The result is displayed in the right side of Figure 2 below.



**Figure 2:** Partial dependence of `cmedv` on `lstat`. *Left:* Default plot. *Right:* Customized plot obtained using the `plotPartial` function.

### Multi-predictor PDPs

The benefit of using **partial** is threefold: (1) it is a generic function that can be used for various types of model fits (not just random forests), (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 (Analytics and Weston, 2015c); we discuss parallel execution in a later section. For example, the following code chunk could be used to assess the joint effect of `lstat` and `rm` on `cmedv` based on a conditional random forest using various options in `plotPartial`. The results are displayed in Figure 3.

```
library(party)
boston.crf <- cforest(cmedv ~ ., data = boston)
pd.lstat.rm <- partial(boston.crf, pred.var = c("lstat", "rm"))
plotPartial(pd.lstat.rm)
plotPartial(pd.lstat.rm, col.regions = colorRampPalette(c("white", "blue")))
plotPartial(pd.lstat.rm, contour = FALSE, zlab = "cmedv", drape = TRUE,
  colorkey = TRUE, screen = list(z = -20, x = -60))
```

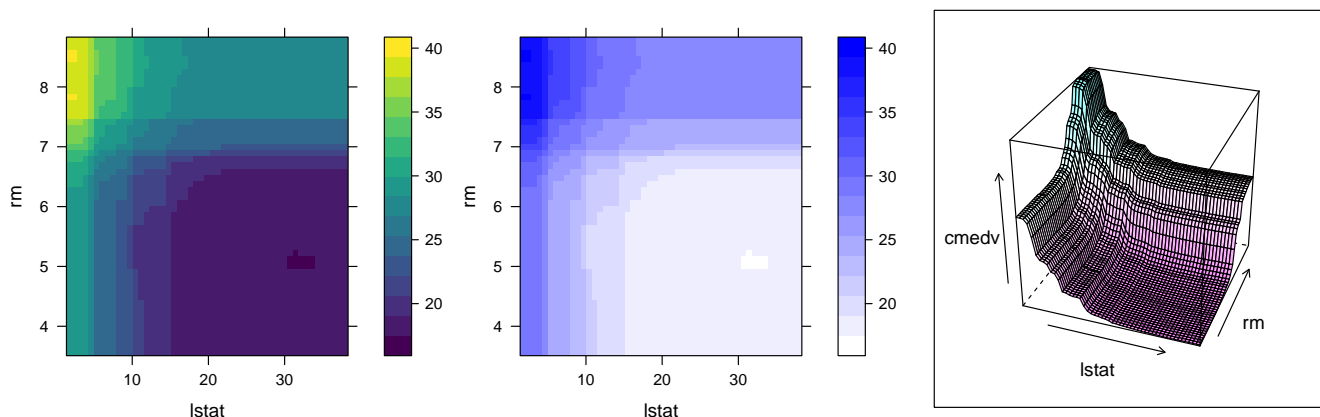
Note that the default color map for contour plots is the Matplotlib "viridis" color map provided through the **viridis** package (Garnier, 2016).

### 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. `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 4. Notice that in both examples we had to provide the original training data to `plotPartial` via the `train` option.

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

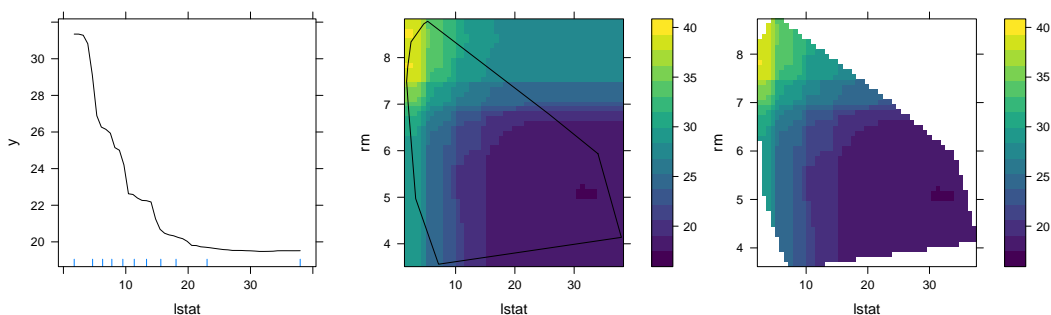
In higher dimensions, plotting the convex hull is more informative; it outlines the region of the predictor space that the model was trained on. When `chull = TRUE`, the convex hull of the first two



**Figure 3:** Partial dependence of `cmedv` on `lstat` and `rm`. *Left:* Default plot. *Middle:* Using a different color palette. *Right:* Using a 3-D surface.

dimensions of  $z_s$  (i.e., the first two variables supplied to `pred.var`) is added to the plot; alternatively, you can set `chull = TRUE` in the call to `partial`, in which case only the region within the convex hull of the first two variables is plotted. Over interpreting the partial dependence plot outside of this region can be dangerous. The right two displays in Figure 4 were produced, respectively, using:

```
plotPartial(pd.lstat.rm, chull = TRUE, train = boston)
partial(boston.rf, pred.var = c("lstat", "rm"), chull = TRUE, plot = TRUE)
```



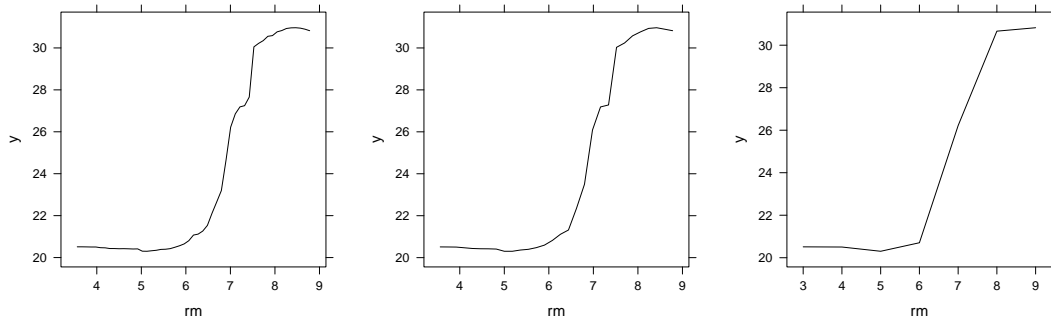
**Figure 4:** Examples of partial dependence plots with the addition of a rug display (left) and a convex hull (right).

### Addressing computational concerns

Constructing PDPs can be quite computationally expensive. Additional options are available to ease the computational intensity for large problems. For example, there is no need to compute partial dependence of median home value using each unique 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 can be controlled using the `n.pts`. Alternatively, a specific set of values (e.g., quantiles of interest) can be supplied through the `pred.grid` argument. To illustrate, the following snippet of code computes the partial dependence of median home value on `rm` using each option; the results are displayed in Figure 5.

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

The partial function relies on the `plyr` package (Wickham, 2011), rather than for loops. This makes it possible 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 functionality, we must load and register a supported parallel backend (e.g., `doMC` (Analytics and



**Figure 5:** Partial dependence of cmedv on rm. *Left:* Default plot. *Middle:* Using a reduced grid size. *Right:* Using a user-specified grid.

Weston, 2015a) or **doParallel** (Analytics and Weston, 2015b)). 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 6.

Setting up a parallel backend is rather straightforward. To illustrate, the following snippet of sets up the partial function to run in parallel on Unix-like systems<sup>1</sup> using the **doParallel**.

```
library(doParallel) # load parallel backend
registerDoParallel(cores = 4) # use 4 cores
```

Now, to run partial in parallel, all we have to do is invoke the `.parallel = TRUE` option and the rest is taken care of by the internal call to **plyr** the parallel backend we loaded:

```
pd <- partial(boston.rf, pred.var = c("lstat", "rm", "ptratio"),
              grid.resolution = 20, .parallel = TRUE)
plotPartial(pd, number = 4, overlap = 0.1)
```

It is important to note that when using more than two variables, `plotPartial` produces a trellis display. The first two variables given to `pred.var` are used for the horizontal and vertical axes; additional variables define the panels. If the panel variables are continuous, then shingles<sup>2</sup> are produced first using the equal count algorithm. Hence, it is probably better to use categorical variables to define the panels in higher dimensional displays when possible.

## Classification problems

For classification problems, partial dependence functions are on a scale similar to the logit; see, for example, Hastie et al. (2009, pp. 369–370). Suppose the response is categorical with  $K$  levels, then for each class we compute

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

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 class depends on different subsets of the predictor variables.

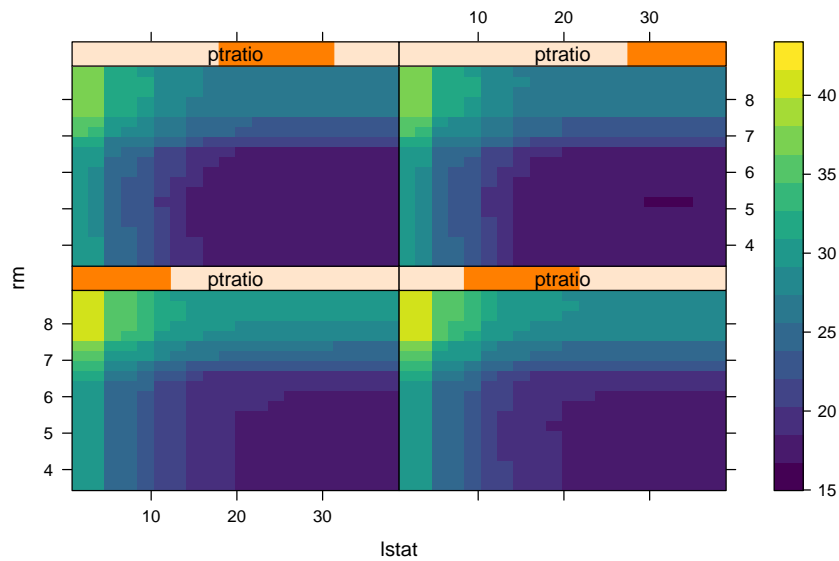
To illustrate, consider Edgar Anderson’s iris data from the **datasets** package. The iris data frame contains the sepal length, sepal width, petal length, and petal width (in centimeters) for 50 flowers from each of three species of iris: setosa, versicolor, and virginica. We fit a support vector machine with a radial basis function kernel to the data using the `svm` function in the **e1071** (the model parameters were determined using 5-fold cross-validation).

```
library(e1071)
iris.svm <- svm(Species ~ ., data = iris, kernel = "radial", gamma = 0.75,
               cost = 0.25, probability = TRUE)
```

The partial function has to be able to extract the predicted probabilities for each class, so it is necessary to invoke the `prob.model = TRUE` option in the call to `svm`.

<sup>1</sup>This example will not run on Windows.

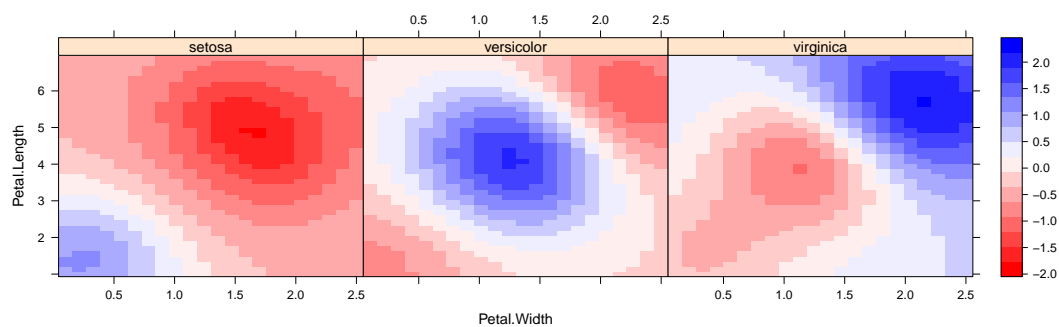
<sup>2</sup>A shingle is a special Trellis data structure that consists of a numeric vector along with intervals that define the “levels” of the shingle. The intervals may be allowed to overlap.



**Figure 6:** Partial dependence of cmedv on lstat, rm, and ptratio. Since ptratio is continuous, it is first converted to a shingle; in this case, four groups with 10% overlap.

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 7.

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



**Figure 7:** Partial dependence of species on petal width and petal length for the iris data.

## Summary

In this paper, we showed how to construct PDPs for various types of black box models in R using the **partial** package. We discussed similar approaches in other packages. Ways to avoid extrapolation and high execution times were demonstrated via examples. There is a bright future for **partial** in terms of growth. For example, we would like to include the ability to construct PDPs for various types of survival models (e.g. conditional random forests with censored response).



## Acknowledgments

TBD.

## Bibliography

- E. Alfaro, M. Gámez, and N. García. *adabag: An R package for classification with boosting and bagging*. *Journal of Statistical Software*, 54(2):1–35, 2013. URL <http://www.jstatsoft.org/v54/i02/>. [p2]
- R. Analytics and S. Weston. *doMC: Foreach Parallel Adaptor for 'parallel'*, 2015a. URL <https://CRAN.R-project.org/package=doMC>. R package version 1.3.4. [p5]
- R. Analytics and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2015b. URL <https://CRAN.R-project.org/package=doParallel>. R package version 1.0.10. [p6]
- R. Analytics and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2015c. URL <https://CRAN.R-project.org/package=foreach>. R package version 1.4.3. [p4]
- S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL <https://CRAN.R-project.org/package=magrittr>. R package version 1.5. [p4]
- F. L. . E. Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2010. URL <https://CRAN.R-project.org/package=mlbench>. R package version 2.1-1. [p3]
- J. Fox. Effect displays in R for generalised linear models. *Journal of Statistical Software*, 8(15):1–27, 2003. URL <http://www.jstatsoft.org/v08/i15/>. [p2]
- J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, second edition, 2011. URL <http://socserv.socsci.mcmaster.ca/jfox/Books/Companion>. [p2]
- J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29: 1189–1232, 2000. [p1]
- J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, 1981. [p2]
- S. Garnier. *viridis: Default Color Maps from 'matplotlib'*, 2016. URL <https://CRAN.R-project.org/package=viridis>. R package version 0.3.4. [p4]
- 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. [p2]
- B. Greenwell. *partial: An R Package for Constructing Partial Dependence Functions*, 2016. URL <https://CRAN.R-project.org/package=partial>. R package version 0.0.1. [p1]
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York, 2009. [p6]
- T. Hothorn and A. Zeileis. *partykit: A Laboratory for Recursive Partytioning*, 2016. URL <https://CRAN.R-project.org/package=partykit>. R package version 1.0-5. [p1]
- A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL <http://www.jstatsoft.org/v11/i09/>. [p2]
- 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]
- D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2015. URL <https://CRAN.R-project.org/package=e1071>. R package version 1.6-7. [p2]
- S. Milborrow. *plotmo: Plot a Model's Response and Residuals*, 2015. URL <https://CRAN.R-project.org/package=plotmo>. R package version 3.1.4. [p1]
- S. Milborrow. *earth: Multivariate Adaptive Regression Splines*, 2016. URL <https://CRAN.R-project.org/package=earth>. R package version 4.4.4. [p2]



- A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2015. URL <https://CRAN.R-project.org/package=ipred>. R package version 0.9-5. [p2]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL <http://lmdvr.r-forge.r-project.org>. ISBN 978-0-387-75968-5. [p4]
- T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2015. URL <https://CRAN.R-project.org/package=rpart>. R package version 4.1-10. [p2]
- C. S. Torsten Hothorn, Kurt Hornik and A. Zeileis. *party: A Laboratory for Recursive Partytioning*, 2015. URL <https://CRAN.R-project.org/package=party>. R package version 1.0-25. [p1]
- W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL <http://www.stats.ox.ac.uk/pub/MASS4>. [p2]
- 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/>. [p5]

Brandon M. Greenwell  
Infoscitex Corporation  
4027 Colonel Glenn Highway  
Suite 210  
Dayton, OH 45431-1672  
United States of America  
[bgreenwell@infoscitex.com](mailto:bgreenwell@infoscitex.com)