Response to Comments from Reviewer 1

Thank you for your time and thoughtful comments. I believe that they have helped to greatly improved, not only the manuscript, but the package it describes as well. Please find below a copy of your comments along with my responses.

Comment 1.1

Bug: A graph is produced as expected if the code below is cut-and-pasted at the command line . However, if the code is put into a file (say `foo.R`) and `source("foo.R")` is executed at the command line, no graph is produced. (But note that if `plot=TRUE` is added to the call to `partial()`, then a plot is produced as expected.)

```r
library(randomForest)
data(airquality)
ozone.rf <- randomForest(Ozone ~ ., data=airquality, na.action=na.omit)
library(pdp)
pd <- partial(ozone.rf, pred.var="Temp")
plotPartial(pd)
```

Response 1.1

This is simply a side effect of using `lattice` to produce the plots. If sourcing a script, then the plot needs to be explicitly printed (e.g., `print(plotPartial(pd))` or `print(partial(fit, pred.var = "x", plot = TRUE))`). This has been clarified in the package documentation and the manuscript (see **Note:** at the bottom of page 3). The fact that this worked when calling `partial` with `plot = TRUE` was a bug (which is now fixed).

Comment 1.2

I couldn't get `partial` to work with `gbm` (version 2.1.1) models (although I didn't try very hard). Perhaps this is because the `gbm` maintainer changed the `gbm` interface in a non-backcompatible way. The following (where `gbm.mod` is a model built using the gbm package)

```r
partial(gbm.mod, pred.var="age")
```

gives

```
Error in paste("Using", n.trees, "trees...\n") :
argument "n.trees" is missing, with no default
```

I also tried

```r
partial(gbm.mod, pred.var="age", n.trees=100)
```

but that gives

```
Error in .fun(piece, ...) : unused argument (n.trees = 100)
```

Response 1.2

This has been fixed in `pdp` (version 0.3.0). The user may still have to supply `n.trees` in the call to `partial` (depending on which version of `gbm` they are using). Also, as suggested in another comment, a `/slowtests` directory is underway and will help prevent future issues like this. Thanks for cathing this.

Comment 1.3

Standard partial matching isn't implemented (e.g., `partial(..., type="r")`).

Response 1.3

Partial matching is now implemented (as of `pdp` version 0.3.0), but only for the `type` argument. Thanks for the suggestion.

Comment 1.4

The following code incorrectly gives an error message

```
"Error in names(pd_df) <- c(pred.var, "y") :
  'names' attribute [2] must be the same length as the vector [1]"
```

It seems to work ok if you change the variable names.

```r
library(pdp)
data <- data.frame(V1 = 1:10, V2 = 1:10)
mod <- lm(V2 ~ V1, data = data)
partial(mod, pred.var = "V1")  # error
```

Response 1.4

This has been fixed in `pdp` (version 0.3.0). This was simply a side effect of `partial`'s internal use of `plyr::adply` which uses default variable names (e.g., `V1`, `V2`, etc.). This caused the column `V1` in the output data frame to be overwritten, producing the error.

```r
library(pdp)
data <- data.frame(V1 = 1:10, V2 = 1:10)
mod <- lm(V2 ~ V1, data = data)
partial(mod, pred.var = "V1")  # no more error
```

```
##    V1 yhat
## 1   1    1
## 2   2    2
## 3   3    3
## 4   4    4
## 5   5    5
## 6   6    6
## 7   7    7
## 8   8    8
## 9   9    9
## 10 10   10
```

Comment 1.5

I think you should mention in the documentation something about the assumptions you make about accessing the model data. The following (somewhat contrived) example fails with `"Error in train[[x]] : subscript out of bounds"`:

```r
data(trees)
foo <- function(data) {
  lm(Volume ~ ., data = data)
}
mod <- foo(trees)
library(pdp)
partial(mod, pred.var = "Girth", plot = TRUE)  # fails
```

and the following silently gives misleading results:

```r
library(pdp)
data(trees)
mod <- lm(Volume ~ ., data = trees)
partial(mod, pred.var = "Girth", plot = TRUE)  # ok
trees <- trees[1:2, ]
partial(mod, pred.var = "Girth", plot = TRUE)  # wrong xlim
```

I'm not necessarily saying that you need to resolve the above two issues, but at least mention in the documentation that this kind of thing can happen, and how to avoid it (basically call partial in the same environment used to build the model, and don't change any of the data used to build the model).

Response 1.5

Agreed! Both the manuscript (see page 3) and the help page for `partial` have been expanded to discuss this. Unfortunately, some functions don't store a copy of the training data. However, I have expanded the code to work harder—especially for `S4` methods—to inform the user to supply the training data when necessary. Re-running your first example should give an informative error message:

```r
data(trees)
foo <- function(data)
{
  lm(Volume ~ ., data = data)
}
mod <- foo(trees)
library(pdp)
partial(mod, pred.var = "Girth", plot = TRUE)  # fails with better error mssg
```

```
## Error in getTrainingData.default(object): The training data could not be extracted from object. Plea
```

Comment 1.6

The following code gives an obscure error message:

```
"Error in seq.default(from = min(train[[x]], na.rm = TRUE),
   to = max(train[[x]],  :
  'from' cannot be NA, NaN or infinite"
```

Some better handholding for the user may be helpful here.

```r
library(pdp)
data(trees)
mod <- lm(Volume~., data=trees)
partial(mod, pred.var="nosuchvariable", plot=TRUE) # obscure err msg
```

Response 1.6

This has been fixed in `pdp` (version 0.3.0) by adding an informative error message listing which variables are the cause of the issue.

```r
library(pdp)
data(trees)
mod <- lm(Volume~., data=trees)
partial(mod, pred.var=c("Girth", "foo", "bar"), plot=TRUE)
```

```
## Error in partial.default(mod, pred.var = c("Girth", "foo", "bar"), plot = TRUE): foo, bar not found
```

Comment 1.7

On page 2 of the paper it says "The columns of the data frame are labeled in the same order as the features supplied to `pred.var`, and the last column is always labeled `y`". What happens if `y` is one of the predictors?

Response 1.7

The last column is now labeled `yhat`, which is less likely to be a predictor name. When one of the predictors is labeled `"y"` `randomForest::partialPlot` fails while `gbm::plot.gbm` does not. I think regardless of what the default column name is, there is always the possibility of this problem. This has been filed as an issue on the `pdp` GitHib page: https://github.com/bgreenwell/pdp/issues/22.

Comment 1.8

Hastie et al. in the book cited in the paper say "Although such a collection [of partial dependence plots] can seldom provide a comprehensive depiction of the approximation, it can often produce helpful clues". Therefore in in the abstract for the paper, to avoid overselling consider changing "relationship between the outcome and predictors of interest can be easily understood" to "relationship between the outcome and predictors of interest can be more easily understood".

Response 1.8

Fixed. Thanks for the suggestion.

Comment 1.9

On page 3 in the PDF of the paper, add a newline to the error message in the paper:

```
Error: The training data could not be extracted from object. Please
supply the raw training data using the ...
```

Response 1.9

Fixed. Thanks for pointing that out.

Comment 1.10

Is there any reason the package is called `pdp` but the main function is called `partial`? Consider giving the function the same name as the package, so e.g., the following works (to help the user get started)

```r
library(pdp)
?pdp  # fails
```

Response 1.10

The package was originally called `partial`, but at the time of its original submission this conflicted with a CRAN package of the same name (but with mixed lowercase/uppercase letters). So, I decided to change the name of the package to `pdp`; besides, I think `partial` and `plotPartial` work well together as function names. However, I did add a help page describing the package, main functions, etc. So, `?pdp` no longer fails. However, I am not opposed to creating an alias to `partial` called `pdp` in the future.

Comment 1.11

`partial` should return the same value regardless of whether or not the `plot` argument is used?

Response 1.11

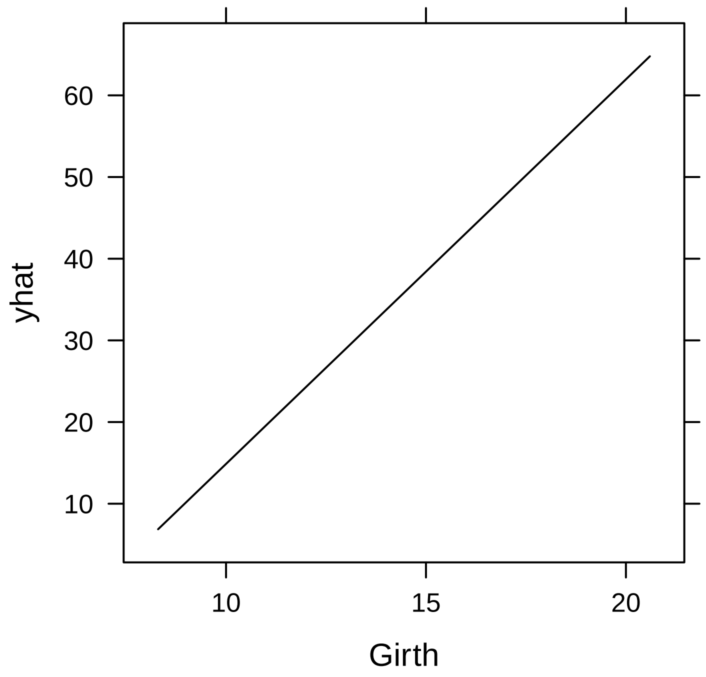
I dont think so, because I want `partial` to behave like a `lattice` function when using `plot = TRUE`; that is, I want it to return a `"trellis"` object. However, I did add the partial dependence data as an attribute to the returned `"trellis"` object so that it can easily be extracted if necessary (e.g., if they do not like the plot and don't want to re-run partial). For example,

```r
library(pdp)
data(trees)
mod <- lm(Volume ~ ., data = trees)
pd.Girth <- partial(mod, pred.var = "Girth")  # data only
pdp.Girth <- partial(mod, pred.var = "Girth", plot = TRUE)  # plot
identical(pd.Girth, attr(pdp.Girth, "partial.data"))  # should be identical
```

```
## [1] TRUE
```

Also, the `lattice` function `trellis.last.object` is now exported when `pdp` is loaded. This would be useful if the user did not store the PDP but needs to retrieve the outout data to avoid re-running `partial` (e.g., when computation time is a concern). For example,

```r
library(pdp)
data(trees)
mod <- lm(Volume ~ ., data = trees)
partial(mod, pred.var = "Girth", plot = TRUE)
```



```
pdp.Girth <- lattice::trellis.last.object()
pd.Girth <- attr(pdp.Girth, "partial.data")
head(pd.Girth)
```

```
##       Girth      yhat
## 1  8.300000  6.873167
## 2  8.773077  9.100489
## 3  9.246154 11.327811
## 4  9.719231 13.555133
## 5 10.192308 15.782455
## 6 10.665385 18.009777
```

This has been noted in the package documentation (see `?partial`). Thanks for the idea!

Comment 1.12

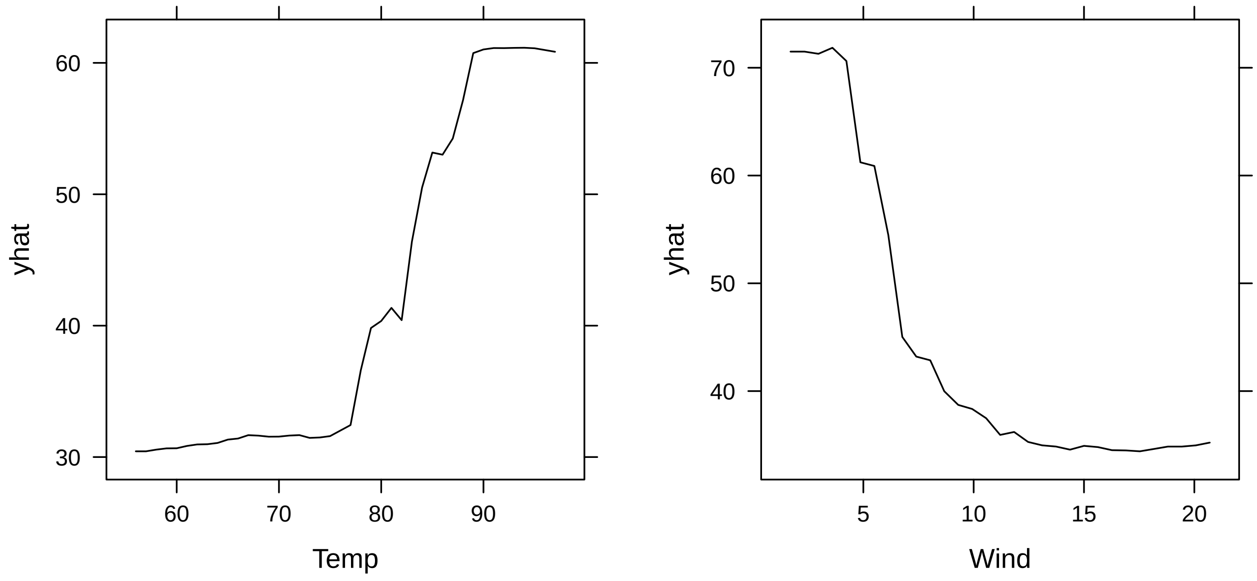
A call to say `par(mfrow = c(2, 2))` before calling `plotPartial` is ignored by `plotPartial`. This is a pity because it would allow multiple plots to be put on one page like this:

```
par(mfrow = c(1, 2))
partial(ozone.rf, pred.var = "Temp", plot = TRUE)
partial(ozone.rf, pred.var = "Wind", plot = TRUE)
```

Response 1.12

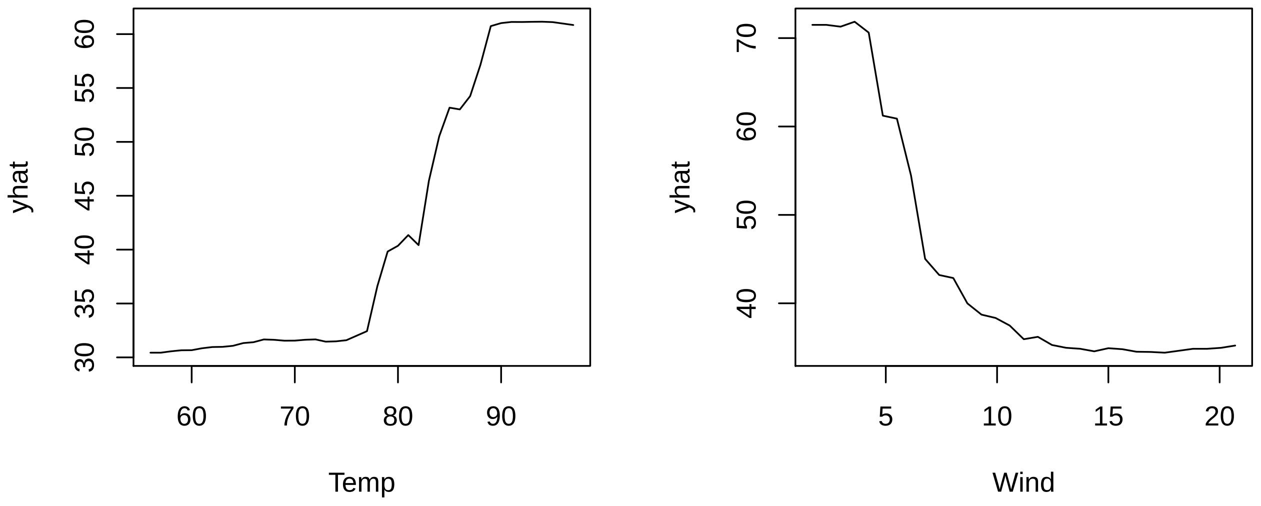
The command `par(mfrow = c(2, 2))` does not work with `"trellis"` objects, such as those produced by `lattice`—which `partial` relies on for producing its plots. Other methods are available. For convenience, `pdp` now imports the `grid.arrange` function from the `gridxtra` package (this has been discussed in the manuscript and incorporated into the examples where appropriate).

```
grid.arrange(
  partial(ozone.rf, pred.var = "Temp", plot = TRUE),
  partial(ozone.rf, pred.var = "Wind", plot = TRUE),
  ncol = 2
)
```



However, since `partial` returns a data frame by default, using `par` would work in the following case

```
par(mfrow = c(1, 2))
plot(partial(ozone.rf, pred.var = "Temp"), type = "l")
plot(partial(ozone.rf, pred.var = "Wind"), type = "l")
```



Comment 1.13

It would be nice if the functions automatically plotted all the (important) variables in a grid of plots, without forcing the user to explicitly specify them.

Response 1.13

I agree, this would be a nice feature, but the problems I see are (1) not all models naturally emit a "variable importance" score; the `caret` package has come a long way in this regard, and (2) too many plots will look cluttered. At this point, I'd rather encourage the user to select which variables they would like to see plotted. I am not opposed to including this feature in the future. I added this as a feature request: https://github.com/bgreenwell/pdp/issues/19. Thanks for the suggestion!

Comment 1.14

It would be nice if the functions automatically got the data from the model used to build the model in more cases without forcing the user to explicitly specify the data. Since XGBoost seems now quite popular, perhaps the authors of that package could be persuaded to make the original model data available for functions like partial (currently XGBoost uses a custom object `xgb.DMatrix` which is currently write-only—one can't access the data stored in `xgb.DMatrix`).

Response 1.14

`kernlab::ksvm` does not store a copy of the training data either, as far as I can tell. As time goes on, the coverage will certainly improve, which will make it easier on the user. For example, `partial` now works with `caret` which, by default, stores the taining data in a component called `trainingData`. Also, I suspect that the package maintainers for `xgboost` (and similar packages designed for working with "large" data sets) would be hesitent in storing a copy of the training data with the fitted model for memory reasons.

Comment 1.15

Consider adding a `slowtests` directory (below `pdp/inst/` or elsewhere) that has tests for all the models supported by the package (Table 1 in the paper) to easily check back compatibility when packages change. (For justification for such tests see for example my comments on `gbm` back compatibility above.) These tests will probably be quite slow, but that won't be an issue because they won't be invoked by CRAN check; the slow tests would be just for the maintainer of the `pdp` package to run manually.

Response 1.15

This is a good suggestion and is currently under development: https://github.com/bgreenwell/pdp/blob/master/slowtests/slowtests.R. In fact, the `gbm` maintainers have already contacted me about a breaking change caused by the next planned release of their package.