ggRandomForests: Random Forests for Regression

John Ehrlinger
Cleveland Clinic

Abstract

Random Forests (Breiman 2001) (RF) are a non-parametric statistical method requiring no distributional assumptions on covariate relation to the response. RF are a robust, nonlinear technique that optimizes predictive accuracy by fitting an ensemble of trees to stabilize model estimates. The **randomForestSRC** package (Ishwaran and Kogalur 2014) is a unified treatment of Breiman's random forests for survival, regression and classification problems.

Predictive accuracy make RF an attractive alternative to parametric models, though complexity and interpretability of the forest hinder wider application of the method. We introduce the **ggRandomForests** package, tools for visually understand random forest models grown in R (R Core Team 2014) with the **randomForestSRC** package. The **ggRandomForests** package is structured to extract intermediate data objects from **randomForestSRC** objects and generates figures using the **ggplot2** (Wickham 2009) graphics package.

This document is structured as a tutorial for building random forests for regression with the **randomForestSRC** package and using the **ggRandomForests** package for investigating how the forest is constructed. We investigate the Boston Housing data (Harrison and Rubinfeld 1978; Belsley, Kuh, and Welsch 1980). We demonstrate random forest variable selection using Variable Importance (VIMP) (Breiman 2001) and Minimal Depth (Ishwaran, Kogalur, Gorodeski, Minn, and Lauer 2010), a property derived from the construction of each tree within the forest. We will also demonstrate the use of variable dependence plots (Friedman 2000) to aid interpretation RF results. We then examine variable interactions between covariates using a minimal depth interactions, and conditional variable dependence plots. The goal of the exercise is to demonstrate the strength of using Random Forest methods for both prediction and information retrieval in regression settings.

Keywords: random forest, regression, VIMP, minimal depth, R, **randomForestSRC**.

About this document

This document is a package vignette for the **ggRandomForests** package for "Visually Exploring Random Forests" (http://CRAN.R-project.org/package=ggRandomForests). The **ggRandomForests** package is designed for use with the **randomForestSRC** package (Ishwaran and Kogalur 2014, http://CRAN.R-project.org/package=randomForestSRC) for growing random forests for survival (time to event response), regression (continuous response) and classification (categorical response) settings and uses the **ggplot2** package (Wickham 2009, http://CRAN.R-project.org/package=ggplot2) for plotting diagnostic and variable association results. **ggRandomForests** is structured to extract data objects from **randomForestSRC**

objects and provides functions for printing and plotting these objects.

The vignette is a tutorial for using the **ggRandomForests** package with the **randomForestSRC** package for building and post-processing random forests for regression settings. In this tutorial, we explore a random forest for regression model constructed for the Boston housing data set (Harrison and Rubinfeld 1978; Belsley *et al.* 1980), available in the **MASS** package (Venables and Ripley 2002). We grow a random forest and demonstrate how **ggRandomForests** can be used when determining how the response depends on predictive variables within the model. The tutorial demonstrates the design and usage of many of **ggRandomForests** functions and features and also how to modify and customize the resulting `ggplot` graphic objects along the way.

The vignette is written in LATEXusing the **knitr** package (Xie 2015, 2014, 2013, `http://CRAN.R-project.org/package=knitr`), which facilitates weaving R (R Core Team 2014) code, results and figures into document text. Throughout this document, R code will be displayed in *code blocks* as shown below. This code block loads the R packages required to run the source code listed in code blocks throughout the remainder of this document.

```
R> ################# Load packages #################
R> library("ggplot2")        # Graphics engine
R> library("RColorBrewer")   # Nice color palettes
R> library("plot3D")         # for 3d surfaces.
R> library("dplyr")          # Better data manipulations
R> library("tidyr")          # gather variables into long format
R> library("parallel")       # mclapply for multicore processing
R>
R> # Analysis packages.
R> library("randomForestSRC") # random forests for survival, regression and
R>                            # classification
R> library("ggRandomForests") # ggplot2 random forest figures (This!)
R>
R> ############### Default Settings #################
R> theme_set(theme_bw())     # A ggplot2 theme with white background
R>
R> ## Set open circle for censored, and x for events
R> event.marks <- c(1, 4)
R> event.labels <- c(FALSE, TRUE)
R>
R> ## We want red for death events, so reorder this set.
R> strCol <- brewer.pal(3, "Set1")[c(2,1,3)]
```

This vignette is available within the **ggRandomForests** package on the Comprehensive R Archive Network (CRAN) (R Core Team 2014, `http://cran.r-project.org`). Once the package has been installed, the vignette can be viewed directly from within R with the following command:

```
R> vignette("randomForestSRC-Regression", package = "ggRandomForests")
```

A development version of the **ggRandomForests** package is also available on GitHub (`https:`

`//github.com`). We invite comments, feature requests and bug reports for this package at `https://github.com/ehrlinger/ggRandomForests`.

1. Introduction

Random Forests (Breiman 2001) (RF) are a fully non-parametric statistical method which requires no distributional or functional assumptions on covariate relation to the response. RF is a robust, nonlinear technique that optimizes predictive accuracy by fitting an ensemble of trees to stabilize model estimates. Random Survival Forests (RSF) (Ishwaran and Kogalur 2007; Ishwaran, Kogalur, Blackstone, and Lauer 2008) are an extension of Breiman's RF techniques to survival settings, allowing efficient non-parametric analysis of time to event data. The **randomForestSRC** package (Ishwaran and Kogalur 2014) is a unified treatment of Breiman's random forests for survival (time to event response), regression (continuous response) and classification (categorical response) problems.

Predictive accuracy make RF an attractive alternative to parametric models, though complexity and interpretability of the forest hinder wider application of the method. We introduce the **ggRandomForests** package for visually exploring random forest models. The **ggRandomForests** package is structured to extract intermediate data objects from **randomForestSRC** objects and generate figures using the **ggplot2** graphics package (Wickham 2009).

Many of the figures created by the **ggRandomForests** package are also available directly from within the **randomForestSRC** package. However **ggRandomForests** offers the following advantages:

- Separation of data and figures: **ggRandomForests** contains functions that operate on either the `randomForestSRC::rfsrc` forest object directly, or on the output from **randomForestSRC** post processing functions (i.e. `plot.variable`, `var.select`, `find.interaction`) to generate intermediate **ggRandomForests** data objects. functions are provide to further process these objects and plot results using the **ggplot2** graphics package. Alternatively, users can use these data objects for their own custom plotting or analysis operations.

- Each data object/figure is a single, self contained object. This allows simple modification and manipulation of the data or **ggplot2** objects to meet users specific needs and requirements.

- The use of **ggplot2** for plotting. We chose to use the **ggplot2** package for our figures to allow users flexibility in modifying the figures to their liking. Each plot function returns either a single `ggplot` object, or a `list` of `ggplot` objects, allowing users to use additional **ggplot2** functions or themes to modify and customize the figures to their liking.

This document is formatted as a tutorial for using the **randomForestSRC** package for building and post-processing random forest models with the **ggRandomForests** package for investigating how the forest is constructed. In this tutorial, we use the Boston Housing Data (Section 2), available in the **MASS** package (Venables and Ripley 2002), to build a random forest for regression (Section 3) and demonstrate the tools in the **ggRandomForests** package for examining the forest construction.

Random forests are not parsimonious, but use all variables available in the construction of a response predictor. We demonstrate a random forest variable selection (Section 4) process using the Variable Importance (Section 4.1) measure (VIMP) (Breiman 2001) as well as Minimal Depth (Section 4.2) (Ishwaran *et al.* 2010), a property derived from the construction of each tree within the forest, to assess the impact of variables on forest prediction.

Once we have an idea of which variables we are want to investigate further, we will use variable dependence plots (Friedman 2000) to understand how a variable is related to the response (Section 5). Marginal dependence plots (Section 5.1) give us an idea of the overall trend of a variable/response relation, while partial dependence plots (Section 5.2) show us a risk adjusted relation. These figures may show strongly non-linear variable/response relations that are not easily obtained through a parametric approach. We are also interested in examining variable interactions within the forest model (Section 6). Using a minimal depth approach, we can quantify how closely variables are related within the forest, and generate marginal dependence (Section 7) and partial dependence (Section!7.1) (risk adjusted) conditioning plots (coplots) (Chambers 1992; Cleveland 1993) to examine these interactions graphically.

2. Data: Boston Housing Values

The Boston Housing data is a standard benchmark data set for regression models. It contains data for 506 census tracts of Boston from the 1970 census (Harrison and Rubinfeld 1978; Belsley *et al.* 1980). The data is available in multiple R packages, but to keep the installation dependencies for the **ggRandomForests** package down, we will use the data contained in the **MASS** package (Venables and Ripley 2002, `http://CRAN.R-project.org/package=MASS`), available with the base install of R. The following code block loads the data into the environment. We include a table of the Boston data set variable names, types and descriptions for reference when we interpret the model results.

```
R> # Load the Boston Housing data
R> data(Boston, package="MASS")
R>
R> # Set modes correctly. For binary variables: transform to logical
R> Boston$chas <- as.logical(Boston$chas)
```

The main objective of the Boston Housing data is to investigate variables associated with predicting the median value of homes (continuous `medv` response) within 506 suburban areas of Boston.

2.1. Exploratory Data Analysis

It is good practice to view your data before beginning an analysis, what Tukey (1977) refers to as Exploratory Data Analysis (EDA). To facilitate this, we use **ggplot2** figures with the `ggplot2::facet_wrap` command to create two sets of panel plots, one for categorical variables with boxplots at each level, and one of scatter plots for continuous variables. Each variable is plotted along a selected continuous variable on the X-axis. These figures help to find outliers, missing values and other data anomalies in each variable before getting deep into the analysis. We have also created a separate **shiny** app (Chang,

Variable	Description	type
crim	Crime rate by town.	numeric
zn	Proportion of residential land zoned for lots over 25,000 sq.ft.	numeric
indus	Proportion of non-retail business acres per town.	numeric
chas	Charles River (tract bounds river).	logical
nox	Nitrogen oxides concentration (10 ppm).	numeric
rm	Number of rooms per dwelling.	numeric
age	Proportion of units built prior to 1940.	numeric
dis	Distances to Boston employment center.	numeric
rad	Accessibility to highways.	integer
tax	Property tax rate per $10,000.	numeric
ptratio	Pupil teacher ratio by town.	numeric
black	Proportion of blacks by town.	numeric
lstat	Lower status of the population (percent).	numeric
medv	Median value of homes ($1000s).	numeric

Table 1: `Boston` housing data dictionary.

Cheng, Allaire, Xie, and McPherson 2015, `http://shiny.rstudio.com`), available at `https://ehrlinger.shinyapps.io/xportEDA`, for creating similar figures with an arbitrary data set, to make the EDA process easier for users.

The Boston housing data consists almost entirely of continuous variables, with the exception of the "Charles river" logical variable. A simple EDA visualization to use for this data is a single panel plot of the continuous variables, with observation points colored by the logical variable. Missing values in our continuous variable plots are indicated by the rug marks along the x-axis, of which there are none in this data. We used the Boston housing response variable, the median value of homes (`medv`), for X variable.

This figure is loosely related to a pairs scatter plot (Becker, Chambers, and Wilks 1988), but in this case we only examine the relation between the response variable against the remainder. Plotting the data against the response also gives us a "sanity check" when viewing our model results. It's pretty obvious from this figure that we should find a strong relation between median home values and the `lstat` and `rm` variables.

3. Random Forest - Regression

A Random Forest is grown by *bagging* (Breiman 1996a) a collection of *classification and regression trees* (CART) (Breiman, Friedman, Olshen, and Stone 1984). The method uses a set of B bootstrap (Efron and Tibshirani 1994) samples, growing an independent tree model on each sub-sample of the population. Each tree is grown by recursively partitioning the population based on optimization of a *split rule* over the p-dimensional covariate space. At each split, a subset of $m \leq p$ candidate variables are tested for the split rule optimization, dividing each node into two daughter nodes. Each daughter node is then split again until the process reaches the *stopping criteria* of either *node purity* or *node member size*, which defines the set of *terminal (unsplit) nodes* for the tree. In regression trees, the split rule is based on minimizing the mean squared error, whereas in classification problems, the Gini index is
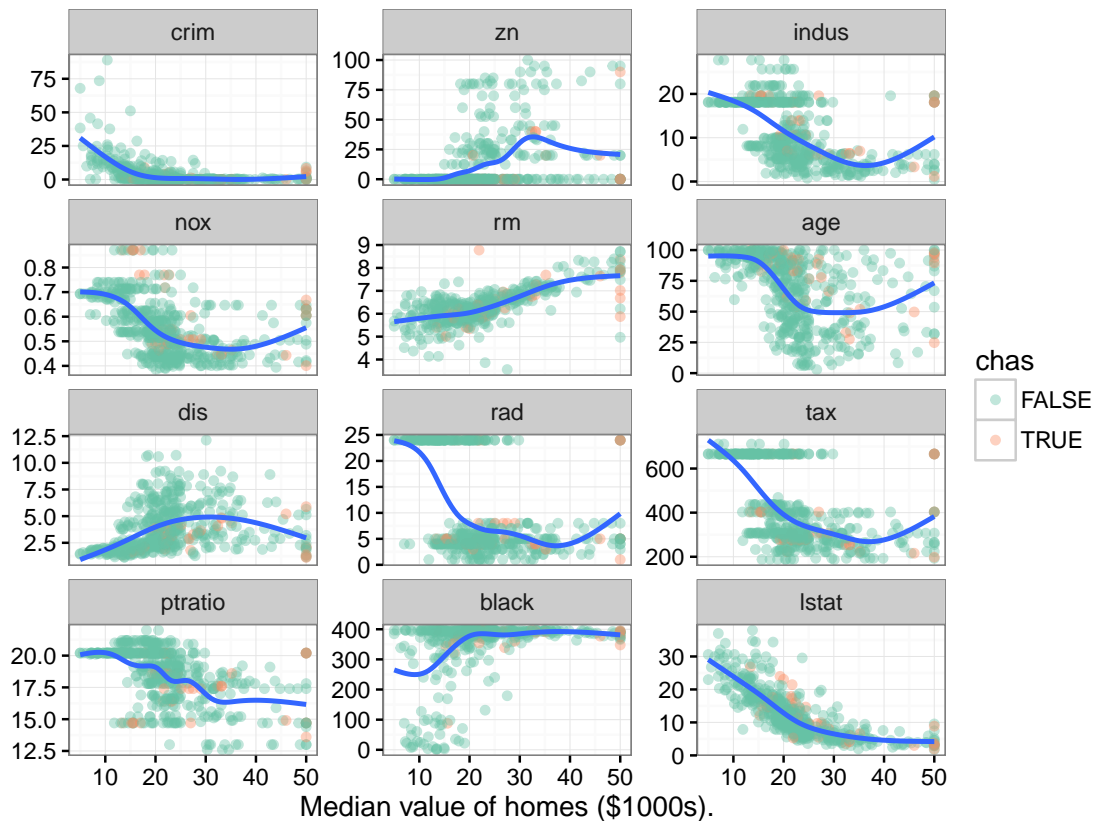
Figure 1: EDA variable plots. Points indicate variable value against the median home value variable. Points are colored according to the chas variable.

used (Friedman 2000).

Random Forests sort each training set observation into one unique terminal node per tree. Tree estimates for each observation are constructed at each terminal node, among the terminal node members. The Random Forest estimate for each observation is then calculated by aggregating, averaging (regression) or votes (classification), the terminal node results across the collection of B trees.

For this tutorial, we grow the random forest for regression using the `rfsrc` command to predict the median home value (`medv` variable) using the remaining 13 independent predictor variables. For this example we will use the default set of $B = 1000$ trees (`ntree` argument), $m = 5$ candidate variables (`mtry`) for each split with a stopping criteria of at most `nodesize=5` observations within each terminal node.

Because growing random forests are computationally expensive, and the **ggRandomForests** package is targeted at the visualization of random forest objects, we will use cached copies of the **randomForestSRC** objects throughout this document. We include the cached objects as data sets in the **ggRandomForests** package. The actual `rfsrc` calls are included in comments within code blocks.

```
R> # Load the data, from the call:
R> # rfsrc_Boston <- rfsrc(medv~., data=Boston)
```

```
R> data(rfsrc_Boston)
R>
R> # print the forest summary
R> rfsrc_Boston
```

```
                         Sample size: 506
                     Number of trees: 1000
           Minimum terminal node size: 5
        Average no. of terminal nodes: 104.393
  No. of variables tried at each split: 5
                Total no. of variables: 13
                             Analysis: RF-R
                               Family: regr
                        Splitting rule: mse
                 % variance explained: 87.61
                           Error rate: 10.48
```

The `randomForestSRC::print.rfsrc` summary details the parameters used for the `rfsrc` call described above, and returns variance and generalization error estimate from the forest training set. The forest is built from 506 observations and 13 independent variables. It was constructed for the continuous `medv` variable using `ntree=1000` regression (`regr`) trees, randomly selecting 5 candidate variables at each node split, and terminating nodes with no fewer than 5 observations.

3.1. Generalization error estimates

One advantage of Random Forests is a built in generalization error estimate. Each bootstrap sample selects approximately 63.2% of the population on average. The remaining 36.8% of observations, the Out-of-Bag (OOB) (Breiman 1996b) sample, can be used as a hold out test set for each of the trees in the forest. An OOB prediction error estimate can be calculated for each observation by predicting the response over the set of trees which were NOT trained with that particular observation. The Out-of-Bag prediction error estimates have been shown to be nearly identical to n–fold cross validation estimates (Hastie, Tibshirani, and Friedman 2009). This feature of Random Forests allows us to obtain both model fit and validation in one pass of the algorithm.

The `gg_error` function operates on the `randomForestSRC::rfsrc` object to extract the error estimates as the forest is grown. The code block demonstrates part the **ggRandom-Forests** design philosophy, to create separate data objects and provide functions to operate on the data objects. The following code block first creates a `gg_error` object, then uses the `plot.gg_error` function to create a `ggplot` object for display.

```
R> # Plot the OOB errors against the growth of the forest.
R> gg_e <- gg_error(rfsrc_Boston)
R> plot(gg_e)
```

This figure demonstrates that it does not take a large number of trees to stabilize the forest prediction error estimate. However, to ensure that each variable has enough of a chance to be
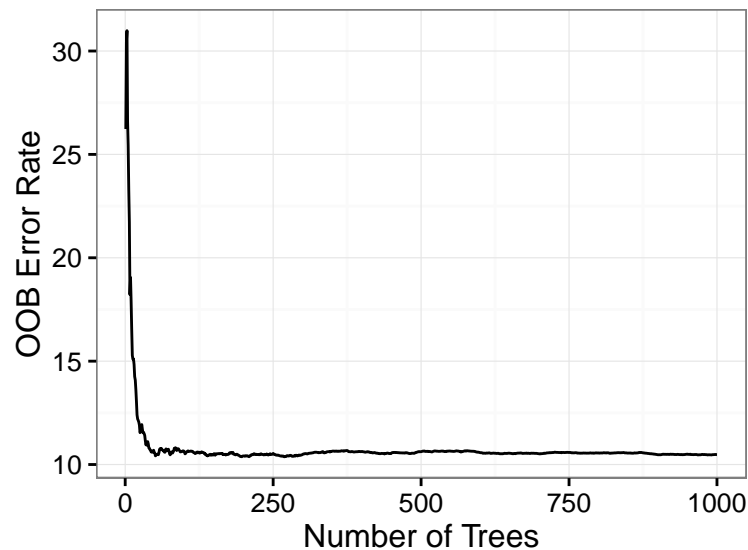
Figure 2: Random forest generalization error. OOB error convergence along the number of trees in the forest.

included in the forest prediction process, we do want to create a rather large random forest of trees.

3.2. Random Forest Prediction

The `gg_rfsrc` function extracts the OOB prediction estimates from the random forest. This code block executes the the data extraction and plotting in one line, since we are not interested in holding the prediction estimates for later reuse. Also note that we add in the additional **ggplot2** command (`coord_cartesian`) to modify the plot object. Each of the **ggRandom-Forests** plot commands return `ggplot` objects, which we can also store for modification or reuse later in the analysis.

```
R> # Plot predicted median home values.
R> plot(gg_rfsrc(rfsrc_Boston), alpha=.5)+
+   coord_cartesian(ylim=c(5,49))
```

The `gg_rfsrc` plot shows the predicted median home value, one point for each observation in the training set. The points are jittered around a single point on the x-axis, since we are only looking at predicted values from the forest. These estimates are Out of Bag, which are analogous to test set estimates. The boxplot is shown to give an indication of the distribution of the prediction estimates. For this analysis the figure is another model sanity check, as we are more interested in exploring the "why" questions for these predictions.

4. Variable Selection

Random forests are not parsimonious, but use all variables available in the construction of a response predictor. Also, unlike parametric models, Random Forests do not require the
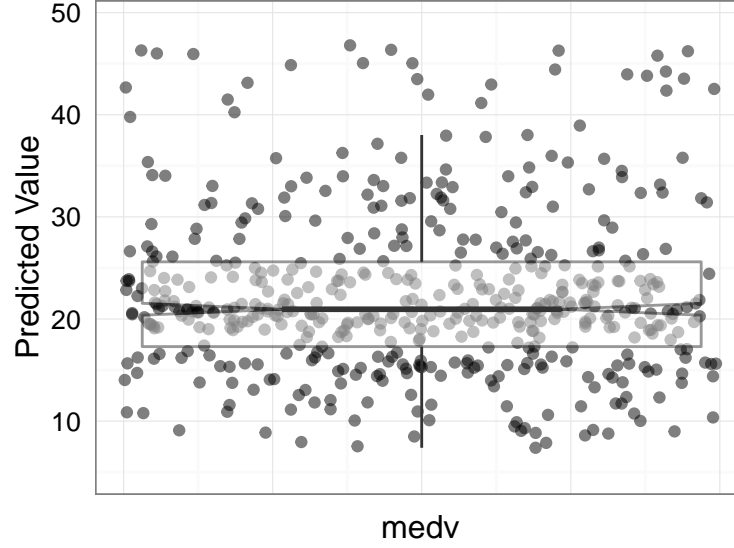
Figure 3: OOB predicted median home values. Points are jittered to help visualize predictions for each observation. Boxplot indicates the distribution of the predicted values.

explicit specification of the functional form of covariates to the response. Therefore there is no explicit p-value/significance test for variable selection with a random forest model. Instead, RF ascertain which variables contribute to the prediction through the split rule optimization, optimally choosing variables which separate observations. We use two separate approaches to explore the RF selection process, Variable Importance (Section 4.1) and Minimal Depth (Section 4.2).

4.1. Variable Importance.

Variable importance (VIMP) was originally defined in CART using a measure involving surrogate variables (see Chapter 5 of Breiman *et al.* (1984)). The most popular VIMP method uses a prediction error approach involving "noising-up" each variable in turn. VIMP for a variable x_v is the difference between prediction error when x_v is noised up by randomly permuting its values, compared to prediction error under the observed values (Breiman 2001; Liaw and Wiener 2002; Ishwaran 2007; Ishwaran *et al.* 2008).

Since VIMP is the difference between OOB prediction error before and after permutation, a large VIMP value indicates that misspecification detracts from the variable predictive accuracy in the forest. VIMP close to zero indicates the variable contributes nothing to predictive accuracy, and negative values indicate the predictive accuracy *improves* when the variable is mispecified. In the later case, we assume noise is more informative than the true variable. As such, we ignore variables with negative and near zero values of VIMP, relying on large positive values to indicate that the predictive power of the forest is dependent on those variables.

The `gg_vimp` function extracts VIMP measures for each of the variables used to grow the forest. The `plot.gg_vimp` function shows the variables, in VIMP rank order, from the largest (Lower Status) at the top, to smallest (Charles River) at the bottom. VIMP measures are shown using bars to compare the scale of the error increase under permutation.

```
R> # Plot the VIMP rankings of independent variables.
R> plot(gg_vimp(rfsrc_Boston), lbls=st.labs)
```
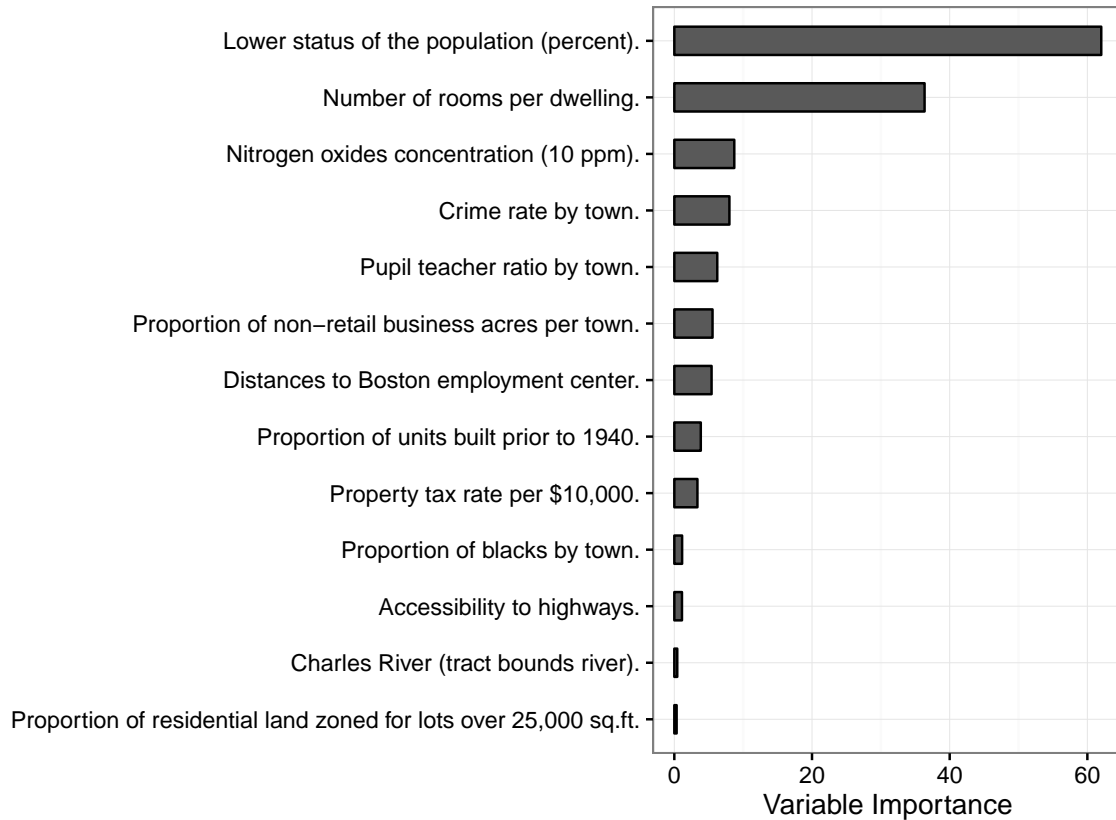

Figure 4: Random forest VIMP plot. Bars are colored by sign of VIMP, longer blue bars indicate more important variables.

For our random forest, the top two variables (`lstat` and `rm`) have the largest VIMP, with a sizable difference to the remaining variables, which mostly have similar VIMP measure. This indicates we should focus attention on these two variables, at least, over the others.

In this example, all VIMP measures are positive, though some are small. When there are both negative and positive VIMP values, the `plot.gg_vimp` function will color VIMP by the sign of the measure. We use the `lbls` argument to pass a named `vector` of meaningful text descriptions to the `plot.gg_vimp` function, replacing the often terse variable names used by default.

4.2. Minimal Depth.

In VIMP, prognostic risk factors are determined by testing the forest prediction under alternative data settings, ranking the most important variables according to their impact on predictive ability of the forest. An alternative method uses inspection of the forest construction to rank variables. *Minimal depth* assumes that variables with high impact on the prediction are those that most frequently split nodes nearest to the trunks of the trees (i.e. at the root node) where they partition large samples of the population.

Within a tree, node levels are numbered based on their relative distance to the trunk of the tree (with the root at 0). Minimal depth measures the important risk factors by averaging the depth of the first split for each variable over all trees within the forest. Lower values of this measure indicate variables important in splitting large groups of patients.

The *maximal subtree* for a variable x is the largest subtree whose root node splits on x. All parent nodes of x's maximal subtree have nodes that split on variables other than x. The largest maximal subtree possible is at the root node. If a variable does not split the root node, it can have more than one maximal subtree, or a maximal subtree may also not exist if there are no splits on the variable. The minimal depth of a variables is a surrogate measure of predictiveness of the variable. The smaller the minimal depth, the more impact the variable has sorting observations, and therefore on the forest prediction.

The `gg_minimal_depth` function is analogous to the `gg_vimp` function for minimal depth. Variables are ranked from most important at the top (minimal depth measure), to least at the bottom (maximal minimal depth). The vertical dashed line indicates the minimal depth threshold where smaller minimal depth values indicate higher importance and larger indicate lower importance.

The `randomForestSRC::var.select` call is again a computationally intensive function, as it traverses the forest finding the maximal subtree within each tree for each variable before averaging the results we use in the `gg_minimal_depth` call. We again use the cached object strategy here to save computational time. The `var.select` call is included in the comment of this code block.

```
R> # Load the data, from the call:
R> # varsel_Boston <- var.select(rfsrc_Boston)
R> data(varsel_Boston)
R>
R> # Save the gg_minimal_depth object for later use.
R> gg_md <- gg_minimal_depth(varsel_Boston)
R>
R> # plot the object
R> plot(gg_md, lbls=st.labs)
```

In general, the selection of variables according to VIMP is to rather arbitrarily examine the values, looking for some point along the ranking where there is a large difference in VIMP measures. The minimal depth threshold method has a more quantitative approach to determine a selection threshold. Given minimal depth is a quantitative property of the forest construction, Ishwaran *et al.* (2010) also construct an analytic threshold for evidence of variable impact. A simple optimistic threshold rule uses the mean of the minimal depth distribution, classifying variables with minimal depth lower than this threshold as important in forest prediction. The minimal depth plot for our model indicates there are ten variables which have a higher impact (minimal depth below the mean value threshold) than the remaining three.

Since the VIMP and Minimal Depth measures use different criteria, we expect the variable ranking to be somewhat different. We use `gg_minimal_vimp` function to compare rankings between minimal depth and VIMP. In this call, we plot the stored `gg_minimal_depth` object
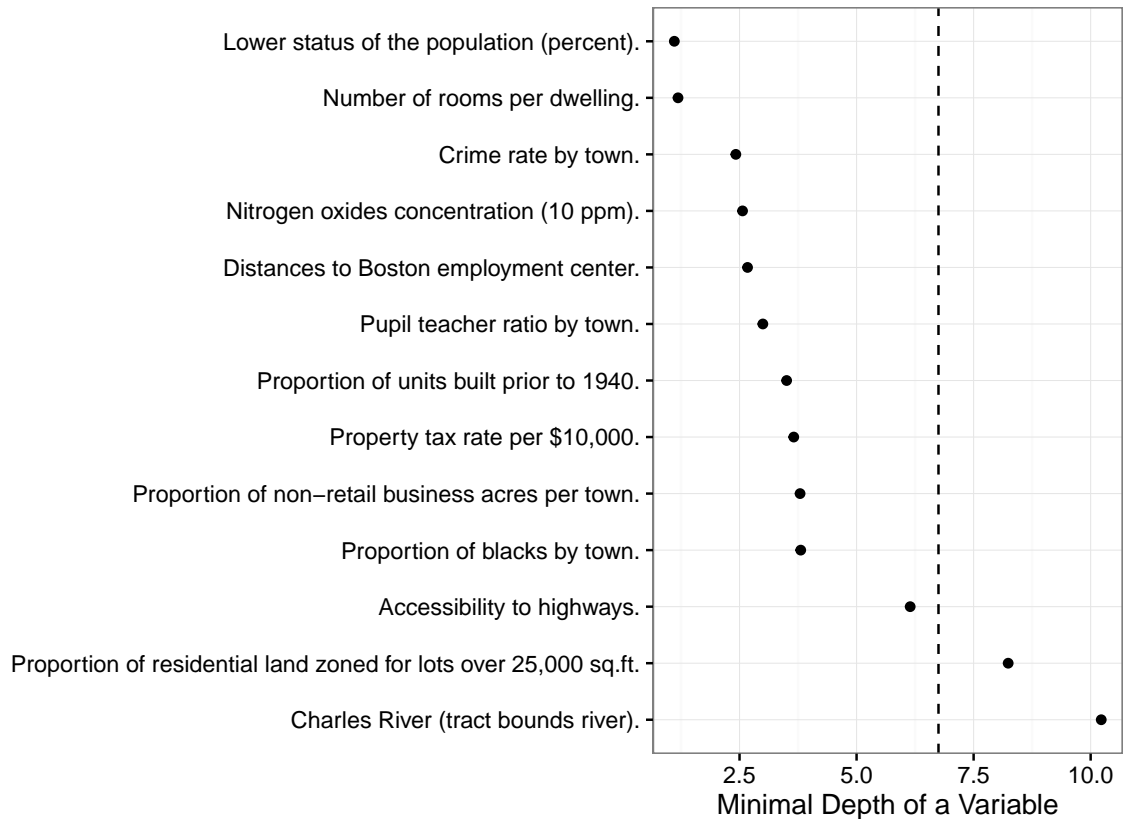
Figure 5: Minimal Depth variables in rank order, most important at the top. Vertical dashed line indicates the maximal minimal depth for important variables.

(gg_md), which would be equivalent to calling `plot.gg_minimal_vimp(varsel_Boston)` or `plot(gg_minimal_vimp(varsel_Boston))`.

```
R> # gg_minimal_depth objects contain information about
R> # both minimal depth and VIMP.
R> plot(gg_minimal_vimp(gg_md))
```

The points along the red dashed line indicates where the measures are in agreement. Points above the red dashed line are ranked higher by VIMP than by minimal depth, indicating the variables are sensitive to misspecification. Those below the line have a higher minimal depth ranking, indicating they are better at dividing large portions of the population. The further the points are from the line, the more the discrepancy between measures. The construction of this figure is skewed towards a minimal depth approach, by ranking variables along the y-axis, though points are colored by the sign of VIMP.

In our example, both minimal depth and VIMP indicate the strong relation of `lstat` and `rm` variables to the forest prediction, which agrees with our expectation from the EDA (Section 2.1) done at the beginning of this document. We now turn to investigating how these, and other variables, are related to the predicted response.
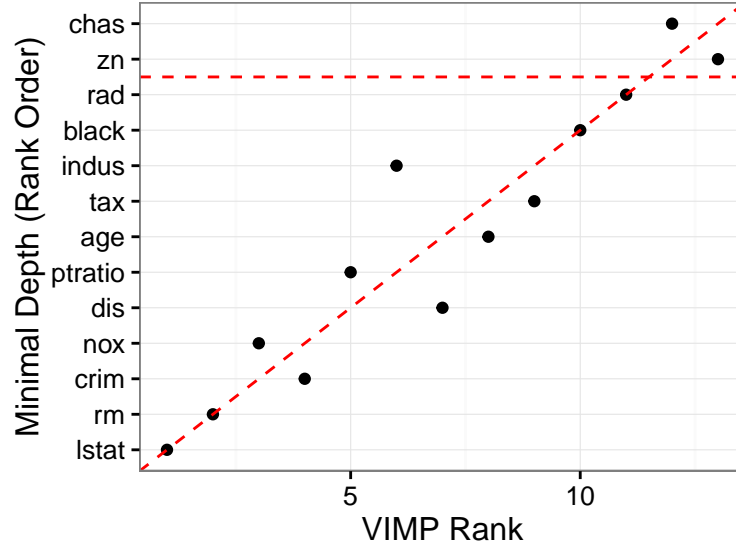
Figure 6: Comparing Minimal Depth and Vimp rankings. Points on the red dashed line are ranked equivalently, points below have higher VIMP, those above have higher minimal depth ranking. Variables are colored by the sign of the VIMP measure.

5. Response/Variable Dependence.

As random forests are not a parsimonious methodology, we can use the minimal depth and VIMP measures to reduce the number of variables we need to examine to a manageable subset. We would like to know how the forest response depends on some specific variables of interest. We often choose to examine variables of interest based on the study question, or other previous knowledge. In the absence of this, we will look at variables that contribute most to the predictive accuracy of the forest.

Although often characterized as a "black box" method, it is possible to express a random forest in functional form. In the end the forest predictor is some function, although complex, of the predictor variables

$$\hat{f}_{rf} = f(x).$$

We use graphical methods to examine the forest predicted response dependency on covariates. We again have two options, variable dependence (Section 5.1) plots are quick and easy to generate, and partial dependence (Section 5.2) plots are computationally intensive but give us a risk adjusted look at the dependence.

5.1. Variable Dependence

Variable dependence plots show the predicted response as a function of a covariate of interest, where each observation is represented by a point on the plot. Each predicted point is an individual observations, dependent on the full combination of all other covariates, not only on the covariate of interest. Interpretation of variable dependence plots can only be in general terms, as point predictions are a function of all covariates in that particular observation. However, variable dependence is straight forward to calculate, only requiring the predicted response for each observation.

We use the `gg_variable` function call to extract the training set variables and the predicted OOB response from `randomForestSRC::rfsrc` and `randomForestSRC::predict` objects. In the following code block, we will store the `gg_variable` data object for later use, as all remaining variable dependence plots can be constructed from this (`gg_v`) object. We will also use the minimal depth selected variables (minimal depth lower than the threshold value) from the previously stored `gg_minimal_depth` object (`gg_md$topvars`) to filter the variables of interest.

The `plot.gg_variable` function call operates in the `gg_variable` object. We pass it the list of variables of interest (`xvar`) and request a single panel (`panel=TRUE`) to display the figures. By default, the `plot.gg_variable` function returns a list of `ggplot` objects, one figure for each variable named in `xvar` argument. The next three arguments are passed to internal `ggplot` plotting routines. The `se` and `span` arguments are used to modify the internal call to `ggplot2::geom_smooth` for fitting smooth lines to the data. The `alpha` argument lightens the coloring points in the `ggplot2::geom_point` call, making it easier to see point over plotting. We also demonstrate modification of the plot labels using the `ggplot2::labs` function.

```
R> # Create the variable dependence object from the random forest
R> gg_v <- gg_variable(rfsrc_Boston)
R>
R> # We want the top ranked minimal depth variables only,
R> # plotted in minimal depth rank order.
R> xvar <- gg_md$topvars
R>
R> # plot the variable list in a single panel plot
R> plot(gg_v, xvar=xvar, panel=TRUE)+
+    labs(y=st.labs["medv"], x="")
```

This figure looks very similar to the EDA (Section 2.1) figure, although with transposed axis as we plot the response variable on the y-axis. The closer the panels match, the better the RF prediction. The panels are sorted to match the order of variables in the `xvar` argument and include a smooth loess line (Cleveland 1981; Cleveland and Devlin 1988), with 95% shaded confidence band, to indicates the trend of the prediction dependence over the covariate values.

There is not a convenient method to panel scatter plots and boxplots together, so we recommend creating panel plots for each variable type separately. The Boston housing data does contain a single categorical variable, the Charles river logical variable. Variable dependence plots for categorical variables are constructed using boxplots to show the distribution of the predictions within each category. Although the Charles river variable has the lowest importance scores in both VIMP and minimal depth measures, we include the variable dependence plot as an example of categorical variable dependence.

```
R> plot(gg_v, xvar="chas", alpha=.4)+
+    labs(y=st.labs["medv"])
R>
R> # , points=FALSE, se=FALSE, notch=TRUE
```

The figure shows that most housing tracts do not border the Charles river (`chas=FALSE`), and comparing the distributions of the predicted median housing values indicates no significant
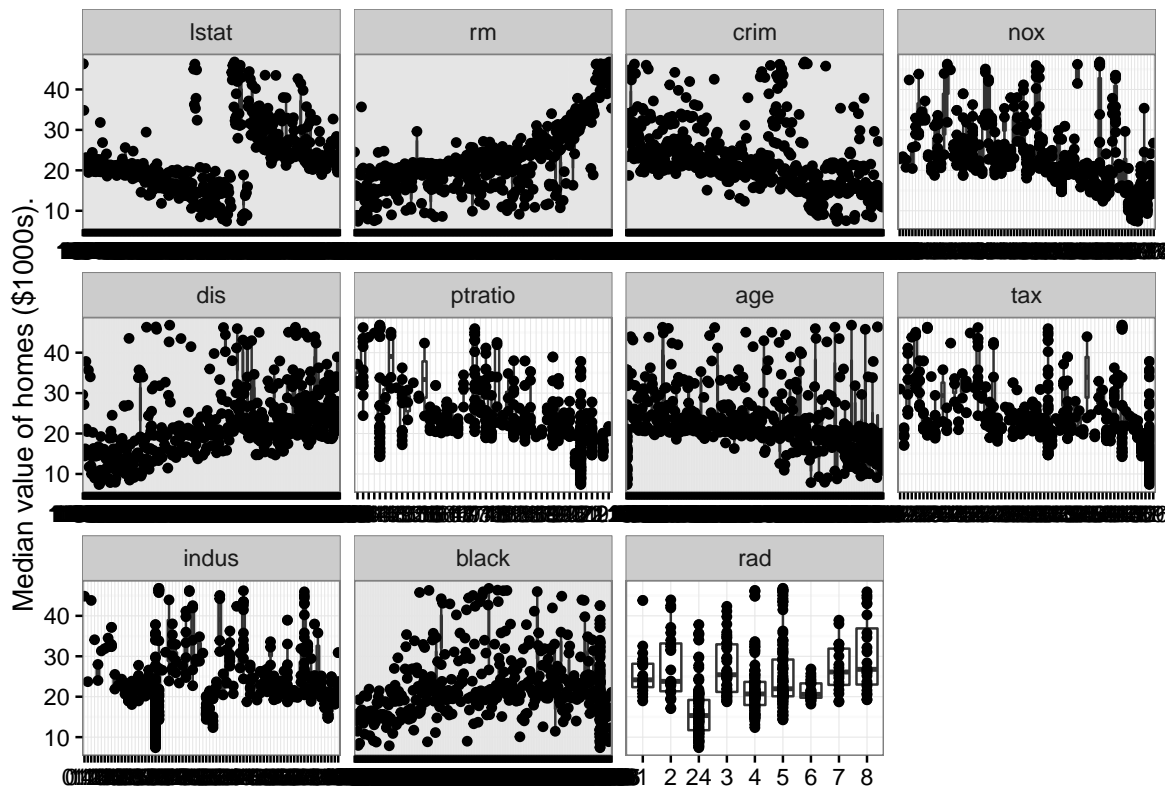
Figure 7: Variable dependence plot. Individual case predictions are marked with points. Loess smooth curve indicates the trend as the variables increase with shaded 95% confidence band.
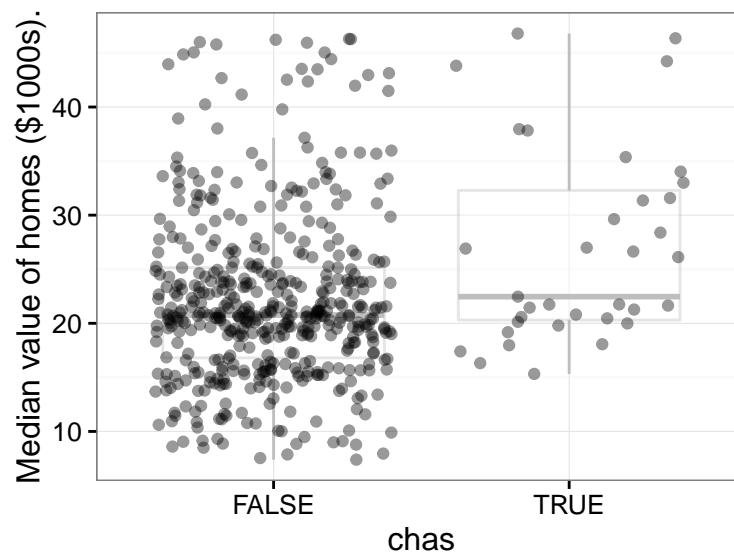

Figure 8: Variable dependence for Charles River logical variable.

difference in home values. This reinforces the findings in both VIMP and Minimal depth, the Charles river variable has very little impact on the forest prediction of median home values.

5.2. Partial Dependence.

Partial variable dependence plots are a risk adjusted alternative to variable dependence. Partial plots are generated by integrating out the effects of all variables beside the covariate of interest. Partial dependence data are constructed by selecting points evenly spaced along the distribution of the X variable of interest. For each value $(X = x)$, we calculate the average RF prediction over all other covariates in X by

$$\tilde{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x, x_{i,o}),$$

where $\hat{f}$ is the predicted response from the random forest and $x_{i,o}$ is the value for all other covariates other than $X = x$ for the observation i (Friedman 2000). Essentially, we average a set of predictions for each observation in the training set at the value of $X = x$. We repeating the process for a sequence of $X = x$ values to generate the estimated points to create a partial dependence plot.

Partial plots are another computationally intensive analysis, especially when there are a large number of observations. We again turn to our data caching strategy here. The default parameters for the `randomForestSRC::plot.variable` function generate partial dependence estimates at `npts=25` points (the default value) along the variable of interest. For each point of interest, the `plot.variable` function averages `n` response predictions. This is repeated for each of the variables of interest and the results are returned for later analysis.

```
R> # Load the data, from the call:
R> # partial_Boston <- plot.variable(rfsrc_Boston,
R> #                                  xvar=gg_md$topvars,
R> #                                  partial=TRUE, sorted=FALSE,
R> #                                  show.plots = FALSE )
R> data(partial_Boston)
R>
R> # generate a list of gg_partial objects, one per xvar.
R> gg_p <- gg_partial(partial_Boston)
R>
R> # plot the variable list in a single panel plot
R> plot(gg_p, panel=TRUE) +  #xvar=xvar, se=FALSE
+   labs(y=st.labs["medv"], x="") +
+   geom_smooth(se=FALSE)
```

We again order the panels by minimal depth ranking. We see again how the `lstat` and `rm` variables are strongly related to the median value response, making the partial dependence of the remaining variables look flat. We also see strong nonlinearity of these two variables. The `lstat` variable looks rather quadratic, while the `rm` shape is more complex.

We could stop here, indicating that the RF analysis has found these ten variables to be important in predicting the median home values. That increasing `lstat` (percentage population of
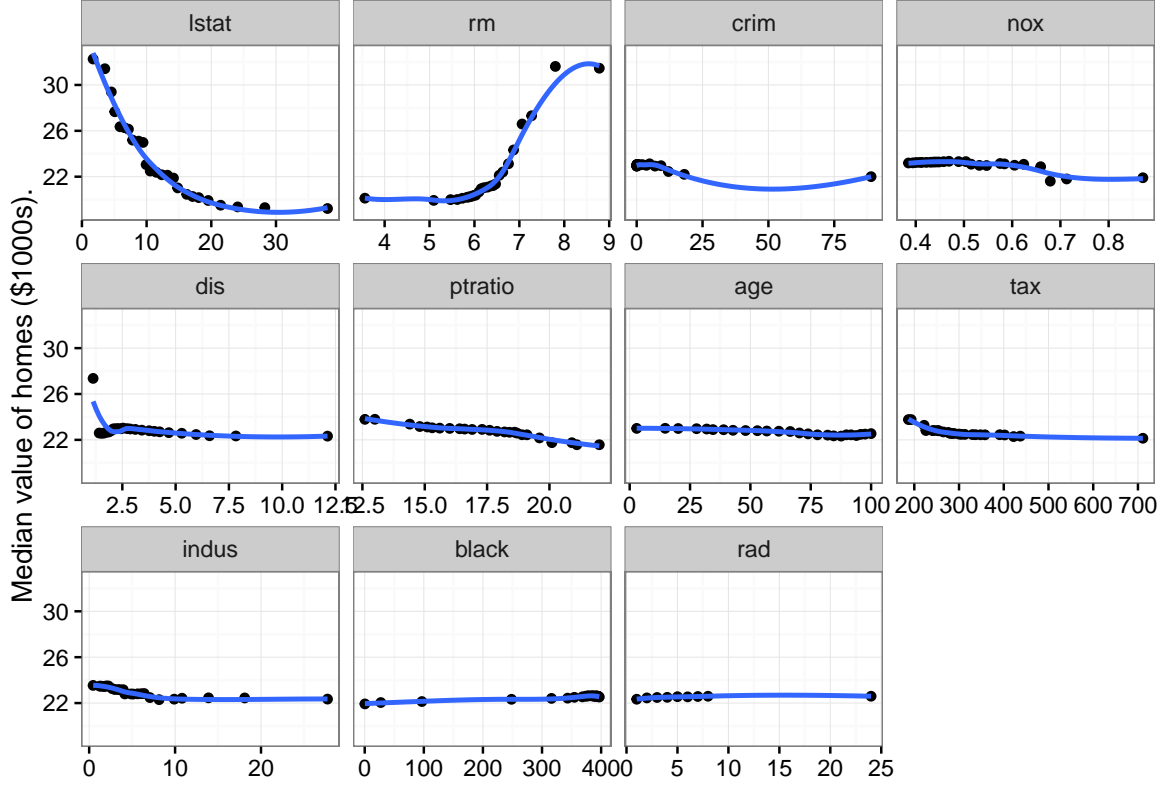
Figure 9: Partial dependence panels. Risk adjusted variable dependence for variables in minimal depth rank order.

lower status) values are associated with decreasing median home values (`medv`) and increasing 'rm > 6' (number of rooms > 6) are associated with increasing median home values. However, we may also be interested in investigating how these variables work together to help improve the random forest prediction of median home values.

6. Variable Interactions

Using the different variable dependence measures, it is also possible to calculate measures of pairwise interactions among variables. Recall that minimal depth measure is defined by averaging the tree depth of variable i relative to the root node. To detect interactions, this calculation can be modified to measure the minimal depth of a variable j with respect to the maximal subtree for variable i (Ishwaran *et al.* 2010; Ishwaran, Kogalur, Chen, and Minn 2011).

The `randomForestSRC::find.interaction` function traverses the forest, calculating all pairwise minimal depth interactions, and returns a $p \times p$ matrix of interaction measures. For each row, the diagonal terms are are related to the minimal depth relative to the root node, though normalized to minimize scaling issues. Each off diagonal minimal depth term is relative to the diagonal term on that row. Small values indicate that an off diagonal term typically splits close to the diagonal term, indicating an forest split proximity of the two variables.

The `gg_interaction` function wraps the `find.interaction` matrix for use with the provided plot and print functions. The `xvar` argument indicates which variables we're interested in looking at. We again use the cache strategy, and collect the figures together using the `panel=TRUE` option.

```
R> # Load the data, from the call:
R> # interaction_Boston <- find.interactions(rfsrc_Boston)
R> data(interaction_Boston)
R>
R> # Plot the results in a single panel.
R> plot(gg_interaction(interaction_Boston),
+       xvar=gg_md$topvars, panel=TRUE)
```
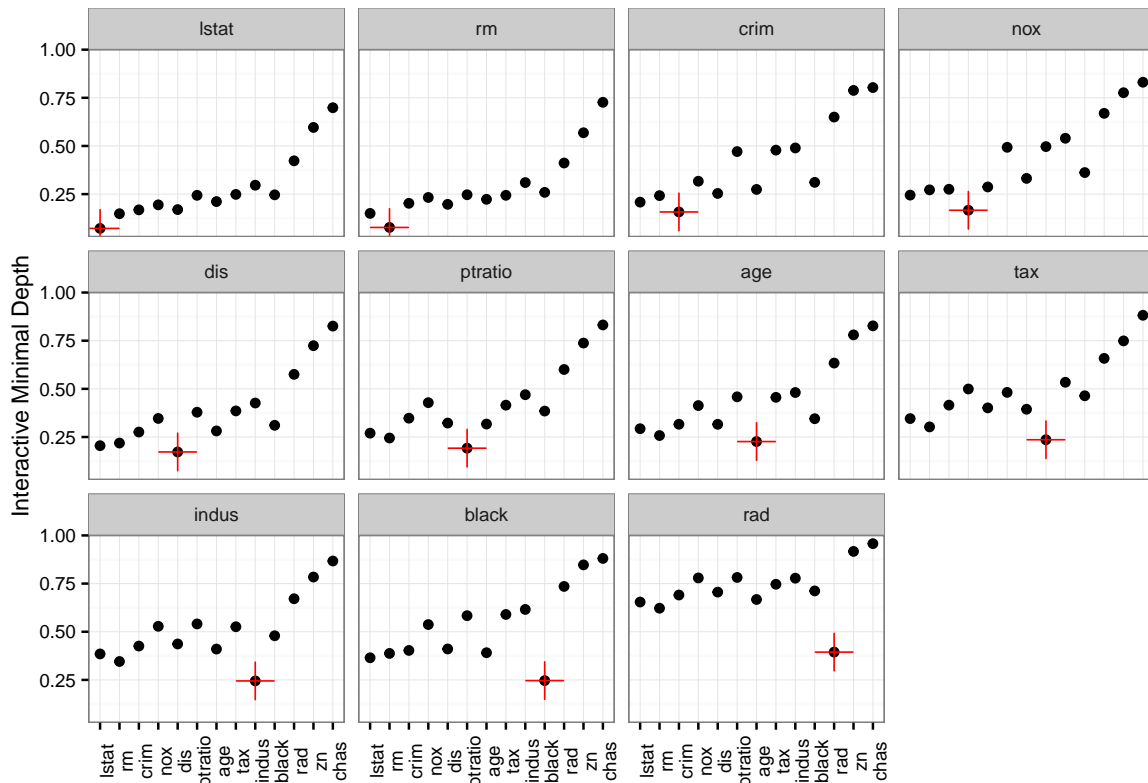

Figure 10: Minimal depth variable interactions. Reference variables are marked with red cross in each panel. Higher values indicate lower interactivity with reference variable.

The `gg_interaction` figure plots the interactions for the target variable (shown in the red cross) with interaction scores for all remaining variables. We expect the covariate with lowest minimal depth (`lstat`) to be associated with almost all other variables, as it typically splits close to the root node, so viewed alone it may not be as informative as looking at a collection of interactive depth plots. Scanning across the panels, we see each successive target depth increasing, as expected. We also see the interactive variables increasing with increasing target depth. Of interest here is the interaction of `lstat` with the `rm` variable shown in the `rm` panel.

Aside from these being the strongest variables by both measures, this interactive measure indicates the strongest connection between variables.

7. Coplots

Conditioning plots (coplots) ([Chambers 1992](); [Cleveland 1993]()) are a powerful visualization tool to efficiently study how a response depends on two or more variables ([Cleveland 1993]()). The method allows us to view data by grouping observations on some conditional membership. The simplest example involves a categorical variable, where we plot our data conditional on class membership, for instance on the Charles river logical variable. We can view a coplot as a stratified variable dependence plot, indicating trends in the RF prediction results within panels of group membership.

Conditional membership with a continuous variable requires stratification at some level. Often we can make these stratification along some feature of the variable, for instance a variable with integer values, or 5 or 10 year age group cohorts. However in the variables of interest in our Boston housing example, we have no "logical" stratification indications. Therefore we will arbitrarily stratify our variables into 6 groups of roughly equal population size using the `quantile_pts` function. We pass the break points located by `quantile_pts` to the `cut` function to create grouping intervals, which we can then add to the `gg_variable` object before plotting with the `plot.gg_variable` function. The simple modification to convert variable dependence plots into condition variable dependence plots is to use the `ggplot2::facet_wrap` command to generate a panel for each grouping interval.

We start by examining the predicted median home value as a function of `lstat` conditional on membership within 6 groups of `rm` "intervals".

```
R> # Find the rm variable points to create 6 intervals of roughly
R> # equal size population
R> rm_pts <- quantile_pts(rfsrc_Boston$xvar$rm, groups=6,
+                        intervals=TRUE)
R>
R> # Pass these variable points to create the 6 (factor) intervals
R> rm_grp <- cut(rfsrc_Boston$xvar$rm, breaks=rm_pts)
R>
R> # Append the group factor to the gg_variable object
R> gg_v$rm_grp <- rm_grp
R>
R> # Modify the labels for descriptive panel titles
R> levels(gg_v$rm_grp) <- paste("rm in ",
+                        levels(gg_v$rm_grp), sep="")
R>
R> # Create a variable dependence (co)plot,
R> # faceted on group membership.
R> plot(gg_v, xvar = "lstat", alpha = .5)+
+   #  method = "loess", span=1.5, se = FALSE) +
+   labs(y = st.labs["medv"], x=st.labs["lstat"]) +
+   theme(legend.position = "none") +
```

```
+    scale_color_brewer(palette = "Set3") +
+    # geom_smooth(se=FALSE) +
+    facet_wrap(~rm_grp)
```
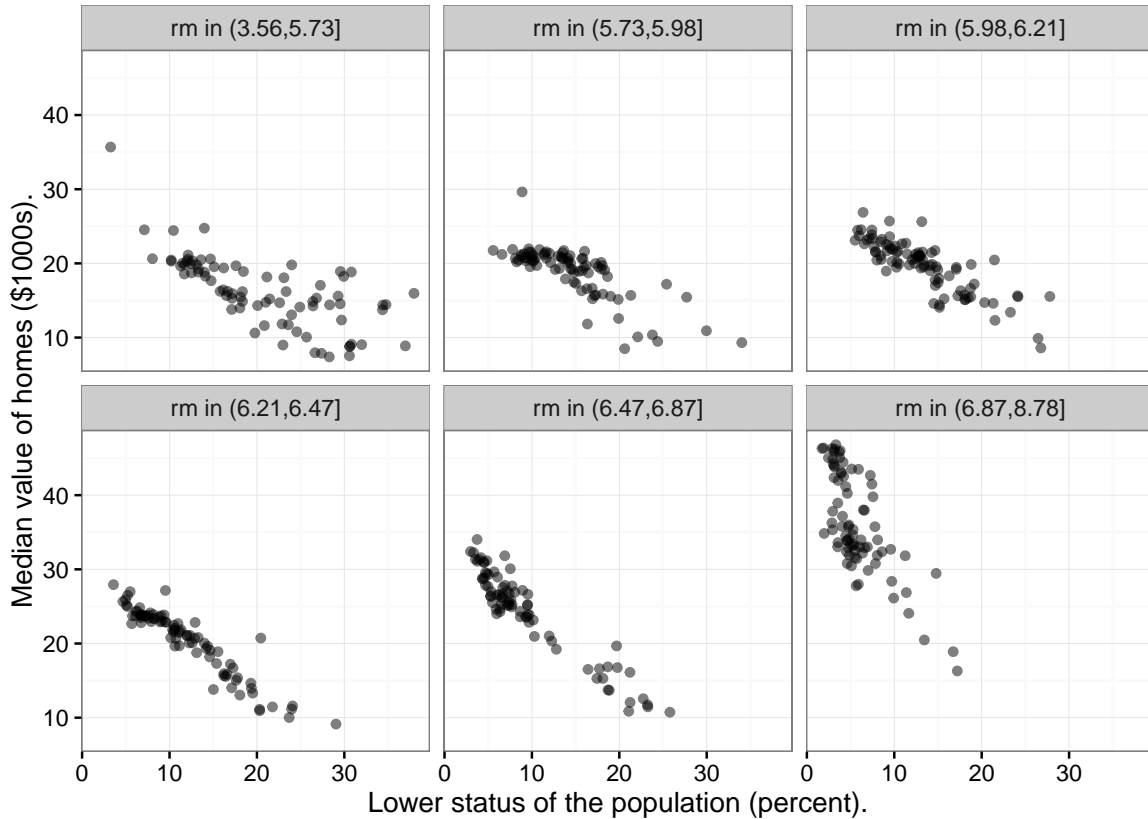

Figure 11: Variable Coplots. Predicted median home values as a function of percentage of lower status population, stratified by average number of rooms groups.

Each point in this figure is the predicted median home value response plotted against `lstat` value conditional on `rm` being on the interval specified. We again use the smooth loess curve to get an idea of the trend within each group. Overall, median values continue to decrease with increasing `lstat`, and increases with increasing `rm`. In addition to trends, we can also examine the conditional distribution of variables. Note that smaller homes (`rm`) in high status (lower `lstat`) neighborhoods still have high predicted median values, and that there are more large homes in the higher status neighborhoods (bottom right panel).

A single coplot gives us a grouped view of a variable (`rm`), along the primary variable dimension (`lstat`). To get a better feel for how the response depends on both variables, it is instructive to look at the complement coplot. We repeat the previous coplot process, predicted median home value as a function of the `rm` variable, conditional on membership within 6 groups `lstat` intervals.

```
R> # Find the lstat variable points to create 6 intervals of roughly
R> # equal size population
R> lstat_pts <- quantile_pts(rfsrc_Boston$xvar$lstat, groups=6, intervals=TRUE)
```

```
R>
R> # Pass these variable points to create the 6 (factor) intervals
R> lstat_grp <- cut(rfsrc_Boston$xvar$lstat, breaks=lstat_pts)
R>
R> # Append the group factor to the gg_variable object
R> gg_v$lstat_grp <- lstat_grp
R>
R> # Modify the labels for descriptive panel titles
R> levels(gg_v$lstat_grp) <- paste("lstat in ", levels(gg_v$lstat_grp), " (%)",sep="")
R>
R> # Create a variable dependence (co)plot, faceted on group membership.
R> plot(gg_v, xvar = "rm", alpha = .5)+
+      #method = "loess", span=1.5, , se = FALSE) +
+    labs(y = st.labs["medv"], x=st.labs["rm"]) +
+    theme(legend.position = "none") +
+    scale_color_brewer(palette = "Set3") +
+  # geom_smooth() +
+    #scale_shape_manual(values = event.marks, labels = event.labels)+
+    facet_wrap(~lstat_grp)
```
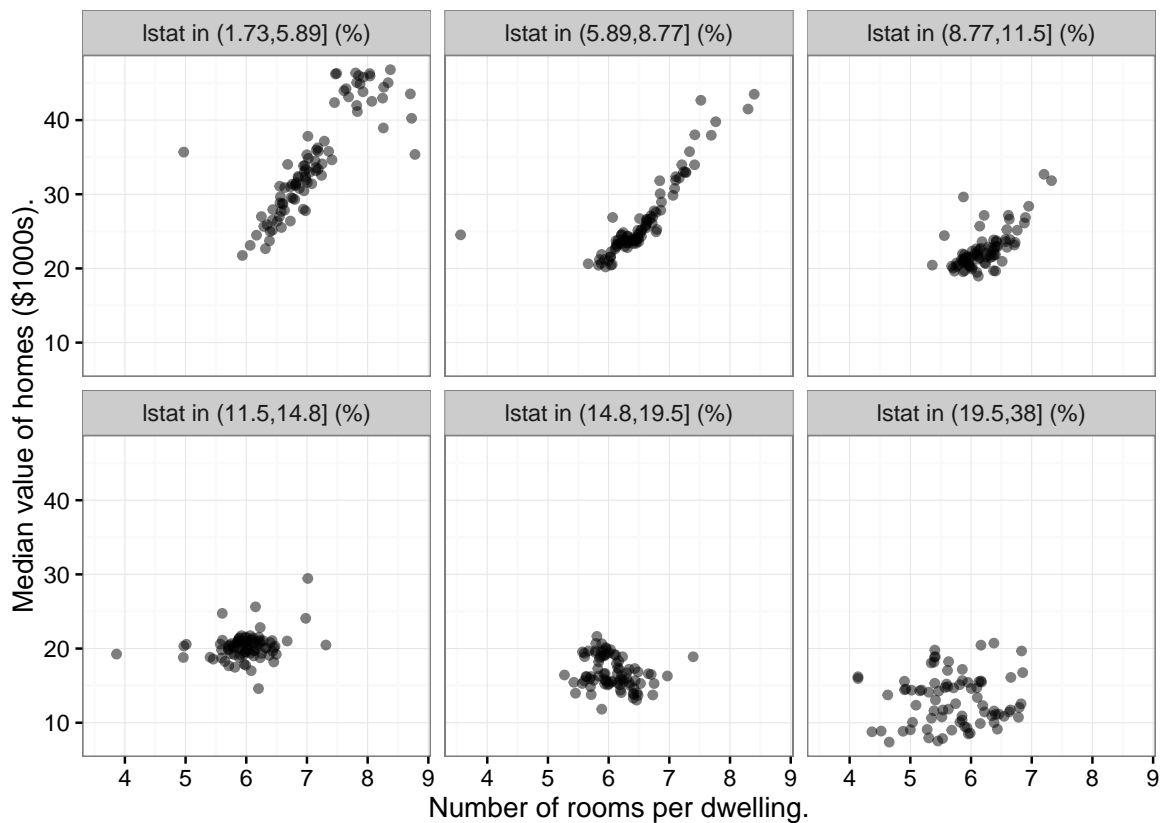

Figure 12: Variable Coplots. Predicted median home value as a function of average number of rooms, stratified by percentage of lower status groups.

We get similar information from this view, predicted median home values decrease with increasing `lstat` percentage and decreasing `rm`. However viewed together we get a better sense of how the `lstat` and `rm` variables work together (interact) in the median value prediction.

Note that typically Cleveland (1993) conditional plots for continuous variables included overlapping intervals along the grouped variable. We chose to use mutually exclusive continuous variable intervals for multiple reasons:

- Simplicity - We can create the coplot figures directly from the `gg_variable` object by adding a conditional group column directly to the object.

- Interpretability - We find it easier to interpret and compare the panels if each observation is only in a single panel.

- Clarity - We prefer using more space for the data portion of the figures than typically displayed in the `coplot` function available in base R, which require the bar plot to present the overlapping segments.

It is still possible to augment the `gg_variable` to include overlapping conditional membership with continuous variables by duplicating rows of the object, and setting the correct conditional group membership. The `plot.gg_variable` function recipe above could then be used to generate the panel plot, with panels ordered according to the factor levels of the grouping variable. We leave this as an exercise for the reader.

7.1. Partial dependence coplots

By characterizing conditional plots as stratified variable dependence plots, the next logical step would be to generate an analogous conditional partial dependence plot. The process is similar to variable dependence coplots, first determine conditional group membership, then calculate the partial dependence estimates on each subgroup using the `randomForestSRC::plot.variable` function with a the `subset` argument for each grouped interval. The `ggRandomForests::gg_partial_coplot` function is a wrapper for generating a conditional partial dependence data object. Given a random forest (`randomForestSRC::rfsrc` object) and a `groups` vector for conditioning the training data set observations, `gg_partial_coplot` calls the `randomForestSRC::plot.variable` function for a set of training set observations conditional on `groups` membership. The function returns a `gg_partial_coplot` object, a sub class of the `gg_partial` object, which can be plotted with the `plot.gg_partial` function.

The following code block will generate the data object for creating partial dependence coplot of the predicted median home value as a function of `lstat` conditional on membership within the 6 groups of `rm` "intervals" that we examined in the previous section.

```
R> partial_coplot_Boston <- gg_partial_coplot(rfsrc_Boston, xvar="lstat",
+                                              groups=rm_grp,
+                                              show.plots=FALSE)
```

Since the `gg_partial_coplot` makes a call to `randomForestSRC::plot.variable` for each group (6) in the conditioning set, we again resort to the data caching strategy, and load the stored result data from the `ggRandomForests` package. We modify the legend label to

indicate we're working with groups of the "Room" variable, and use the `palette="Set1"` from the **RColorBrewer** package (Neuwirth 2014) to choose a nice color theme for displaying the six curves.

```
R> # Load the stored partial coplot data.
R> data(partial_coplot_Boston)
R>
R> # # Partial coplot
R> # plot(partial_coplot_Boston) + ## Looks like a dangling or missing '+' characer in the
R> ggplot(partial_coplot_Boston, aes(x=lstat, y=yhat, col=group, shape=group))+
+    geom_point()+
+    geom_smooth(se=FALSE, alpha=.25)+
+    labs(x=st.labs["lstat"], y=st.labs["medv"],
+         color="Room", shape="Room")+
+    scale_color_brewer(palette="Set1")
```
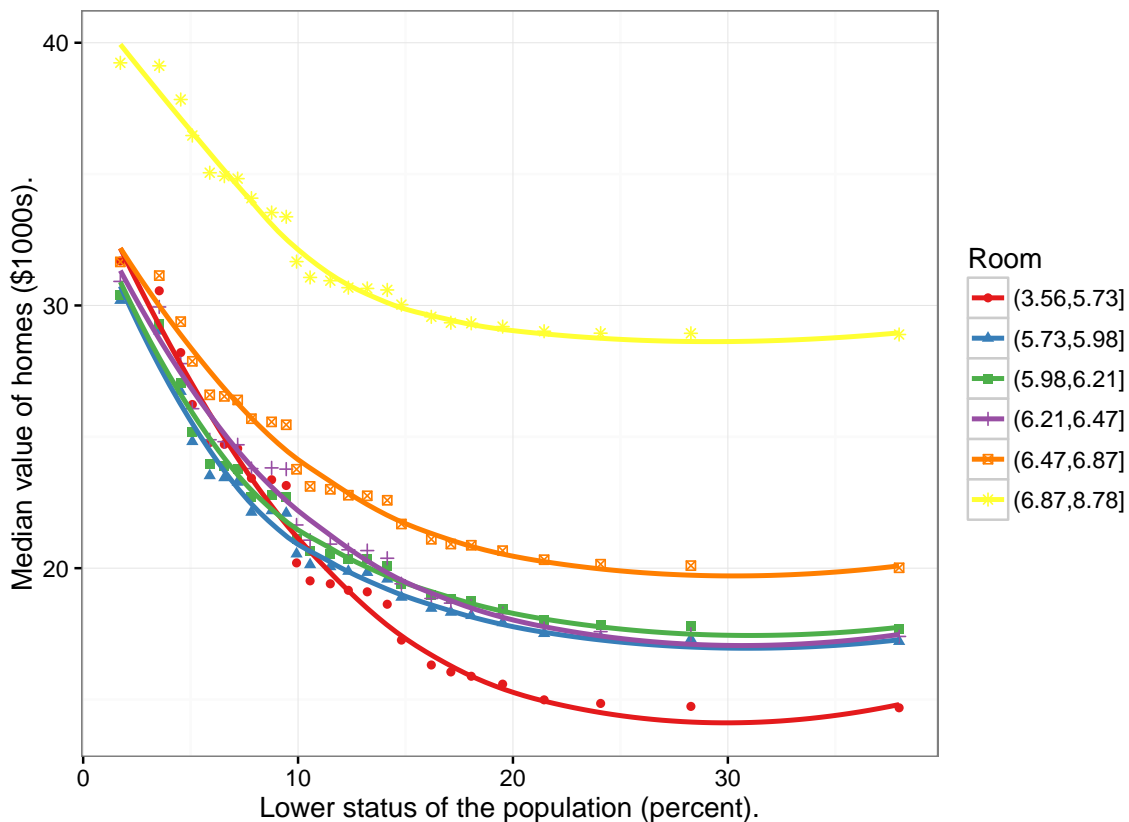

Figure 13: Partial Coplots. Risk adjusted predicted median value as a function of Lower Status, conditional on groups of average number of rooms.

Unlike variable dependence coplots, we do not need to use a panel format for partial dependence coplots because we are looking risk adjusted estimates (points) instead of population estimates. The figure has a loess curve through the point estimates conditional on the `rm` interval groupings. The figure again indicates that larger homes (`rm` from 6.87 and up, shown in

yellow) have a higher median value then the others. In neighborhoods with higher `lstat` percentage, the Median values decrease with `rm` until it stabilizes from the intervals between 5.73 and 6.47, then decreases again for values smaller than 5.73. In lower `lstat` neighborhoods, the effect of smaller `rm` is not as noticeable.

We can view the partial coplot curves as slices along a surface viewed into the page, either along increasing or decreasing `rm` values. This is made more difficult by our choice to select groups of similar population size, as the curves are not evenly spaced along the `rm` variable. We return to this problem in the next section.

We also construct the complement view, for partial dependence coplot of the predicted median home value as a function of `rm` conditional on membership within the 6 groups of `lstat` "intervals", and cache the following `gg_partial_coplot` data call, and plot the results with the `plot.gg_variable` call:

```
R> partial_coplot_Boston2 <- gg_partial_coplot(rfsrc_Boston, xvar="rm",
+                                               groups=lstat_grp,
+                                               show.plots=FALSE)


R> # Load the stored partial coplot data.
R> data(partial_coplot_Boston2)
R>
R> # Partial coplot
R> #plot(partial_coplot_Boston2)+ ## again plot.gg_partial_coplot
R> ggplot(partial_coplot_Boston, aes(x=lstat, y=yhat, col=group, shape=group))+
+   geom_point()+
+   geom_smooth(se=FALSE)+
+   labs(x=st.labs["rm"], y=st.labs["medv"],
+        color="Lower Status", shape="Lower Status")+
+   scale_color_brewer(palette="Set1")
```

This figure indicates that the median home value does not change much until the `rm` increases above 6.5, then flattens again above 8, regardless of the `lstat` value. This agrees well with the `rm` partial plot(Section 5.2) shown earlier. Again, care must be taken in interpreting the even spacing of these curves along the percentage of `lstat` groupings, as again, we chose these groups to have similar sized populations, not to be evenly spaced along the `lstat` variable.

8. Partial plot surfaces

Visualizing two dimensional projections of three dimensional data is difficult, though there are tools available to make the data more understandable. To make the interplay of lower status and average room size a bit more understandable, we will generate a contour partial plot of the median home values. We could generate this figure with the coplot data we already have, but the resolution would be a bit strange. To generate the plot of `lstat` conditional on `rm` groupings, we would end up with contours over a grid of `lstat`= 25× `rm`= 6, for the alternative `rm` conditional on `lstat` groups, we'd have the transpose grid of `lstat`= 6× `rm`= 25.
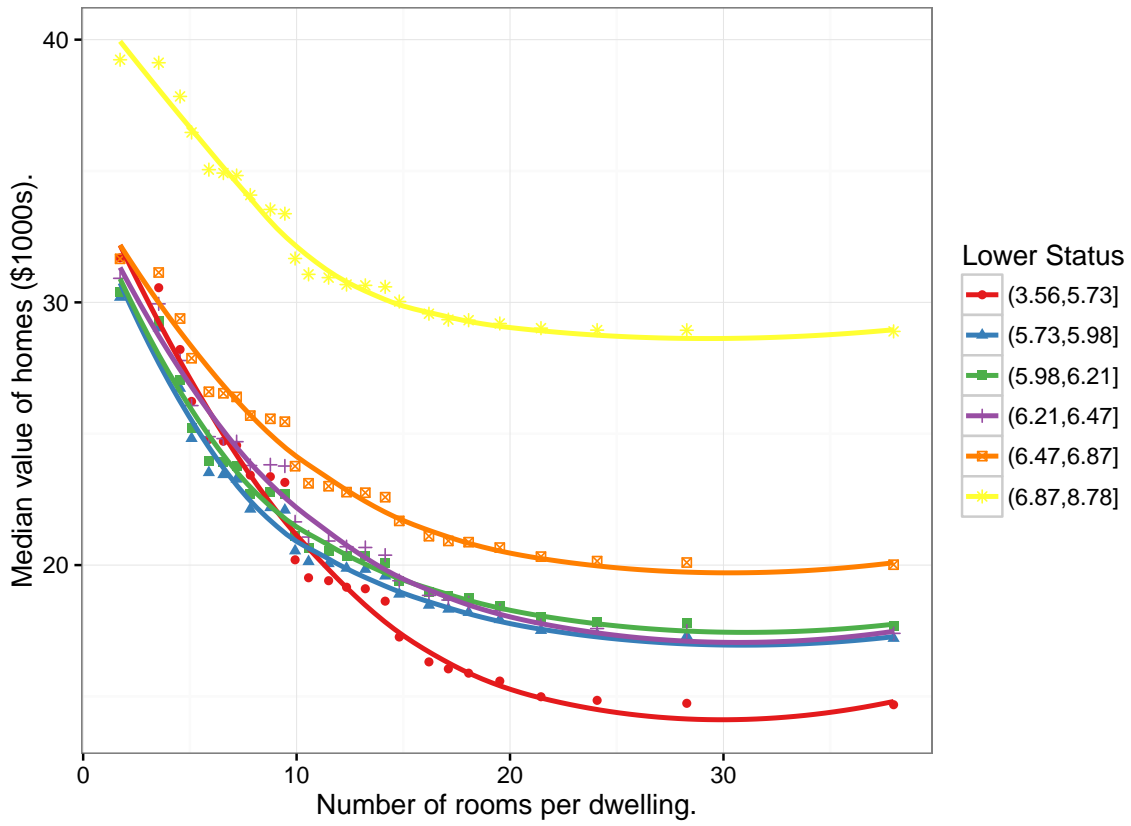
Figure 14: Partial Coplots. Risk adjusted predicted median value as a function of average number of rooms, conditional on groups of percentage of lower status population.

Since we are already using the data caching strategy, we will generate another set of `gg_partial` objects with increased resolution in both the `lstat` and `rm` dimensions. For this exercise, we will find 50 points evenly spaced along the `rm` variable values, and generate a partial plot curve for each point. For these partial plots, we will evaluate the risk adjusted median home value over 'npts=50' points along the `lstat` variable. This code block finds 50 `rm` values evenly spaced along the distribution of `rm`.

```
R> # Find the quantile points to create 50 cut points
R> rm_pts <- quantile_pts(rfsrc_Boston$xvar$rm, groups=50)
```

We use the following data call to generate the partial plots with the `randomForestSRC::plot.variable` call. Within the lapply call, we use scope to modify the value of the `rm` variable within the `rfsrc_Boston` training set. Since all values in the training set are the same, the averaged value of `rm` places each partial plot curve at a specific value of `rm`. This code block took about 20 minutes to run on a quad core Mac Air using a single processor.

The cached data is stored in the `partial_Boston_surf` data set in the `ggRandomForests` package. The data set is a `list` of 50 `plot.variable` objects. This code block loads the data, converts the `plot.variable` objects to `gg_partial` objects, attaches numeric values for the `rm` variable, and generates the contour plot.

```
R> # Generate the gg_partial_coplot data object
R> system.time(partial_Boston_surf <- lapply(rm_pts, function(ct){
+    rfsrc_Boston$xvar$rm <- ct
+    plot.variable(rfsrc_Boston, xvar = "lstat", time = 1,
+                  npts = 50, show.plots = FALSE,
+                  partial = TRUE)
+ }))
R> #      user   system  elapsed
R> # 1109.641    76.516 1199.732


R> # Load the stored partial coplot data.
R> data(partial_Boston_surf)
R>
R> # Instead of groups, we want the raw rm point values,
R> # To make the dimensions match, we need to repeat the values
R> # for each of the 50 points in the lstat direction
R> rm.tmp <- do.call(c,lapply(rm_pts,
+                           function(grp){rep(grp, 50)}))
R>
R> # Convert the list of plot.variable output to
R> partial_surf <- do.call(rbind,lapply(partial_Boston_surf, gg_partial))
R>
R> # attach the data to the gg_partial_coplot
R> partial_surf$rm <- rm.tmp
R>
R> # ggplot2 contour plot of x, y and z data.
R> ggplot(partial_surf, aes(x=lstat, y=rm, z=yhat))+
+    stat_contour(aes(colour = ..level..), binwidth = .5)+
+    labs(x=st.labs["lstat"], y=st.labs["rm"],
+         color="Median Home Values")+
+    scale_colour_gradientn(colours=topo.colors(10))
```

The contours are generated over the raw `gg_partial` estimation points, not smooth curves as shown in the partial plot (Section 5.2) and partial coplot (Section 7.1) figures previously. Contour lines, like topographic maps, are concentrated where the slope of the surface is large. We use color to indicate the direction of the contour lines, so that lower median home values are concentrated in the lower right hand corner, and the values increase along the diagonal toward the upper right. The close contour lines indicate some thing like a step in values at 7 and 7.5 rooms, and at 5, 10 and 15% lstat.

Contour plots are still a little difficult to interpret. However, we can also generate a surface with this data using the **plot3D** package (Soetaert 2014) and the `plot3D::surf3D` function. Viewed in 3D, a surface can help to better understand what the contour lines are showing us.

```
R> # Modify the figure margins to make the figure larger
R> par(mai = c(0,0,0,0))
R>
```
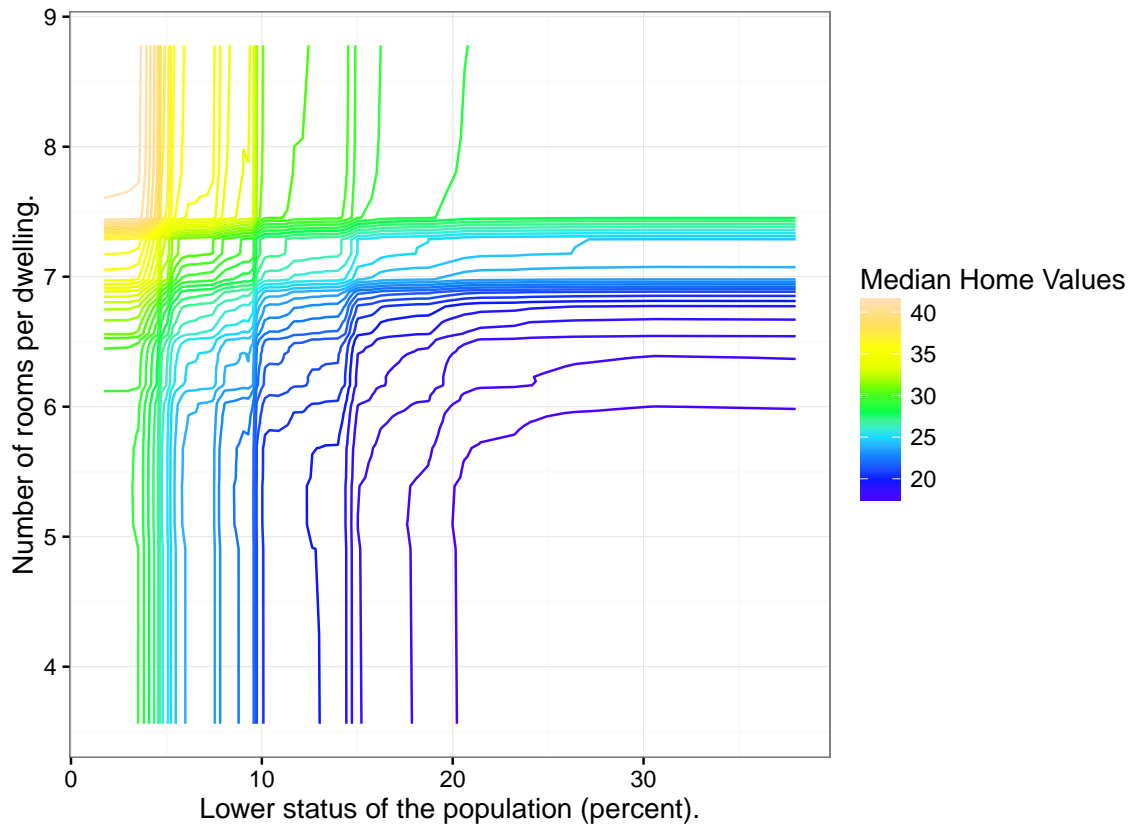
Figure 15: Partial coplot contour plot. Contours of median home value along the lstat/rm plane.

```
R> # Transform the gg_partial_coplot object into a list of three named matrices
R> # for surface plotting with plot3D::surf3D
R> suppressWarnings(
+    srf <- surface_matrix(partial_surf, c("lstat", "rm", "yhat"))
+ )
R>
R> # Generate the figure.
R> surf3D(x=srf$x, y=srf$y, z=srf$z, col=topo.colors(10),
+        colkey=FALSE, border = "black", bty="b2",
+        shade = 0.5, expand = 0.5,
+        lighting = TRUE, lphi = -50,
+        xlab="Lower Status", ylab="Average Rooms", zlab="Median Value"
+ )
```

These figures reinforce the previous findings, where lower home values are associated with higher `lstat` percentage, and higher values are associated with larger `rm`. The difference in this figure is we can see how the predicted values change as we move around the map of `lstat` and `rm` combinations. We do still need to be careful though, as partial plots average over values on the surface that are note supported by actual observations.
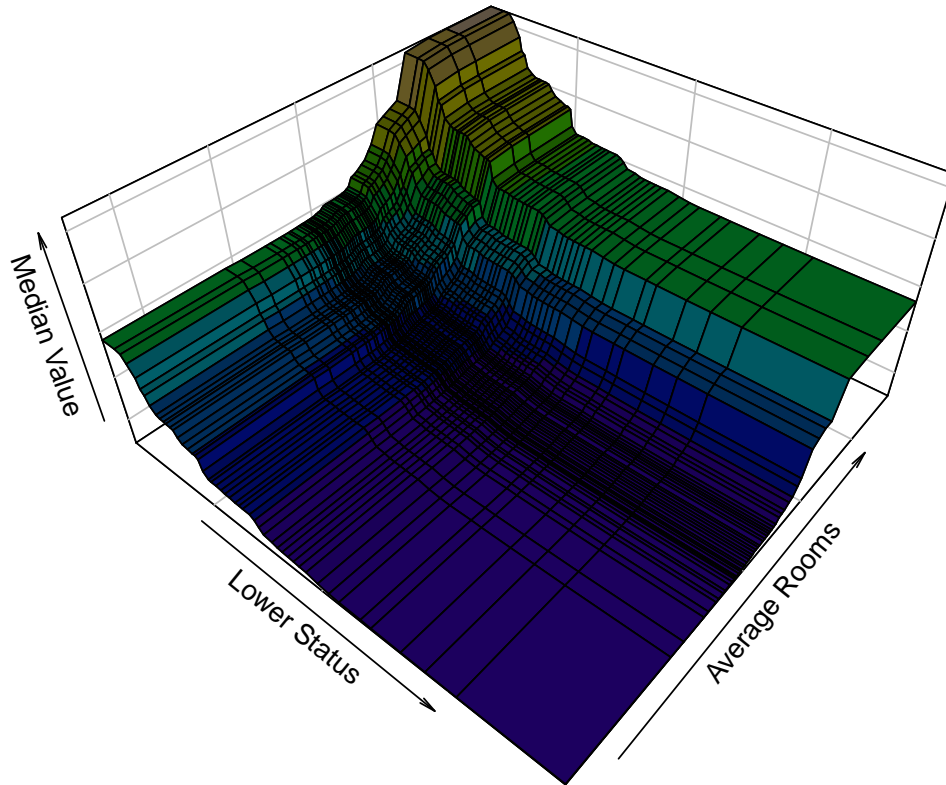
Figure 16: Partial plot surface.

9. Conclusion

In this vignette, we have demonstrated the use of the **ggRandomForests** package to explore a regression random forest built with the **randomForestSRC** package. We have shown how to create a random forest model (Section 3) and determine which variables contribute to the forest prediction accuracy using both VIMP (Section 4.1) and Minimal Depth (Section 4.2) measures. We outlined how to investigate variable associations with the response variable using variable dependence (Section 5.1) and the risk adjusted partial dependence (Section 5.2) plots. We've also explored variable interactions by using pairwise minimal depth interactions (Section 6) and directly viewed these interactions using variable dependence coplots (Section 7) and partial dependence coplots (Section 7.1). Along the way, we've demonstrated the use of additional commands from the **ggplot2** package for modifying and customizing results from **ggRandomForests**.

References

Becker RA, Chambers JM, Wilks AR (1988). *The New S Language.* Wadsworth & Brooks/Cole.

Belsley D, Kuh E, Welsch R (1980). *Regression Diagnostics. Identifying Influential Data and Sources of Collinearity.* John Wiley & Sons, New York.

Breiman L (1996a). "Bagging Predictors." *Machine Learning*, **26**, 123–140.

Breiman L (1996b). "Out–Of–Bag Estimation." *Technical report*, Statistics Department, University of California,Berkeley, CA. 94708. URL `ftp://ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps.Z`.

Breiman L (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32.

Breiman L, Friedman JH, Olshen R, Stone C (1984). *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA.

Chambers JM (1992). *Statistical Models in S.* Wadsworth & Brooks/Cole.

Chang W, Cheng J, Allaire J, Xie Y, McPherson J (2015). **shiny**: *Web Application Framework for R.* R package version 0.11.1, URL `http://CRAN.R-project.org/package=shiny`.

Cleveland WS (1981). "LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression." *The American Statistician*, **35**(1), 54.

Cleveland WS (1993). *Visualizing Data.* Summit Press.

Cleveland WS, Devlin SJ (1988). "Locally-Weighted Regression: An Approach to Regression Analysis by Local Fitting." *Journal of the American Statistical Association*, **83**(403), 596–610.

Efron B, Tibshirani R (1994). *An Introduction to the Bootstrap.* Chapman & Hall/CRC. ISBN 0412042312.

Friedman JH (2000). "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics*, **29**, 1189–1232.

Harrison D, Rubinfeld D (1978). "Hedonic prices and the demand for clean air." *J. Environ. Economics and Management*, **5**, 81âĂŞ102.

Hastie T, Tibshirani R, Friedman JH (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second edition. Springer-Verlag, New York. ISBN 978-0-387-84857-0.

Ishwaran H (2007). "Variable Importance in Binary Regression Trees and Forests." *Electronic Journal of Statistics*, **1**, 519–537.

Ishwaran H, Kogalur UB (2007). "Random Survival Forests for R." *R News*, **7**, 25–31.

Ishwaran H, Kogalur UB (2014). "Random Forests for Survival, Regression and Classification (RF-SRC), R package version 1.6." URL `http://CRAN.R-project.org/package=randomForestSRC`.

Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS (2008). "Random Survival Forests." *The Annals of Applied Statistics*, **2**(3), 841–860.

Ishwaran H, Kogalur UB, Chen X, Minn AJ (2011). "Random Survival Forests for High–Dimensional Data." *Statist. Anal. Data Mining*, **4**, 115–132.

Ishwaran H, Kogalur UB, Gorodeski EZ, Minn AJ, Lauer MS (2010). "High–Dimensional Variable Selection for Survival Data." *J. Amer. Statist. Assoc.*, **105**, 205–217.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." R *News*, **2**(3), 18–22.

Neuwirth E (2014). ***RColorBrewer**: ColorBrewer Palettes.* R package version 1.1-2, URL http://CRAN.R-project.org/package=RColorBrewer.

R Core Team (2014). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Soetaert K (2014). ***plot3D**: Plotting multi-dimensional data.* R package version 1.0-2, URL http://CRAN.R-project.org/package=plot3D.

Tukey JW (1977). *Exploratory Data Analysis.* Pearson.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with* S. Fourth edition. Springer-Verlag, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis.* Springer-Verlag, New York. ISBN 978-0-387-98140-6.

Xie Y (2013). *Dynamic Documents with* R *and **knitr**.* Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1482203530, URL http://yihui.name/knitr/.

Xie Y (2014). "**knitr**: A Comprehensive Tool for Reproducible Research in R." In V Stodden, F Leisch, RD Peng (eds.), *Implementing Reproducible Computational Research.* Chapman and Hall/CRC. ISBN 978-1466561595, URL http://www.crcpress.com/product/isbn/9781466561595.

Xie Y (2015). ***knitr**: A General-Purpose Package for Dynamic Report Generation in* R. R package version 1.9, URL http://yihui.name/knitr/.

Affiliation:

John Ehrlinger
Quantitative Health Sciences
Lerner Research Institute
Cleveland Clinic
9500 Euclid Ave
Cleveland, Ohio 44195
E-mail: john.ehrlinger@gmail.com
URL: http://www.lerner.ccf.org/qhs/people/ehrlinj/