ggRandomForests: Survival with Random Forests

John Ehrlinger
Cleveland Clinic

Jeevanantham Rajeswaran
Cleveland Clinic

Hemant Ishwaran
University of Miami

Udaya B. Kogalur
Kogalur & Company, Inc.

Eugene H. Blackstone
Cleveland Clinic

Abstract

Random Forests (Breiman 2001) (RF) are a fully 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. Random Forests for survival (Ishwaran and Kogalur 2007; Ishwaran, Kogalur, Blackstone, and Lauer 2008) (RF-S) 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, 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 creating and plotting data structures to visually understand random forest models grown in R with the **randomForestSRC** package. The **ggRandomForests** package is structured to extract intermediate data objects from **randomForestSRC** objects and generate figures using the **ggplot2** (Wickham 2009) graphics package.

This document is formatted as a tutorial for using the **randomForestSRC** for building random forests for survival and **ggRandomForests** package for investigating how the forest is constructed. This tutorial uses the Primary Biliary Cirrhosis (PBC) Data from the Mayo Clinic (Fleming and Harrington 1991) available in the **randomForestSRC** package. We use Variable Importance measure (VIMP) (Breiman 2001) as well as Minimal Depth (Ishwaran, Kogalur, Gorodeski, Minn, and Lauer 2010), a property derived from the construction of each tree within the forest, to assess the impact of variables on forest prediction. We will also demonstrate the use of variable dependence plots (Friedman 2000) to aid interpretation RF results in different response settings. We also will investigate interactions between covariates to demonstrate the strength of the Random Forest method in survival settings.

Keywords: random forest, survival, 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 (Ish-

waran and Kogalur 2014, http://CRAN.R-project.org/package=randomForestSRC) for survival, regression and classification forests 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 a survival random forest. In this tutorial, we explore a random forest for survival model constructed for the primary biliary cirrhosis (PBC) of the liver data set (Fleming and Harrington 1991), available in the **randomForestSRC** package. We grow a survival random forest and demonstrate how **ggRandomForests** can be used when determining how the survival response depends on predictive variables within the model. The tutorial demonstrates the design and usage of many of **ggRandomForests** functions and features 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 the combination of R code directly into documents, weaving code, results and figures into dialog 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 R code listed in the remaining code blocks.

```
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("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)]
```

The latest version of 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-Survival", 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 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 *et al.* 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, `http://CRAN.R-project.org/package=ggRandomForests`) 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 (`http://CRAN.R-project.org/package=ggRandomForests`) 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, `http://CRAN.R-project.org/package=ggplot2`).

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`) to generate intermediate **ggRandomForests** data objects. **ggRandomForests** 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 `ggplot` objects to meet users specific needs and requirements.
- We chose to use the **ggplot2** package for our figures for flexibility in modifying the output. Each **ggRandomForests** plot function returns either a single `ggplot` object, or a `list` of `ggplot` objects, allowing the use of additional **ggplot2** functions and/or themes to modify and customize the final figures.

This document is structured as a tutorial for using the **randomForestSRC** package for building and post-processing random survival forest models and using the **ggRandomForests** package for understanding how the forest is constructed. In this tutorial, we will build a random survival forest for the primary biliary cirrhosis (PBC) of the liver data set (Fleming and Harrington 1991), available in the **randomForestSRC** package. We present the data in Section 2 and summarize the analysis of Fleming and Harrington (1991).

In Section 3, we describe how to grow a random survival forest. Random forests are not parsimonious, but use all variables available in the construction of a response predictor. We demonstrate random forest variable selection techniques (Section 4) using Variable Importance (VIMP) (Breiman 2001) in Section 4.1 as well as Minimal Depth (Ishwaran *et al.* 2010) in Section 4.2. We use both methods to assess the impact of variables on forest prediction.

Once we have an idea of which variables we are most interested in, we use variable dependence plots (Friedman 2000) (Section 5) to understand how a variable is related to the response. Marginal variable dependence (Section 5.1) plots 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. Variable dependence plots often show strongly non-linear variable/response relations that are not easily obtained through parametric modeling.

We examine forest variable interactions in Section 6. Using a minimal depth approach, we quantify how closely variables are related within the forest. We then demonstrate the use of marginal dependence and partial dependence (risk adjusted) conditioning plots (coplots) (Chambers 1992; Cleveland 1993) to examine interactions among variables of interest graphically (Section 7).

2. Data Summary: Primary Biliary Cirrhosis (PBC) Data

For this tutorial, we will use data obtained from a Mayo Clinic randomized trial in *primary biliary cirrhosis* of the liver (PBC) conducted between 1974 and 1984. The data consists of 424 PBC patients referred to Mayo Clinic which met eligibility criteria for a randomized placebo controlled trial of the drug D-penicillamine (DPCA). The data is described in (Fleming and Harrington 1991, Chapter 0.2) and a partial likelihood model (Cox proportional hazards) is developed in Chapter 4.4. The `pbc` data set, included in the **randomForestSRC** package, contains 418 observations (Fleming and Harrington 1991, Appendix D). Of these observations, 312 patients participated in the randomized trial.

```
R> data(pbc, package = "randomForestSRC")
```

For this analysis, we modify some of the data for formatting results. Since the data contains about 12 years of follow up, we prefer using `years` instead of `days` survival. We also convert the `age` variable to years, and the `treatment` variable to a factor containing levels of `c("DPCA", "placebo")`. The variable names, type and description are outlined in Table 1.

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 this end, we use **ggplot2** figures with the `facet_wrap` command and create two sets of panel plots, one for categorical variables using histograms (Figure 1), and another of scatter plots for continuous variables (Figure 2). Variables are plotted along a continuous variable on the X-axis, in this case the length of follow up (survival time in `years`). These figures help to find outliers, missing values and other data anomalies within each variable before getting deep into the analysis.

In categorical EDA plots (Figure 1), we are looking for patterns of missing data (grey portion of bars). We often use surgical date for our X-axis variable to look for periods of low enrollment. There is no comparable variable available in the `pbc` data set, so instead we used follow

Variable	Description	Type
years	survival time (years)	numeric
status	event indicator (F = censor, T = death)	logical
treatment	treament (DPCA, Placebo)	factor
age	age in years	numeric
sex	Female	logical
ascites	Asictes	logical
hepatom	Hepatomegaly	logical
spiders	Spiders	logical
edema	edema	factor
bili	serum bilirubin (mg/dl)	numeric
chol	serum cholesterol (mg/dl)	integer
albumin	albumin (gm/dl)	numeric
copper	urine copper (ug/day)	integer
alk	alkaline phosphatase (U/liter)	numeric
sgot	SGOT (U/ml)	numeric
trig	triglicerides (mg/dl)	integer
platelet	platelets per cubic ml/1000	integer
prothrombin	prothrombin time (sec)	numeric
stage	histologic stage	factor

Table 1: `pbc` data descriptions.

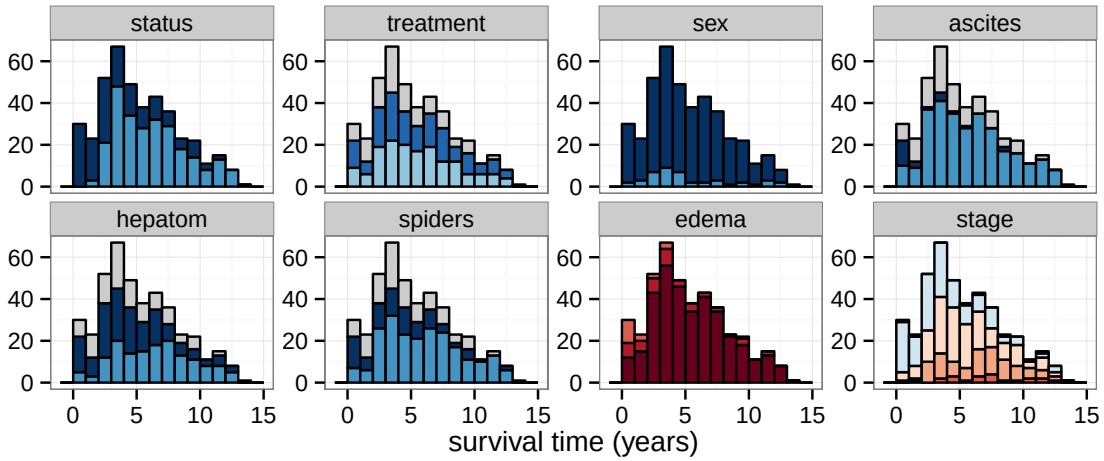

Figure 1: Categorical variable EDA plots. Bars indicate counts within 1 year of followup for each categorical variable. Bars are colored according to the class membership within each variable. Missing values are colored grey.

up time (`years`). Another reasonable choice may have been to use the patient `age` variable. The important quality of the variable is to spread the observations out to aid in finding data anomalies.

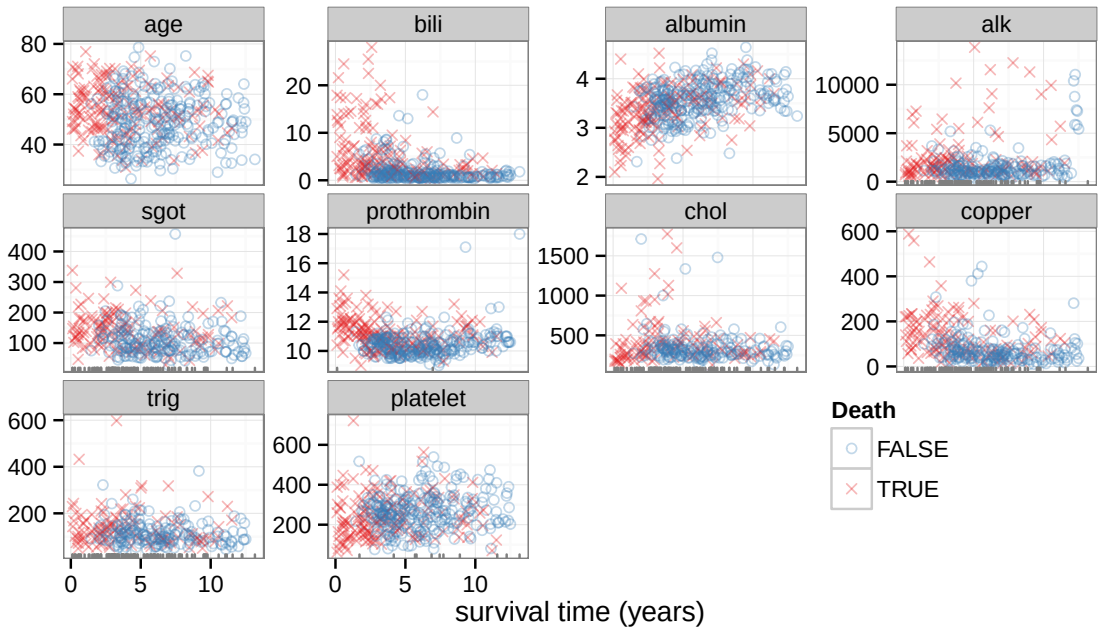

Figure 2: Continuous variable EDA plots. Points indicate variable value against the follow up time in years. Points are colored according to the death event in the `status` variable. Missing values are indicated by the rug marks along the X-axis

In continuous data EDA plots (Figure 2), we look for missingness (rug marks) and extreme

	pbc	pbc.trial
treatment	106	0
ascites	106	0
hepatom	106	0
spiders	106	0
chol	134	28
copper	108	2
alk	106	0
sgot	106	0
trig	136	30
platelet	11	4
prothrombin	2	0
stage	6	0

Table 2: Missing value counts in `pbc` data set.

values. For survival settings, we color and shape the points corresponds to the censoring/event indicator (`status`) variable using a red "x" to indicate an event, and a blue circle to indicate a censored observation.

Both figures indicate quite a bit of missing data. Table 2 details the number of missing values in each variable of the `pbc` data set. Of the 19 variables in the data, 12 have missing values. The `full` column details variables with missing data in the full `pbc` data set, though there are patients that were not randomized into the trial. If we restrict the data to the trial only, most of the missing values are also removed, leaving only 4 variables with missing values. We focus on the 312 observations from the clinical trial for the remainder of this document. We will discuss how **randomForestSRC** handles missing values in Section 3.3.

2.2. Fleming and Harrington (1991) Model Summary (`gg_survival`)

We'll conclude our data set investigation with a summary of Fleming and Harrington (1991) model results from Chapter 4.4. We start by generating Kaplan–Meier (KM) survival estimates comparing the treatment groups of DPCA and placebo. We use the **ggRandomForests** `gg_survival` function to generate these estimates from the data set as follows.

```
R> # Create the trial and test data sets.
R> pbc.trial <- pbc %>% filter(!is.na(treatment))
R> pbc.test <- pbc %>% filter(is.na(treatment))
R>
R> # Create the gg_survival object
R> gg_dta <- gg_survival(interval = "years",
+                        censor = "status",
+                        by = "treatment",
+                        data = pbc.trial,
+                        conf.int = .95)
```

The code block first reduces the `pbc.trial` data set to only include observations from the

clinical trial, and sorts the remainder into the `pbc.test` data set for later use. The **ggRandomForests** package is designed to use a two step process in figure generation. The first step is data generation, where we store a `gg_survival` data object in the `gg_dta` object. The `gg_survival` function uses the `data` set, follow up `interval`, `censor` indicator and an optional grouping argument (`by`). By default `gg_survival` also calculates 95% confidence band, which we can control with the `conf.int` argument.

In the figure generation step, we use the **ggRandomForests** plot routine `plot.gg_survival` as shown in the following code block. The plot function uses the data object to plot the survival estimate curves for each group and corresponding confidence interval ribbons. We have used additional **ggplot2** commands to modify the axis and legend labels (`labs`), the legend location (`theme`) and control the plot range of the y-axis (`coord_cartesian`) for this figure.

```
R> plot(gg_dta) +
+   labs(y = "Survival Probability",
+        x = "Observation Time (years)",
+        color = "Treatment", fill = "Treatment")+
+   theme(legend.position = c(.2,.2))+
+   coord_cartesian(y = c(0,1.01))
```

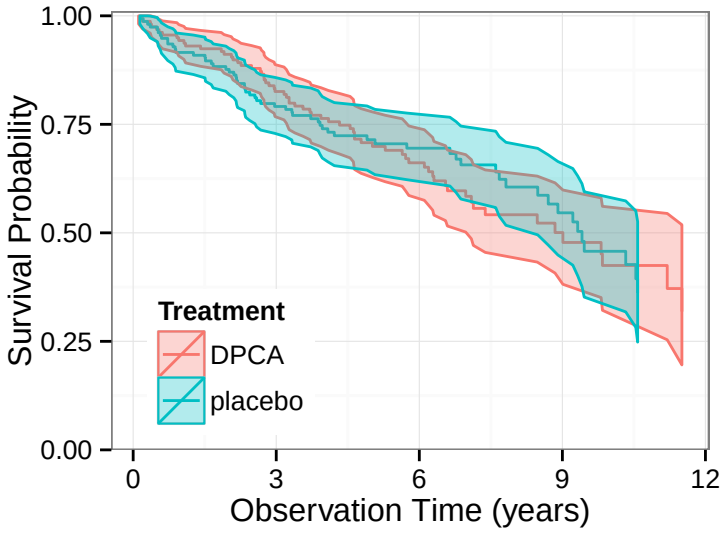

Figure 3: Kaplan–Meier `pbc` data survival estimates comparing the `DPCA` treatment (red) with `placebo` (blue). Median survival with shaded 95% confidence band.

The `gg_survival` plot of Figure 3 is analogous to Fleming and Harrington (1991) Figure 0.2.3 and Figure 4.4.1, showing there is little difference between the treatment and control groups.

The `gg_survival` function generates a variety of time-to-event estimates, including the cumulative hazard. The follow code block creates a cumulative hazard plot (Fleming and Harrington 1991, Figure 0.2.1) in Figure 4. The red `DPCA` line is equivalent to Figure 0.2.1, and we add the cumulative hazard estimates for the `placebo` population in blue.

```
R> plot(gg_dta, type="cum_haz") +
+   labs(y = "Cumulative Hazard",
+        x = "Observation Time (years)",
+        color = "Treatment", fill = "Treatment")+
+   theme(legend.position = c(.2,.8))
```

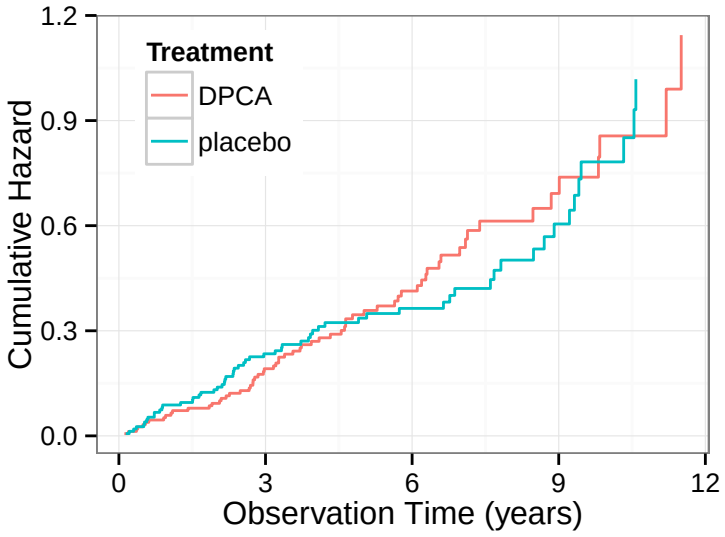

Figure 4: Kaplan–Meier pbc data cumulative hazard estimates comparing the DPCA treatment (red) with placebo (blue).

In Figure 3, we demonstrated grouping on the categorical variable (`treatment`). To demonstrate plotting grouped survival on a continuous variable, we examine KM estimates of survival within stratified groups of bilirubin measures. The groupings are obtained directly from Fleming and Harrington (1991) Figure 4.4.2, where they presented univariate model results of predicting survival on a function of bilirubin.

We set up the `bili` groups on a temporary data set (`pbc.bili`) using the `cut` function with intervals matching the reference figure. For this example we combine the data generation and plot steps into a single line of code. The `error` argument of the `plot.gg_survival` is used to control display of the confidence bands. We suppress the intervals for this figure with `error = "none"` and again modify the plot display with **ggplot2** commands to generate Figure 5.

```
R> # Duplicate the trial data and group by bilirubin values
R> pbc.bili <- pbc.trial
R> pbc.bili$bili_grp <- cut(pbc.bili$bili, breaks = c(0, .8, 1.3, 3.4, 29))
R>
R> plot(gg_survival(interval = "years",censor = "status",
+                   by = "bili_grp", data = pbc.bili),
+      error = "none") +
+   labs(y = "Survival Probability",
+        x = "Observation Time (years)",
+        color = "Bilirubin")
```

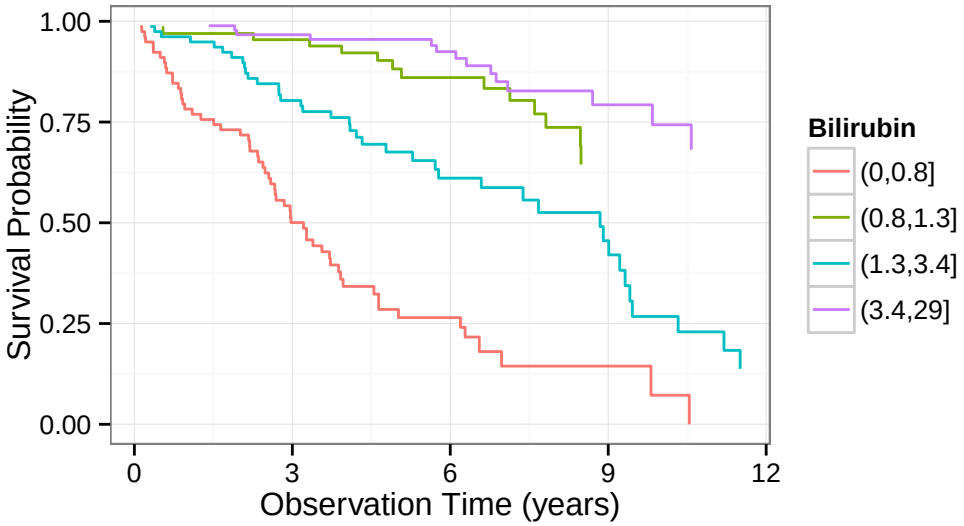

Figure 5: Kaplan–Meier pbc data survival estimates comparing Bilirubin measures. Groups defined in Fleming and Harrington (1991).

	Coef.	Std. Err.	Z stat.
Age	0.033	0.009	3.84
log(Albumin)	-3.055	0.724	-4.22
log(Bilirubin)	0.879	0.099	8.90
Edema	0.785	0.299	2.62
log(Prothrombin Time)	3.016	1.024	2.95

Table 3: Regression model summary (Fleming and Harrington 1991, Chapter 4). 312 randomized cases in `pbc.trial` data set.

In Chapter 4, Fleming and Harrington (1991) use partial likelihood methods to build a linear model with log transformations on some variables. We summarize the final, biologically reasonable model in Table 3 for later comparison with our random forest results.

3. Random Survival Forest

A Random Forest (Breiman 2001) 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, node impurity is measured by mean squared error, whereas in classification problems, the Gini

index is 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.

Random Forests for survival (Ishwaran 2007; Ishwaran *et al.* 2008) (RF-S) are an extension of Breiman (2001) Random Forests for right censored time to event data. A forest of survival trees is grown using a log-rank splitting rule to select the optimal candidate variables. Survival estimate for each observation are constructed with a Kaplan–Meier (KM) estimator within each terminal node, at each event time.

Random Forests for survival adaptively discover nonlinear effects and interactions and are fully nonparametric. Averaging over trees, with randomization while growing a tree, enables RF-S to approximate complex survival functions, including non-proportional hazards, while maintaining low prediction error. Ishwaran and Kogalur (2010) showed that RF-S is uniformly consistent and that survival forests have a uniform approximating property in finite-sample settings, a property not possessed by individual survival trees.

The **randomForestSRC** `rfsrc` function call grows the forest, determining the type of forest by the response supplied in the `formula` argument. In the following code block, we grow a random forest for survival, by passing a survival (`Surv`) object to the forest. The forest uses all remaining variables in the `pbc.trial` data set to generate survival estimates.

```
R> # Grow and store the random survival forest
R> rfsrc_pbc <- rfsrc(Surv(years, status) ~ .,
+                     data = pbc.trial,
+                     nsplit = 10,
+                     na.action = "na.impute")
R>
R> # Print the forest summary
R> rfsrc_pbc
```

```
                         Sample size: 312
                     Number of deaths: 125
                     Was data imputed: yes
                      Number of trees: 1000
           Minimum terminal node size: 3
        Average no. of terminal nodes: 60.163
No. of variables tried at each split: 5
              Total no. of variables: 17
                            Analysis: RSF
                              Family: surv
                      Splitting rule: logrank *random*
        Number of random split points: 10
                            Error rate: 16.14%
```

The `print.rfsrc` function returns information on how the random forest was grown. Here the `family = "surv"` forest has `ntree = 1000` trees (the default `ntree` argument). We used

`nsplit = 10` random split points to select random split rule, instead of an optimization on each variable at each split for performance reasons.

3.1. Generalization error (`gg_error`)

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 (Breiman 1996b) (OOB) sample, can be used as a hold out test set for each tree. 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. 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 random forest (`rfsrc_pbc`) object to extract the error estimates as a function of the number of trees in the forest. The following code block first creates a `gg_error` data object, then uses the `plot.gg_error` function to create a `ggplot` object for display.

```
R> # Data extraction
R> ggerr <- gg_error(rfsrc_pbc)
R>
R> # Figure creation
R> plot(ggerr)+
+   coord_cartesian(y = c(.09,.31))
```

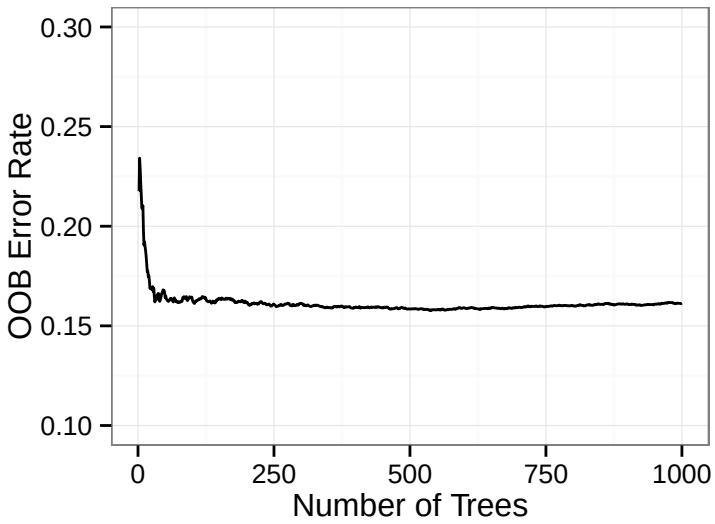

Figure 6: Random forest prediction error estimates as a function of the number of trees in the forest.

The `gg_error` plot of Figure 6 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 included in the forest prediction process, we do want to create a rather large random forest of trees.

3.2. Training Set Prediction (`gg_rfsrc`)

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. Note that we again use additional **ggplot2** commands to modify the display of the plot object. Each of the **ggRandomForests** plot commands return `ggplot` objects, which we can also store for modification or reuse later in the analysis (`ggRFsrc` object).

```
R> # Data extraction
R> gg_dta <- gg_rfsrc(rfsrc_pbc)
R>
R> # Save the ggplot2 object
R> ggRFsrc <- plot(gg_dta, alpha = .2) +
+   scale_color_manual(values = strCol) +
+   theme(legend.position = "none") +
+   labs(y = "Survival Probability", x = "time (years)")+
+   coord_cartesian(y = c(-.01,1.01))
R>
R> # Display the figure
R> show(ggRFsrc)
```

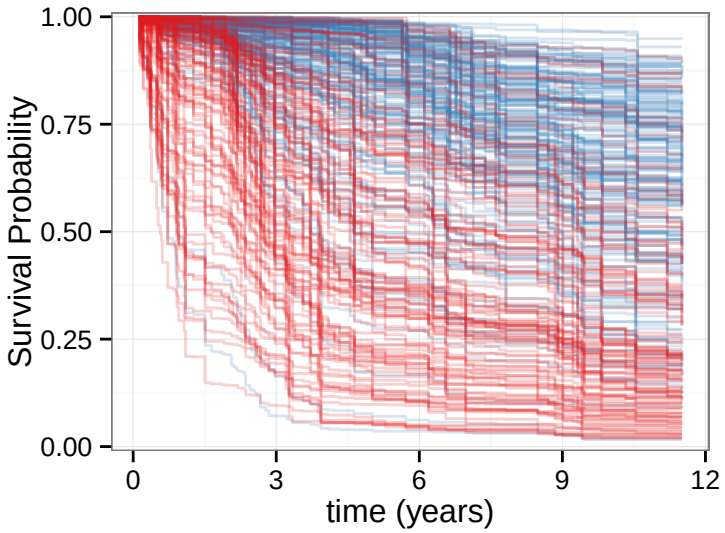

Figure 7: Random forest predicted survival. Blue lines correspond to censored observations, red lines correspond to patients who experienced the event (death).

The `gg_rfsrc` plot of Figure 7 shows the predicted survival from our RF-S model. One survival line for each patient in the training data set, where censored patients are colored blue, and patients experiencing the event are colored in red.

Interpretation of Figure 7 is difficult because of the number of curves displayed. We extend all predicted survival curves to the longest follow up time (12 years), regardless of the actual length of a patient's follow up time. To get more interpretable results, it is preferable to plot a summary of the survival results. The following code block compares the predicted survival between treatment groups, as we did in Figure 3.

```
R> plot(gg_rfsrc(rfsrc_pbc, by="treatment")) +
+   theme(legend.position = c(.2,.2)) +
+   labs(y = "Survival Probability", x = "time (years)")+
+   coord_cartesian(y = c(-.01,1.01))
```

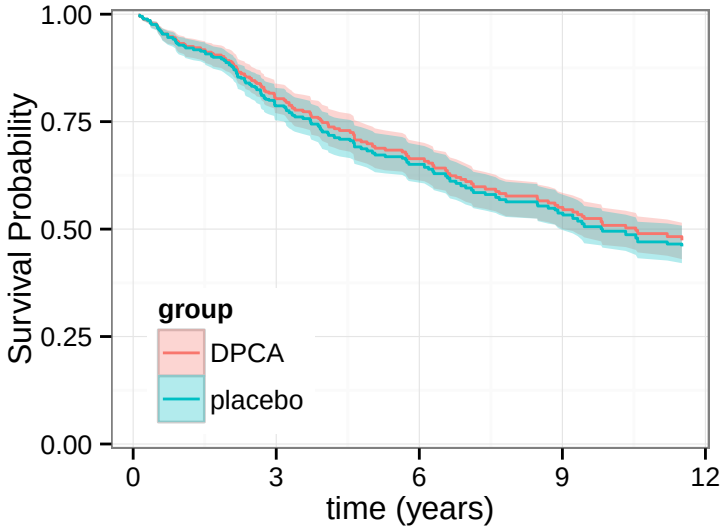

Figure 8: Mean value random forest predicted survival with shaded 95% confidence band. `DPCA` group in red, `placebo` in blue.

The `gg_rfsrc` plot of Figure 8 shows the median survival with a 95% shaded confidence band for the `DPCA` group in red, and the `placebo` group in blue. When calling `gg_rfsrc` with either a `by` argument or a `conf.int` argument, the function calculates a bootstrap confidence interval around the median survival line. By default, the function will calculate the `conf.int=.95` confidence interval, with the number of `bs.samples` equal to the number of observations.

3.3. Random Forest Imputation

There are two modeling issues when dealing with missing data values: "How does the algorithm build a model when values are missing from the training data?", and "How does the algorithm predict a response when values are missing from the test data?". The standard procedure for linear models is to either remove or impute the missing data values before modelling. Removing the missingness is done by either removing the variable with missing values (column wise) or removing the observations (row wise). Removal is a simple solution, but may bias results when either observations or variables are scarce. However, imputing missing values before modelling may discard information if the missingness is not truly at random. **?**.

The **randomForestSRC** package has an internal missing value imputation algorithm within the `rfsrc` function Ishwaran *et al.* (2008). Rather than impute all missing values before growing the forest, the algorithm takes a "just–in–time" approach. At each node split, the set of `mtry` candidate variables is checked for missing data. Missing values are imputed by randomly drawing values from non-missing values within the node before calculating the split-statistic. The split-statistic is then calculated on observations without missing data. The imputed values are used to sort observations into the subsequent daughter nodes and then discarded before the next split occurs. The process is repeated until terminal nodes are reached.

A final imputation step can be used to fill in missing values from within the terminal nodes. This step uses a process similar to the previous imputation but uses the OOB non-missing terminal node data for the random draws. These values are aggregated (averaging for continuous variables, voting for categorical variables) over the `ntree` trees in the forest to estimate an imputed data set. By default, the missing values are not filled into the training data, but are available within the forest object for later use if desired.

At each imputation step, the random forest assumes that similar observations are grouped together within each node. The random draws used to fill in missing data do not bias the split rule, but only sort observations similar in non-missing data into like nodes. A feature of this approach is the ability of predicting on test set observations with missing values.

3.4. Test Set Predictions

The importance of the forest imputation methodology becomes clear when doing prediction on new observations. If we want to predict survival for patients that did not participate in the trial, using the model we created in Section 3, we need to somehow account for the missing values detailed in Table 2.

The `predict.rfsrc` call takes the forest object (`rfsrc_pbc`), and the test data set (`pbc_test`) and returns a predicted survival using the same forest imputation method for missing values within the test data set (`na.action="na.impute"`).

```
R> # Predict survival for 106 patients not in randomized trial
R> rfsrc_pbc_test <- predict(rfsrc_pbc,
+                            newdata = pbc.test,
+                            na.action = "na.impute")
R>
R> # Print prediction summary
R> rfsrc_pbc_test


  Sample size of test (predict) data: 106
       Number of deaths in test data: 36
                Was test data imputed: yes
                 Number of grow trees: 1000
  Average no. of grow terminal nodes: 60.163
         Total no. of grow variables: 17
                             Analysis: RSF
                               Family: surv
                 Test set error rate: 19.48%
```

The forest summary indicates there are 106 test set observations with 36 deaths and the predicted error rate is 19.1%. We plot the predicted survival just as we did the training set estimates.

```
R> # Test set predicted survival
R> plot(gg_rfsrc(rfsrc_pbc_test), alpha=.2)+
+   scale_color_manual(values = strCol) +
+   theme(legend.position = "none") +
+   labs(y = "Survival Probability", x = "time (years)")+
+   coord_cartesian(y = c(-.01,1.01))
```

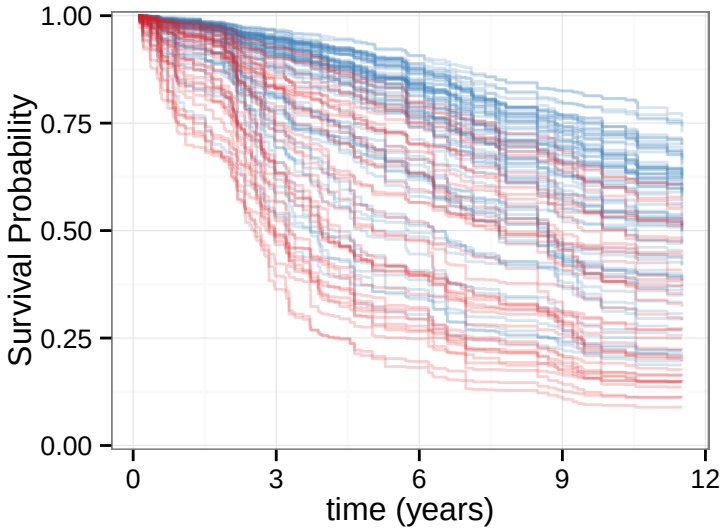

Figure 9: Test set prediction: 106 observations with missing value imputation. Censored observations shown in blue, events shown in red.

The `gg_rfsrc` plot of Figure 9 shows the test set predictions, similar to the training set predictions in Figure 7, though with fewer patients the survival curves do not cover the same area of the figure. It is important to note that because Figure 7 is constructed with OOB estimates, the survival results are comparable as estimates from unseen observations.

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 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 (`gg_vimp`)

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-u" each variable in turn. VIMP for a variable x_v is the difference between prediction error when x_v is randomly permuted, 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 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 misspecified. 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, labeled with the named vector in the `lbls=st.labs` argument.

```
R> plot.gg_vimp(rfsrc_pbc, lbls = st.labs) +
+   theme(legend.position = c(.8,.2))+
+   labs(fill = "VIMP > 0")+
+   scale_fill_brewer(palette = "Set1")
```

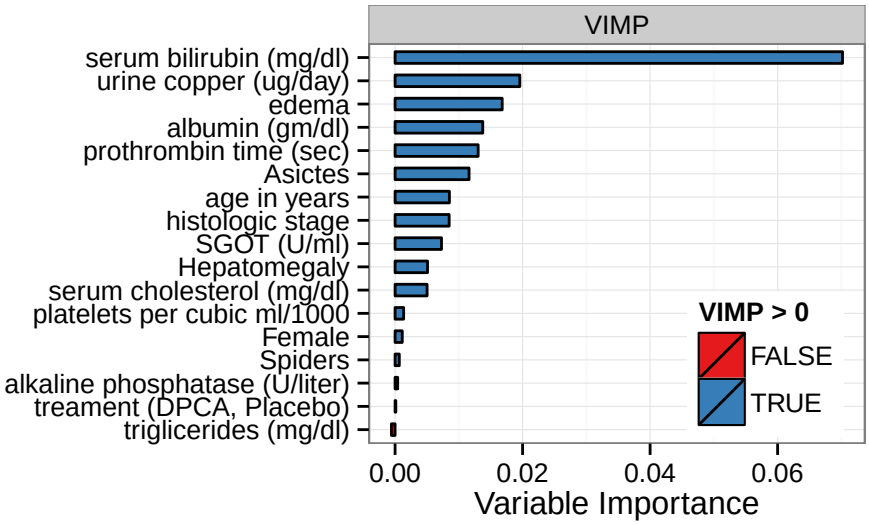

Figure 10: Random forest variable Importance (VIMP). Blue bars indicate important variables (positive VIMP), red indicates noise variables (negative VIMP).

The `gg_vimp` plot of Figure 10 details VIMP ranking for the `pbc` trial observations, from the largest (serum bilirubin) at the top, to smallest (Treatment) at the bottom. VIMP measures are shown using bars to compare the scale of the error increase under permutation and colored

by the sign of the measure (red for negative values). Note that four of the five highest ranking variables by VIMP match those selected by the Fleming and Harrington (1991) model listed in Table 3, with urine copper (2) ranking higher than age (8).

4.2. Minimal Depth (`gg_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* (Ishwaran *et al.* 2010; Ishwaran, Kogalur, Chen, and Minn 2011) assumes that variables with high impact on the prediction are those that most frequently split nodes nearest to the root node, where they partition the largest samples of the population.

Within a tree, node levels are numbered based on their relative distance to the root of the tree (with the root at 0). Minimal depth measures 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 one or more than one maximal subtree. A maximal subtree may not exist if there are no splits on the variable. The smaller the minimal depth, the more impact the variable has sorting observations, and therefore on the forest prediction.

The **randomForestSRC** `var.select` function uses the minimal depth methodology for variable selection, returning an object with both minimal depth and vimp measures. The **ggRandomForests** `gg_minimal_depth` function is analogous to the `gg_vimp` function. Variables are ranked from most important at the top (minimal depth measure), to least at the bottom (maximal minimal depth).

```
R> varsel_pbc <- var.select(rfsrc_pbc)
R> gg_md <- gg_minimal_depth(varsel_pbc, lbls = st.labs)
R> print(gg_md)

-----------------------------------------------------------
gg_minimal_depth
model size         : 12
depth threshold    : 5.5942

PE :[1] 16.142
-----------------------------------------------------------

Top variables:
            depth      vimp
bili         1.68  0.070179
albumin      2.51  0.013752
copper       2.69  0.019565
```

```
prothrombin  2.88  0.013030
chol         3.42  0.005020
age          3.51  0.008519
sgot         3.56  0.007281
edema        3.64  0.016798
platelet     3.67  0.001341
alk          3.90  0.000408
trig         4.45 -0.000574
stage        4.53  0.008481
-----------------------------------------------------------
```

The `gg_minimal_depth` summary mostly reproduces the output when running the `var.select` command from the **randomForestSRC** package. We report the minimal depth threshold (5.58) and the number of variables with depth below that threshold (12). We also list a table of the top selected variables, in minimal depth order with the associated VIMP measures. The minimal depth numbers indicate that `bili` tends to split closest to the root node, and the next three variables (`albumin`, `copper`, `prothrombin`) split close to the second level on average.

In general, to select variables according to VIMP, we examine the VIMP values, looking for some point along the ranking where there is a large difference in VIMP measures. Given minimal depth is a quantitative property of the forest construction, Ishwaran *et al.* (2010) also derive 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. Minimal depth for our model indicates there are twelve variables which have a higher impact (minimal depth below the mean value threshold) than the remaining five.

```
R> plot(gg_md, lbls = st.labs)
```

The `gg_minimal_depth` plot of Figure 11 is similar to the `gg_vimp` plot in Figure 10, ranking variables 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.

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.

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.

4.3. Model Selection Comparison

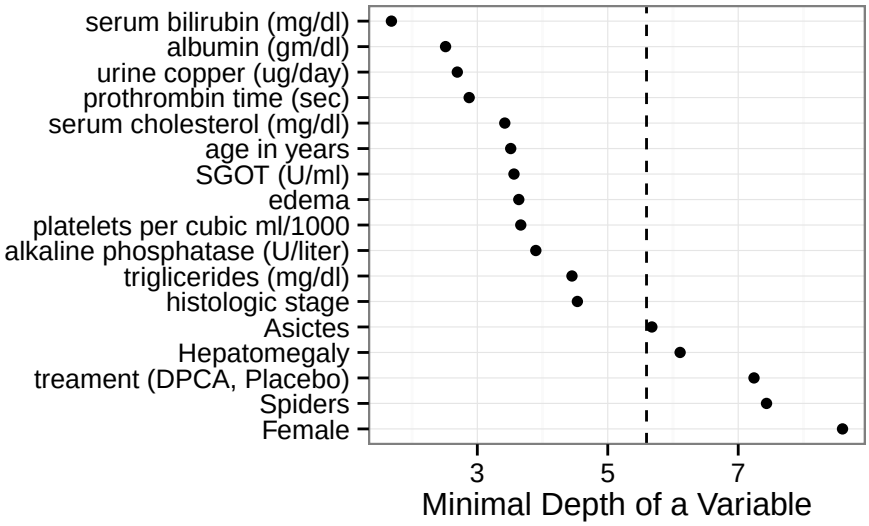

Figure 11: Minimal Depth variable selection. Low minimal depth indicates important variables. The dashed line is the threshold of maximum value for variable selection.

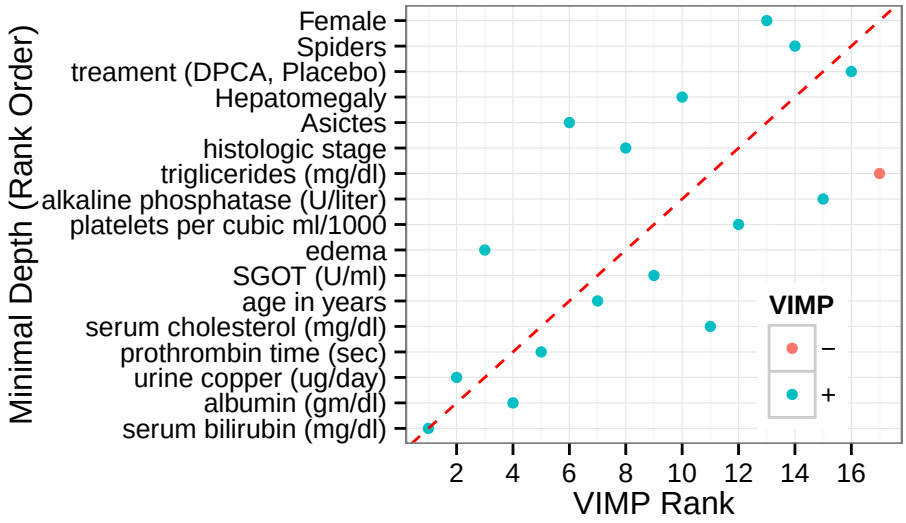

Figure 12: Comparing Minimal Depth and Vimp rankings. Points on the red dashed line are ranked equivalently, points below have higher VIMP, those above have higher minimal depth ranking.

Table 4 compares the Fleming and Harrington (1991) model of Table 3 with variables selected by minimal depth and VIMP. The table is constructed by taking the top ranked minimal depth variables (below the selection threshold) and matching the VIMP ranking and Fleming and Harrington (1991) model transforms. We see all three methods indicate a strong relation of serum bilirubin to survival, and overall, the minimal depth and VIMP rankings agree reasonably well with the Fleming and Harrington (1991) model.

Variable	Min Depth	VIMP	FH
bili	1	1	log(Bilirubin)
albumin	2	4	log(Albumin)
copper	3	2	-
prothrombin	4	5	log(Prothrombin Time)
chol	5	11	-
age	6	7	Age
sgot	7	9	-
edema	8	3	Edema
platelet	9	12	-
alk	10	15	-
trig	11	17	-
stage	12	8	-

Table 4: Comparison of model selection criteria. Minimal Depth, Vimp and proportional hazards model (Fleming and Harrington 1991, Chapter 4).

The minimal depth select process reduced the number of variables of interest from 17 to 12, which is still a rather large subset of interest. An obvious selection set is to examine the five variables selected by Fleming and Harrington (1991). There is additional evidence that `copper` and possibly `chol` may be of interest based on minimal depth and VIMP measures. Though minimal depth does not indicate the `edema` variable is very interesting, VIMP ranking does agree with the proportional hazards model, indicating we might not want to remove the `edema` variable.

One point about the `chol` variable is the amount of missing values. Recall from Table 2 that in the trial data set, there were 28 observations missing `chol` values. By definition, the forest was constructed by randomly sorting this observations into daughter nodes when using the `chol` variable, which is equivalent to how VIMP is calculated. Because of this, we will ignore the `chol` variable also.

Having selected the five Fleming and Harrington (1991) variables, plus the `copper` variable as interesting. We will examine how these six variables are related to survival using variable dependence. We are interested in the direction of the effect and would like to verify the transforms used in Fleming and Harrington (1991).

5. Variable Dependence

As random forests are not a parsimonious methodology, we use the minimal depth and VIMP measures to reduce the number of variables we need to examine to a manageable subset. Once we have an idea of which variables contribute most to the predictive accuracy of the forest, we would like to know how the response depends on these variables.

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 variable dependence.

5.1. Marginal Dependence (gg_variable)

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 represents an individual observation, 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, involving only the getting the predicted response for each observation.

In survival settings, we must account for the additional dimension of time. In this case, we plot the response at a specific time points of interest, for example survival at 1 or 3 years.

```
R> ggRFsrc +
+   geom_vline(aes(xintercept = c(1, 3)), linetype = "dashed") +
+   coord_cartesian(x = c(0, 4))
```

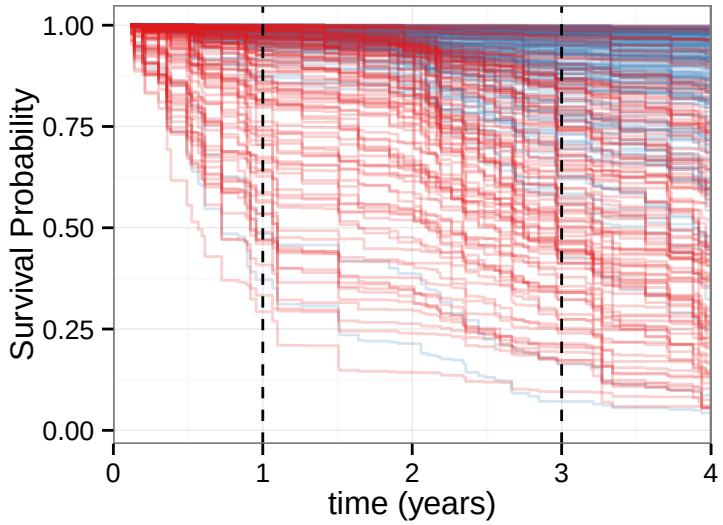

Figure 13: Random forest OOB predicted patient survival. Red curves correspond to patients which have died, blue corresponds to alive (or censored) cases. Vertical dashed lines indicate the 1 and 3 year survival estimates.

We create Figure 13 by adding vertical dashed lines to Figure 7 (stored in the ggRFsrc variable) at the 1 and 3 year survival time. A variable dependence plot is generated from the the predicted value of each curve at that intersecting time line plotted against covariate value for that 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 store the gg_variable data object for later use, as all remaining variable dependence plots can be constructed from this (gg_v) object.

```
R> gg_v <- gg_variable(rfsrc_pbc, time = c(1, 3),
+                      time.labels = c("1 Year", "3 Years"))
R>
R> # Plot the "bili" variable dependence plot
R> plot(gg_v, xvar = "bili", se = .95, alpha = .3) +
+   labs(y = "Survival", x = st.labs["bili"]) +
+   theme(legend.position = "none") +
+   scale_color_manual(values = strCol, labels = event.labels) +
+   scale_shape_manual(values = event.marks, labels = event.labels)+
+   coord_cartesian(y = c(-.01,1.01))
```

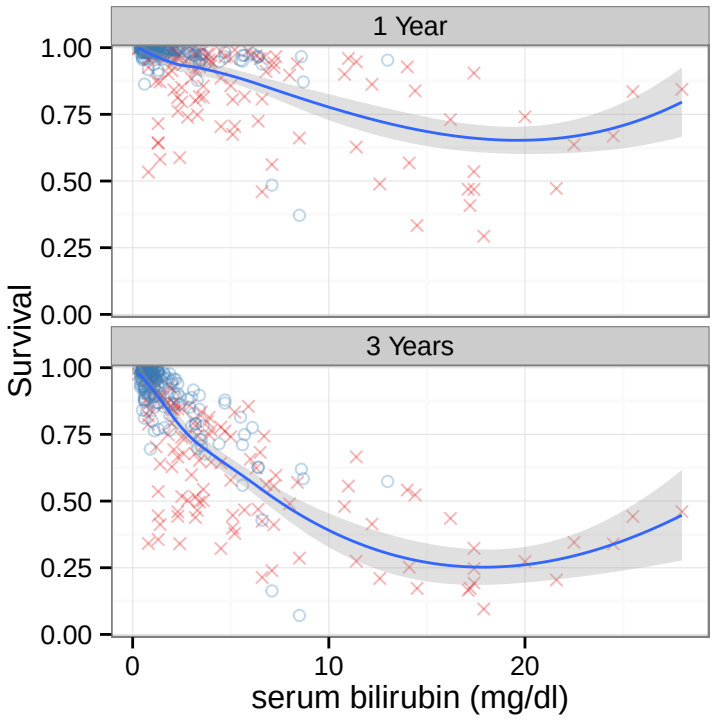

Figure 14: Bilirubin variable dependence at 1 and 3 years. Individual cases are marked with blue circles (alive or censored) and red xs (dead). Loess smooth curve with shaded 95% confidence band indicates the survival trend with increasing bilirubin.

The resulting Figure 14 shows the variable dependence for the `bili` variable. Again censored cases are shown as blue circles, events are indicated by the red "x" symbols. Each predicted point is dependent on the full combination of all other covariates, not only on the covariate displayed in the dependence plot. The smooth loess line (Cleveland 1981; Cleveland and Devlin 1988) indicates the trend of the prediction over surgical date progression.

The `plot.gg_variable` function call operates in the `gg_variable` object. We pass it the list of variables of interest (`xvar`). 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 **ggplot2** plotting routines. The `se` and `span` arguments are used to modify the internal call to `geom_smooth` for fitting smooth lines to the data. The

`alpha` argument lightens the coloring points in the `geom_point` call, making it easier to see point over plotting. We also demonstrate modification of the plot labels using the `labs` function.

It is possible to generate a single panel (`panel = TRUE`) of multiple variable dependence plots. In this code block, we use the minimal depth selected variables (minimal depth lower than the threshold value) from the stored `gg_minimal_depth` object (`gg_md$topvars`) to filter the variables of interest. We separate the categorical variables (`edema`) from the remaining continuous variables before generating a panel of continuous variable dependence plots.

```
R> # Get the minimal depth selected variables
R> xvar <- gg_md$topvars[c(1:4, 6)]
R>
R> # Remove the categorical variables
R> xvar.cat <- c("edema")
R>
R> # panel plot the next 5 continuous variable dependence plots.
R> plot(gg_v, xvar = xvar[-1], panel = TRUE,
+      se = FALSE, alpha = .3,
+      method = "glm", formula = y~poly(x,2)) +
+   labs(y = "Survival") +
+   theme(legend.position = "none") +
+   scale_color_manual(values = strCol, labels = event.labels) +
+   scale_shape_manual(values = event.marks, labels = event.labels)+
+   coord_cartesian(y = c(-.01,1.01))
```

Figure 15 displays a panel of the next 4 continuous variable dependence plots, since Figure 14 already shows the variable dependence of `bili`, the highest ranked minimal depth variable. The panels are sorted in the order of variables in the `xvar` argument and include a smooth loess line (Cleveland 1981; Cleveland and Devlin 1988) to indicate 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. Variable dependence plots for categorical variables are constructed using boxplots to show the distribution of the predictions within each category.

```
R> plot(gg_v, xvar = xvar.cat, panel = TRUE, notch = TRUE, alpha = .3) +
+   labs(y = "Survival") +
+   theme(legend.position = "none") +
+   scale_color_manual(values = strCol, labels = event.labels) +
+   scale_shape_manual(values = event.marks, labels = event.labels)+
+   coord_cartesian(y = c(-.01,1.02))
```

Figure 16 shows variable dependence boxplots created for categorical variables.

5.2. Partial Dependence (`gg_partial`)

Partial dependence plots are a risk adjusted alternative to marginal variable dependence. Partial plots are generated by integrating out the effects of variables beside the covariate of

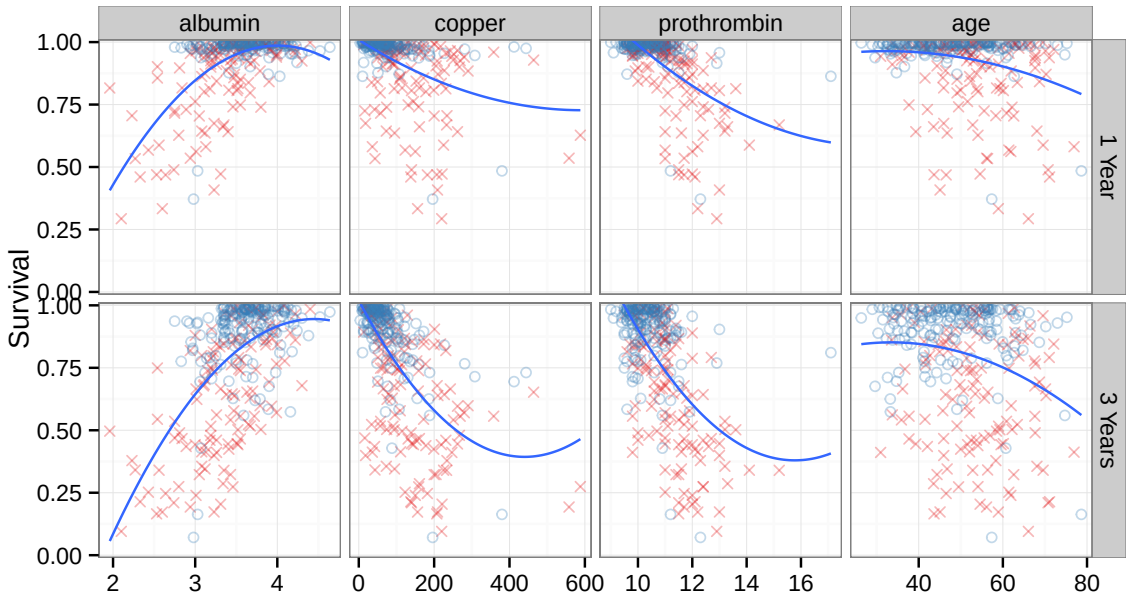

Figure 15: Bilirubin variable dependence at 1 and 3 years. Individual cases are marked with blue circles (alive or censored) and red xs (dead). Loess smooth curve with shaded 95% confidence band indicates the survival trend with increasing bilirubin.

interest. The figures are constructed by selecting points evenly spaced along the distribution of the X variable. For each of these values ($X = x$), we calculate the average Random Forest prediction over all other covariates in X by

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

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). Partial dependence plots in time to event settings are shown at specific time points, similar to variable dependence.

Partial plots are computationally intensive to create, especially when there are a large number of observations. The default parameters for the `randomForestSRC::plot.variable` function generate partial dependence estimates at `npts = 25` points 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> xvar <- c(xvar, xvar.cat)
R> # Calculate the 1, 3 and 5 year partial dependence
R> partial_pbc <- mclapply(c(1,3,5), function(tm){
+   plot.variable(rfsrc_pbc, surv.type = "surv",
+                 time = tm,
+                 xvar.names = xvar, partial = TRUE,
+                 show.plots = FALSE)
+   })
```

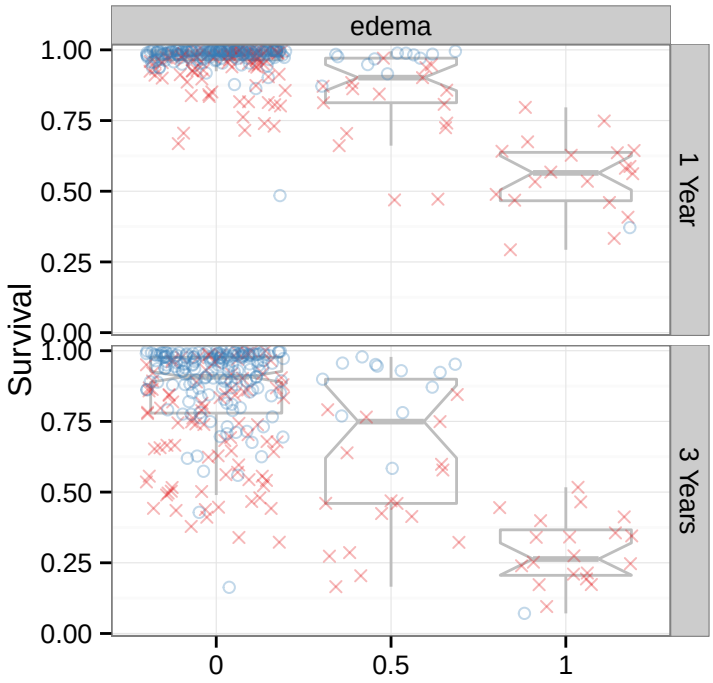

Figure 16: Variable dependence plots at 1 and 3 years for continuous variables age, albumin, copper and prothrombin. Individual cases are marked with blue circles (alive or censored) and red xs (dead). Loess smooth curve indicates the survival trend with increasing variable value.

```
R> # Convert all partial plots to gg_partial objects
R> gg_dta <- mclapply(partial_pbc, gg_partial)
R>
R> # Combine the timed gg_partial objects together.
R> pbc_ggpart <- combine.gg_partial(gg_dta[[1]], gg_dta[[2]],
+                                   lbls = c("1 Year", "3 Years"))
```

Figure **??** shows the partial dependence of one (red) and three (blue) survival on bilirubin. Non-proportional hazards are evident in Figure **??**.

```
R> # Create a temporary holder and remove extra variables
R> ggpart <- pbc_ggpart
R> ggpart$edema <- NULL
R>
R> # Panel partial dependence plots.
R> plot(ggpart, se = FALSE, panel = TRUE) +
+   labs(x = "", y = "Survival", color = "Time", shape = "Time") +
+   scale_color_brewer(palette = "Set2") +
+   theme(legend.position = c(.5, .2)) +
+   coord_cartesian(y = c(25,101))
```

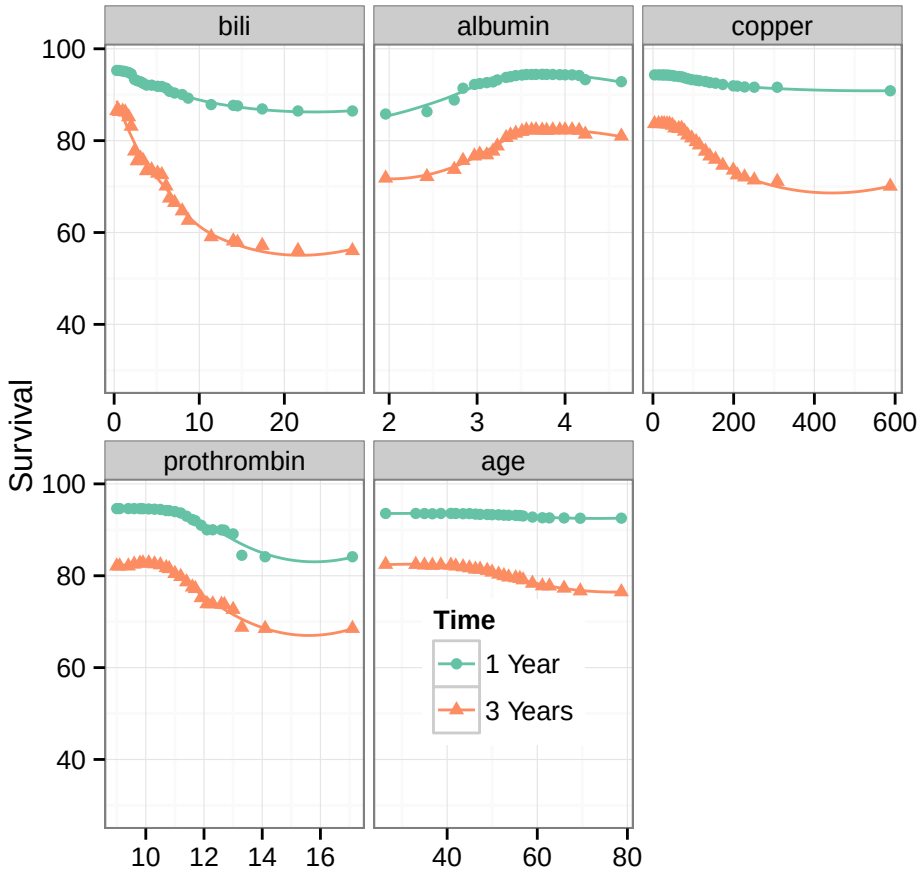

Figure 17: Partial dependence plot of (risk adjusted) predicted survival probability as a function continuous variables `prothrombin`, `albumin`, `age` and `copper` at 1 year (red circle) and 3 years (blue triangle).

In Figure 17, we again order the panels by minimal depth ranking. We see again how the variables are strongly related to survival, making the partial dependence of the remaining variables look flat. We also see strong nonlinearity of these variables.

```
R> plot.gg_partial(pbc_ggpart[["edema"]], panel=TRUE,
+                       notch = TRUE, alpha = .3, outlier.shape = NA) +
+   labs(x = "", y = "Survival (%)", color="Time", shape="Time")+
+   scale_color_brewer(palette = "Set2")+
+   theme(legend.position = c(.2, .2))+
+   coord_cartesian(y = c(25,101))
```

We could stop here, indicating that the RF analysis has found these ten variables to be important in predicting the median home values. That strongest associations to home values where there is a . However, we may also be interested in investigating how variables these work together to help random forest prediction.

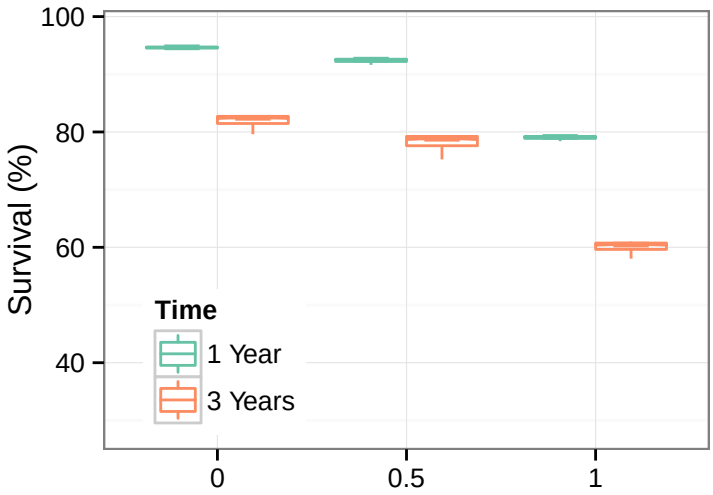

Figure 18: Partial dependence plot of (risk adjusted) predicted survival probability as a function of `edema` (categorical variable) at 1 year (red) and 3 years (blue triangle). Points indicate risk adjusted prediction for all patients within each edema group. Box plots indicate distributional properties within each group.

5.3. Time and Partial Dependence

Previously, calculated the partial dependence for 2 time points at 1 and 3 years. To get a feel for the time variation, we calculate partial dependence for

Source code is shown in Appendix B.

Figure 19 is a partial dependence surface of predicted survival as a function of `bili` over a five year follow up. We used the **plot3D** `http://CRAN.R-project.org/package=plot3D` package and the `plot3D::surf3D` function. Viewed in 3D, a surface can help to better understand

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, 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. The diagonal terms are normalized to the root node, and off diagonal terms are normalized measures of pairwise variable interaction.

```
R> ggint <- gg_interaction(rfsrc_pbc)
```

The `gg_interaction` function wraps the `find.interaction` matrix for use with the provided S3 plot and print functions. The `xvar` argument indicates which variables we're interested in

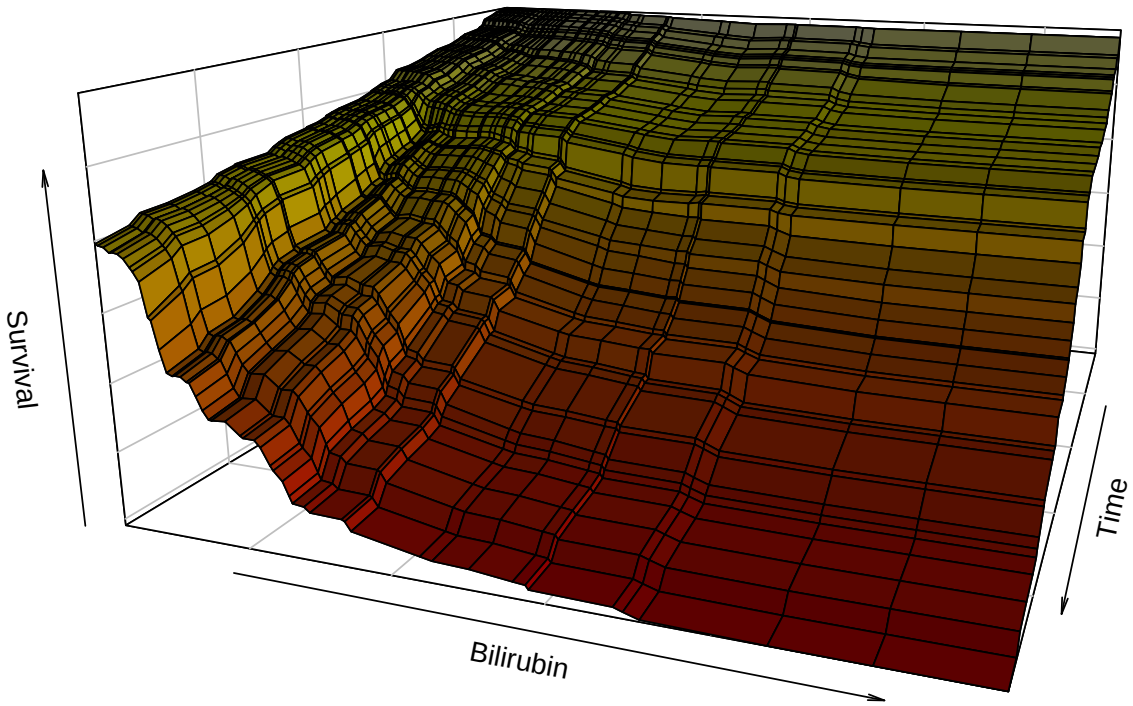

Figure 19: Partial coplot surface.

looking at. We again use the cache strategy, and collect the figures together using the `panel = TRUE` option.

```
R> plot(ggint, xvar = xvar) +
+   labs(y = "Interactive Minimal Depth") +
+   theme(legend.position = "none")
```

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 (`bili`) 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.

7. Conditional Dependence Plots

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.

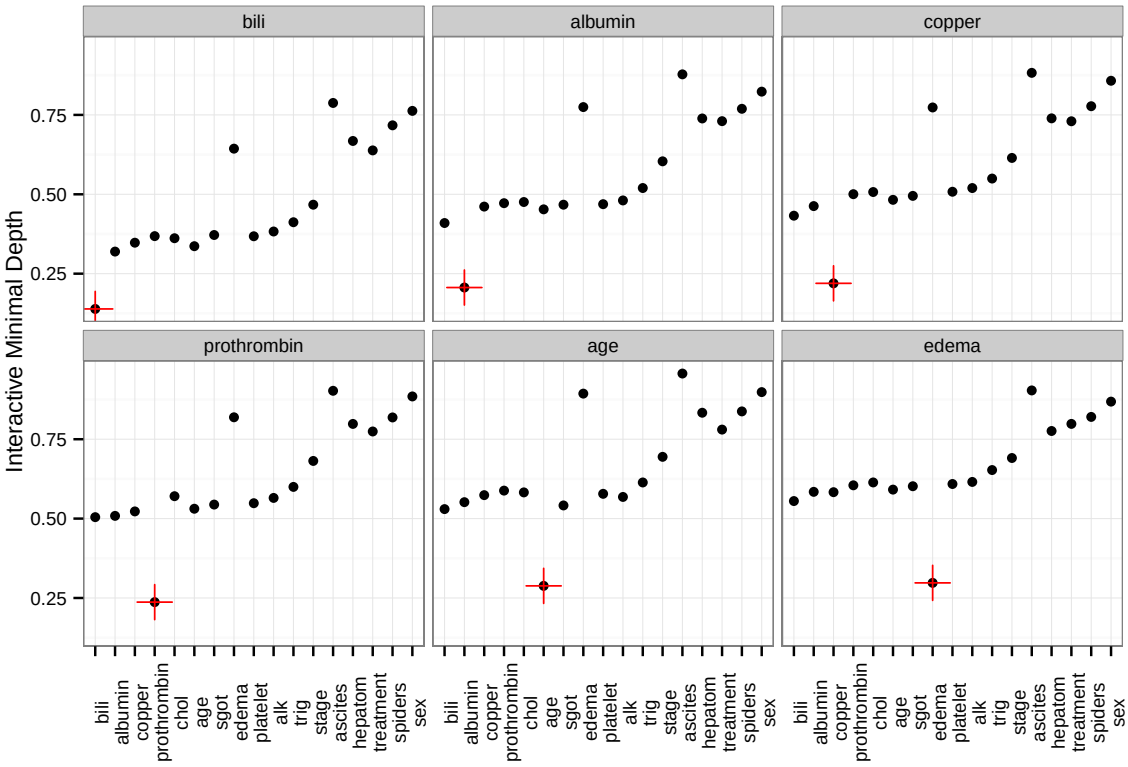

Figure 20: Minimal depth variable interaction plot. Higher values indicate lower interactivity with target variable.

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.

Interactions with categorical data are straight forward, and can be generated directly from variable dependence plots. Recall the 1 year variable dependence for bilirubin, shown in Figure 21.

```
R> ggvar <- gg_variable(rfsrc_pbc, time = 1)
R> ggvar$stage <- paste("stage = ", ggvar$stage, sep = "")
R> ggvar$edema <- paste("edema = ", ggvar$edema, sep = "")
R>
R> var_dep <- plot(ggvar, xvar = "bili",
+                  method = "glm",
+                  alpha = .5, se = FALSE) +
+   labs(y = "Survival",
+        x = st.labs["bili"]) +
+   theme(legend.position = "none") +
+   scale_color_manual(values = strCol, labels = event.labels) +
+   scale_shape_manual(values = event.marks, labels = event.labels)+
```

```
+   coord_cartesian(y = c(-.01,1.01))
R>
R> var_dep
```

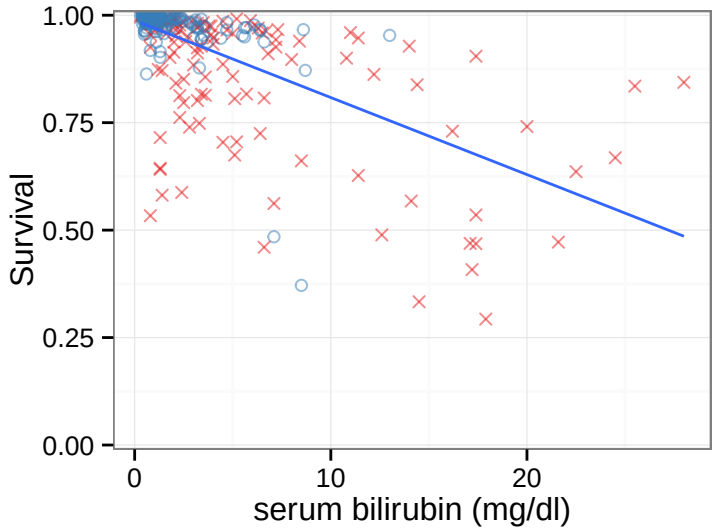

Figure 21: Variable dependence plot. Survival at 1 year against `bili` variable. Individual cases are marked with blue circles (alive or censored) and red x (dead). Loess smooth curve indicates the trend.

We can view the conditional dependence of survival against bilirubin, versus other categorical covariates, say `edema` and `stage` (categorical variables), by adding a facet argument.

```
R> var_dep +
+   facet_grid(~edema)
```

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 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_cuts` function. We pass the break points located by `quantile_cuts` 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.

```
R> # Find intervals with similar number of observations and create groups.
R> copper_cts <-quantile_pts(ggvar$copper, groups = 6, intervals = TRUE)
R> ggvar$copper_grp <- cut(ggvar$copper, breaks = copper_cts)
R>
R> # Adjust naming for facets
R> levels(ggvar$copper_grp) <- paste("copper = ",levels(ggvar$copper_grp), sep = "")
```

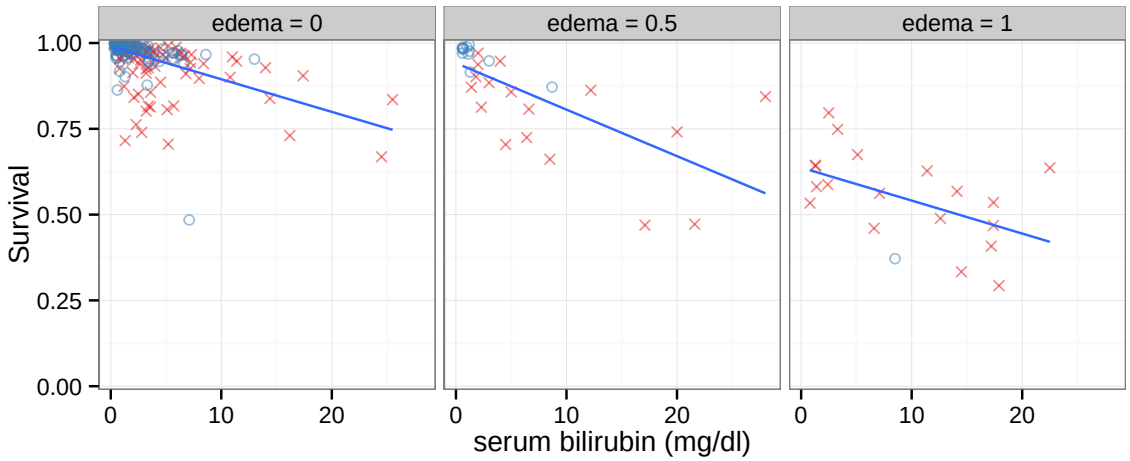

Figure 22: Variable dependence coplot. Survival at 1 year against `bili`, stratified by conditional group membership of `edema` and `stage`.

```
R>
R> plot(ggvar[-which(is.na(ggvar$copper)),], xvar = "bili",
+                  method = "glm", alpha = .5, se = FALSE) +
+   labs(y = "Survival", x = st.labs["bili"]) +
+   theme(legend.position = "none") +
+   scale_color_manual(values = strCol, labels = event.labels) +
+   scale_shape_manual(values = event.marks, labels = event.labels)+
+   facet_wrap(~copper_grp)+
+   coord_cartesian(y = c(-.01,1.01))
```

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 survival as a function of the `copper` variable, conditional on membership within 6 groups `bili` intervals.

We get similar information from this view, However viewed together we get a better sense of how the 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.

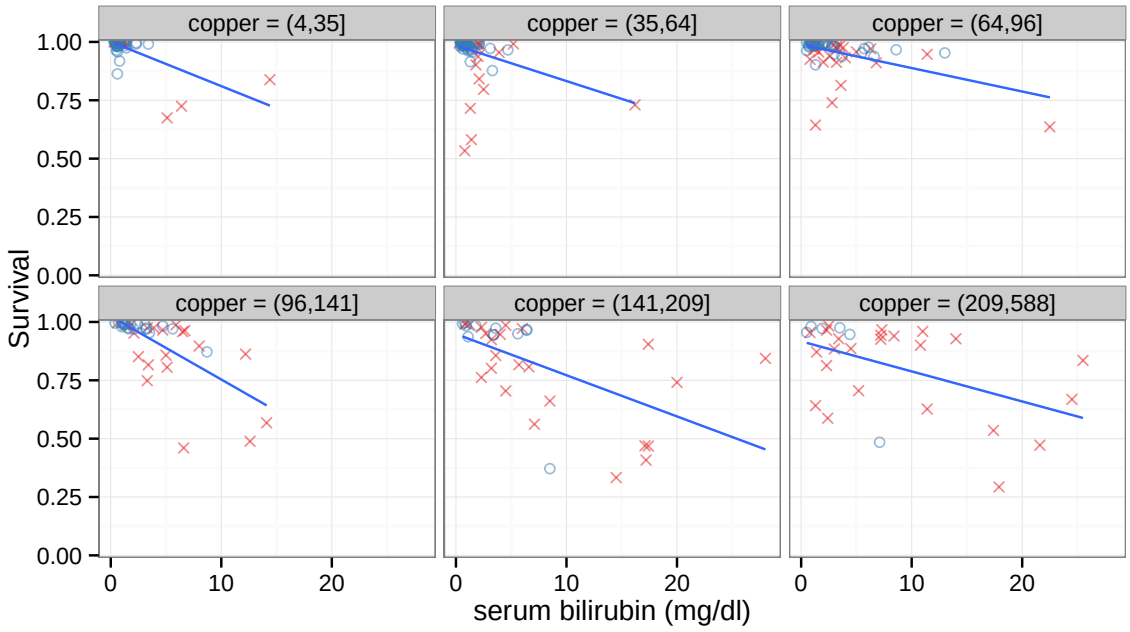

Figure 23: Variable dependence coplot. Survival at 1 year against `bili`, stratified by conditonal membership in `copper` measurement intervals.

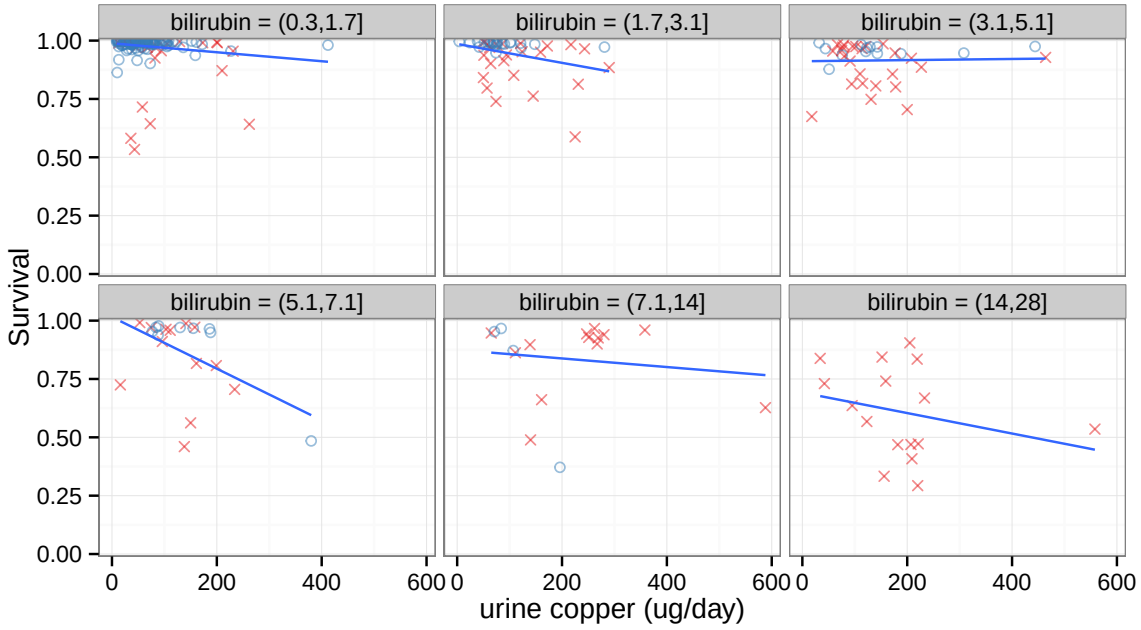

Figure 24: Variable dependence coplot. Survival at 1 year against `bili`, stratified by conditonal membership in `copper` measurement intervals.

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 (`gg_partial_coplot`)

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 `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 `bili` conditional on membership within the 6 groups of `copper` "intervals" that we examined in the previous section.

```
R> partial_coplot_pbc <- gg_partial_coplot(rfsrc_pbc, xvar = "bili",
+                                          groups = copper_grp,
+                                          surv_type = "surv",
+                                          time = 1,
+                                          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 , and use the `palette = "Set2"` Color Brewer(http://colorbrewer2.org/) color palette to choose a nice color theme for displaying the six curves.

```
R> plot(partial_coplot_pbc, se = FALSE)+
+   labs(x = st.labs["bili"], y = "Survival at 1 year (%)",
+        color = "Urine Copper", shape = "Urine Copper")+
+   scale_color_brewer(palette = "Set2")+
+   coord_cartesian(y = c(49,101))
```

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.

We can view the partial coplot curves as slices along a surface viewed into the page, either along increasing or decreasing 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 `copper` variable. We return to this problem in the next section.

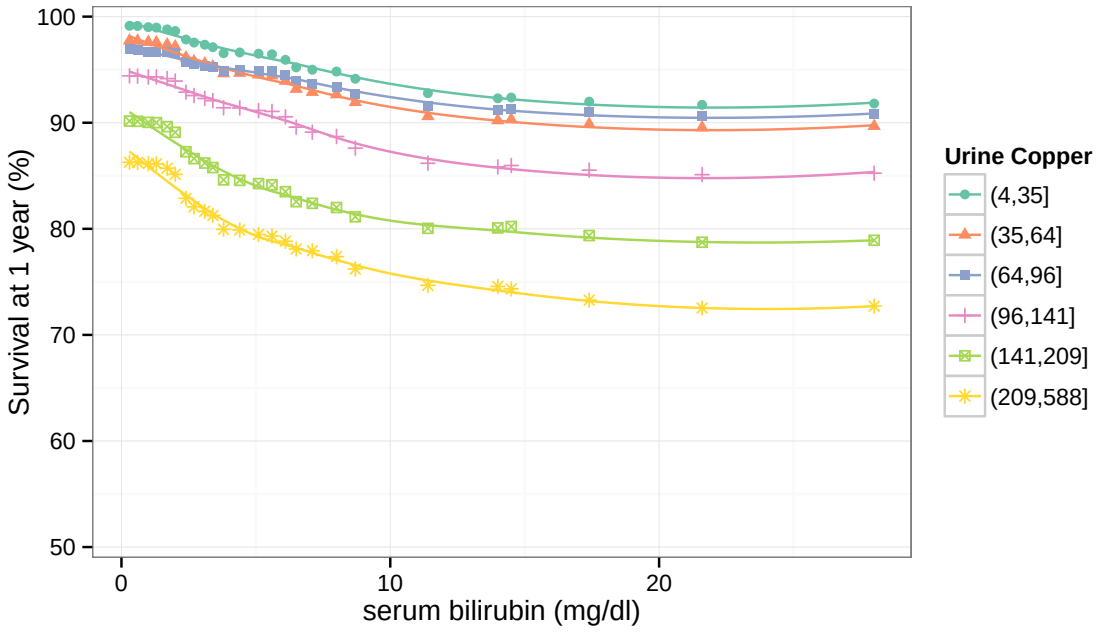

Figure 25: Partial (risk adjusted) variable dependence coplot. Survival at 1 year against `bili`, stratified by `copper` groups. Points mark risk adjusted estimates, loess smooth indicates predicted trend within each group as a function of `bili`.

We also construct the complement view, for partial dependence coplot of the "intervals", and cache the following `gg_partial_coplot` data call.

```
R> partial_coplot_pbc2 <- gg_partial_coplot(rfsrc_pbc, xvar = "copper",
+                                           groups = bili_grp,
+                                           surv_type = "surv",
+                                           time = 1,
+                                           show.plots = FALSE)

R> plot(partial_coplot_pbc2, se = FALSE)+
+   labs(x = st.labs["copper"], y = "Survival at 1 year (%)",
+        color = "Bilirubin", shape = "Bilirubin")+
+   scale_color_brewer(palette = "Set2")+
+   coord_cartesian(y = c(49,101))
```

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 plot of the median home values. We could generate this figure with the data we already have, but the resolution would be a bit strange. To generate the plot of `bili` conditional on `copper`

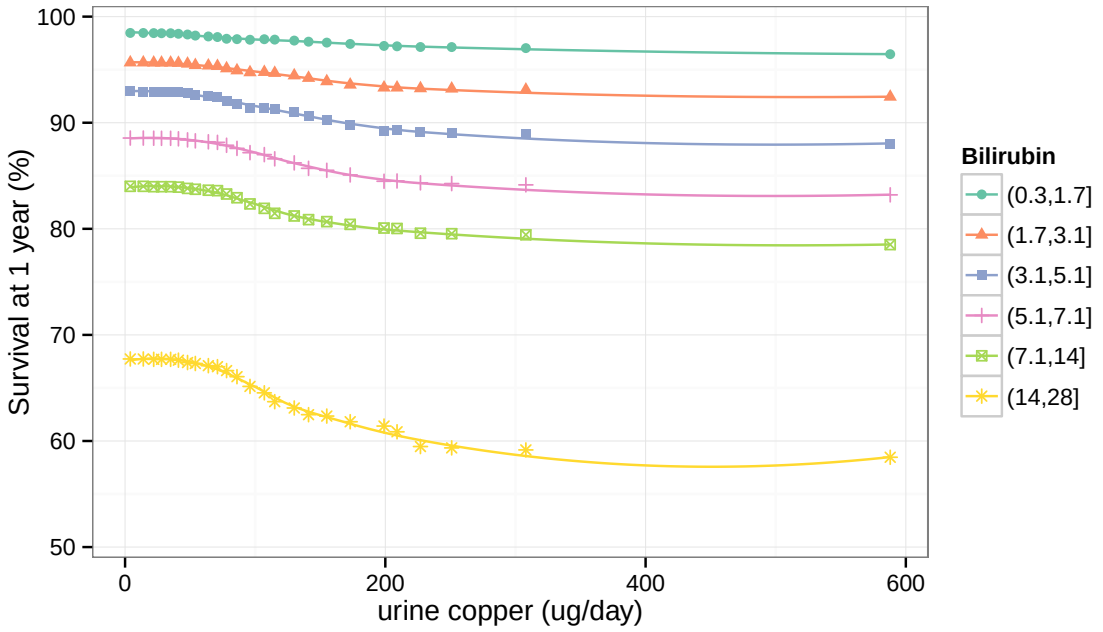

Figure 26: Partial (risk adjusted) variable dependence coplot. Survival at 1 year against `bili`, stratified by `copper` groups. Points mark risk adjusted estimates, loess smooth indicates predicted trend within each group as a function of `bili`.

groupings, we would end up with contours over a grid of `bili` = 25× `copper` = 6, for the alternative `copper` conditional on `bili` groups, we'd have the transpose grid of `bili` = 6× `copper` = 25.

Since we are already using the data caching strategy, we will generate another `gg_partial_coplot` data set with increased resolution in both the `bili` and `copper` dimensions. For this exercise, we will create 50 `copper` groups and generate the partial plot data at `npts = 50` points along the `bili` dimension for each group within the `plot.variable` call. This code block generates the 50 `copper` groups, each containing about 9 observations.

We use the following data call to generate the `gg_partial_coplot` data object. This took about 15 minutes to run on a quad core Mac Air.

The cached `gg_partial_coplot` data object is included as a data set in the **ggRandomForests** package. We load the data, attach numeric values for the `copper` groups, and generate the figure.

The contours are generated over the raw `gg_partial` estimation points, not smooth curves as shown in the partial plot and coplot figures. We can also generate a surface with this data using the **plot3D** `http://CRAN.R-project.org/package=plot3D` package and the `plot3D::surf3D` function. Viewed in 3D, a surface can help to better understand what the contour lines mean.

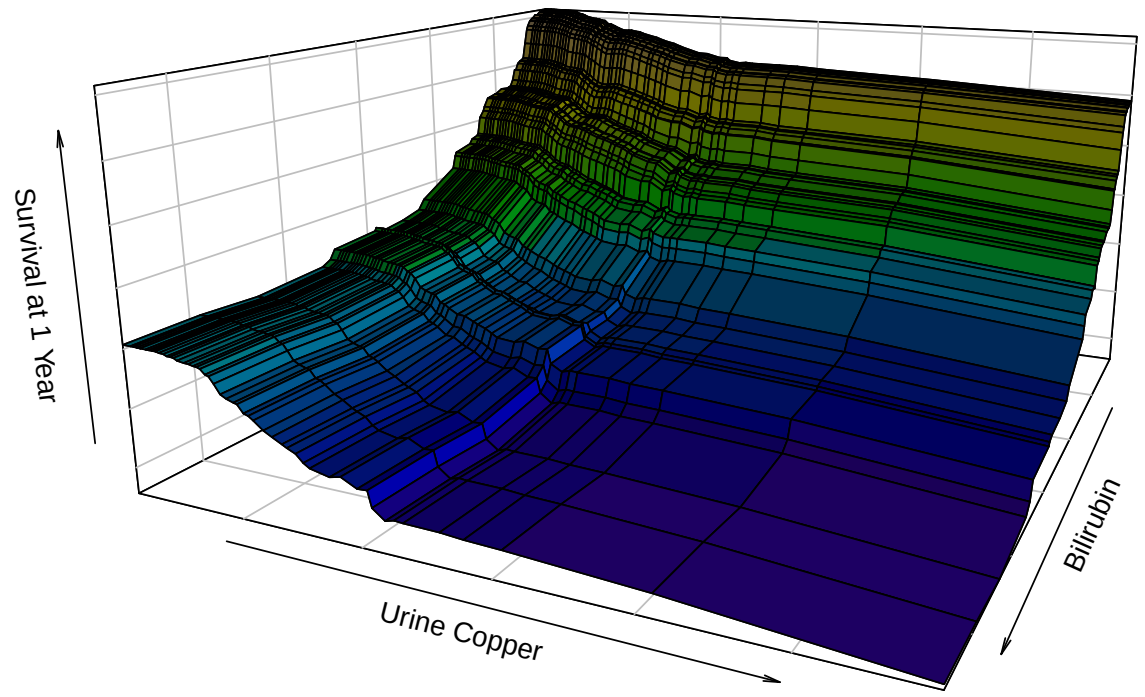

Figure 27: Partial coplot surface.

9. Conclusion

References

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.

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.

Fleming TR, Harrington DP (1991). *Counting processes and survival analysis*. Wiley, New York.

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

Hastie T, Tibshirani R, Friedman JH (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2 edition. Springer. 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 (2010). "Consistency of random survival forests." *Statistics and Probability Letters*, **80**, 1056–1064.

Ishwaran H, Kogalur UB (2014). "Random Forests for Survival, Regression and Classification (RF-SRC), R package version 1.5.5."

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.

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/.

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

Wickham H (2009). *ggplot2: elegant graphics for data analysis*. Springer 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/.

A. Bilirubin Coplot

```
R> # Find intervals with similar number of observations.
R> bili_cts <-quantile_pts(ggvar$bili, groups = 6, intervals = TRUE)
R>
R> # We need to move the minimal value so we include that observation
R> bili_cts[1] <- bili_cts[1] - 1.e-7
R>
R> # Create the conditional groups and add to the gg_variable object
R> ggvar$bili_grp <- cut(ggvar$bili, breaks = bili_cts)
R>
R> # Adjust naming for facets
R> levels(ggvar$bili_grp) <- paste("bilirubin = ",levels(ggvar$bili_grp), sep = "")
R>
R> # plot.gg_variable
R> plot(ggvar[-which(is.na(ggvar$copper)),], xvar = "copper",
+                 method = "glm", alpha = .5, se = FALSE) +
+   labs(y = "Survival", x = st.labs["copper"]) +
+   theme(legend.position = "none") +
+   scale_color_manual(values = strCol, labels = event.labels) +
+   scale_shape_manual(values = event.marks, labels = event.labels)+
+   facet_wrap(~bili_grp)+
+   coord_cartesian(y = c(-.01,1.01))
```

B. Partial Dependence in Time Dimension

```
R> # Restrict the time of interest to less than 5 years.
R> time_pts <- rfsrc_pbc$time.interest[which(rfsrc_pbc$time.interest<=5)]
R>
R> # Find the 50 points in time, evenly space along the distribution of
R> # event times for a series of partial dependence curves
R> time_cts <-quantile_pts(time_pts, groups = 50)
R>
R> # Generate the gg_partial_coplot data object
R> system.time(partial_pbc_time <- lapply(time_cts, function(ct){
+   plot.variable(rfsrc_pbc, xvar = "bili", time = ct,
+                 npts = 50, show.plots = FALSE,
+                 partial = TRUE, surv.type="surv")
+   }))
R> #     user   system  elapsed
```

```
R> # 2561.313   81.446 2641.707
R>
R> # We need to attach the time points of interest to our data.
R> time.tmp <- do.call(c,lapply(time_cts,
+                               function(grp){rep(grp, 50)}))
R>
R> # Convert the list of plot.variable output to gg_partial
R> partial_time <- do.call(rbind,lapply(partial_pbc_time, gg_partial))
R>
R> # attach the time data to the gg_partial_coplot
R> partial_time$time <- time.tmp
R>
R> # Modify the figure margins to make it larger
R> par(mai = c(0,0.3,0,0))
R>
R> # Transform the gg_partial_coplot object into a list of three named matrices
R> # for surface plotting with plot3D::surf3D
R> srf <- surface_matrix(partial_time, c("time", "bili", "yhat"))
R>
R> # Generate the figure.
R> surf3D(x = srf$x, y = srf$y, z = srf$z, col = heat.colors(25),
+        colkey = FALSE, border = "black", bty = "b2",
+        shade = 0.5, expand = 0.5, theta=50,phi=15,
+        lighting = TRUE, lphi = -50,
+        ylab = "Bilirubin", xlab = "Time", zlab = "Survival"
+ )
```

C. Partial Dependence in Multiple Variable Dimensions

```
R> # Find the quantile points to create 50 cut points for 49 groups
R> copper_cts <-quantile_pts(ggvar$copper, groups = 50)
R>
R> system.time(partial_pbc_surf <- lapply(copper_cts, function(ct){
+   rfsrc_pbc$xvar$copper <- ct
+   plot.variable(rfsrc_pbc, xvar = "bili", time = 1,
+                 npts = 50, show.plots = FALSE,
+                 partial = TRUE, surv.type="surv")
+   }))
R> # user   system  elapsed
R> # 2547.482   91.978 2671.870
R>
R> # Load the stored partial coplot data.
R> data(partial_pbc_surf)
R>
R> # Instead of groups, we want the raw copper point values,
```

```
R> # To make the dimensions match, we need to repeat the values
R> # for each of the 50 points in the copper direction
R> copper.tmp <- do.call(c,lapply(copper_cts,
+                                function(grp){rep(grp, 50)}))
R>
R> # Convert the list of plot.variable output to
R> partial_surf <- do.call(rbind,lapply(partial_pbc_surf, gg_partial))
R>
R> # attach the data to the gg_partial_coplot
R> partial_surf$copper <- copper.tmp
R>
R> # Modify the figure margins to make the figure larger
R> par(mai = c(0,03,0,0))
R>
R> # Transform the gg_partial_coplot object into a list of three named matrices
R> # for surface plotting with plot3D::surf3D
R> srf <- surface_matrix(partial_surf, c("bili", "copper", "yhat"))
R>
R> # Generate the figure.
R> surf3D(x = srf$x, y = srf$y, z = srf$z, col = topo.colors(25),
+        colkey = FALSE, border = "black", bty = "b2",
+        shade = 0.5, expand = 0.5,
+        lighting = TRUE, lphi = -50,
+        xlab = "Bilirubin", ylab = "Urine Copper", zlab = "Survival at 1 Year"
+ )
```

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/