Random Forest

Jeffrey Strickland

2/24/2022

## Random Forest using R

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees’ habit of overfitting to their training set. The source data for this and the following examples are at: <https://github.com/stricje1/random_forest>.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set with responses , bagging repeatedly (B times) selects random samples with replacement of the training set and fits trees to these samples:

1. Each sample may be comprised of different mixes members of the population, i.e., sample will have some members in common and some will be different

2. For , and , there will be N training sets comprised of m members sampled with replacement

3. Each sample may be comprised of a different mix of variables from the original set X

4. Each tree (regression or classification) will train using one of the training sets

5. After training, predictions are made by either averaging of the prediction from individual trees or by majority vote in the case of classification trees

Random Forest algorithm is built in randomForest package of R and same name function allows us to use the Random Forest in R.

###Load libraries

if(!require(randomForest)) install.packages("randomForest")   
if(!require(colorspace)) install.packages("colorspace")   
if(!require(reshape)) install.packages("reshape")   
if(!require(ggplot2)) install.packages("ggplot2")  
  
library(randomForest)  
library(colorspace)   
library(reshape)  
library(ggplot2)  
library(ggRandomForests)  
library(randomForestSRC)

Some of the commonly used parameters of randomForest functions are • x: Random Forest Formula

• data: Input data frame

• ntree: Number of decision trees to be grown

• replace: Takes True and False and indicates whether to take sample with/without replacement

• sampsize: Sample size to be drawn from the input data for growing decision tree

• importance: Whether independent variable importance in random forest be assessed

• proximity: Whether to calculate proximity measures between rows of a data frame

## Medical longevity Study of primary biliary cirrhosis (PBC)

Data was obtained from a Mayo Clinic randomized trial in primary biliary cirrhosis (PBC) of the liver conducted between 1974 and 1984. A total of 424 PBC patients, referred to Mayo Clinic during that ten-year interval met eligibility criteria for the randomized placebo-controlled trial of the drug D-penicillamine (DPCA). The data and partial likelihood model is described in Fleming and Harrington (1991). The variables in the data set are:

* case number
* number of days between registration and the earlier of death, transplantation, or study analysis time in July 1986
* status: 0=alive, 1=liver transplant, 2=dead
* drug: 1= D-penicillamine, 2=placebo
* age in days
* sex: 0=male, 1=female
* presence of ascites: 0=no 1=yes
* presence of hepatomegaly 0=no 1=yes
* presence of spiders 0=no 1=yes
* presence of edema 0=no edema and no diuretic therapy for edema; .5 = edema present without diuretics, or edema resolved by diuretics; 1 = edema despite diuretic therapy
* serum bilirubin in mg/dl
* serum cholesterol in mg/dl
* albumin in gm/dl
* urine copper in ug/day
* alkaline phosphatase in U/liter
* SGOT in U/ml
* triglycerides in mg/dl
* platelets per cubic ml / 1000
* prothrombin time in seconds
* histologic stage of disease

data(pbc)   
summary(pbc)

## days status treatment age   
## Min. : 41 Min. :0.0000 Min. :1.000 Min. : 9598   
## 1st Qu.:1093 1st Qu.:0.0000 1st Qu.:1.000 1st Qu.:15644   
## Median :1730 Median :0.0000 Median :1.000 Median :18628   
## Mean :1918 Mean :0.3852 Mean :1.494 Mean :18533   
## 3rd Qu.:2614 3rd Qu.:1.0000 3rd Qu.:2.000 3rd Qu.:21273   
## Max. :4795 Max. :1.0000 Max. :2.000 Max. :28650   
## NA's :106   
## sex ascites hepatom spiders   
## Min. :0.0000 Min. :0.00000 Min. :0.0000 Min. :0.0000   
## 1st Qu.:1.0000 1st Qu.:0.00000 1st Qu.:0.0000 1st Qu.:0.0000   
## Median :1.0000 Median :0.00000 Median :1.0000 Median :0.0000   
## Mean :0.8947 Mean :0.07692 Mean :0.5128 Mean :0.2885   
## 3rd Qu.:1.0000 3rd Qu.:0.00000 3rd Qu.:1.0000 3rd Qu.:1.0000   
## Max. :1.0000 Max. :1.00000 Max. :1.0000 Max. :1.0000   
## NA's :106 NA's :106 NA's :106   
## edema bili chol albumin   
## Min. :0.0000 Min. : 0.300 Min. : 120.0 Min. :1.960   
## 1st Qu.:0.0000 1st Qu.: 0.800 1st Qu.: 249.5 1st Qu.:3.243   
## Median :0.0000 Median : 1.400 Median : 309.5 Median :3.530   
## Mean :0.1005 Mean : 3.221 Mean : 369.5 Mean :3.497   
## 3rd Qu.:0.0000 3rd Qu.: 3.400 3rd Qu.: 400.0 3rd Qu.:3.770   
## Max. :1.0000 Max. :28.000 Max. :1775.0 Max. :4.640   
## NA's :134   
## copper alk sgot trig   
## Min. : 4.00 Min. : 289.0 Min. : 26.35 Min. : 33.00   
## 1st Qu.: 41.25 1st Qu.: 871.5 1st Qu.: 80.60 1st Qu.: 84.25   
## Median : 73.00 Median : 1259.0 Median :114.70 Median :108.00   
## Mean : 97.65 Mean : 1982.7 Mean :122.56 Mean :124.70   
## 3rd Qu.:123.00 3rd Qu.: 1980.0 3rd Qu.:151.90 3rd Qu.:151.00   
## Max. :588.00 Max. :13862.4 Max. :457.25 Max. :598.00   
## NA's :108 NA's :106 NA's :106 NA's :136   
## platelet prothrombin stage   
## Min. : 62.0 Min. : 9.00 Min. :1.000   
## 1st Qu.:188.5 1st Qu.:10.00 1st Qu.:2.000   
## Median :251.0 Median :10.60 Median :3.000   
## Mean :257.0 Mean :10.73 Mean :3.024   
## 3rd Qu.:318.0 3rd Qu.:11.10 3rd Qu.:4.000   
## Max. :721.0 Max. :18.00 Max. :4.000   
## NA's :11 NA's :2 NA's :6

The first thing we want to do is transform variable values: years to days; 0-1 to T-F, transform age to years, and impute missing statuses.

pbc1<-pbc   
pbc1$Years<-pbc$days/365   
pbc1$age<-pbc$age/365   
pbc1$Status <- NULL   
pbc1$status

## [1] 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1  
## [38] 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 0 1 0 0 0 0 1  
## [75] 1 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 0 1 0 1 0 1 0  
## [112] 1 1 1 0 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 1  
## [149] 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0  
## [186] 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1  
## [223] 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [260] 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [297] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1  
## [334] 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 1 1  
## [371] 1 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0  
## [408] 0 0 0 0 0 0 1 0 0 0 0

The output for the last step was omitted but you might want to print it out. Now we pull in the transformed data

# Get transformed data

pbc2<-pbc1[,2:20]   
head(pbc2)

## status treatment age sex ascites hepatom spiders edema bili chol albumin  
## 1 1 1 58.80548 1 1 1 1 1.0 14.5 261 2.60  
## 2 0 1 56.48493 1 0 1 1 0.0 1.1 302 4.14  
## 3 1 1 70.12055 0 0 0 0 0.5 1.4 176 3.48  
## 4 1 1 54.77808 1 0 1 1 0.5 1.8 244 2.54  
## 5 0 2 38.13151 1 0 1 1 0.0 3.4 279 3.53  
## 6 1 2 66.30411 1 0 1 0 0.0 0.8 248 3.98  
## copper alk sgot trig platelet prothrombin stage Years  
## 1 156 1718.0 137.95 172 190 12.2 4 1.095890  
## 2 54 7394.8 113.52 88 221 10.6 3 12.328767  
## 3 210 516.0 96.10 55 151 12.0 4 2.772603  
## 4 64 6121.8 60.63 92 183 10.3 4 5.273973  
## 5 143 671.0 113.15 72 136 10.9 3 4.120548  
## 6 50 944.0 93.00 63 NA 11.0 3 6.857534

## Reshaping Variable

Let’s talk briefly about reshaping data. Data Reshaping in R is something like arranged rows and columns in our own way to use it as per our requirements. We take most of the data we use as a data frame format in R to do data processing using functions like rbind(), cbind(), etc. (Smith, 2020) In this process, we reshape or re-organize the data into rows and columns. Reshaped data is reorganized data transformed a particular way which we need the data for further processing. Now we’ll reshape continuous variables data for exploratory analysis, using the melt() function.

If you have used R before, you may recall a package called reshape2. Now, tidyr supersedes reshape2 (2010-2014) and reshape (2005-2010). tidyr is designed specifically for tidying data, not general reshaping (reshape2), or the general aggregation (reshape). The data.table package provides high-performance implementations of melt() and dcast(). However, we used reshape2 here and it seems to still work.

dtb1<- melt(pbc2, id.vars=c("age","Years","status"))   
dtb2<- melt(pbc2, id.vars=c("bili","Years","status"))   
dtb3<- melt(pbc2, id.vars=c("albumin","Years","status"))   
dtb4<- melt(pbc2, id.vars=c("alk","Years","status"))   
dtb5<- melt(pbc2, id.vars=c("sgot","Years","status"))   
dtb6<- melt(pbc2, id.vars=c("prothrombin","Years","status"))   
dtb7<- melt(pbc2, id.vars=c("chol","Years","status"))   
dtb8<- melt(pbc2, id.vars=c("copper","Years","status"))   
dtb9<- melt(pbc2, id.vars=c("trig","Years","status"))   
dtb10<- melt(pbc2, id.vars=c("platelet","Years","status"))

# Plot contuinuous variables

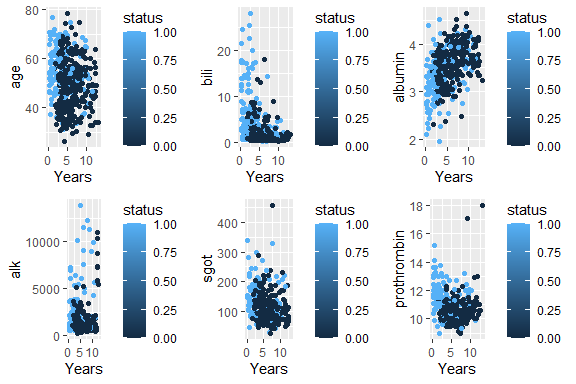
Next, we define 10 ggplots, one for each variable and year, (x,year). These will come in handy later.

gg1<-ggplot(data=dtb1, aes(x=Years, y=age)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg2<-ggplot(data=dtb2, aes(x=Years, y=bili)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg3<-ggplot(data=dtb3, aes(x=Years, y=albumin)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg4<-ggplot(data=dtb4, aes(x=Years, y=alk)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg5<-ggplot(data=dtb5, aes(x=Years, y=sgot)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg6<-ggplot(data=dtb6, aes(x=Years, y=prothrombin)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg7<-ggplot(data=dtb7, aes(x=Years, y=chol)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg8<-ggplot(data=dtb8, aes(x=Years, y=copper)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg9<-ggplot(data=dtb9, aes(x=Years, y=trig)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")   
gg10<-ggplot(data=dtb10, aes(x=Years, y=platelet)) +  
 geom\_point(aes(x=Years, color=status)) +  
 scale\_fill\_brewer(type="seq", palette = "Set1")

# Show multiple pots in a window

Here we demonstrate one way to print more than one plot in one window, simultaneously (there are other ways to do this). So, we will use the ggplot() that we just defined shown below.

if(require(!gridExtra)) install.packages("gridExtra")  
library(gridExtra)  
grid.arrange(gg1,gg2,gg3,gg4,gg5,gg6,nrow=2)



### Create Trial and Test Sets

Now, we create a trial using only randomized patients and a test set using the remaining patients, i.e., those that were not randomized. We will also create a survival object and plot the survival probability function.

pbc.trial <- pbc2[-which(is.na(pbc2$treatment)),]  
pbc.test <- pbc2[which(is.na(pbc2$treatment)),]   
head(pbc.trial)

## status treatment age sex ascites hepatom spiders edema bili chol albumin  
## 1 1 1 58.80548 1 1 1 1 1.0 14.5 261 2.60  
## 2 0 1 56.48493 1 0 1 1 0.0 1.1 302 4.14  
## 3 1 1 70.12055 0 0 0 0 0.5 1.4 176 3.48  
## 4 1 1 54.77808 1 0 1 1 0.5 1.8 244 2.54  
## 5 0 2 38.13151 1 0 1 1 0.0 3.4 279 3.53  
## 6 1 2 66.30411 1 0 1 0 0.0 0.8 248 3.98  
## copper alk sgot trig platelet prothrombin stage Years  
## 1 156 1718.0 137.95 172 190 12.2 4 1.095890  
## 2 54 7394.8 113.52 88 221 10.6 3 12.328767  
## 3 210 516.0 96.10 55 151 12.0 4 2.772603  
## 4 64 6121.8 60.63 92 183 10.3 4 5.273973  
## 5 143 671.0 113.15 72 136 10.9 3 4.120548  
## 6 50 944.0 93.00 63 NA 11.0 3 6.857534

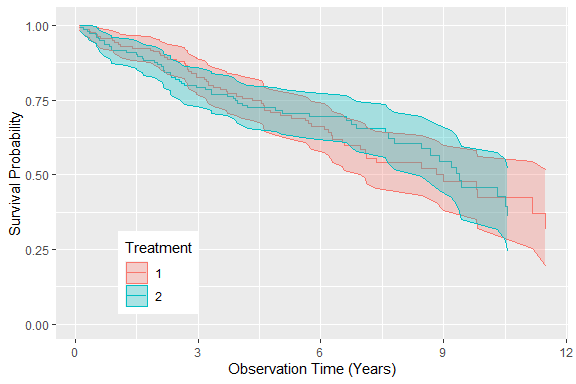
###Create the Survival Probability Function Here, we use gg\_survival to create an object representing the probability of surviving PBC.

gg\_dta <- gg\_survival(interval = "Years",   
 censor = "status",   
 by = "treatment",   
 data = pbc.trial,   
 conf.int = .95)

### Plot the survival Probability Function

We now plot the probability function we just created. This will reflect both the treatment and the placebo (or treatment 2) and will show a range of values that increase over time. That is, we will see there is greater uncertainty regarding a patient’s survival as the observation time increase.

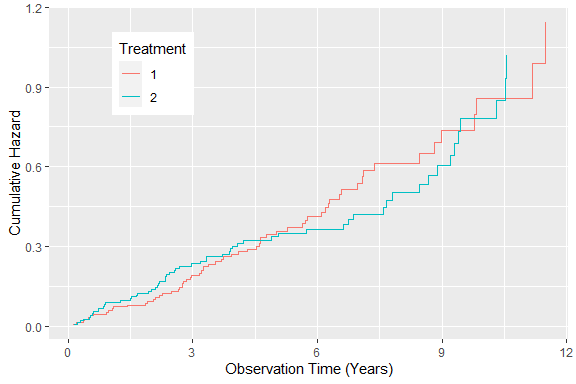
plot(gg\_dta) + labs(y = "Survival Probability",   
 x = "Observation Time (Years)",   
 color = "Treatment", fill = "Treatment") +   
 theme(legend.position = c(.2,.2)) +   
 coord\_cartesian(y = c(0,1.01))



# Plot the cumulative hazard function

Now, we plot the cumulative hazard function for the treatment and placebo.

plot(gg\_dta, type="cum\_haz") +   
 labs(y = "Cumulative Hazard",   
 x = "Observation Time (Years)",   
 color = "Treatment", fill = "Treatment") +   
 theme(legend.position = c(.2,.8))



# Duplicate the trial data

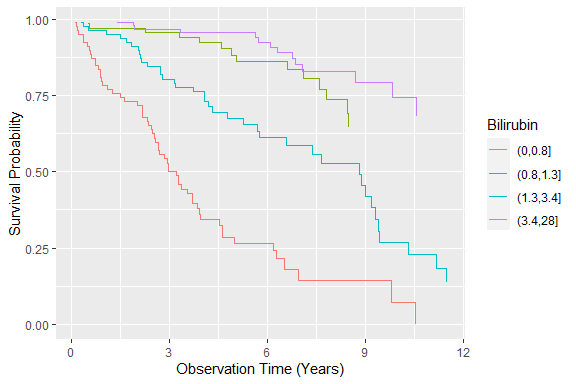
Now, we want to duplicate the trial data and group it by bilirubin values. Bilirubin forms due to the natural breakdown of hemoglobin in red blood cells, so too much bilirubin in the urine or bloodstream is not a good thing. It indicates problems with your red blood cells or with your liver.

pbc.bili <- pbc.trial  
pbc.bili$bili\_grp <- cut(pbc.trial$bili,   
 breaks = c(0, .8, 1.3, 3.4,   
 max(pbc.trial$bili)))

# Plot the gg\_survival object directly

When now plot the survival probability for four different values of bilirubin. The normal adult range is 0.8 to 1.3 mg/dL. If you are in the high range, you can develop jaundice.

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")



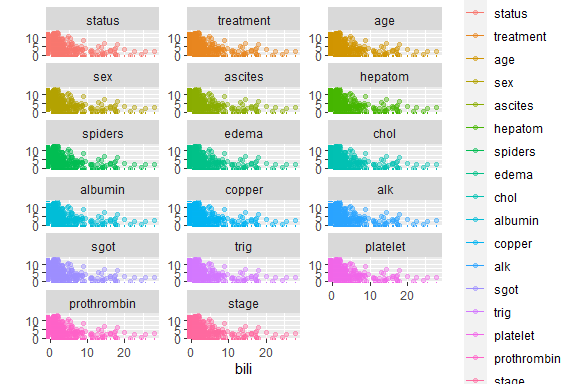
if(!require(reshape2)) install.packages(reshape2)   
library(reshape2)  
dta <- melt(pbc2, id.vars=c("bili","Years"))   
dtb <- melt(pbc2, id.vars=c("Years","status"))   
head(dtb)

## Years status variable value  
## 1 1.095890 1 treatment 1  
## 2 12.328767 0 treatment 1  
## 3 2.772603 1 treatment 1  
## 4 5.273973 1 treatment 1  
## 5 4.120548 0 treatment 2  
## 6 6.857534 1 treatment 2

# Analog to: choose\_palette(gui = “shiny”)

We’ll talk more about building Shiny apps and their GUI. For now, Shiny carries out mapping between assorted color spaces including RGB, HSV, HLS, CIEXYZ, CIELUV, HCL (polar CIELUV), CIELAB and polar CIELAB. Qualitative, sequential, and diverging color palettes based on HCL colors are provided along with corresponding.

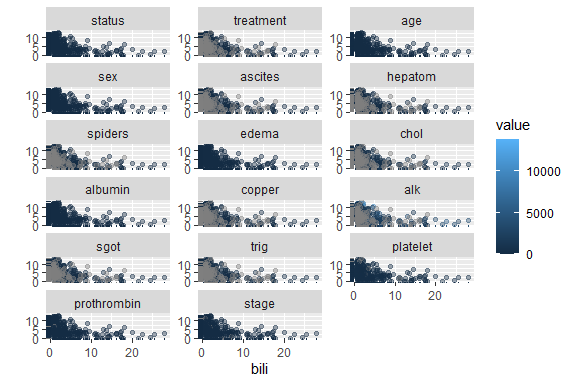
library(colorspace)  
ggplot(dta, aes(x=bili, y=Years, color=variable)) +   
 geom\_point(alpha=.4) +   
 geom\_rug(data=dta) +   
 labs(y="", x="bili") +   
 scale\_fill\_gradientn(colours =   
 colorspace::rainbow\_hcl(17)) +   
 facet\_wrap(~variable, scales="free\_y", ncol=3)



The plots in gave us nice colors, but little information. So, let’s look using shades of blue.

Now we repeat the plots with shade of blue as seen

ggplot(dta, aes(x=bili, y=Years, color=value)) +   
 geom\_point(alpha=.4) +   
 geom\_rug(data=dta) +   
 labs(y="", x="bili") +   
 scale\_fill\_brewer (type = "seq", palette = "Set2") +   
 facet\_wrap(~variable, scales="free\_y", ncol=3)



# Grow and store the random survival forest

Bagging (Bootstrap Aggregating) generates m new training data sets. Each new training data set picks a sample of observations with replacement (bootstrap sample) from original data set. By sampling with replacement, some observations may be repeated in each new training data set. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

Out-of-Bag error (miscalculation rate) is equivalent to validation or test data. In random forests, there is no need for a separate test set to validate result. It is estimated internally, during the run, as follows: As the forest is built on training data , each tree is tested on the 1/3rd of the samples (36.8%) not used in building that tree (similar to validation data set). This is the out of bag error estimate - an internal error estimate of a random forest as it is being constructed.

Fast Unified Random Forests for Survival, Regression, and Classification (RF-SRC) uses fast OpenMP parallel computing of random forests for regression, classification, survival analysis, competing risks, multivariate, unsupervised, quantile regression, and class imbalanced q-classification. Different splitting rules invoked under deterministic or random splitting are available for all families. Different types of variable importance (VIMP), holdout VIMP, as well as confidence regions can be calculated for single and grouped variables. Minimal depth variable selection. Fast interface for missing data imputation using a variety of different random forest methods.

1. **Types of forests**. There is no need to set the type of forest as the package automagically determines the underlying random forest requested from the type of outcome and the formula supplied. There are several possible scenarios:
2. Regression forests for continuous outcomes.
3. Classification forests for factor outcomes.
4. Multivariate forests for continuous and/or factor outcomes and for mixed (both type) of outcomes.
5. Unsupervised forests when there is no outcome.
6. Survival forests for right-censored survival.
7. Competing risk survival forests for competing risk.
8. **Splitting**
9. Splitting rules are specified by the option split rule.
10. For all families, pure random splitting can be invoked by setting splitrule=“random”.
11. For all families, computational speed can be increased using randomized splitting invoked by the option nsplit. See Improving Computational Speed.

rfsrc\_pbc <- rfsrc(Surv(Years, status) ~ .,   
 data = pbc.trial)

Use random splitting (nsplit = 10) and impute missing values (na.action = “na.impute”)

rfsrc\_pbc2 <- rfsrc(Surv(Years, status) ~ .,   
 data = pbc.trial,   
 nsplit = 10,   
 na.action = "na.impute")

# Print the forest summary

summary(rfsrc\_pbc)

## Length Class Mode   
## call 3 -none- call   
## family 1 -none- character  
## n 1 -none- numeric   
## ntree 1 -none- numeric   
## nimpute 1 -none- numeric   
## mtry 1 -none- numeric   
## nodesize 1 -none- numeric   
## nodedepth 1 -none- numeric   
## nsplit 1 -none- numeric   
## yvar 2 data.frame list   
## yvar.names 2 -none- character  
## xvar 17 data.frame list   
## xvar.names 17 -none- character  
## event.info 7 -none- list   
## subj 0 -none- NULL   
## subj.names 0 -none- NULL   
## xvar.wt 17 -none- numeric   
## split.wt 17 -none- numeric   
## cause.wt 1 -none- numeric   
## leaf.count 500 -none- numeric   
## proximity 0 -none- NULL   
## forest 49 rfsrc list   
## forest.wt 0 -none- NULL   
## distance 0 -none- NULL   
## membership 0 -none- NULL   
## tdc.membership 0 -none- NULL   
## splitrule 1 -none- character  
## inbag 0 -none- NULL   
## var.used 0 -none- NULL   
## imputed.indv 0 -none- NULL   
## imputed.data 0 -none- NULL   
## split.depth 0 -none- NULL   
## node.stats 0 -none- NULL   
## ensemble 1 -none- character  
## holdout.array 0 -none- NULL   
## block.size 1 -none- numeric   
## holdout.blk 0 -none- NULL   
## empr.risk 0 -none- NULL   
## oob.empr.risk 0 -none- NULL   
## ctime.internal 1 -none- numeric   
## ctime.external 5 proc\_time numeric   
## chf 30084 -none- numeric   
## chf.oob 30084 -none- numeric   
## predicted 276 -none- numeric   
## predicted.oob 276 -none- numeric   
## hazard 0 -none- NULL   
## hazard.oob 0 -none- NULL   
## survival 30084 -none- numeric   
## survival.oob 30084 -none- numeric   
## cif 0 -none- NULL   
## cif.oob 0 -none- NULL   
## err.rate 500 -none- numeric   
## time.interest 109 -none- numeric   
## ndead 1 -none- numeric

summary(rfsrc\_pbc2)

## Length Class Mode   
## call 5 -none- call   
## family 1 -none- character  
## n 1 -none- numeric   
## ntree 1 -none- numeric   
## nimpute 1 -none- numeric   
## mtry 1 -none- numeric   
## nodesize 1 -none- numeric   
## nodedepth 1 -none- numeric   
## nsplit 1 -none- numeric   
## yvar 2 data.frame list   
## yvar.names 2 -none- character  
## xvar 17 data.frame list   
## xvar.names 17 -none- character  
## event.info 7 -none- list   
## subj 0 -none- NULL   
## subj.names 0 -none- NULL   
## xvar.wt 17 -none- numeric   
## split.wt 17 -none- numeric   
## cause.wt 1 -none- numeric   
## leaf.count 500 -none- numeric   
## proximity 0 -none- NULL   
## forest 49 rfsrc list   
## forest.wt 0 -none- NULL   
## distance 0 -none- NULL   
## membership 0 -none- NULL   
## tdc.membership 0 -none- NULL   
## splitrule 1 -none- character  
## inbag 0 -none- NULL   
## var.used 0 -none- NULL   
## imputed.indv 36 -none- numeric   
## imputed.data 19 data.frame list   
## split.depth 0 -none- NULL   
## node.stats 0 -none- NULL   
## ensemble 1 -none- character  
## holdout.array 0 -none- NULL   
## block.size 1 -none- numeric   
## holdout.blk 0 -none- NULL   
## empr.risk 0 -none- NULL   
## oob.empr.risk 0 -none- NULL   
## ctime.internal 1 -none- numeric   
## ctime.external 5 proc\_time numeric   
## chf 38064 -none- numeric   
## chf.oob 38064 -none- numeric   
## predicted 312 -none- numeric   
## predicted.oob 312 -none- numeric   
## hazard 0 -none- NULL   
## hazard.oob 0 -none- NULL   
## survival 38064 -none- numeric   
## survival.oob 38064 -none- numeric   
## cif 0 -none- NULL   
## cif.oob 0 -none- NULL   
## err.rate 500 -none- numeric   
## time.interest 122 -none- numeric   
## ndead 1 -none- numeric

### Predict Patient Survival

Now, we predict the survival for 106 patients not in randomized trial:

pbc.test$status<-ifelse(pbc.test$status == "T",1,0)   
head(pbc.test)

## status treatment age sex ascites hepatom spiders edema bili chol  
## 313 0 NA 60.04110 1 NA NA NA 0.0 0.7 NA  
## 314 0 NA 65.04384 1 NA NA NA 0.5 1.4 NA  
## 315 0 NA 54.03836 1 NA NA NA 0.0 0.7 NA  
## 316 0 NA 75.05205 1 NA NA NA 0.5 0.7 NA  
## 317 0 NA 62.04384 1 NA NA NA 0.0 0.8 NA  
## 318 0 NA 43.03014 1 NA NA NA 0.0 0.7 NA  
## albumin copper alk sgot trig platelet prothrombin stage Years  
## 313 3.65 NA NA NA NA 378 11.0 NA 11.128767  
## 314 3.04 NA NA NA NA 331 12.1 4 9.756164  
## 315 4.03 NA NA NA NA 226 9.8 4 7.791781  
## 316 3.96 NA NA NA NA NA 11.3 4 5.673973  
## 317 2.48 NA NA NA NA 273 10.0 NA 8.301370  
## 318 3.68 NA NA NA NA 306 9.5 2 4.602740

rfsrc\_pbc\_test <- predict(rfsrc\_pbc,   
 newdata = pbc.test,   
 na.action = "na.impute")  
rfsrc\_pbc

## Sample size: 276  
## Number of deaths: 111  
## Number of trees: 500  
## Forest terminal node size: 15  
## Average no. of terminal nodes: 13.994  
## No. of variables tried at each split: 5  
## Total no. of variables: 17  
## Resampling used to grow trees: swor  
## Resample size used to grow trees: 174  
## Analysis: RSF  
## Family: surv  
## Splitting rule: logrank \*random\*  
## Number of random split points: 10  
## (OOB) CRPS: 0.12701315  
## (OOB) Requested performance error: 0.17530337

rfsrc\_pbc2

## Sample size: 312  
## Number of deaths: 125  
## Was data imputed: yes  
## Number of trees: 500  
## Forest terminal node size: 15  
## Average no. of terminal nodes: 15.402  
## No. of variables tried at each split: 5  
## Total no. of variables: 17  
## Resampling used to grow trees: swor  
## Resample size used to grow trees: 197  
## Analysis: RSF  
## Family: surv  
## Splitting rule: logrank \*random\*  
## Number of random split points: 10  
## (OOB) CRPS: 0.12282725  
## (OOB) Requested performance error: 0.17122

# Print prediction summary

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.

rfsrc\_pbc\_test

## Sample size of test (predict) data: 106  
## Was test data imputed: yes  
## Number of grow trees: 500  
## Average no. of grow terminal nodes: 13.994  
## Total no. of grow variables: 17  
## Resampling used to grow trees: swor  
## Resample size used to grow trees: 67  
## Analysis: RSF  
## Family: surv

Next, we setup the subsequent prediction summary:

rfsrc\_pbc\_test2 <- predict(rfsrc\_pbc2,   
 newdata = pbc.test,   
 na.action = "na.impute")

Then, we print the subsequent prediction summary:

rfsrc\_pbc\_test2

## Sample size of test (predict) data: 106  
## Was test data imputed: yes  
## Number of grow trees: 500  
## Average no. of grow terminal nodes: 15.402  
## Total no. of grow variables: 17  
## Resampling used to grow trees: swor  
## Resample size used to grow trees: 67  
## Analysis: RSF  
## Family: surv

# Variable Importance

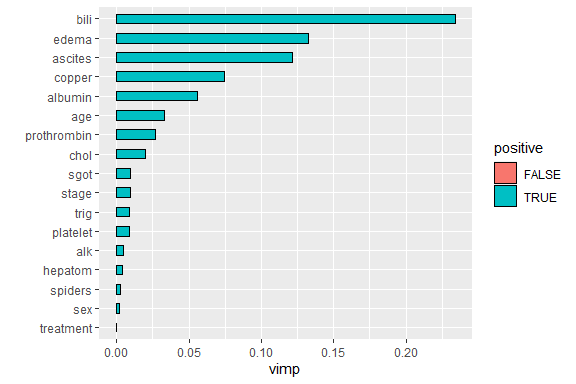
Variable importance (VIMP) was originally deﬁned in CART using a measure involving surrogate variables (Breiman, H., Olshen, & Stone, 1984). The most popular VIMP method uses a prediction error approach involving “noising-up” each variable in turn. VIMP for a variable xv is the diﬀerence between prediction error when xv is noised up by randomly permuting its values, compared to prediction error under the observed values (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010).

Since VIMP is the diﬀerence in Out-of-Bag prediction error before and after permutation, a large VIMP value indicates that misspeciﬁcation detracts from the predictive accuracy in the forest. If VIMP is close to zero the variable contributes nothing to predictive accuracy, and negative values indicate the predictive accuracy improves when the variable is misspeciﬁed. In the latter 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 in , labeled with the named vector in the lbls argument. Now, we extract VIMP measures for each of the variables used to grow the forest.

gg\_variable(rfsrc\_pbc)   
gg\_dta<-gg\_vimp(rfsrc\_pbc)  
gg\_dta

## vars set vimp positive  
## 1 bili VIMP 0.233931275 TRUE  
## 2 edema VIMP 0.132606188 TRUE  
## 3 ascites VIMP 0.121736631 TRUE  
## 4 copper VIMP 0.074217833 TRUE  
## 5 albumin VIMP 0.055842637 TRUE  
## 6 age VIMP 0.033015087 TRUE  
## 7 prothrombin VIMP 0.026667339 TRUE  
## 8 chol VIMP 0.019600383 TRUE  
## 9 sgot VIMP 0.009166524 TRUE  
## 10 stage VIMP 0.009148748 TRUE  
## 11 trig VIMP 0.008944525 TRUE  
## 12 platelet VIMP 0.008566034 TRUE  
## 13 alk VIMP 0.004899239 TRUE  
## 14 hepatom VIMP 0.004070648 TRUE  
## 15 spiders VIMP 0.002311568 TRUE  
## 16 sex VIMP 0.001897845 TRUE  
## 17 treatment VIMP -0.000252970 FALSE

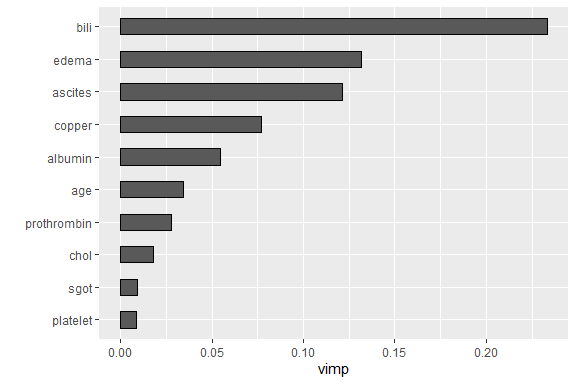
plot(gg\_dta)



gg\_dta\_10<-gg\_vimp(rfsrc\_pbc, nvar=10)  
gg\_dta\_10

## vars set vimp positive  
## 1 bili VIMP 0.233403489 TRUE  
## 2 edema VIMP 0.131917474 TRUE  
## 3 ascites VIMP 0.121207616 TRUE  
## 4 copper VIMP 0.076919356 TRUE  
## 5 albumin VIMP 0.054682550 TRUE  
## 6 age VIMP 0.034017371 TRUE  
## 7 prothrombin VIMP 0.027512452 TRUE  
## 8 chol VIMP 0.017812147 TRUE  
## 9 sgot VIMP 0.008983854 TRUE  
## 10 platelet VIMP 0.008376653 TRUE

plot(gg\_dta\_10)



Now we plot the VIMP the top ten variables using default settings.

library(ggRandomForests)  
cls <- sapply(pbc, class)  
dta.labs <- data.frame(cbind(names = colnames(pbc), label = labels, type = cls))  
# Put the "years" variable on top.  
dta.labs <- rbind(dta.labs[nrow(dta.labs),], dta.labs[-nrow(dta.labs),])  
  
st.labs <- as.character(dta.labs$label)  
names(st.labs) <- rownames(dta.labs)  
library(ggRandomForests)  
cls <- sapply(pbc, class)  
dta.labs <- data.frame(cbind(names = colnames(pbc), label = labels, type = cls))

# Minimal Depth

In VIMP, predictive risk factors are determined by testing the forest model prediction under alternative data settings, ranking the most important variables according to their impact on predictive ability of the forest. Another method uses inspection of the forest construction to rank variables. Minimal depth (Ishwaran H. , Kogalur, Chen, & J., 2011) (Ishwaran H. , Kogalur, Gorodeski, & Minn, 2010) 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 each 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. We assume that in the metric smaller minimal depth values indicate the variable separates large groups of observations, and therefore has a large impact on the forest prediction.

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.

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)

# Return an object with both minimal depth and vimp measures

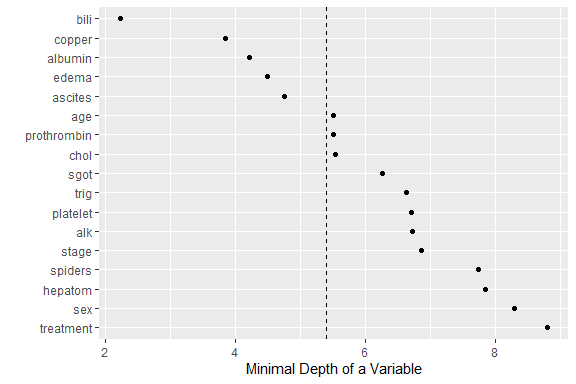
varsel\_pbc <- var.select(rfsrc\_pbc)

## minimal depth variable selection ...  
##   
##   
## -----------------------------------------------------------  
## family : surv   
## var. selection : Minimal Depth   
## conservativeness : medium   
## x-weighting used? : TRUE   
## dimension : 17   
## sample size : 276   
## ntree : 500   
## nsplit : 10   
## mtry : 5   
## nodesize : 15   
## refitted forest : FALSE   
## model size : 5   
## depth threshold : 5.3956   
## PE (true OOB) : 17.5303   
##   
##   
## Top variables:  
## depth vimp  
## bili 2.236 NA  
## copper 3.852 NA  
## albumin 4.220 NA  
## edema 4.500 NA  
## ascites 4.752 NA  
## -----------------------------------------------------------

ggMindepth <- gg\_minimal\_depth(varsel\_pbc, lbls = Years)   
print(ggMindepth)

## -----------------------------------------------------------  
## gg\_minimal\_depth  
## model size : 5   
## depth threshold : 5.3956   
##   
## PE :[1] 17.53  
## -----------------------------------------------------------  
##   
## Top variables:  
## depth vimp  
## bili 2.24 NA  
## copper 3.85 NA  
## albumin 4.22 NA  
## edema 4.50 NA  
## ascites 4.75 NA  
## -----------------------------------------------------------

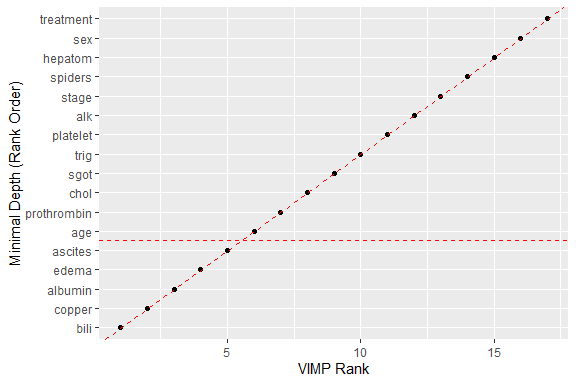
plot(ggMindepth)



# Both minimal depth and VIMP

VIMP and Minimal Depth measures use diﬀerent criteria, so we expect the variable ranking to be slightly 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 (gg\_md), which would be equivalent to calling plot.gg\_minimal\_vimp(ggMindepth) or plot(gg\_minimal\_vimp(ggMindepth)).

plot(gg\_minimal\_vimp(ggMindepth))



# Get the minimal depth selected variables

xvar <- varsel\_pbc$topvars   
xvar

## [1] "bili" "copper" "albumin" "edema" "ascites"

The points along the red dashed line indicate where the measures agree. Points above the red dashed line are ranked higher by VIMP than by minimal depth, indicating the variables are more 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.

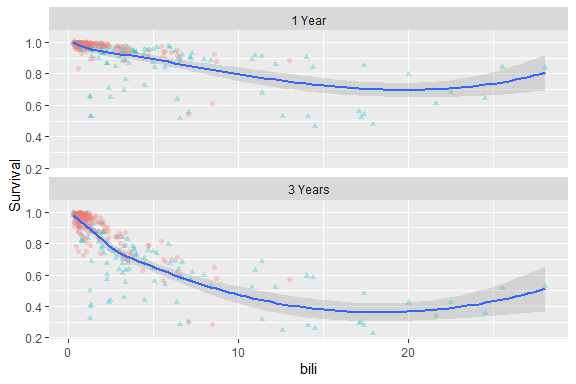
# Data generation

ggrf <- gg\_variable(rfsrc\_pbc, time = c(1, 3),   
 time.labels = c("1 Year", "3 Years"))

# Bilirubin variable dependence plot

Here we generate variable dependence plots of survival at 1 and 3 years on bilirubin variable. Individual cases are marked with red circles (alive or censored) and green triangles (dead). Loess smooth curve with shaded 95% conﬁdence band indicates decreasing survival with increasing bilirubin.

plot(ggrf, xvar = "bili", se = .95, alpha = .3) +   
 labs(y = "Survival", x = "bili") +   
 theme(legend.position = "none")



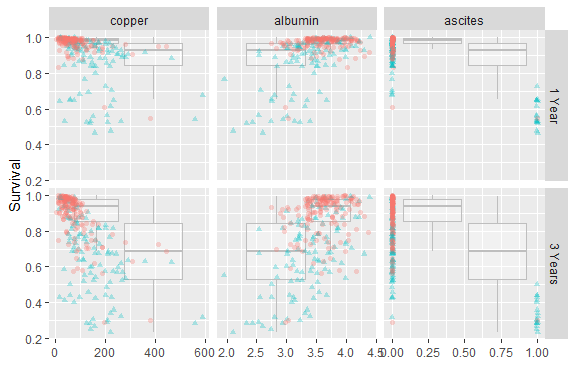
# Pull the categorical variables

xvar.cat <- c("edema", "stage")   
xvar <- xvar[-which(xvar %in% xvar.cat)]

# Continuous Variable Dependence Plots.

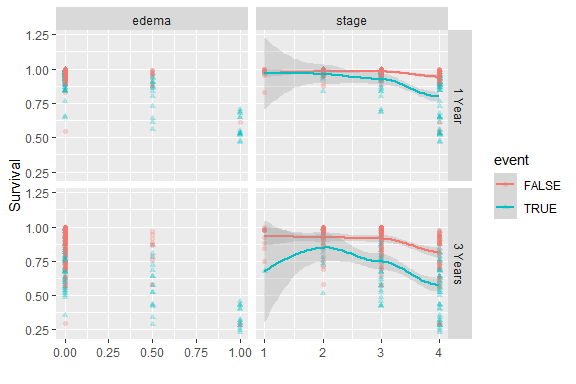
We now generate variable dependence plots of predicted survival at 1 and 3 years on continuous variables of interest. Individual cases are marked with red circles for censored cases and green triangles for death events. Loess smooth curve indicates the survival trend with increasing values.

plot(ggrf, xvar = xvar[2:6], panel = TRUE,   
 se = FALSE, alpha = .3,   
 method = "glm", formula = y~poly(x,2)) +   
 labs(y = "Survival") +   
 theme(legend.position = "none") #optional



Variable dependence plots for categorical variables are constructed using boxplots to show the distribution of the predictions within each category. Here we demonstrate variable dependence of survival 1 and 3 years on the edema categorical variable. Symbols with red circles indicate censored cases and green triangles indicate death events. Boxplots indicate distribution of predicted survival for all observations within each edema group.

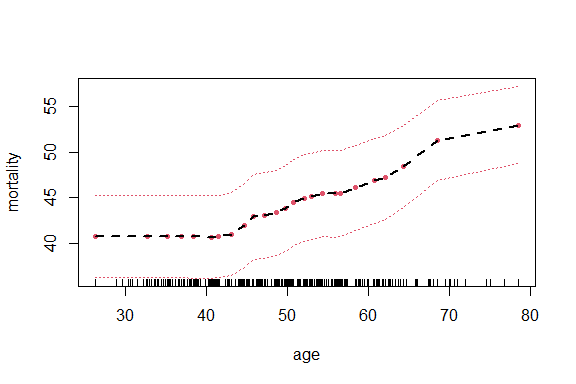
plot(ggrf, xvar = xvar.cat, panel = TRUE, notch = TRUE, alpha = .3) +   
 labs(y = "Survival") +   
 scale\_fill\_gradientn(colours = colorspace::rainbow\_hcl(17))



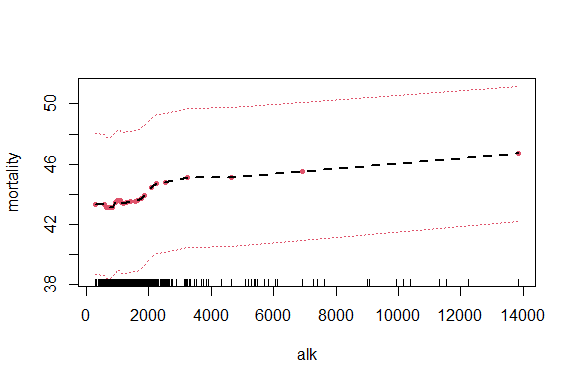
### Partial Dependence Plot

The partial dependence plot is a global method: The method considers all instances and gives a statement about the global relationship of a feature with the predicted outcome. We calculate the 1- and 3-year partial dependence plots for age and alkaline phosphatase

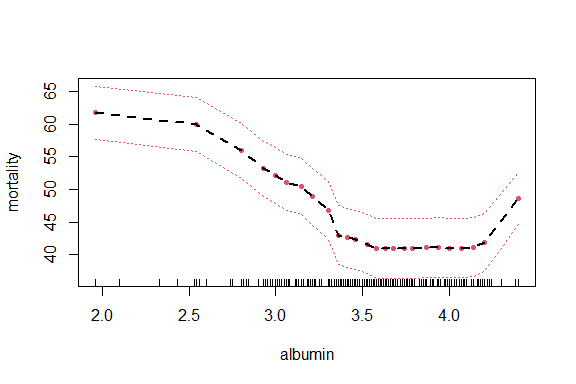
partial\_age <- plot.variable(rfsrc\_pbc, xvar.names = "age",   
 partial=TRUE)



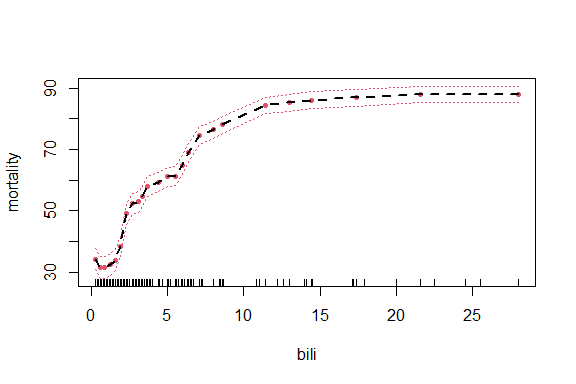
partial\_alk <- plot.variable(rfsrc\_pbc, xvar.names = "alk",   
 partial=TRUE)



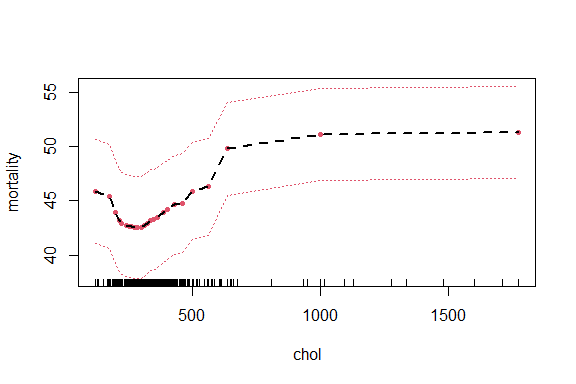
partial\_alb <- plot.variable(rfsrc\_pbc, xvar.names = "albumin",  
 partial=TRUE)



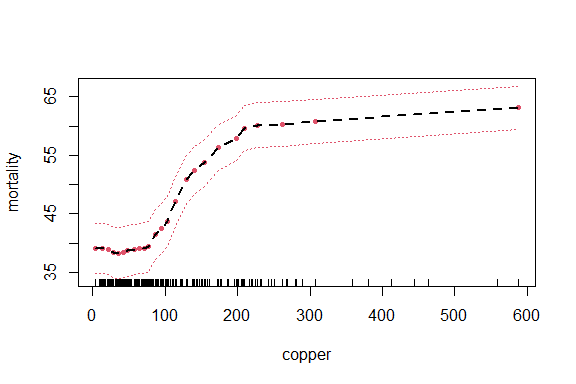
partial\_bili <- plot.variable(rfsrc\_pbc, xvar.names = "bili",   
 partial=TRUE)



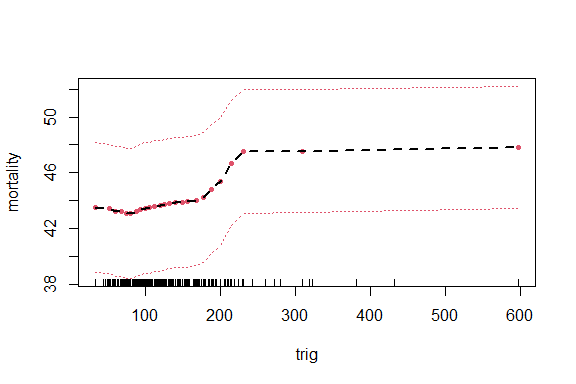
partial\_chol <- plot.variable(rfsrc\_pbc, xvar.names = "chol",   
 partial=TRUE)



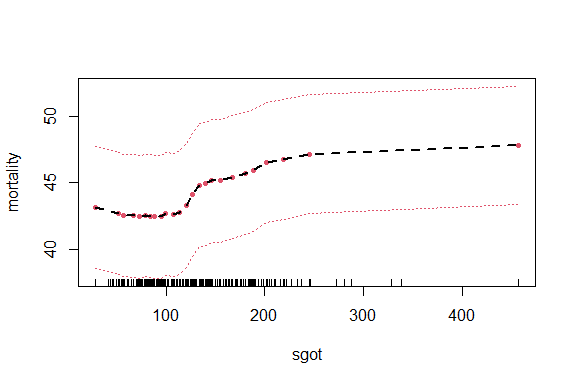
partial\_copp <- plot.variable(rfsrc\_pbc, xvar.names = "copper",   
 partial=TRUE)



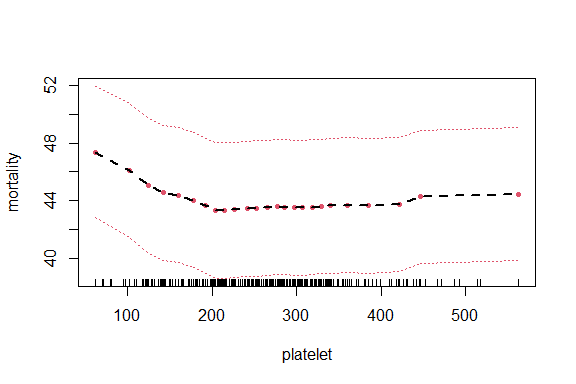
partial\_trig <- plot.variable(rfsrc\_pbc, xvar.names = "trig",   
 partial=TRUE)



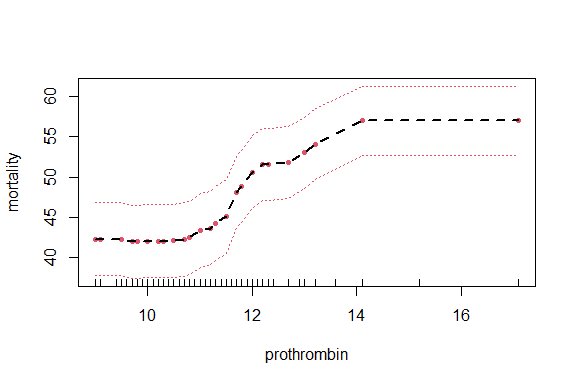
partial\_sgot <- plot.variable(rfsrc\_pbc, xvar.names = "sgot",   
 partial=TRUE)



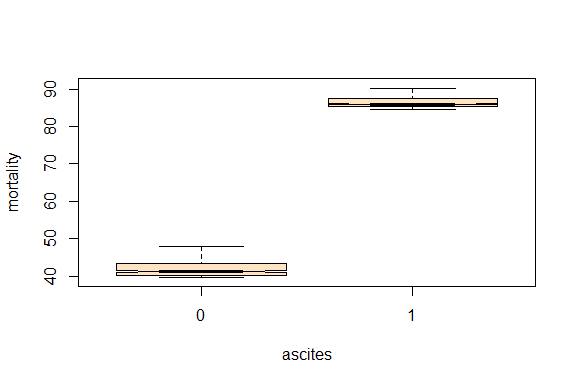
partial\_plat <- plot.variable(rfsrc\_pbc, xvar.names = "platelet",   
 partial=TRUE)



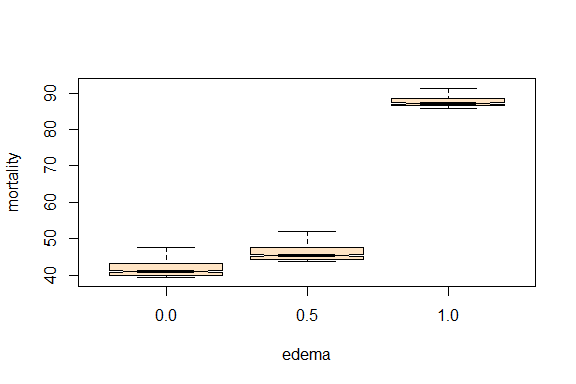
partial\_pro <- plot.variable(rfsrc\_pbc, xvar.names = "prothrombin",   
 partial=TRUE)



partial\_asci <- plot.variable(rfsrc\_pbc, xvar.names = "ascites",  
 partial=TRUE)

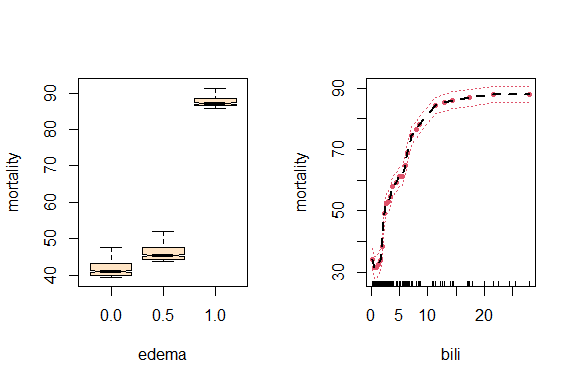


partial\_edema<- plot.variable(rfsrc\_pbc, xvar.names = "edema",   
 partial=TRUE)

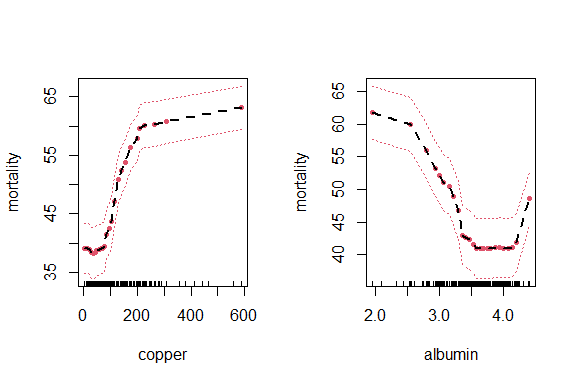


Categorical Features So far, we have only thought about numerical features. For categorical features, the partial dependence is very easy to compute. For each of the categories, like “yes” or “no” for the presence of ascites we get a PDP estimate by forcing all data instances to have the same category as in Figure 10 18. For another example, if we look at the PBC dataset and are interested in the partial dependence plot for the edema, we get 3 numbers, one for each category. To compute the value for “no edema and no diuretic therapy for edema”, we replace the edema of all data instances with “no edema and no diuretic therapy for edema” and average the predictions. We plot this instance along with for continuous variables and with 1-year and 3-year in plots.

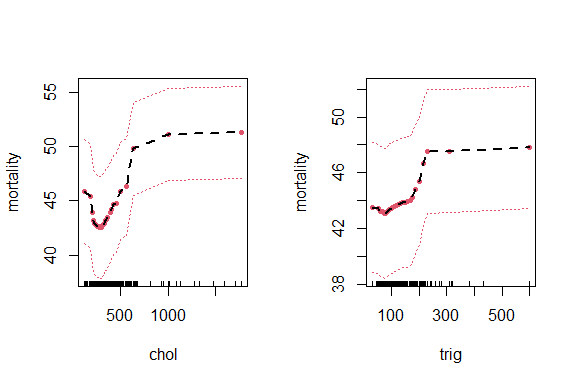
part\_var01 <- plot.variable(rfsrc\_pbc, xvar.names =   
c("edema","bili"), partial = TRUE)



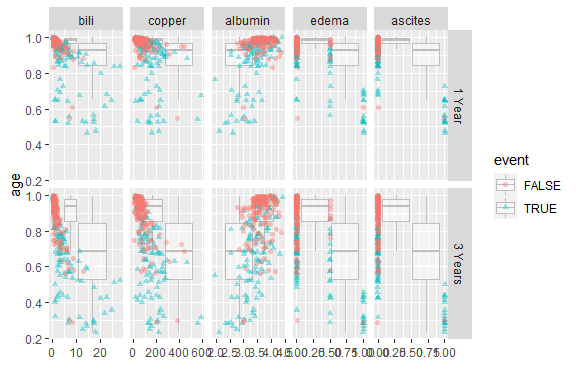
part\_var02 <- plot.variable(rfsrc\_pbc, xvar.names =   
c("copper","albumin"), partial = TRUE)



part\_var03 <- plot.variable(rfsrc\_pbc, xvar.names =   
c("chol","trig"), partial = TRUE)

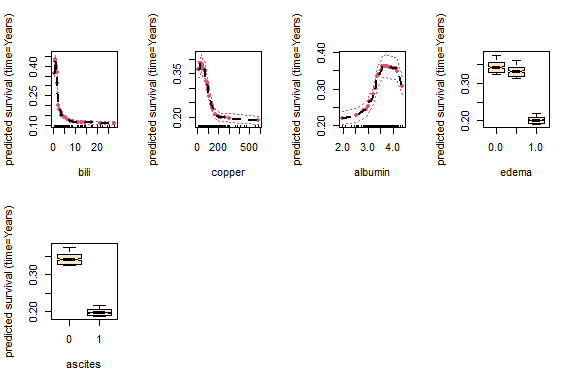
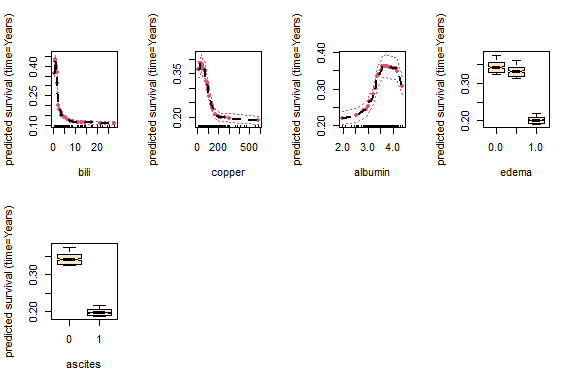
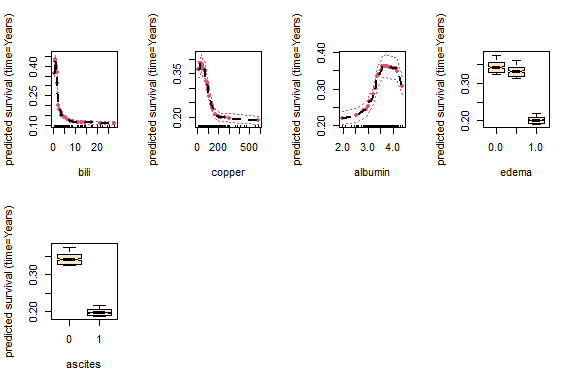


xvar <- ggMindepth$topvars   
plot(ggrf, xvar=xvar, panel=TRUE, alpha=.4) + labs(y="age", x="")

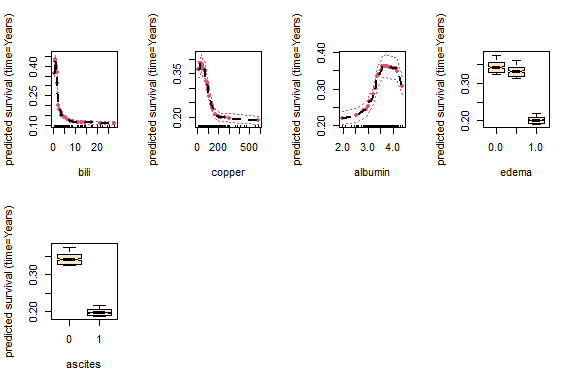


Now, we calculate the 1, 3 and 5 year partial dependence

partial\_pbc <- lapply(c(1,3,5), function(tm){   
 plot.variable(rfsrc\_pbc, surv.type = "surv",   
 time = "Years",   
 xvar.names = xvar,   
 partial = TRUE,   
 show.plots = TRUE)   
})



plot.variable(rfsrc\_pbc, surv.type = "surv",   
 time = "Years",   
 xvar.names = xvar,   
 partial = TRUE,   
 show.plots = TRUE)



Recall that presence of edema has three values representing categorical flags:

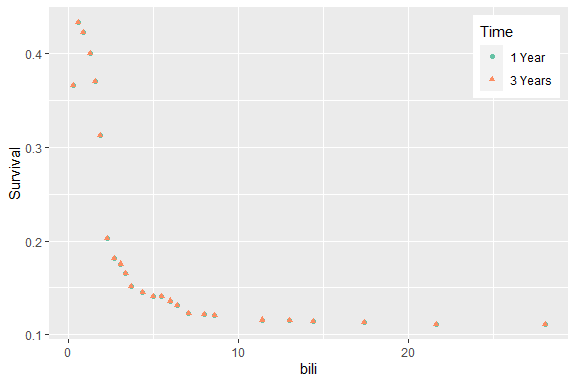
* 0.0 = no edema and no diuretic therapy for edema;
* 0.5 = edema present without diuretics, or edema resolved by diuretics;
* 1.0 = edema despite diuretic therapy

The distributions for these categories are surprisingly similar and we might need to investigate the physiological and pathophysiological aspect. Next, we convert all partial plots to gg\_partial objects.

gg\_dta <- lapply(partial\_pbc, gg\_partial)

Now, we combine the objects to get multiple time curves along variables on a single figure.

gg\_dta <- lapply(partial\_pbc, gg\_partial)   
pbc\_ggpart <- combine.gg\_partial(gg\_dta[[1]], gg\_dta[[2]], lbls = c("1 Year", "3 Years"))   
#pbc\_ggpart2 <- combine.gg\_partial(ggRandomForests::gg\_partial(gg\_dta[[1]], gg\_dta[[2]]), lbls = c("1 Year", "3 Years"))   
plot(pbc\_ggpart[["bili"]]) +   
 theme(legend.position = c(.9, .85)) +   
 labs(y = "Survival",   
 x = "bili",   
 color = "Time", shape = "Time") +   
 scale\_color\_brewer(palette = "Set2")

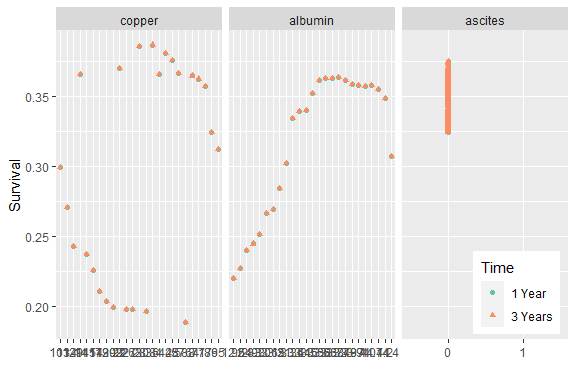


Let’s create a temporary holder and remove the stage and edema data.

ggpart <- pbc\_ggpart   
ggpart$edema <- ggpart$stage <- NULL   
ggpart$bili <- ggpart$sgot <- ggpart$chol <- NULL   
ggpart$platelet <- ggpart$trig <- ggpart$alk <- NULL

We also panel plot the remainder.

plot(ggpart, panel = TRUE) +   
 labs(x = "", y = "Survival", color = "Time", shape = "Time") +   
 scale\_color\_brewer(palette = "Set2") +   
 theme(legend.position = c(.9, .15))



Now, we generate the partial dependence plot for stage and edema by survival rates at 1-year and 3-year marks.

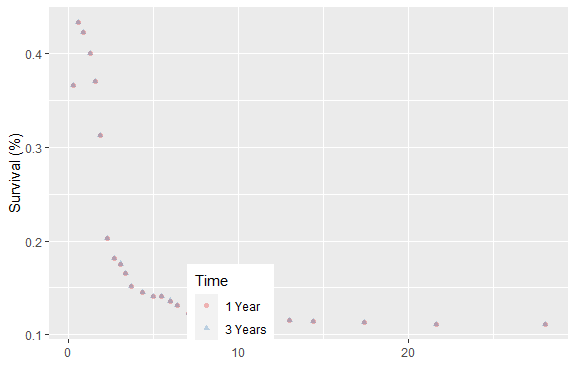
ggpart <- pbc\_ggpart   
head(ggpart$edema)

## yhat edema group  
## 1 0.3247720 0 1 Year  
## 2 0.3435546 0 1 Year  
## 3 0.3354199 0 1 Year  
## 4 0.3393505 0 1 Year  
## 5 0.3317216 0 1 Year  
## 6 0.3549816 0 1 Year

ggpart$stage

## NULL

names(ggpart) <- c("edema", "stage")   
class(ggpart) <- c("gg\_partial\_list", class(ggpart))   
plot(ggpart$edema, panel=TRUE, notch = TRUE, alpha = .3) +   
 labs(x = "", y = "Survival (%)", color="Time", shape="Time") +   
 scale\_color\_brewer(palette = "Set1") +   
 theme(legend.position = c(.35, .1))



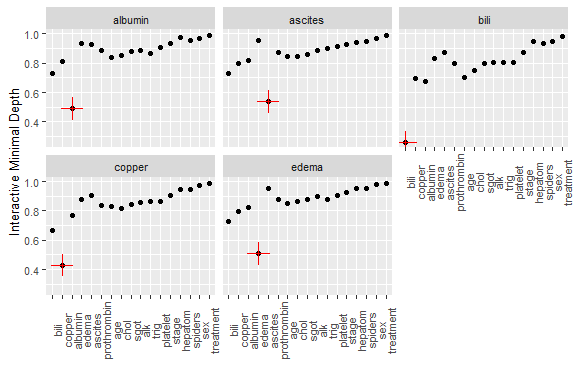
The gg\_interaction() function wraps the find.interaction() matrix for use with the provided S3 plot and print functions.

interaction\_pbc <- find.interaction(rfsrc\_pbc)

##   
## Method: maxsubtree  
## No. of variables: 17  
## Variables sorted by minimal depth?: TRUE  
##   
## bili copper albumin edema ascites prothrombin age chol sgot alk  
## bili 0.26 0.70 0.68 0.83 0.87 0.80 0.70 0.75 0.80 0.81  
## copper 0.67 0.43 0.77 0.88 0.91 0.84 0.83 0.82 0.84 0.85  
## albumin 0.73 0.81 0.49 0.93 0.93 0.88 0.84 0.85 0.88 0.89  
## edema 0.73 0.80 0.82 0.51 0.95 0.88 0.85 0.86 0.88 0.90  
## ascites 0.73 0.79 0.82 0.95 0.54 0.88 0.85 0.85 0.86 0.89  
## prothrombin 0.83 0.90 0.88 0.95 0.96 0.62 0.91 0.90 0.92 0.92  
## age 0.84 0.90 0.88 0.97 0.97 0.93 0.62 0.90 0.91 0.92  
## chol 0.85 0.89 0.86 0.95 0.97 0.93 0.91 0.63 0.92 0.92  
## sgot 0.88 0.90 0.91 0.97 0.99 0.95 0.92 0.92 0.70 0.94  
## alk 0.90 0.93 0.92 0.98 0.99 0.96 0.94 0.94 0.94 0.75  
## trig 0.89 0.93 0.91 0.99 0.99 0.97 0.95 0.94 0.94 0.94  
## platelet 0.92 0.93 0.94 0.98 0.98 0.96 0.96 0.96 0.96 0.96  
## stage 0.89 0.92 0.93 0.97 0.99 0.96 0.94 0.94 0.93 0.95  
## hepatom 0.94 0.95 0.95 0.98 0.99 0.97 0.97 0.95 0.96 0.97  
## spiders 0.93 0.95 0.95 0.99 0.98 0.98 0.96 0.97 0.96 0.97  
## sex 0.96 0.98 0.98 0.99 1.00 0.98 0.98 0.98 0.98 0.98  
## treatment 1.00 1.00 1.00 1.00 1.00 1.00 0.99 1.00 1.00 1.00  
## trig platelet stage hepatom spiders sex treatment  
## bili 0.80 0.81 0.87 0.95 0.93 0.95 0.98  
## copper 0.86 0.86 0.91 0.95 0.94 0.97 0.99  
## albumin 0.87 0.90 0.93 0.98 0.95 0.97 0.99  
## edema 0.88 0.91 0.92 0.95 0.95 0.98 0.99  
## ascites 0.90 0.91 0.93 0.94 0.95 0.97 0.99  
## prothrombin 0.93 0.94 0.94 0.98 0.98 0.98 0.99  
## age 0.92 0.93 0.95 0.98 0.97 0.99 0.99  
## chol 0.94 0.92 0.95 0.97 0.98 0.98 0.99  
## sgot 0.93 0.94 0.97 0.98 0.98 0.98 1.00  
## alk 0.94 0.94 0.97 0.99 0.99 0.99 1.00  
## trig 0.75 0.94 0.97 0.99 0.98 0.98 0.99  
## platelet 0.95 0.75 0.97 0.99 0.98 0.99 1.00  
## stage 0.95 0.96 0.76 0.99 0.99 0.98 1.00  
## hepatom 0.96 0.98 0.98 0.87 1.00 0.99 1.00  
## spiders 0.96 0.97 0.98 0.99 0.87 0.99 1.00  
## sex 0.98 0.98 0.99 0.99 0.99 0.93 1.00  
## treatment 1.00 1.00 1.00 1.00 1.00 1.00 0.98

Now, we generate the interactive plots for selected variables, albumin, ascites, serum bilirubin, serum cholesterol, copper, edema, and prothrombin

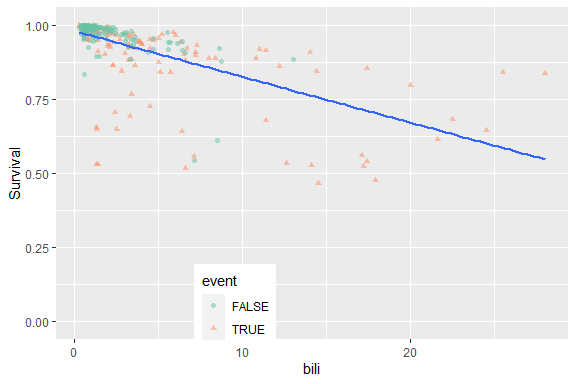
ggint <- gg\_interaction(interaction\_pbc)   
plot(ggint, xvar = xvar) +   
 labs(y = "Interactive Minimal Depth") +   
 theme(legend.position = "none")



# Conditional Dependence Plots

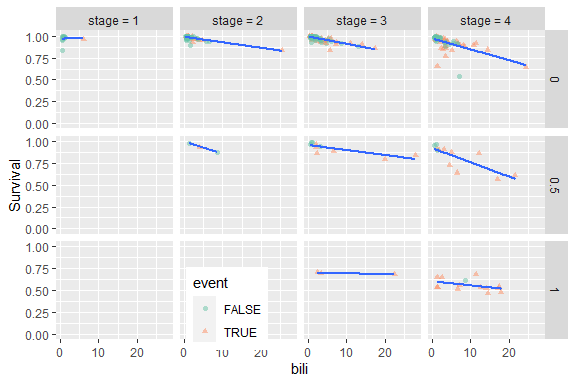
Now, we generate the conditional dependence plot for serum bilirubin with fitted line.

ggvar <- gg\_variable(rfsrc\_pbc, time = 1)   
ggvar$stage <- paste("stage = ", ggvar$stage, sep = "")   
var\_dep <- plot(ggvar, xvar = "bili",   
 method = "glm", alpha = .5, se = FALSE) +   
 labs(y = "Survival", x = "bili") +   
 theme(legend.position = c(.35, .1)) +   
 scale\_color\_brewer(palette = "Set2") +   
 scale\_shape(solid=TRUE) +   
 coord\_cartesian(y = c(-.01,1.01))  
var\_dep



We now show the conditional dependence of survival against bilirubin, versus other categorical covariates, say edema and stage.

var\_dep + facet\_grid(edema~stage)



Furthermore, we find intervals with similar number of observations.

copper\_cts <-quantile\_pts(ggvar$copper, groups = 6, intervals = TRUE)

Next, we create the conditional groups and add to the gg\_variable() object.

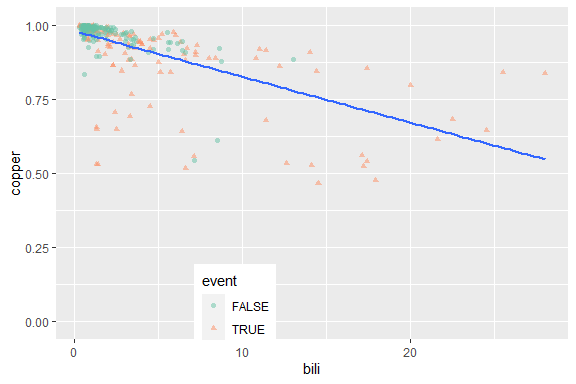
copper\_grp <- cut(ggvar$copper, breaks = copper\_cts)   
ggvar$copper\_grp <- copper\_grp

Finally, we adjust naming for facets before plotting.

levels(ggvar$copper\_grp) <- paste("copper = ",levels(copper\_grp), sep = "")

So, the variable dependency plot we generate is serum cholesterol versus serum bilirubin.

var\_dep <- plot(ggvar, xvar = "bili",   
 method = "glm", alpha = .5, se = FALSE) +   
 labs(y = "copper", x = "bili") +   
 theme(legend.position = c(.35, .1)) +   
 scale\_color\_brewer(palette = "Set2") +   
 scale\_shape(solid=TRUE) +   
 coord\_cartesian(y = c(-.01,1.01))  
var\_dep



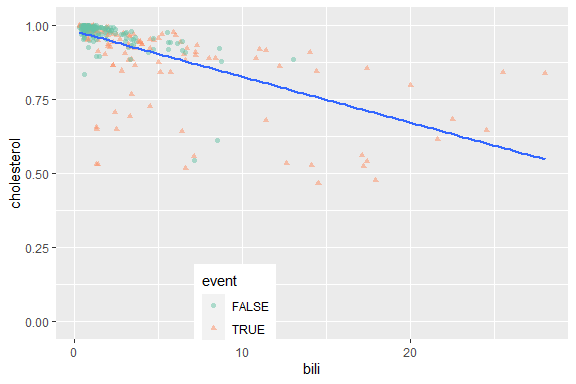
chol\_cts <-quantile\_pts(ggvar$chol, groups = 6, intervals = TRUE)   
chol\_grp <- cut(ggvar$chol, breaks = chol\_cts)   
ggvar$chol\_grp <- chol\_grp

# Adjust naming for facets

levels(ggvar$chol\_grp) <- paste("chol = ",levels(chol\_grp), sep = "")

# Plot.gg\_variable

var\_dep <- plot(ggvar, xvar = "bili",   
 method = "glm", alpha = .5, se = FALSE) +   
 labs(y = "cholesterol", x = "bili") +   
 theme(legend.position = c(.35, .1)) +   
 scale\_color\_brewer(palette = "Set2") +   
 scale\_shape(solid=TRUE) +   
 coord\_cartesian(y = c(-.01,1.01))  
var\_dep



### Partial dependence coplots

data(rfsrc\_pbc, package="ggRandomForests")

For the partial dependence coplots, we create the variable plot.

ggvar <- gg\_variable(rfsrc\_pbc, time = 1)

Then, we find intervals with similar number of observations.

copper\_cts <-quantile\_pts(ggvar$copper, groups = 6, intervals = TRUE)

# Create the conditional groups and add to the gg\_variable object

copper\_grp <- cut(ggvar$copper, breaks = copper\_cts)   
partial\_coplot\_pbc <- gg\_partial\_coplot(rfsrc\_pbc,   
 xvar = "bili",   
 groups = copper\_grp,   
 surv\_type = "surv",   
 time = 1,   
 show.plots = FALSE)

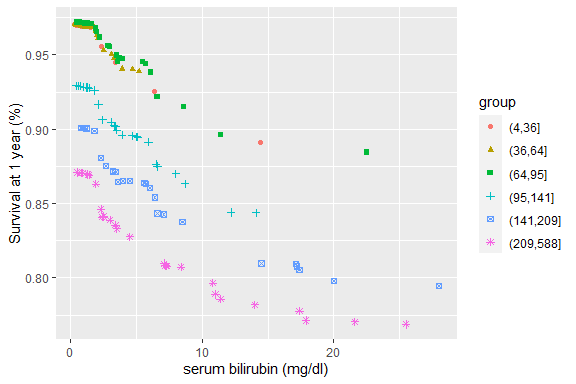
Now, we load the cached set.

data(partial\_coplot\_pbc, package="ggRandomForests")

# Partial Coplot

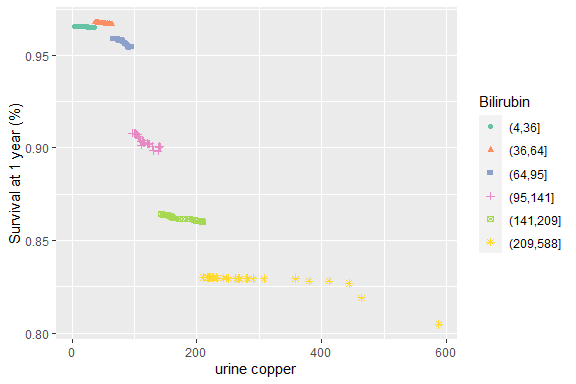
Finally, we display the partial dependence coplots by the conditional groups.

plot(partial\_coplot\_pbc) +   
 labs(x = "serum bilirubin (mg/dl)",   
 y = "Survival at 1 year (%)")



Next, we generate the plot of urine copper by serum bilirubin for 1-year survival percentage

partial\_coplot\_pbc <- gg\_partial\_coplot(rfsrc\_pbc,   
 xvar = "copper",   
 groups = copper\_grp,   
 surv\_type = "surv",   
 time = 1,   
 show.plots = FALSE)   
  
data(partial\_coplot\_pbc, package = "ggRandomForests")  
plot(partial\_coplot\_pbc) +   
 labs(x = "urine copper",   
 y = "Survival at 1 year (%)",   
 color = "Bilirubin",   
 shape = "Bilirubin") +   
 scale\_color\_brewer(palette = "Set2")



## Business Application - Banking

### Scenario and dataset

A marketing department of a bank runs various marketing campaigns for cross-selling products, improving customer retention and customer services. In this example, the bank wanted to cross-sell term deposit product to its customers. Contacting all customers is costly and does not create good customer experience. So, the bank wanted to build a predictive model which will identify customers who are more likely to respond to term deport cross sell campaign. We will use sample Marketing Data sample for building random forest based model using R.

### Read and Explore data

bank<-read.csv(file="./Banking.csv",header = T)   
bank<-data.frame(bank)  
str(bank)

## 'data.frame': 45212 obs. of 15 variables:  
## $ job : chr "management" "blue-collar" "unknown" "entrepreneur" ...  
## $ marital : chr "married" "married" "single" "divorced" ...  
## $ education: chr "tertiary" "unknown" "unknown" "tertiary" ...  
## $ age : int 58 47 33 42 58 58 57 57 25 39 ...  
## $ balance : int 2143 1506 1 2 121 71 162 52 50 255 ...  
## $ homeowner: int 1 1 0 1 1 1 1 1 1 1 ...  
## $ loans : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ default : int 0 0 0 1 0 0 0 0 0 0 ...  
## $ contact : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ length : int 261 92 198 380 50 71 174 38 342 296 ...  
## $ campaign : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ pdays : int -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...  
## $ previous : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ poutcome : int 2 2 2 2 2 2 2 2 2 2 ...  
## $ RESP : chr "NO" "NO" "NO" "NO" ...

names(bank)

## [1] "job" "marital" "education" "age" "balance" "homeowner"  
## [7] "loans" "default" "contact" "length" "campaign" "pdays"   
## [13] "previous" "poutcome" "RESP"

head(bank)

## job marital education age balance homeowner loans default contact  
## 1 management married tertiary 58 2143 1 0 0 0  
## 2 blue-collar married unknown 47 1506 1 0 0 0  
## 3 unknown single unknown 33 1 0 0 0 0  
## 4 entrepreneur divorced tertiary 42 2 1 0 1 0  
## 5 retired married primary 58 121 1 0 0 0  
## 6 technician married unknown 58 71 1 0 0 0  
## length campaign pdays previous poutcome RESP  
## 1 261 1 -1 0 2 NO  
## 2 92 1 -1 0 2 NO  
## 3 198 1 -1 0 2 NO  
## 4 380 1 -1 0 2 NO  
## 5 50 1 -1 0 2 NO  
## 6 71 1 -1 0 2 NO

Input dataset has 20 independent variables and a target variable. The target variable y is binary.

table(bank$RESP)/nrow(bank)

##   
## NO YES   
## 0.8830178 0.1169822

Eleven % of the observations has target variable “yes” and remaining 89% observations take value “no”.

Now, we will split the data sample into development and validation samples.

sample.ind <- sample(2, nrow(bank), replace = T,prob = c(0.6,0.4))

#sample.ind <- sample(2, 10000, replace=F)

library(forcats)  
bank.train <- bank[sample.ind==1,]  
bank.test <- bank[sample.ind==2,]  
table(bank.train$RESP)/nrow(bank.train)

##   
## NO YES   
## 0.8844077 0.1155923

table(bank.test$RESP)/nrow(bank.test)

##   
## NO YES   
## 0.880914 0.119086

bank.train$RESP=as\_factor(bank.train$RESP)  
bank.test$RESP=as\_factor(bank.test$RESP)  
bank.test$RESP

## [1] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [19] NO NO NO NO NO NO NO NO NO NO YES NO NO NO NO NO NO NO   
## [37] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [55] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [73] NO NO NO NO NO NO NO NO NO NO NO YES NO NO NO NO NO NO   
## [91] NO NO NO NO YES NO NO YES NO NO NO NO NO NO NO NO NO NO   
## [109] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [127] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [145] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [163] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [181] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
## [199] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO   
...

## [17839] NO YES NO YES YES NO YES NO NO YES YES NO YES YES YES YES YES YES  
## [17857] YES YES YES NO YES YES NO YES NO NO NO NO NO NO NO NO NO NO   
## [17875] NO YES YES YES NO NO NO YES NO NO NO NO YES NO NO NO NO NO   
## [17893] NO NO NO NO NO NO NO NO YES NO YES NO YES NO YES NO NO NO   
## [17911] YES YES NO YES YES NO YES YES NO NO NO NO NO YES YES YES NO NO   
## [17929] NO NO NO NO NO YES YES NO NO NO NO YES NO NO NO YES NO NO   
## [17983] YES NO YES YES YES  
## Levels: NO YES

Both development and validation samples have similar target variable distribution. This is just a sample validation.

If target variable is factor, classification decision tree is built. We can check the type of response variable.

class(bank.train$RESP)

## [1] "factor"

class(bank.test$RESP)

## [1] "factor"

Class of target or response variable is factor, so a classification Random Forest will be built. The current data frame has a list of independent variables, so we can make it formula and then pass as a parameter value for randomForest.

### Make Formula

varNames <- names(bank.train)

# Exclude ID or Response variable

varNames <- varNames[!varNames %in% c("RESP")]

# add + sign between exploratory variables

varNames1 <- paste(varNames, collapse = "+")

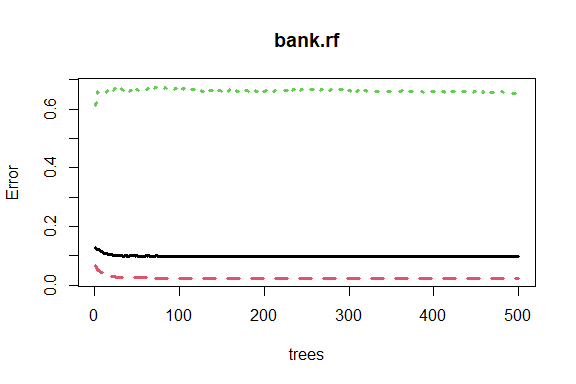
# Add response variable and convert to a formula object

rf.form <- as.formula(paste("RESP", varNames1, sep = " ~ "))

# Building Random Forest using R

Now, we have a sample data and formula for building Random Forest model. Let’s build 500 decision trees using Random Forest and plot the outcome.

bank.rf <- randomForest(rf.form, bank.train, ntree=500,importance=T)   
par(mfrow = c(1, 1))  
plot(bank.rf, lwd = 3)



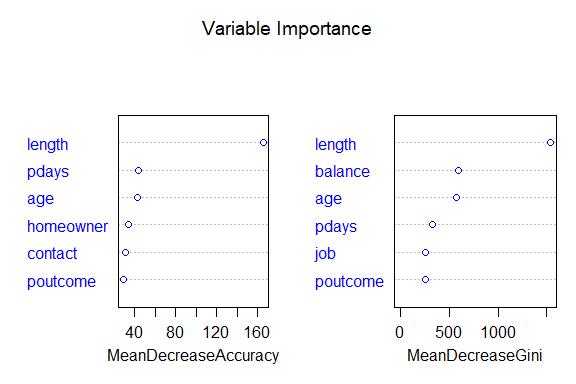
500 decision trees or a forest has been built using the random forest algorithm-based learning. We can plot the error rate across decision trees. The plot seems to indicate that after 100 decision trees, there is not a significant reduction in error rate.

YES and NO are lines for error in prediction for that specific label, and OOB (our first column) is simply the weighted-average of the two. As the number of trees increase, your OOB error gets lower because you get a better prediction from more trees.

### Variable Importance Plot

Variable importance plot is also a useful tool and can be plotted using varImpPlot function. Top 5 variables are selected and plotted based on Model Accuracy and Gini value. We can also get a table with decreasing order of importance based on a measure (1 for model accuracy and 2 node impurity).

varImpPlot(bank.rf, sort = T,   
 main="Variable Importance", n.var = 6,  
 lwd = 2, col = "blue")



# Variable Importance Table

var.imp <- data.frame(importance(bank.rf, type=2))

Here, we make row names as columns:

var.imp$Variables <- row.names(var.imp)   
var.imp[order(var.imp,decreasing = T),]

## MeanDecreaseGini Variables  
## NA NA <NA>  
## NA.1 NA <NA>  
## NA.2 NA <NA>  
## NA.3 NA <NA>  
## NA.4 NA <NA>  
## NA.5 NA <NA>  
## NA.6 NA <NA>  
## NA.7 NA <NA>  
## NA.8 NA <NA>  
## NA.9 NA <NA>  
## NA.10 NA <NA>  
## NA.11 NA <NA>  
## NA.12 NA <NA>  
## NA.13 NA <NA>  
## length 1538.88126 length  
## balance 593.76169 balance  
## age 574.24512 age  
## pdays 327.82808 pdays  
## job 258.00680 job  
## poutcome 256.01278 poutcome  
## campaign 206.26769 campaign  
## previous 161.56182 previous  
## homeowner 152.42646 homeowner  
## education 143.65515 education  
## contact 125.26639 contact  
## marital 116.94263 marital  
## loans 54.81676 loans  
## default 11.17271 default

Based on random forest variable importance, the variables could be selected for any other predictive modelling techniques or machine learning. Now, we want to measure the accuracy of the Random Forest model. Some of the other model performance statistics are: - KS - Lift Chart - ROC curve

### Predict Response Variable Value using Random Forest

Generic predict function can be used for predicting response variable using Random Forest object. Here we use a predicting response variable:

bank.train$predicted.response <- predict(bank.rf, bank.train)

### Confusion Matrix

confusionMatrix() function from caret package can be used for creating confusion matrix based on actual response variable and predicted value.

# Load Library or packages

library(e1071)  
library(caret)  
#if(!require(e1071)) install.packages("e1071")   
#if(!require(caret)) install.packages("caret")

# Create Confusion Matrix

confusionMatrix(bank.train$predicted.response, bank.train$RESP)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction NO YES  
## NO 24078 807  
## YES 0 2340  
##   
## Accuracy : 0.9704   
## 95% CI : (0.9683, 0.9723)  
## No Information Rate : 0.8844   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8368   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 1.0000   
## Specificity : 0.7436   
## Pos Pred Value : 0.9676   
## Neg Pred Value : 1.0000   
## Prevalence : 0.8844   
## Detection Rate : 0.8844   
## Detection Prevalence : 0.9140   
## Balanced Accuracy : 0.8718   
##   
## 'Positive' Class : NO   
##

# Predicting response variable

library(ModelMetrics)  
truth <- bank.test$RESP  
pred <- predict(bank.rf, bank.test)  
mse(pred, truth)

## [1] 0.101351

It has accuracy of 99.81%, which is fantastic. Now we can predict response for the validation sample and calculate model accuracy for the sample.

# Create Confusion Matrix

confusionMatrix(pred, truth)

## [,1] [,2]  
## [1,] 0 0  
## [2,] 0 16896

Accuracy level has dropped to 91.4% but still significantly higher. We can perform a little more thorough exploration, but we really need to see how a random forest works.

## How random forest works

Each tree is grown as follows:

1. Random Record Selection : Each tree is trained on roughly 2/3rd of the total training data (exactly 63.2%) . Cases are drawn at random with replacement from the original data. This sample will be the training set for growing the tree.
2. Random Variable Selection : Some predictor variables (say, m) are selected at random out of all the predictor variables and the best split on these m is used to split the node. By default, m is square root of the total number of all predictors for classification. For regression, m is the total number of all predictors divided by 3. The value of m is held constant during the forest growing.

Note : In a standard tree, each split is created after examining every variable and picking the best split from all the variables.

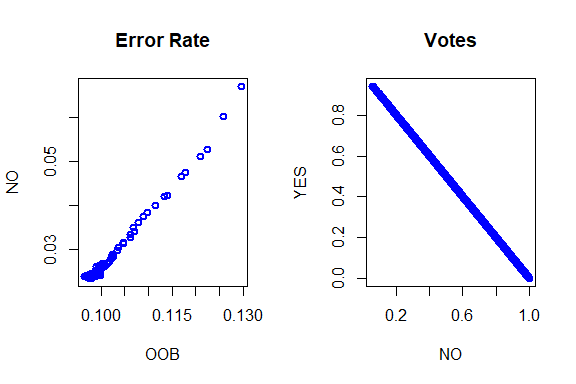
1. For each tree, using the leftover (36.8%) data, calculate the misclassification rate - out of bag (OOB) error rate. Aggregate error from all trees to determine overall OOB error rate for the classification. If we grow 200 trees then on average a record will be OOB for about .37\*200=74 trees.
2. Each tree gives a classification on leftover data (OOB), and we say the tree “votes” for that class. The forest chooses the classification having the most votes over all the trees in the forest. For a binary dependent variable, the vote will be YES or NO, count up the YES votes. This is the RF score and the percent YES votes received is the predicted probability. In regression case, it is average of dependent variable.

For example, suppose we fit 500 trees, and a case is out-of-bag in 200 of them: - 160 trees votes class 1 - 40 trees votes class 2

In this case, RF score is class1. Probability for that case would be 0.8 which is 160/200. Similarly, it would be an average of target variable for regression problem.

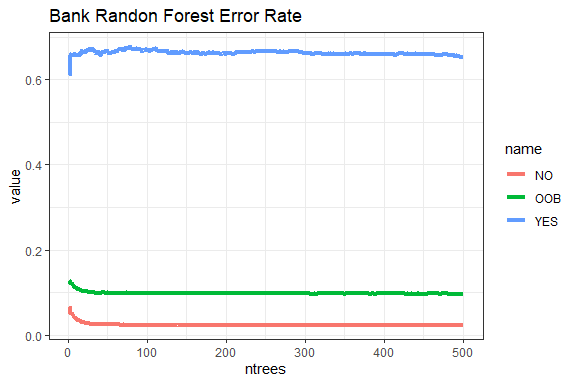
Having this knowledge, We can also gain insight by plotting error rates and votes.

par(mfrow = c(1, 2))  
plot(bank.rf$err.rate, main = "Error Rate", lwd = 2, col = "blue")  
plot(bank.rf$votes, main = "Votes", lwd = 2, col = "blue")



The pivot\_longer() “lengthens” data, increasing the number of rows and decreasing the number of columns. We can use it to plot the error rate with the different values of the reponse.

library(tidyr)  
plotdf <- pivot\_longer(data.frame(ntrees=1:nrow(bank.rf$err.rate),bank.rf$err.rate),-ntrees)  
ggplot(plotdf,aes(x=ntrees,y=value,col=name)) +   
 geom\_line(lwd=1.5) + ggtitle("Bank Randon Forest Error Rate") + theme\_bw()



NO and YES are lines for error in prediction for that specific label, and OOB (your first column) is simply the average of the two. As the number of trees increase, your OOB error gets lower because you get a better prediction from more trees.

## Fit a Random Forest to the fgl data and compare with SVM

The fgl data frame has 214 rows and 10 columns. It was collected by B. German on fragments of glass collected in forensic work.

#if(!require(MASS)) install.packages("MASS")   
library(MASS)   
data(fgl)   
set.seed(17)   
fgl.rf <- randomForest(type ~ ., data = fgl, mtry = 2,   
 importance = TRUE, do.trace = 100)

## ntree OOB 1 2 3 4 5 6  
## 100: 20.09% 14.29% 18.42% 58.82% 23.08% 22.22% 13.79%  
## 200: 21.03% 10.00% 22.37% 64.71% 23.08% 33.33% 13.79%  
## 300: 18.69% 10.00% 18.42% 64.71% 23.08% 11.11% 13.79%  
## 400: 19.16% 11.43% 17.11% 64.71% 23.08% 22.22% 13.79%  
## 500: 19.63% 11.43% 19.74% 58.82% 23.08% 22.22% 13.79%

print(fgl.rf)

##   
## Call:  
## randomForest(formula = type ~ ., data = fgl, mtry = 2, importance = TRUE, do.trace = 100)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 19.63%  
## Confusion matrix:  
## WinF WinNF Veh Con Tabl Head class.error  
## WinF 62 6 2 0 0 0 0.1142857  
## WinNF 11 61 1 1 1 1 0.1973684  
## Veh 7 3 7 0 0 0 0.5882353  
## Con 0 2 0 10 0 1 0.2307692  
## Tabl 0 2 0 0 7 0 0.2222222  
## Head 1 3 0 0 0 25 0.1379310

We can compare random forests with support vector machines by doing ten repetitions of 10-fold cross-validation, using the errorest functions in the ipred package. Errorest() performs resampling based estimates of prediction error: misclassification error, root mean squared error or Brier score for survival data.

#if(!require(ipred)) install.packages("ipred")   
library(ipred)   
set.seed(131)   
error.RF <- numeric(10)   
for(i in 1:10)   
 error.RF[i] <- errorest(type ~ ., data = fgl,   
 model = randomForest, mtry = 2)$error   
summary(error.RF)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.1729 0.1869 0.1963 0.1967 0.2091 0.2150

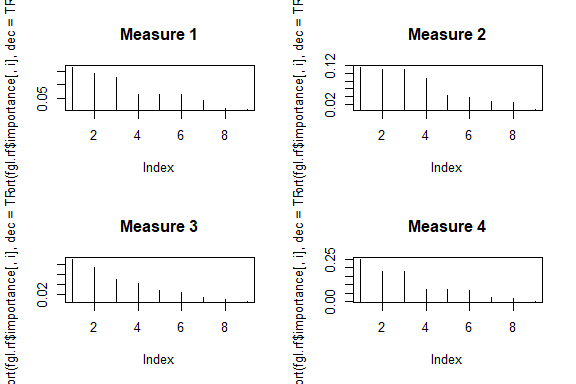
# SVM

set.seed(563)   
error.SVM <- numeric(10)   
for (i in 1:10)   
 error.SVM[i] <- errorest(type ~ ., data = fgl,   
 model = svm, cost = 10, gamma = 1.5)$error   
summary(error.SVM)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.3505 0.3657 0.3762 0.3762 0.3820 0.4112

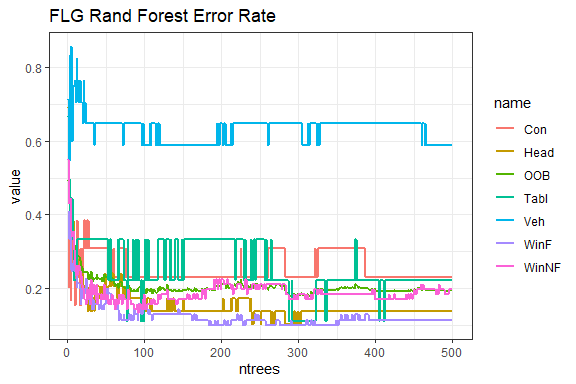
We see that the random forest compares quite favorably with SVM. We have found that the variable importance measures produced by random forests can sometimes be useful for model reduction

par(mfrow = c(2, 2))   
for (i in 1:4)   
 plot(sort(fgl.rf$importance[,i], dec = TRUE),   
 type = "h", main = paste("Measure", i))



Now we can also check the error rate of the FLG random forest for all its variables.

plotdf <- pivot\_longer(data.frame(ntrees=1:nrow(fgl.rf$err.rate),fgl.rf$err.rate),-ntrees)  
ggplot(plotdf,aes(x=ntrees,y=value,col=name)) +   
 geom\_line(lwd=1) + ggtitle("FLG Rand Forest Error Rate") + theme\_bw()



The legend shows that the variables are the colored lines for error in prediction (except the green line) for that specific label, and OOB (the green line) is simply the average of the all the variables. As the number of trees increase, your OOB error gets lower because you get a better prediction from more trees.

We can compare random forests with support vector machines by doing ten repetitions of 10-fold cross-validation, using the errorest functions in the ipred package. errorest performs resampling, based estimates of prediction error: misclassification error, root mean squared error or Brier score for survival data.

library(ipred)   
  
set.seed(131)  
error.RF <- numeric(10)  
for(i in 1:10) error.RF[i] <- errorest(type ~ .,   
data = fgl, model = randomForest, mtry = 2)$error  
summary(error.RF)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.1729 0.1869 0.1963 0.1967 0.2091 0.2150

set.seed(563)  
error.SVM <- numeric(10)  
for (i in 1:10) error.SVM[i] <- errorest(type ~ ., data = fgl, model = svm, cost = 10, gamma = 1.5)$error  
summary(error.SVM)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.3505 0.3657 0.3762 0.3762 0.3820 0.4112

We see that the random forest compares quite favorably with SVM. We have found that the variable importance measures produced by random forests can sometimes be useful for model reduction.