Random forest exercises

Helene Charlotte Rytgaard and Mark Bech Knudsen

October, 2025

Contents

0 Loading the randomForestSRC package and the Epo data

If this is the first time you use the package, you need to install it. You do this by running the line below:

```
install.packages("randomForestSRC")
```

```
Installing package into '/home/helene/R/x86_64-pc-linux-gnu-library/3.6'
(as 'lib' is unspecified)
trying URL 'http://cran.fhcrc.org/src/contrib/randomForestSRC_2.9.3.tar.gz'
Content type 'application/x-gzip' length 1129754 bytes (1.1 MB)
==================================================
downloaded 1.1 MB

* installing *source* package 'randomForestSRC' ...
** package 'randomForestSRC' successfully unpacked and MD5 sums checked
```

```
** using staged installation
checking for gcc... gcc -std=gnu99
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc -std=gnu99 accepts -g... yes
checking for gcc -std=gnu99 option to accept ISO C89... none needed
checking for gcc -std=gnu99 option to support OpenMP... -fopenmp
configure: creating ./config.status
config.status: creating src/Makevars
** libs
gcc -std=gnu99 -I"/usr/share/R/include" -DNDEBUG    -fopenmp  -fpic  -g -O2 -fdebug-prefix-map=/buil
gcc -std=gnu99 -I"/usr/share/R/include" -DNDEBUG    -fopenmp  -fpic  -g -O2 -fdebug-prefix-map=/buil
gcc -std=gnu99 -I"/usr/share/R/include" -DNDEBUG    -fopenmp  -fpic  -g -O2 -fdebug-prefix-map=/buil
gcc -std=gnu99 -shared -L/usr/lib/R/lib -Wl,-Bsymbolic-functions -Wl,-z,relro -o randomForestSRC.so
installing to /home/helene/R/x86_64-pc-linux-gnu-library/3.6/00LOCK-randomForestSRC/00new/randomFore
** R
** data
** inst
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
** checking absolute paths in shared objects and dynamic libraries
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (randomForestSRC)

The downloaded source packages are in
'/tmp/RtmpuPwdKe/downloaded_packages'
```

In any case you need to load the package by writing:

```
library("randomForestSRC")
```

The Epo data can be read into R by writing:

```
Epo <- read.csv("https://biostatistics.dk/teaching/advtopics/data/Epo.csv",
    stringsAsFactors=TRUE)
```

Likewise, the newpatient data are read into R by writing:

```
newpatient <-
  read.csv("https://biostatistics.dk/teaching/advtopics/data/newpatient.csv",
    stringsAsFactors=TRUE)
```

1 Exercise 1: From trees to forests

In this exercise, we will use the `rfsrc()` function from the `R`-package `randomForestSRC` to grow single trees on the `Epo` data. Next you will combine those trees to make forest predictions.

Follow the steps outlined below. If you have trouble running `R`-code, Section 1.1 shows an example of a solution to the last part; here you can also find the numbers to fill out the table of the first part.

1. Get the tree prediction for individual $i = 25$:
   - Set the seed.
   - Use `rfsrc()` to grow a forest with only one tree.
   - Get the oob prediction for individual $i = 25$.
2. Repeat 10 times and fill out the table below
   - Each time, set a new seed.
   - Use `rfsrc` to grow a forest with one tree.
   - Get the oob predictions for individual $i = 25$.

seed	prediction
1	
2	
3	
⋮	

3. A random forest is an algorithm that is based on many (can be 1000 or 10000) decision trees. Specifically, the random forest predictions are obtained as averages over the individual trees.
   - Compute your own "forest prediction" based on the 10 tree predictions you have computed (leave out NA's).
   - Increase the number of trees and plot the changing average.

1.1 Code for Exercise 1

The following code produces 1000 individual tree predictions for individual $i = 25$:

```
M <- 1000
pred <- rep(0, M)
for (ii in 1:M) {
    tree1 <- rfsrc(Y~age+sex+HbBase+Treat+Resection,
           Epo, ntree=1, seed=ii)
    pred[ii] <- tree1$predicted.oob[25]
}
```

Look at the first 10 predictions:

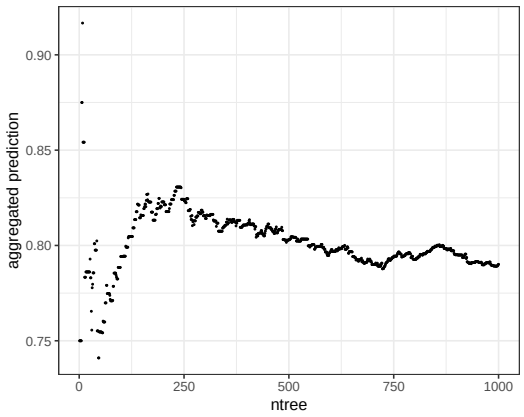
```
pred[1:10]
```

```
[1] 0.8  NA  NA  NA  NA 1.0  NA 1.0 1.0  NA
```

We see only see a number when individual $i = 25$ is oob (not included in the bootstrap sample used to grow that tree).

We can generate the result plot for the last part with the following code:

```
pred.mean <- sapply(1:M, function(ii) {
    mean(na.omit(pred[1:ii]))
})
plot(pred.mean)
```



2 Exercise 2: Get predictions for `newpatient`

In this exercise, we will use the `rfsrc()` function from the `randomForestSRC` package to get the forest prediction for `newpatient` for different values of hyperparameters.

Follow the steps outlined below. Section 2.1 shows an example of a solution; here you can also find numbers to fill out the tables.

1. Run the random forest on the `Epo` data and report the prediction for `newpatient`.

2. Run the random forest with new values for `ntree` and report again the prediction for `newpatient` (fix: nodesize=1, mtry=3). Repeat with a new seed and fill out the table:

seed	ntree=50	ntree=100	ntree=1000
1			
2			
3			

3. Run the random forest with other values of `nodesize` and report again the prediction for `newpatient` (fix: ntree=1000, mtry=3). Repeat with a new seed and fill out the table:

seed	nodesize=1	nodesize=3	nodesize=5	nodesize=10
1				
2				
3				

4. Run the random forest with other value of `mtry` and report again the prediction for `newpatient` (fix: ntree=1000, nodesize=1). Repeat with a new seed and fill out the table:

seed	mtry=1	mtry=3	mtry=6
1			
2			
3			

2.1 Code for Exercise 2

To compute the forest prediction for `newpatient` for different combinations of the hyperparameter, we first define the different combinations of hyperparameters that we are interested in. Here I define a hypergrid for all possible combinations of the hyperparameters considered in this exercise (and three different seed values):

```
hyper.grid.prediction <- expand.grid(
    mtry = c(1,3,6),
    nodesize = c(1,3,5,10),
    ntree=c(50, 100, 1000),
    seed=c(1,5,12),
    prediction.newpatient=NA
)
```

Then I make a loop to get the prediction for `newpatient` for all combinations:

```
for (j in 1:nrow(hyper.grid.prediction)) {
    tmp.forest <-
    rfsrc(Y~age+sex+HbBase+Treat+Resection,
          Epo,
          mtry=hyper.grid.prediction[j, "mtry"],
          nodesize=hyper.grid.prediction[j, "nodesize"],
          ntree=hyper.grid.prediction[j, "ntree"],
          seed=hyper.grid.prediction[j, "seed"])
    hyper.grid.prediction[j, "prediction.newpatient"] <-
    predict(tmp.forest, newdata=newpatient)$predicted
}
```

The results are as follows:

```
hyper.grid.prediction[order(hyper.grid.prediction$mtry,
                hyper.grid.prediction$nodesize,
                hyper.grid.prediction$ntree), ]
```

```
    mtry nodesize ntree seed prediction.newpatient
1      1        1    50    1              0.4188417
37     1        1    50    5              0.5586870
73     1        1    50   12              0.5412642
13     1        1   100    1              0.4939789
49     1        1   100    5              0.5328155
85     1        1   100   12              0.5789148
25     1        1  1000    1              0.5352921
61     1        1  1000    5              0.5340832
97     1        1  1000   12              0.5437999
4      1        3    50    1              0.4450314
40     1        3    50    5              0.5565447
76     1        3    50   12              0.5538213
16     1        3   100    1              0.5181860
52     1        3   100    5              0.5244570
88     1        3   100   12              0.5534081
28     1        3  1000    1              0.5317648
64     1        3  1000    5              0.5227657
100    1        3  1000   12              0.5230066
```

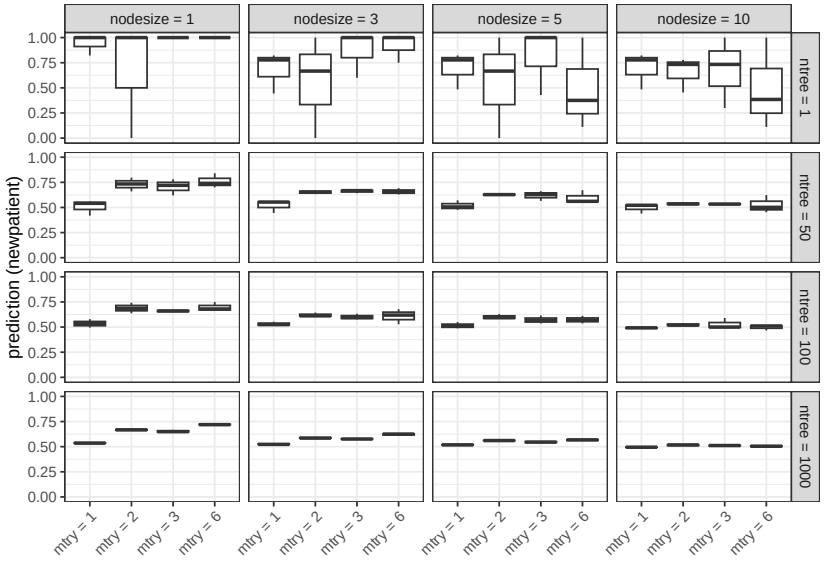
7	1	5	50	1	0.4758092
43	1	5	50	5	0.5708545
79	1	5	50	12	0.5049862
19	1	5	100	1	0.5086548
55	1	5	100	5	0.4870693
91	1	5	100	12	0.5496585
31	1	5	1000	1	0.5142464
67	1	5	1000	5	0.5250320
103	1	5	1000	12	0.5176816
10	1	10	50	1	0.4403129
46	1	10	50	5	0.5214462
82	1	10	50	12	0.5230330
22	1	10	100	1	0.4843340
58	1	10	100	5	0.4925440
94	1	10	100	12	0.4932428
34	1	10	1000	1	0.4944094
70	1	10	1000	5	0.4962791
106	1	10	1000	12	0.4947379
2	3	1	50	1	0.7200000
38	3	1	50	5	0.6200000
74	3	1	50	12	0.7800000
14	3	1	100	1	0.6500000
50	3	1	100	5	0.6600000
86	3	1	100	12	0.6700000
26	3	1	1000	1	0.6670625
62	3	1	1000	5	0.6495000
98	3	1	1000	12	0.6440000
5	3	3	50	1	0.6673333
41	3	3	50	5	0.6416667
77	3	3	50	12	0.6686667
17	3	3	100	1	0.5761667
53	3	3	100	5	0.5948333
89	3	3	100	12	0.6326667
29	3	3	1000	1	0.5739458
65	3	3	1000	5	0.5760500
101	3	3	1000	12	0.5800000
8	3	5	50	1	0.5657857
44	3	5	50	5	0.6283810
80	3	5	50	12	0.6618016
20	3	5	100	1	0.5361667
56	3	5	100	5	0.5648929
92	3	5	100	12	0.6148016
32	3	5	1000	1	0.5455931
68	3	5	1000	5	0.5394611
104	3	5	1000	12	0.5532052
11	3	10	50	1	0.5382526
47	3	10	50	5	0.5337879
83	3	10	50	12	0.5254880
23	3	10	100	1	0.4947090
59	3	10	100	5	0.5000131

95	3	10	100	12	0.5905100
35	3	10	1000	1	0.5130352
71	3	10	1000	5	0.5057267
107	3	10	1000	12	0.5115004
3	6	1	50	1	0.8400000
39	6	1	50	5	0.7000000
75	6	1	50	12	0.7400000
15	6	1	100	1	0.6600000
51	6	1	100	5	0.7500000
87	6	1	100	12	0.6800000
27	6	1	1000	1	0.7190000
63	6	1	1000	5	0.7260000
99	6	1	1000	12	0.7180000
6	6	3	50	1	0.6920000
42	6	3	50	5	0.6286667
78	6	3	50	12	0.6553333
18	6	3	100	1	0.5283333
54	6	3	100	5	0.6775000
90	6	3	100	12	0.6193333
30	6	3	1000	1	0.6089500
66	6	3	1000	5	0.6251833
102	6	3	1000	12	0.6302833
9	6	5	50	1	0.5610079
45	6	5	50	5	0.5534444
81	6	5	50	12	0.6711587
21	6	5	100	1	0.5362341
57	6	5	100	5	0.6117579
93	6	5	100	12	0.5698730
33	6	5	1000	1	0.5579921
69	6	5	1000	5	0.5713698
105	6	5	1000	12	0.5677782
12	6	10	50	1	0.4526877
48	6	10	50	5	0.5009685
84	6	10	50	12	0.6232897
24	6	10	100	1	0.4659117
60	6	10	100	5	0.5163693
96	6	10	100	12	0.5105581
36	6	10	1000	1	0.4985670
72	6	10	1000	5	0.5047324
108	6	10	1000	12	0.5119349

Rather than looking at the results in a table, we here plot the predictions for `newpatient` for all combinations:

```
library(ggplot2)
ggplot() + theme_bw() +
  facet_grid(ntree~nodesize) +
  geom_boxplot(data=hyper.grid.prediction, aes(y=prediction.newpatient,
                           x=factor(mtry),
                           group=mtry)) +
  xlab("") + ylab("prediction (newpatient)") +
  theme(legend.position="bottom") +
```

```
  labs(shape="mtry")
```



3 Exercise 3: Identifying risk factors

In this exercise we consider the paper:

Identifying Important Risk Factors for Survival in Patient With Systolic Heart Failure Using Random Survival Forests

Eileen Hsich, MD; Eiran Z. Gorodeski, MD, MPH; Eugene H. Blackstone, MD;
Hemant Ishwaran, PhD; Michael S. Lauer, MD

Background—Heart failure survival models typically are constructed using Cox proportional hazards regression. Regression modeling suffers from a number of limitations, including bias introduced by commonly used variable selection methods. We illustrate the value of an intuitive, robust approach to variable selection, random survival forests (RSF), in a large clinical cohort. RSF are a potentially powerful extensions of classification and regression trees, with lower variance and bias.

Methods and Results—We studied 2231 adult patients with systolic heart failure who underwent cardiopulmonary stress testing. During a mean follow-up of 5 years, 742 patients died. Thirty-nine demographic, cardiac and noncardiac comorbidity, and stress testing variables were analyzed as potential predictors of all-cause mortality. An RSF of 2000 trees was constructed, with each tree constructed on a bootstrap sample from the original cohort. The most predictive variables were defined as those near the tree trunks (averaged over the forest). The RSF identified peak oxygen consumption, serum urea nitrogen, and treadmill exercise time as the 3 most important predictors of survival. The RSF predicted survival similarly to a conventional Cox proportional hazards model (out-of-bag C-index of 0.705 for RSF versus 0.698 for Cox proportional hazards model).

Conclusions—An RSF model in a cohort of patients with heart failure performed as well as a traditional Cox proportional hazard model and may serve as a more intuitive approach for clinicians to identify important risk factors for all-cause mortality. **(*Circ Cardiovasc Qual Outcomes*. 2011;4:39-45.)**

Key Words: heart failure ■ prognosis ■ statistics ■ survival analyses

1. Read page 39–40 of the paper
   - What is the aim of the paper?
   - What method did they use?
2. Read the Results section (p. 41f)
   - What are the results of the analysis?
   - How are the results presented? (See Figures p. 44)

4 Exercise 4: Predicting tumor class

In this practical we will work with a dataset containing information on 38 tumor mRNA samples from 38 individuals and the gene expression values from 3051 genes. The data comes from the leukemia microarray study[1].

The dataset is loaded into `R` as follows:

```
tumor.data <- read.csv("https://biostatistics.dk/teaching/advtopics/data/tumor.csv")
```

The first column `y` in the dataframe indicates the tumor class, 27 acute lymphoblastic leukemia (ALL) cases (code 0) and 11 acute myeloid leukemia (AML) cases (code 1). The remaining 3051 columns are numeric vectors encoding gene expression levels. We want to predict the tumor class.

4.1 Applying a random forest to the data

Follow the steps outlined below to analyze the data; you can also find example solutions in the form of `R`-code in the following subsections.

1. Try to grow a forest using all the gene expressions in the data.
2. Patient 1 and patient 28 correspond to two different tumor types. Look at their (oob) predictions. If we classify a patient to tumor class 1 if the oob prediction is greater than 0.5 and class 0 otherwise, how many samples are classified correctly?
3. Set a new seed and grow a new forest. Get the two predictions for individuals 1 and 28. How do they change?
4. Compute the forest prediction for patient 28 for different values of hyperparameters and for different random seeds. Comment on these results.
5. Compute (or extract) the estimated error rates of the forest models for different values of hyperparameters. Comment on these results.

You can use the following loss function:

```
loss.fun <- function(Y, Phat) mean((Y-Phat)^2)
```

To compute the estimated OOB error rate for a particular forest (here named `forest`) as follows:

```
loss.fun(tumor.data$y, forest$predicted.oob)
```

6. Continue with the (tuned) random forest model, corresponding to the one minimizing the estimated error rate.
7. Get the predicted values from the tuned forest for patient 1 and patient 28.
8. Compute the VIMP measures for the predictor variables of the tuned forest. Which variables seem important with this method?
9. Compute the minimal depth of each variable (you may use the code below to do this; note the `tuned.forest` is the name of the tuned forest object) and conclude what variables are important using this method.

[1] T. R. Golub et al., Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. Science 286, 531-537(1999). DOI: 10.1126/science.286.5439.531

```
# --- this code computes the minimal depth:
md.obj <- max.subtree(tuned.forest, max.order=0)
md <- sapply(1:dim(md.obj$order)[1], function(i) mean(md.obj$order[i, ]))
names(md) <- names(md.obj$order[, 1])
#---  The threshold value can be extracted as follows:
md.obj$threshold
```

10. Produce PDP plots for the 9 most and the 9 least important predictor variables (you decide if you evaluate importance based on VIMP or minimal depth) to look at how prediction depends on their values.

11. Use the code below to produce the ICE plot for the variable `x.896`. Here we have colored the curves according to values of the variable `x.2124`. Can you see differing patterns across individuals?

```
ice.grid <- expand.grid(
  x.896=seq(min(tumor.data$x.896), max(tumor.data$x.896), length=200), pred=NA
)

ice.dat <- do.call("rbind", lapply(1:nrow(tumor.data), function(i) {
  tmp <- ice.grid
  tmp$id <- i

  for (xvar in colnames(tumor.data[,-1])) {
    if (xvar!="x.896")
      tmp[, xvar] <- tumor.data[i, xvar]
  }

  tmp$pred <- predict(tuned.forest.tumor, newdata=tmp, type="response")$predicted
  return(tmp)
}))

library(ggplot2)
plot.ice <-
  ggplot() + theme_bw() +
  geom_line(data=ice.dat, aes(x=x.896, y=pred, group=id, col=x.2124)) +
  xlab("x.896") + ylab("Predicted probability")
print(plot.ice)
```

4.2 Getting forest predictions

1. Try to grow a forest using all the gene expressions in the data.

We apply the forest with 500 trees and otherwise default values of hyperparameters as follows:

```
forest1.tumor <- rfsrc(y~., tumor.data, ntree=500, seed=1)
```

(Note that the period "`.`" in the formula "`y~.`" above means that *all variables of the dataset are used in the formula* (except "`y`"). Thus it allows us to include all predictors, without having to specify their names).

We can look of the summary of the model:

```
forest1.tumor
```

```
                       Sample size: 38
                  Number of trees: 500
        Forest terminal node size: 5
    Average no. of terminal nodes: 2.072
No. of variables tried at each split: 1017
          Total no. of variables: 3051
    Resampling used to grow trees: swor
 Resample size used to grow trees: 24
                        Analysis: RF-R
                          Family: regr
                  Splitting rule: mse *random*
    Number of random split points: 10
                  (OOB) R squared: 0.74437694
(OOB) Requested performance error: 0.05399719
```

2. Patient 1 and patient 28 correspond to two different tumor types. Look at their (oob) predictions. If we classify a patient to tumor class 1 if the oob prediction is greater than 0.5 and class 0 otherwise, how many samples are classified correctly?

```
forest1.tumor$predicted.oob[1]
forest1.tumor$predicted.oob[28]
```

```
[1] 0.08263242
[1] 0.4651665
```

We can get a quick look at how many samples that are classified correctly as follows:

```
table(classification=ifelse(forest1.tumor$predicted.oob>0.5, 1, 0),
      observed=tumor.data$y)
```

```
              observed
classification  0  1
             0 27  1
             1  0 10
```

3. Set a new seed and grow a new forest. Get the two predictions for individuals 1 and 28. How do they change?

```
forest2.tumor <- rfsrc(y~.,
               tumor.data, ntree=500, seed=2)
```

```
forest2.tumor$predicted.oob[1]
forest2.tumor$predicted.oob[28]
```

```
[1] 0.06857844
[1] 0.4427403
```

We can again get a quick look at how many samples that are classified correctly as follows:

```
table(classification=ifelse(forest2.tumor$predicted.oob>0.5, 1, 0),
      observed=tumor.data$y)
```

```
              observed
classification  0  1
             0 27  1
             1  0 10
```

4.3 Growing a forest for different values of hyperparameters

4. In the output of the code below, we look at the forest prediction for patient 28 for different values of hyperparameters and for different random seeds. Comment on these results.

We define the hypergrid of hyperparameter values:

```
hyper.grid.prediction <- expand.grid(
  mtry = floor((ncol(tumor.data)-1)/c(12,8,4,3,2)),
  ntree = c(100, 1000),
  nodesize = c(1,3,5,10),
  seed = c(2, 1240, 1919133, 111099, 867),
  prediction = NA
)
```

We get the oob prediction for patient 28 for each combination of hyperparameter values:

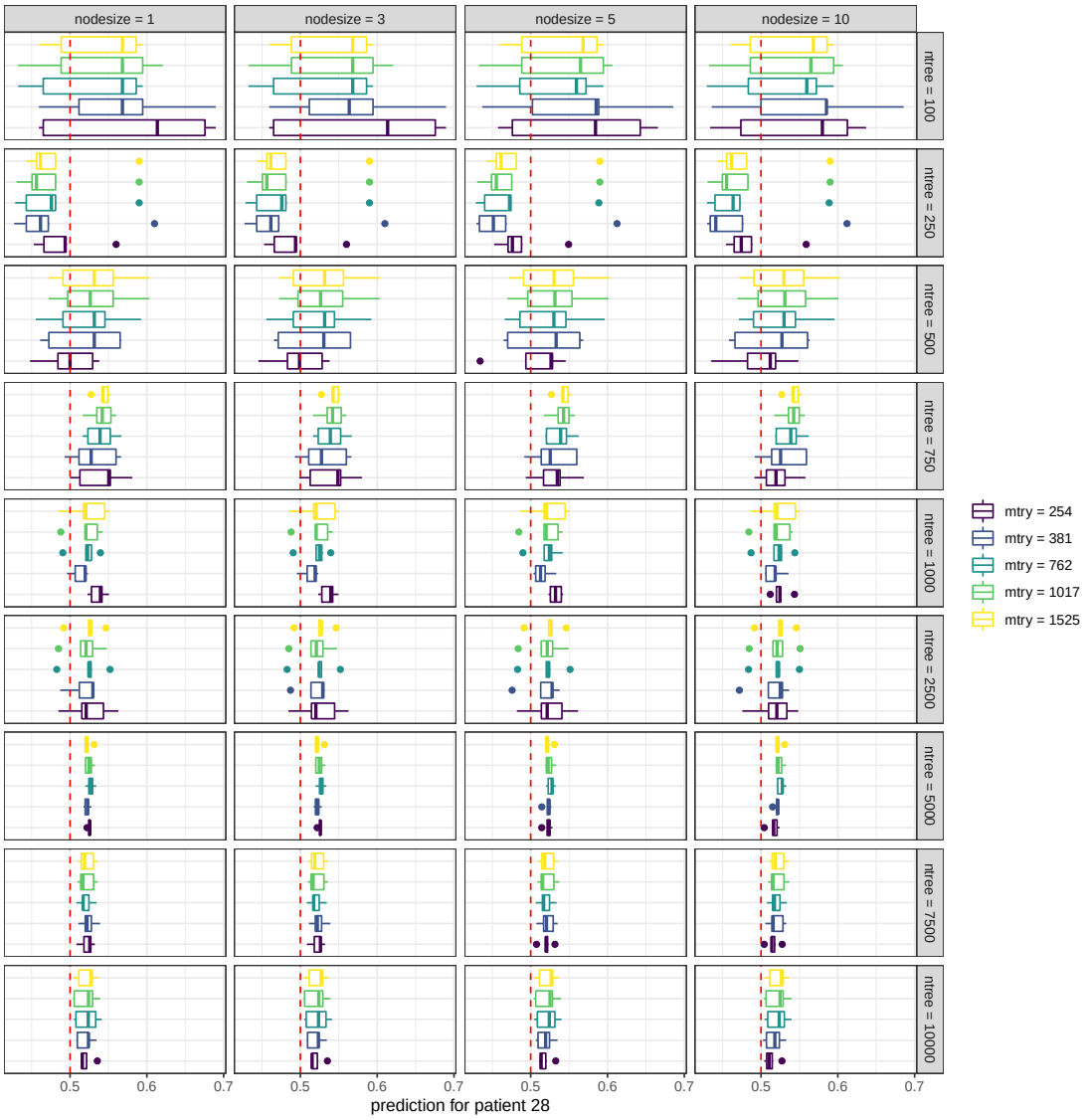
```
for (j in 1:nrow(hyper.grid.prediction)) {
  tmp.forest <-
    rfsrc(y~.,
      tumor.data,
      mtry=hyper.grid.prediction[j, "mtry"],
      nodesize=hyper.grid.prediction[j, "nodesize"],
      ntree=hyper.grid.prediction[j, "ntree"],
      seed=hyper.grid.prediction[j, "seed"])
  hyper.grid.prediction[j, "prediction"] <-
    tmp.forest$predicted.oob[28]
}
```

We can look at some of the predictions in plain text as follows, or get an overview with a boxplot as shown further below. Note that the boxplot shows predictions for many different choices of `ntree`, to investigate when the predictions seem to stabilize. You could do this as well, but note that it takes a long time to run the code.

```
print(hyper.grid.prediction[order(hyper.grid.prediction$ntree),][1:50,])
```

```
   mtry ntree nodesize    seed prediction
1   254   100        1       2  0.5517241
2   381   100        1       2  0.6551724
3   762   100        1       2  0.6206897
4  1017   100        1       2  0.6206897
5  1525   100        1       2  0.5172414
11  254   100        3       2  0.5517241
12  381   100        3       2  0.6551724
```

13	762	100	3	2	0.6206897
14	1017	100	3	2	0.6206897
15	1525	100	3	2	0.5172414
21	254	100	5	2	0.5772441
22	381	100	5	2	0.6423782
23	762	100	5	2	0.5860016
24	1017	100	5	2	0.6007800
25	1525	100	5	2	0.5123153
31	254	100	10	2	0.5868464
32	381	100	10	2	0.6442939
33	762	100	10	2	0.5860016
34	1017	100	10	2	0.6007800
35	1525	100	10	2	0.5123153
41	254	100	1	1240	0.5909091
42	381	100	1	1240	0.4545455
43	762	100	1	1240	0.4772727
44	1017	100	1	1240	0.5000000
45	1525	100	1	1240	0.4772727
51	254	100	3	1240	0.5909091
52	381	100	3	1240	0.4545455
53	762	100	3	1240	0.4772727
54	1017	100	3	1240	0.5000000
55	1525	100	3	1240	0.4772727
61	254	100	5	1240	0.5591631
62	381	100	5	1240	0.4597312
63	762	100	5	1240	0.4864268
64	1017	100	5	1240	0.4913871
65	1525	100	5	1240	0.4772727
71	254	100	10	1240	0.5529442
72	381	100	10	1240	0.4578024
73	762	100	10	1240	0.4876229
74	1017	100	10	1240	0.4913871
75	1525	100	10	1240	0.4772727
81	254	100	1	1919133	0.4594595
82	381	100	1	1919133	0.4594595
83	762	100	1	1919133	0.5405405
84	1017	100	1	1919133	0.5405405
85	1525	100	1	1919133	0.5135135
91	254	100	3	1919133	0.4594595
92	381	100	3	1919133	0.4594595
93	762	100	3	1919133	0.5405405
94	1017	100	3	1919133	0.5405405
95	1525	100	3	1919133	0.5135135



4.4 Tuning the forest

5. In the output of the code below, we look at the estimated error rate of the forest predictors for different values of hyperparameters. Comment on these results.

We define the loss function:

```
loss.fun <- function(Y, Phat) mean((Y-Phat)^2)
```

We define the hypergrid of hyperparameter values. We choose to fix `ntree` = 5000 since the predictions for patient 28 have more or less stabilized at that point (see results above).

```
hyper.grid <- expand.grid(
  mtry = floor((ncol(tumor.data)-1)/c(12,8,4,3,2)),
  ntree = 5000,
  nodesize = c(1,3,5,10),
```

```
  seed = c(2, 1240, 1919133, 111099, 867),
  oob.error = NA
)
```

Note that we also consider a grid of different random seeds to minimize dependence on randomness. We now measure the error rate for each combination of hyperparameter values:

```
for (j in 1:nrow(hyper.grid)) {
  tmp.forest <-
    rfsrc(y~.,
      tumor.data,
      mtry=hyper.grid[j, "mtry"],
      nodesize=hyper.grid[j, "nodesize"],
      ntree=hyper.grid[j, "ntree"],
      seed=hyper.grid[j, "seed"])
  hyper.grid[j, "oob.error"] <-
    loss.fun(tumor.data$y, tmp.forest$predicted.oob)
}
```

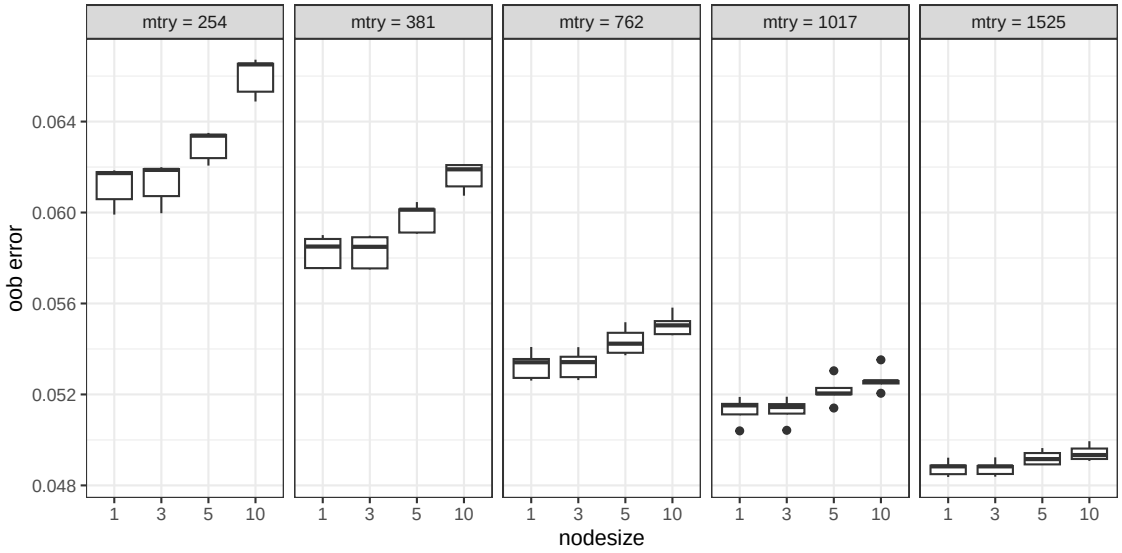
```
print(hyper.grid[order(hyper.grid$oob.error), ])
```

```
    mtry ntree nodesize    seed  oob.error
85  1525  5000        1     867 0.04837043
90  1525  5000        3     867 0.04838410
45  1525  5000        1 1919133 0.04849497
50  1525  5000        3 1919133 0.04850278
25  1525  5000        1    1240 0.04883778
30  1525  5000        3    1240 0.04883778
70  1525  5000        3  111099 0.04885429
65  1525  5000        1  111099 0.04886995
55  1525  5000        5 1919133 0.04892207
95  1525  5000        5     867 0.04892401
60  1525  5000       10 1919133 0.04907796
75  1525  5000        5  111099 0.04915705
100 1525  5000       10     867 0.04916161
5   1525  5000        1       2 0.04921879
10  1525  5000        3       2 0.04923332
80  1525  5000       10  111099 0.04933441
35  1525  5000        5    1240 0.04942336
40  1525  5000       10    1240 0.04962135
15  1525  5000        5       2 0.04964197
20  1525  5000       10       2 0.04994361
64  1017  5000        1  111099 0.05039836
69  1017  5000        3  111099 0.05042273
4   1017  5000        1       2 0.05112309
9   1017  5000        3       2 0.05115788
74  1017  5000        5  111099 0.05140190
49  1017  5000        3 1919133 0.05144879
44  1017  5000        1 1919133 0.05151868
89  1017  5000        3     867 0.05157427
84  1017  5000        1     867 0.05158421
24  1017  5000        1    1240 0.05189045
```

29	1017	5000	3	1240	0.05189815
54	1017	5000	5	1919133	0.05201077
14	1017	5000	5	2	0.05204240
79	1017	5000	10	111099	0.05205224
94	1017	5000	5	867	0.05228531
59	1017	5000	10	1919133	0.05248232
19	1017	5000	10	2	0.05258104
99	1017	5000	10	867	0.05258975
43	762	5000	1	1919133	0.05260989
48	762	5000	3	1919133	0.05263250
3	762	5000	1	2	0.05272654
8	762	5000	3	2	0.05276165
34	1017	5000	5	1240	0.05304046
63	762	5000	1	111099	0.05341443
68	762	5000	3	111099	0.05342110
39	1017	5000	10	1240	0.05352146
83	762	5000	1	867	0.05355752
88	762	5000	3	867	0.05365725
13	762	5000	5	2	0.05372350
53	762	5000	5	1919133	0.05383180
28	762	5000	3	1240	0.05407945
23	762	5000	1	1240	0.05408733
73	762	5000	5	111099	0.05423255
18	762	5000	10	2	0.05460383
58	762	5000	10	1919133	0.05465257
93	762	5000	5	867	0.05471034
78	762	5000	10	111099	0.05503800
33	762	5000	5	1240	0.05517539
98	762	5000	10	867	0.05522623
38	762	5000	10	1240	0.05581729
47	381	5000	3	1919133	0.05749698
42	381	5000	1	1919133	0.05750986
7	381	5000	3	2	0.05754260
2	381	5000	1	2	0.05755784
67	381	5000	3	111099	0.05849142
62	381	5000	1	111099	0.05850342
22	381	5000	1	1240	0.05883689
27	381	5000	3	1240	0.05891296
87	381	5000	3	867	0.05898009
82	381	5000	1	867	0.05900785
52	381	5000	5	1919133	0.05906826
12	381	5000	5	2	0.05911931
1	254	5000	1	2	0.05990913
6	254	5000	3	2	0.05996825
92	381	5000	5	867	0.06012453
72	381	5000	5	111099	0.06013890
32	381	5000	5	1240	0.06045967
41	254	5000	1	1919133	0.06058709
46	254	5000	3	1919133	0.06072192
17	381	5000	10	2	0.06073983

```
57   381  5000        10 1919133 0.06114891
21   254  5000         1    1240 0.06173436
81   254  5000         1     867 0.06177710
61   254  5000         1  111099 0.06185154
26   254  5000         3    1240 0.06187403
97   381  5000        10     867 0.06190025
86   254  5000         3     867 0.06190987
66   254  5000         3  111099 0.06199171
11   254  5000         5       2 0.06206414
37   381  5000        10    1240 0.06208635
77   381  5000        10  111099 0.06211711
51   254  5000         5 1919133 0.06239267
91   254  5000         5     867 0.06337872
71   254  5000         5  111099 0.06341410
31   254  5000         5    1240 0.06350094
56   254  5000        10 1919133 0.06488202
16   254  5000        10       2 0.06531322
76   254  5000        10  111099 0.06651278
96   254  5000        10     867 0.06654340
36   254  5000        10    1240 0.06672058
```

We can plot the results as boxplots (across the random seeds) as follows:



6. We continue with the tuned model by running the code below.

```
# Find mean error across seeds
hyper.grid.mean <- aggregate(
  oob.error ~ mtry + nodesize + ntree,
  data = hyper.grid,
  FUN = mean
)
opt.hypers <- hyper.grid.mean[which.min(hyper.grid.mean$oob.error),]
```

This corresponds to the following values of hyperparameters:

```
opt.hypers
```

```
  mtry nodesize ntree  oob.error
5 1525        1  5000 0.04875838
```

```
tuned.forest.tumor <-
  rfsrc(y~.,
    tumor.data,
    mtry=opt.hypers[1, "mtry"],
    nodesize=opt.hypers[1, "nodesize"],
    ntree=opt.hypers[1, "ntree"],
    seed=11)
```

7. Get the predicted values from the tuned forest for patient 1 and patient 28.

```
tuned.forest.tumor$predicted.oob[1]
```

```
[1] 0.08583465
```

```
tuned.forest.tumor$predicted.oob[28]
```

```
[1] 0.529185
```

Let's look get a quick look at how many samples that are classified correctly by this model:

```
table(classification=ifelse(tuned.forest.tumor$predicted.oob>0.5, 1, 0),
      observed=tumor.data$y)
```

```
              observed
classification  0  1
             0 27  0
             1  0 11
```

4.5 Interpretable machine learning measures

8. Now we want to find the potentially important variables. We first look at the VIMP measure by running the code below.

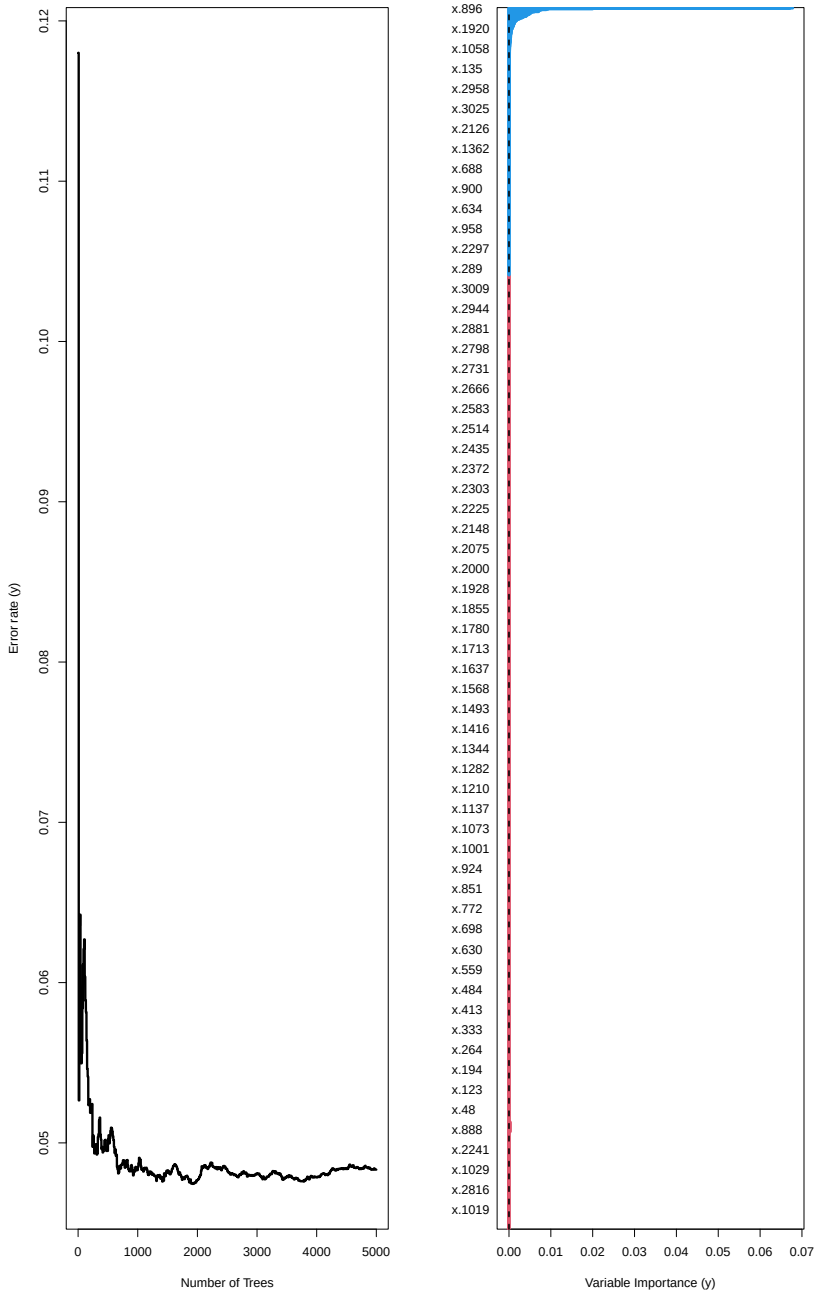
```
tuned.forest.tumor.vimp <-
  rfsrc(y~.,
    tumor.data,
    mtry=opt.hypers[1, "mtry"],
    nodesize=opt.hypers[1, "nodesize"],
    ntree=opt.hypers[1, "ntree"],
    seed=11,
    importance=TRUE)
```

There are many variables. Let's look at the 25 with the highest value:

```
data.frame(ranked.no=1:25, vimp=rev(sort(tuned.forest.tumor.vimp$importance))[1:25])
```

```
       ranked.no        vimp
x.2124         1 0.067773155
x.896          2 0.067103179
x.829          3 0.019998863
x.2386         4 0.009313317
x.2600         5 0.008909779
x.808          6 0.007704314
x.2670         7 0.007355977
x.766          8 0.007154893
x.108          9 0.007070632
x.394         10 0.007053976
x.1413        11 0.005957548
x.1037        12 0.005438542
x.2002        13 0.005219238
x.786         14 0.005130525
x.515         15 0.005061805
x.937         16 0.004930090
x.2939        17 0.004736090
x.1995        18 0.004591159
x.894         19 0.004289931
x.283         20 0.004183965
x.1907        21 0.004104198
x.1448        22 0.004047757
x.1834        23 0.003623340
x.848         24 0.003373666
x.523         25 0.003233393
```

We can also plot the VIMP:



9. Let's also look at the minimal depth of each variable and conclude what variables are important using this method.

There is an extra step involved when getting the minimal depth. We first apply the function `max.subtree()` to the forest object:

```
md.obj <- max.subtree(tuned.forest.tumor.vimp, max.order=0)
```

Then we can compute the minimal depth:

```
md <- sapply(1:dim(md.obj$order)[1], function(i) mean(md.obj$order[i, ]))
names(md) <- names(md.obj$order[, 1])
```

We look only at the with the 25 with the lowest value:

```
data.frame(ranked.no=1:25, "minimal depth"=sort(md)[1:25])
```

```
       ranked.no minimal.depth
x.2124         1        1.0534
x.896          2        1.0542
x.829          3        1.1230
x.2386         4        1.1498
x.2600         5        1.1498
x.808          6        1.1506
x.2670         7        1.1526
x.108          8        1.1538
x.1037         9        1.1566
x.2002        10        1.1570
x.394         11        1.1574
x.1413        12        1.1574
x.766         13        1.1584
x.2939        14        1.1594
x.1995        15        1.1606
x.515         16        1.1624
x.786         17        1.1630
x.1834        18        1.1632
x.1448        19        1.1634
x.283         20        1.1646
x.894         21        1.1650
x.1907        22        1.1650
x.937         23        1.1660
x.848         24        1.1678
x.523         25        1.1688
```

The threshold computed by the function is:

```
md.obj$threshold
```

```
[1] 1.179554
```

And we can count the number of "important" variables with this method as:

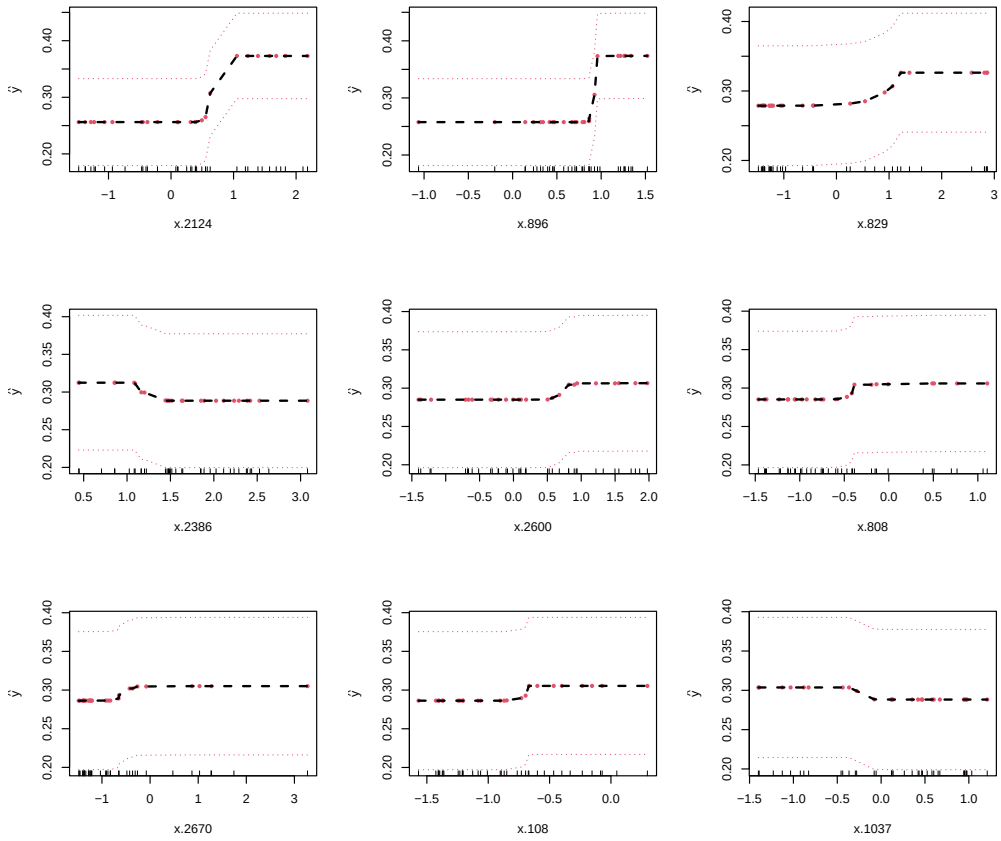
```
sum(md<=md.obj$threshold)
```

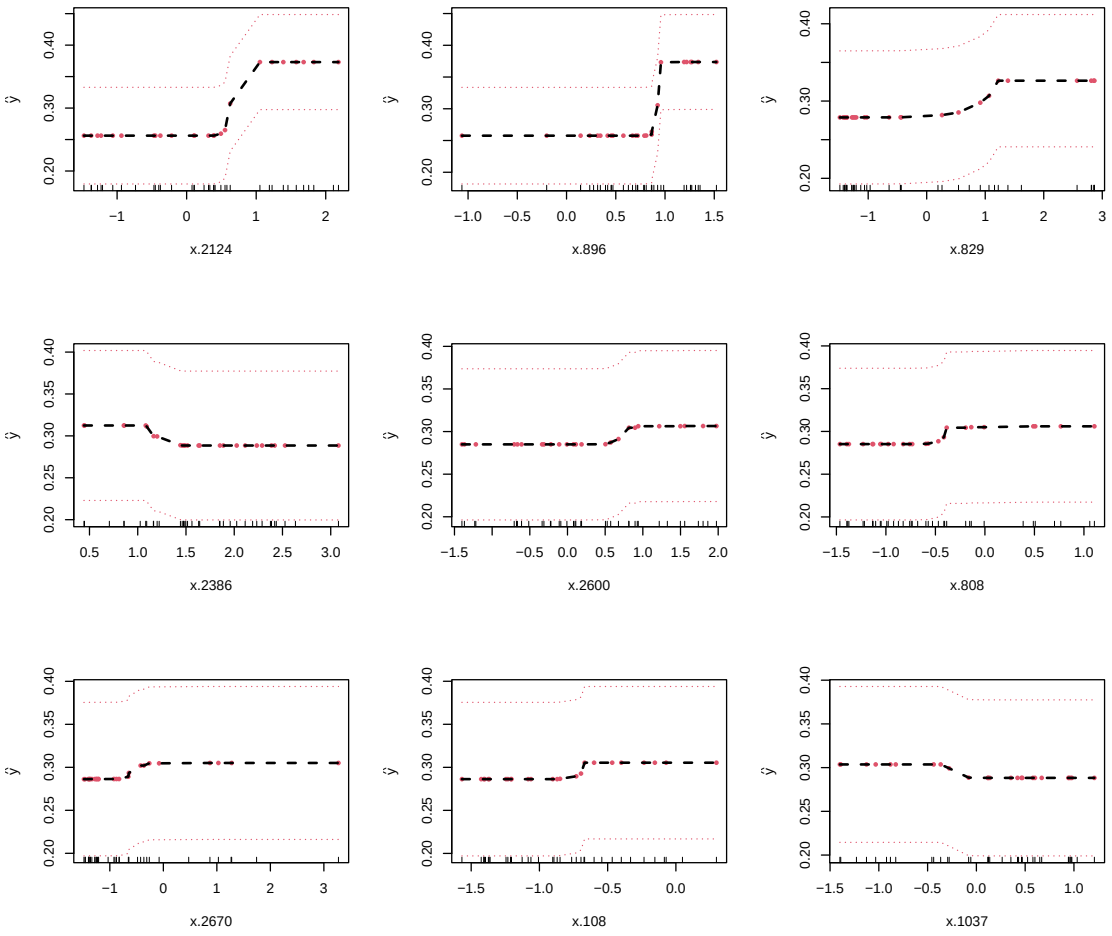
```
[1] 217
```

10. We can produce PDP plots for the some of the variables to look at how prediction depends on their values.

It takes too long to make the PDP plot for all variables, so we will simply consider first the 9 with lowest minimal depth:

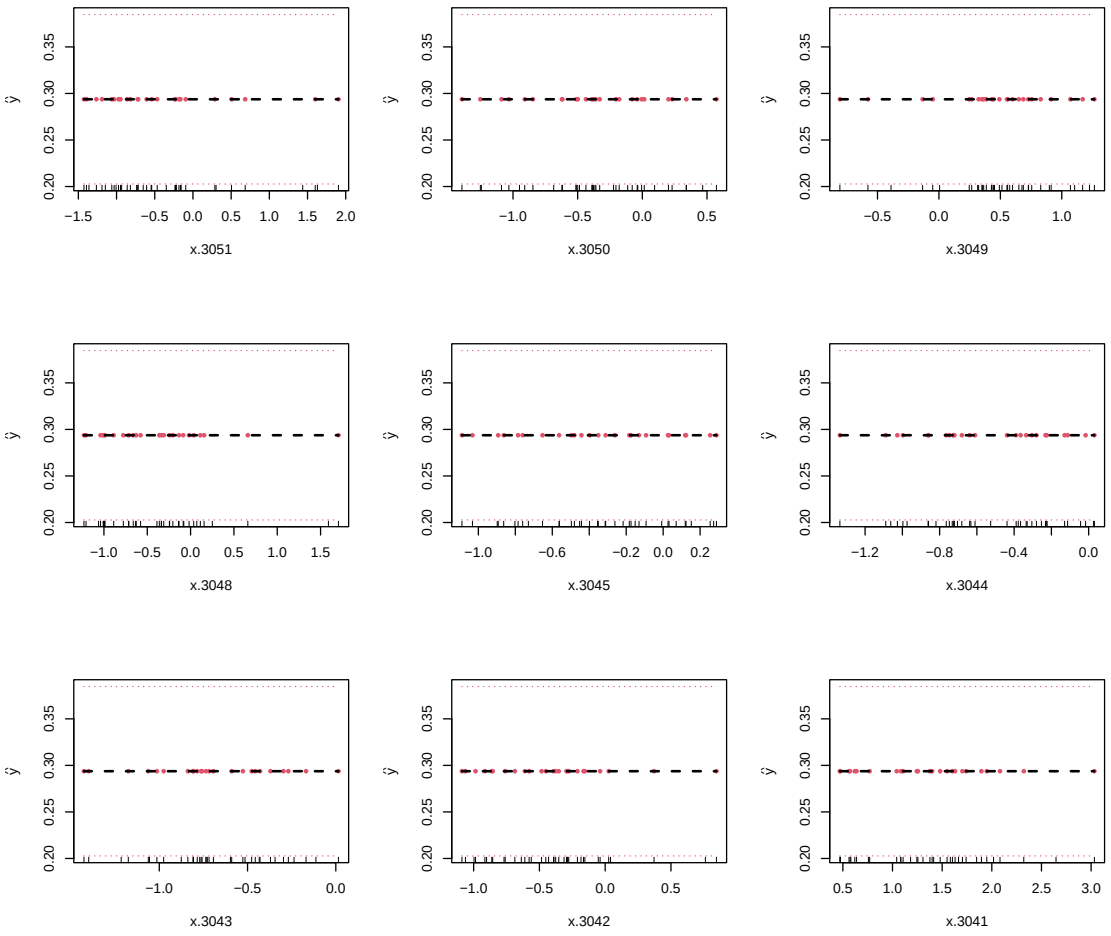
```
plot.variable(tuned.forest.tumor, partial=TRUE, plots.per.page=3,
          xvar.names=names(sort(md))[1:9])
```





We can plot the least important as well:

```
plot.variable(tuned.forest.tumor, partial=TRUE, plots.per.page=3,
          xvar.names=names(rev(sort(md)))[1:9])
```



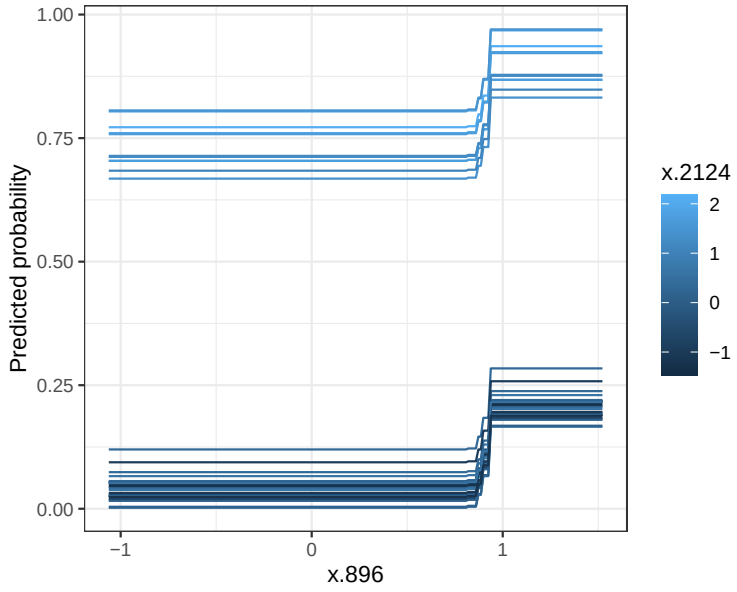
11. We can also look at ICE plots. The code below produces the ICE plot for the variable `x.896`. We want to look for differing patterns across individuals. Note that the plot is colored according to values of the variable `x.2124`.

```
ice.grid <- expand.grid(
    x.896=seq(min(tumor.data$x.896), max(tumor.data$x.896), length=200), pred=NA
)

ice.dat <- do.call("rbind", lapply(1:nrow(tumor.data), function(i) {
    tmp <- ice.grid
    tmp$id <- i

    for (xvar in colnames(tumor.data[,-1])) {
    if (xvar!="x.896")
        tmp[, xvar] <- tumor.data[i, xvar]
    }

    tmp$pred <- predict(tuned.forest.tumor, newdata=tmp, type="response")$predicted
    return(tmp)
```

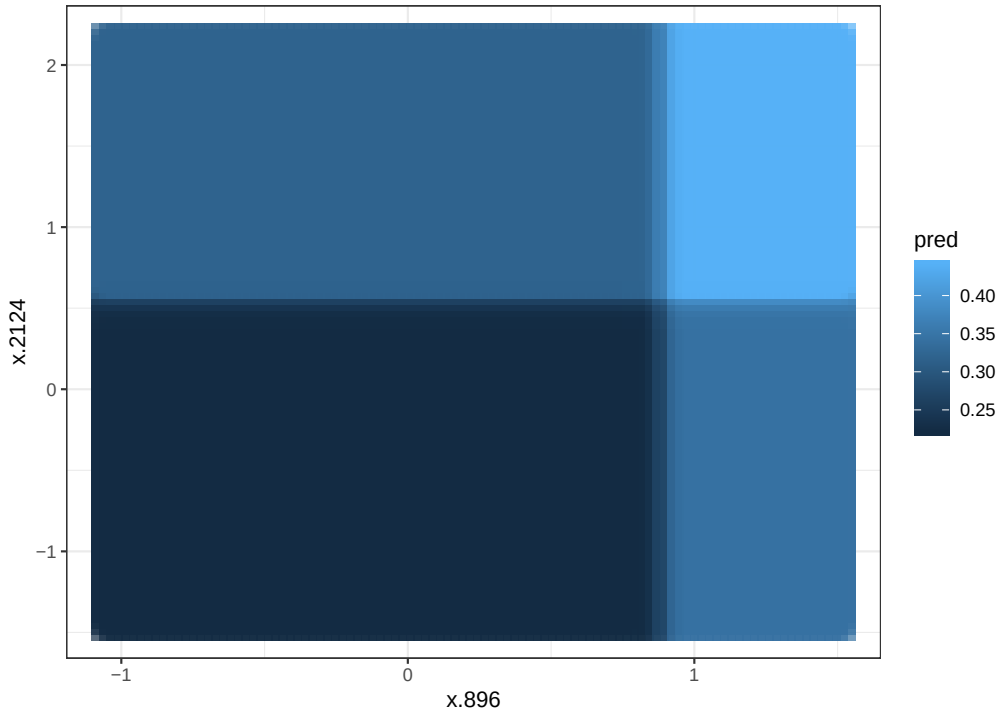
```
}))
```

```
library(ggplot2)
plot.ice <-
  ggplot() + theme_bw() +
  geom_line(data=ice.dat, aes(x=x.896, y=pred, group=id, col=x.2124)) +
  xlab("x.896") + ylab("Predicted probability")
print(plot.ice)
```

```
library(ggplot2)
plot.ice <-
  ggplot() + theme_bw() +
  geom_line(data=ice.dat, aes(x=x.2124, y=pred, group=id, col=x.896)) +
  xlab("x.896") + ylab("Predicted probability")
print(plot.ice)
```



The following two-way PDP graph is perhaps more informative to show the interaction:


2
1
0
−1
x.2124
−1
0
1
x.896
pred
0.40
0.35
0.30
0.25