Random forest exercises

Advanced Statistical Topics in Health Research A

November 22, 2022

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 are read into R by writing:

```
Epo <- read.csv("http://publicifsv.sund.ku.dk/~helene/Epo.csv", stringsAsFactors=TRUE)
```

(NB: somehow does not work to copy the symbol "~"; if you copied the string above and got error messages, try to rewrite "~").

The newpatient data are read into R by writing:

```
newpatient <- read.csv("http://publicifsv.sund.ku.dk/~helene/newpatient",
    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:

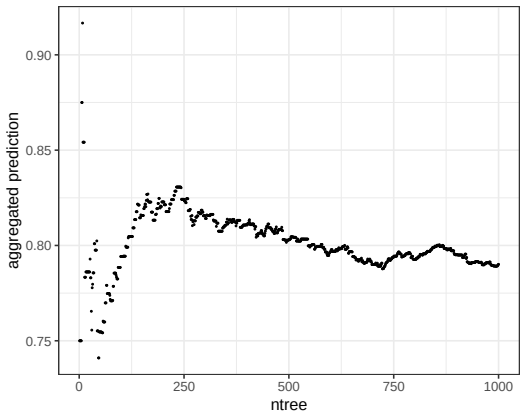
```
pred[1:10]
```

```
[1] 0.7500000        NA        NA        NA        NA 1.0000000        NA 1.0000000 0.6666667
```

We see an NA every time individual $i = 25$ was oob (not included in the bootstrap sample of 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
)
```

The 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              0.4159781
37     1        1    50    5              0.5499598
73     1        1    50   12              0.5478496
13     1        1   100    1              0.5074093
49     1        1   100    5              0.5088901
85     1        1   100   12              0.5673316
25     1        1  1000    1              0.5346341
61     1        1  1000    5              0.5354126
97     1        1  1000   12              0.5445230
4      1        3    50    1              0.4410087
40     1        3    50    5              0.5488174
76     1        3    50   12              0.5509323
16     1        3   100    1              0.5158999
52     1        3   100    5              0.5048235
88     1        3   100   12              0.5423568
28     1        3  1000    1              0.5272854
64     1        3  1000    5              0.5212890
100    1        3  1000   12              0.5227943
```

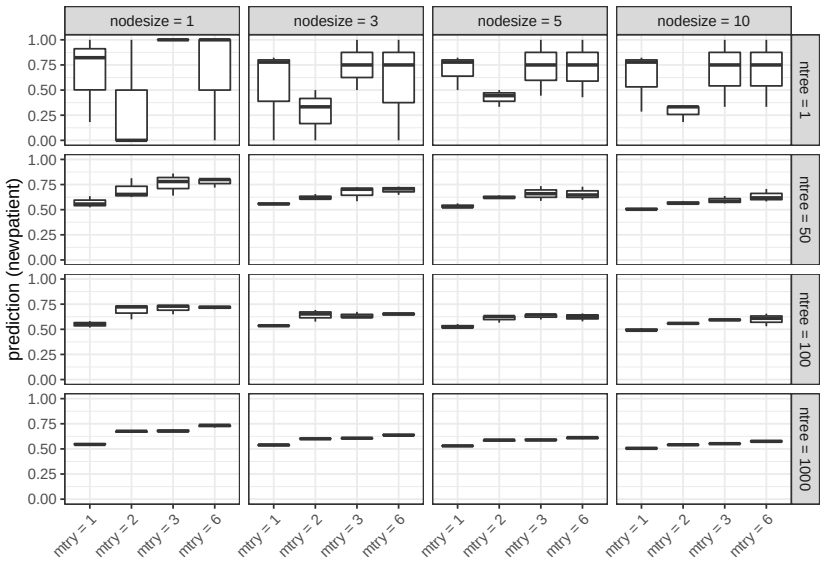
7	1	5	50	1	0.4610091
43	1	5	50	5	0.5910949
79	1	5	50	12	0.5100511
19	1	5	100	1	0.4978793
55	1	5	100	5	0.4980036
91	1	5	100	12	0.5274211
31	1	5	1000	1	0.5156784
67	1	5	1000	5	0.5262168
103	1	5	1000	12	0.5220628
10	1	10	50	1	0.4403129
46	1	10	50	5	0.5214462
82	1	10	50	12	0.5230330
22	1	10	100	1	0.4843340
58	1	10	100	5	0.4925440
94	1	10	100	12	0.4907428
34	1	10	1000	1	0.4945875
70	1	10	1000	5	0.4958726
106	1	10	1000	12	0.4947986
2	3	1	50	1	0.7400000
38	3	1	50	5	0.6400000
74	3	1	50	12	0.8600000
14	3	1	100	1	0.6700000
50	3	1	100	5	0.6900000
86	3	1	100	12	0.6800000
26	3	1	1000	1	0.6767625
62	3	1	1000	5	0.6635000
98	3	1	1000	12	0.6590000
5	3	3	50	1	0.6753333
41	3	3	50	5	0.6573333
77	3	3	50	12	0.7140000
17	3	3	100	1	0.5951667
53	3	3	100	5	0.6041667
89	3	3	100	12	0.6395000
29	3	3	1000	1	0.5816625
65	3	3	1000	5	0.5763333
101	3	3	1000	12	0.5810333
8	3	5	50	1	0.5517857
44	3	5	50	5	0.6178254
80	3	5	50	12	0.6593016
20	3	5	100	1	0.5505556
56	3	5	100	5	0.5527619
92	3	5	100	12	0.6332857
32	3	5	1000	1	0.5480772
68	3	5	1000	5	0.5423873
104	3	5	1000	12	0.5555028
11	3	10	50	1	0.5292141
47	3	10	50	5	0.5312144
83	3	10	50	12	0.5354880
23	3	10	100	1	0.5007439
59	3	10	100	5	0.5019254

95	3	10	100	12	0.5905100
35	3	10	1000	1	0.5113350
71	3	10	1000	5	0.5055761
107	3	10	1000	12	0.5112855
3	6	1	50	1	0.8200000
39	6	1	50	5	0.7000000
75	6	1	50	12	0.8000000
15	6	1	100	1	0.7600000
51	6	1	100	5	0.7500000
87	6	1	100	12	0.7800000
27	6	1	1000	1	0.7430000
63	6	1	1000	5	0.7450000
99	6	1	1000	12	0.7290000
6	6	3	50	1	0.6983333
42	6	3	50	5	0.6110000
78	6	3	50	12	0.7183333
18	6	3	100	1	0.6128333
54	6	3	100	5	0.6533333
90	6	3	100	12	0.6766667
30	6	3	1000	1	0.6275000
66	6	3	1000	5	0.6220333
102	6	3	1000	12	0.6360833
9	6	5	50	1	0.5762778
45	6	5	50	5	0.5432460
81	6	5	50	12	0.6599444
21	6	5	100	1	0.5242579
57	6	5	100	5	0.5753254
93	6	5	100	12	0.6485754
33	6	5	1000	1	0.5664397
69	6	5	1000	5	0.5640353
105	6	5	1000	12	0.5773206
12	6	10	50	1	0.5080578
48	6	10	50	5	0.4826231
84	6	10	50	12	0.5610407
24	6	10	100	1	0.4704323
60	6	10	100	5	0.4990995
96	6	10	100	12	0.5745990
36	6	10	1000	1	0.4964534
72	6	10	1000	5	0.4999047
108	6	10	1000	12	0.5089207

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) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  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 of Golub et al. (1999).

The dataset is loaded as follows:

```
golub <- read.csv("http://publicifsv.sund.ku.dk/~helene/golub.csv")[, -1]
golub.cl <- read.csv("http://publicifsv.sund.ku.dk/~helene/golub.cl.csv")[, -1]
```

The dataframe `golub.cl` is a numeric vector indicating the tumor class, 27 acute lymphoblastic leukemia (ALL) cases (code 0) and 11 acute myeloid leukemia (AML) cases (code 1). We want to predict the tumor class.

Note that the form of the gene expression data is wrong compared to how many functions in `R` would want it. They want individuals (mRNA samples) as the rows and the individuals genes as the columns. Consequently, we start by transforming the data matrix.

```
gdata <- t(golub) # tranpose data
dim(gdata) # check dimensions
```

```
[1]   38 3051
```

The functions in the `randomForestSRC` package require that the data are stored in a data frame:

```
tumor.data <- data.frame(y=golub.cl, x=gdata)
```

This is the data we will work with.

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 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.

```
# --- 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*. 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: 2.008
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
                     (OOB) R squared: 0.79938198
   (OOB) Requested performance error: 0.04237806
```

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] 0.09110809
[1] 0.5012195
```

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  0
             1  0 11
```

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] 0.0718232
[1] 0.4961656
```

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, 250, 500, 750, 1000, 2500, 5000, 7500, 10000),
  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:

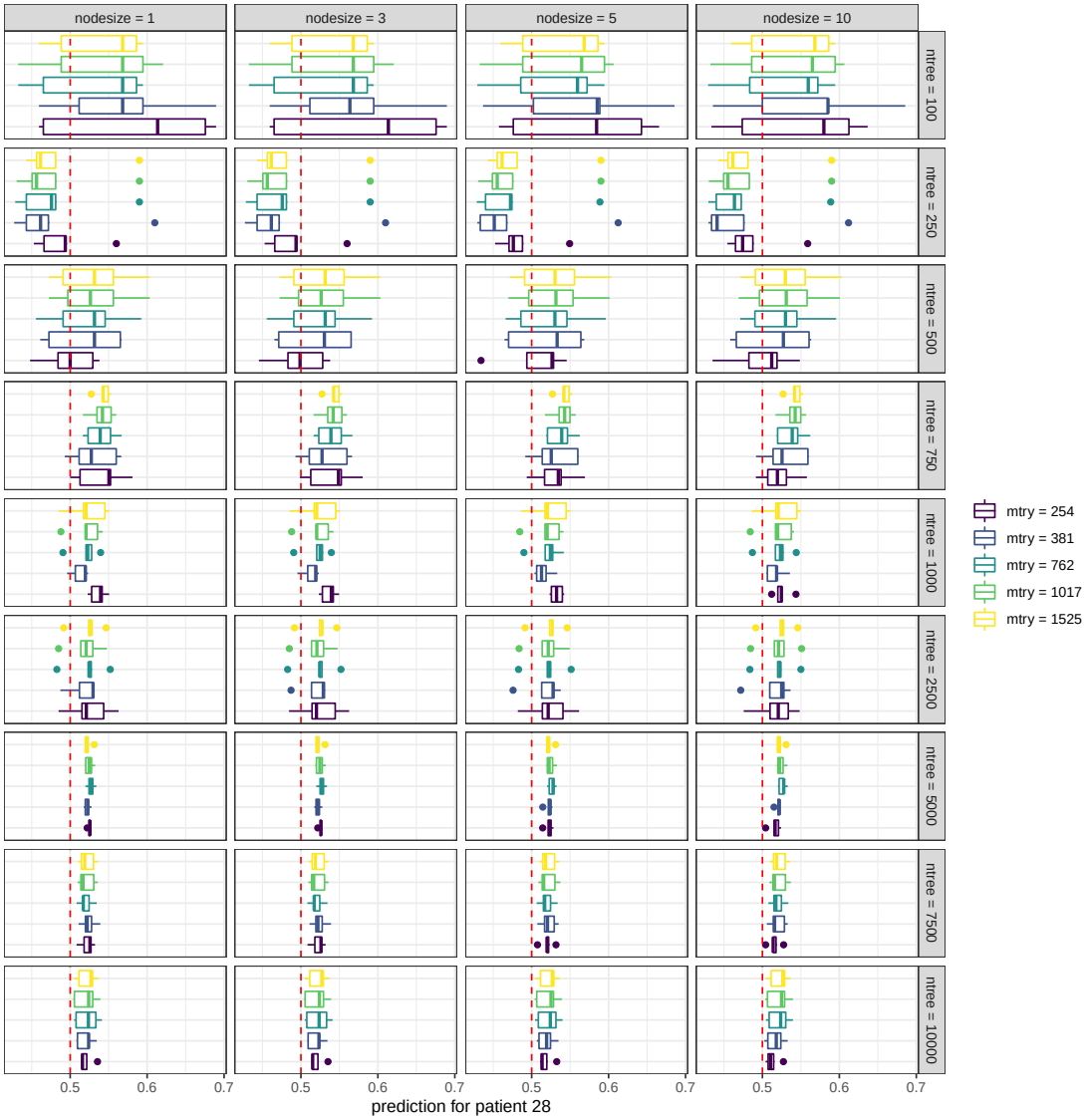
```
for (j in 1:nrow(hyper.grid.prediction)) {
  tmp.forest <-
    rfsrc(y~.,
      tumor.data,
      mtry=hyper.grid.prediction[j, "mtry"],
      nodesize=hyper.grid.prediction[j, "nodesize"],
      ntree=hyper.grid.prediction[j, "ntree"],
      seed=hyper.grid.prediction[j, "seed"])
  hyper.grid.prediction[j,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

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

	mtry	ntree	nodesize	prediction
1	254	100	1	0.4736842
2	381	100	1	0.4473684
3	762	100	1	0.4473684
4	1017	100	1	0.4473684
5	1525	100	1	0.4473684
21	254	100	3	0.4736842
22	381	100	3	0.4473684
23	762	100	3	0.4473684
24	1017	100	3	0.4473684
25	1525	100	3	0.4473684
41	254	100	5	0.4532164
42	381	100	5	0.4334795
43	762	100	5	0.4407895
44	1017	100	5	0.4473684
45	1525	100	5	0.4473684
61	254	100	10	0.4283626
62	381	100	10	0.4349415
63	762	100	10	0.4407895
64	1017	100	10	0.4473684
65	1525	100	10	0.4473684
6	254	500	1	0.5365854
7	381	500	1	0.5365854
8	762	500	1	0.5073171
9	1017	500	1	0.5024390
10	1525	500	1	0.4975610
26	254	500	3	0.5336585
27	381	500	3	0.5356098
28	762	500	3	0.5073171
29	1017	500	3	0.5024390
30	1525	500	3	0.4975610
46	254	500	5	0.5371235
47	381	500	5	0.5328901
48	762	500	5	0.5125339
49	1017	500	5	0.5012195
50	1525	500	5	0.4975610
66	254	500	10	0.5311636
67	381	500	10	0.5279638
68	762	500	10	0.5125339
69	1017	500	10	0.5012195
70	1525	500	10	0.4975610
11	254	1000	1	0.5012594
12	381	1000	1	0.4861461
13	762	1000	1	0.4962217
14	1017	1000	1	0.4937028
15	1525	1000	1	0.4987406
31	254	1000	3	0.5032746
32	381	1000	3	0.4856423
33	762	1000	3	0.4962217
34	1017	1000	3	0.4937028

```
35 1525   1000          3   0.4987406
```



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:

```
hyper.grid <- expand.grid(
  mtry = floor((ncol(tumor.data)-1)/c(12,8,4,3,2)),
  ntree = c(100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000),
```

```
  nodesize = c(1,3,5,10),
  seed = c(2, 1240, 1919133, 111099, 867),
  oob.error = NA
)
```

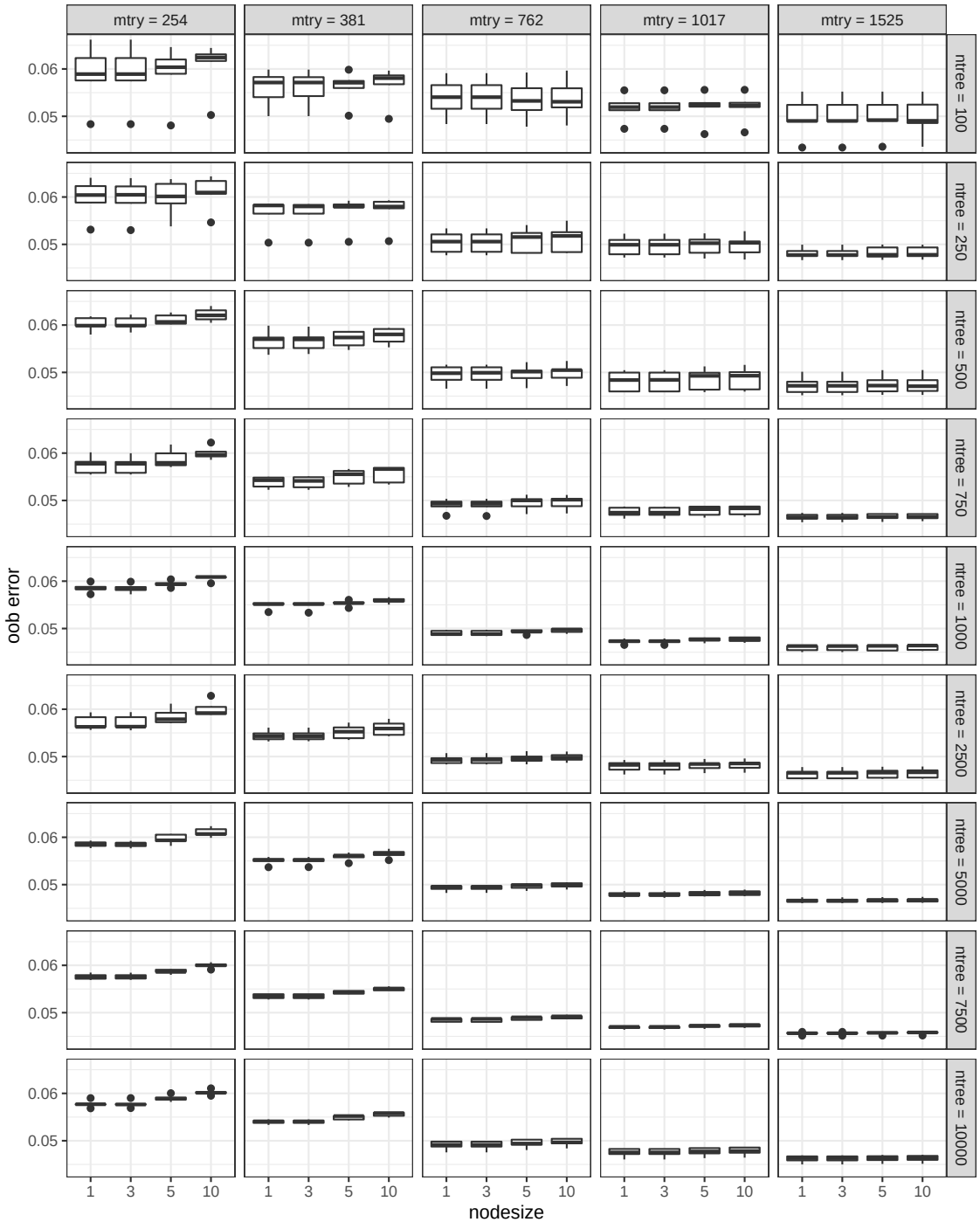
Note that we also consider a grid of different random seeds to minimize dependence on randomness. We now measure the error rate for each combination of hyperparameter values:

```
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 nodesize  oob.error
5  1525        1 0.04484345
10 1525        3 0.04484345
20 1525       10 0.04515708
15 1525        5 0.04516235
4  1017        1 0.04623228
9  1017        3 0.04623228
3   762        1 0.04669852
8   762        3 0.04669852
14 1017        5 0.04677993
19 1017       10 0.04716625
18  762       10 0.04787095
13  762        5 0.04801718
7   381        3 0.05292729
2   381        1 0.05303804
12  381        5 0.05369741
17  381       10 0.05395683
1   254        1 0.05602255
6   254        3 0.05620059
11  254        5 0.05822075
16  254       10 0.06005685
```

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.

```
(j <-
   (1:nrow(hyper.grid))[hyper.grid$mean.oob.error==min(hyper.grid$mean.oob.error)][1])
```

This corresponds to the following values of hyperparameters:

```
print(unique(hyper.grid[j, 1:3]))
```

```
   mtry ntree nodesize
85 1525  7500        3
```

```
tuned.forest.tumor <-
  rfsrc(y~.,
    tumor.data,
    mtry=hyper.grid[j, "mtry"],
    nodesize=hyper.grid[j, "nodesize"],
    ntree=hyper.grid[j, "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] 0.08481825
```

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

```
[1] 0.5218349
```

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.

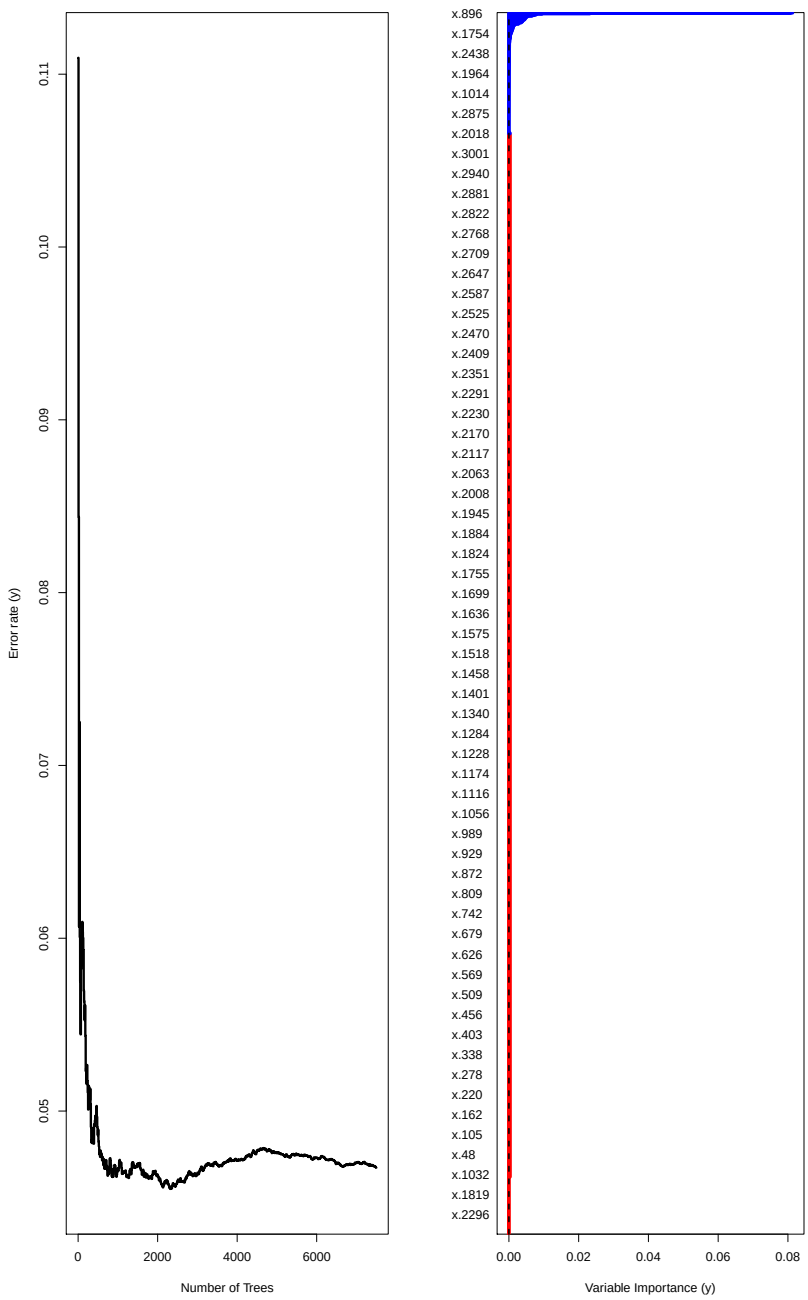
```
tuned.forest.tumor.vimp <-
  rfsrc(y~.,
    tumor.data,
    mtry=hyper.grid[j, "mtry"],
    nodesize=hyper.grid[j, "nodesize"],
    ntree=hyper.grid[j, "ntree"],
    seed=11,
    importance=TRUE)
```

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 0.081350874
x.896          2 0.080776020
x.829          3 0.023233174
x.2386         4 0.009473027
x.2600         5 0.007917316
x.2670         6 0.007725406
x.766          7 0.007323853
x.394          8 0.006851619
x.1413         9 0.006519383
x.2939        10 0.005498903
x.808         11 0.005104383
x.786         12 0.005083040
x.283         13 0.004931823
x.515         14 0.004750448
x.108         15 0.004703811
x.937         16 0.004608217
x.1995        17 0.004600471
x.523         18 0.004522196
x.2002        19 0.004313443
x.1448        20 0.004151597
x.894         21 0.003930679
x.1907        22 0.003889323
x.1037        23 0.003883062
x.849         24 0.003728416
x.1834        25 0.003573958
```

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     0.8836000
x.896          2     0.8837333
x.829          3     0.9649333
x.2386         4     0.9956000
x.2600         5     0.9973333
x.2670         6     0.9976000
x.394          7     1.0000000
x.766          8     1.0002667
x.1413         9     1.0028000
x.2939        10     1.0044000
x.808         11     1.0050667
x.283         12     1.0053333
x.786         13     1.0057333
x.1995        14     1.0060000
x.108         15     1.0062667
x.523         16     1.0069333
x.1834        17     1.0070667
x.515         18     1.0072000
x.2002        19     1.0072000
x.1448        20     1.0074667
x.894         21     1.0077333
x.937         22     1.0081333
x.1907        23     1.0081333
x.849         24     1.0089333
x.2198        25     1.0090667
```

The threshold computed by the function is:

```
md.obj$threshold
```

```
[1] 1.020992
```

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

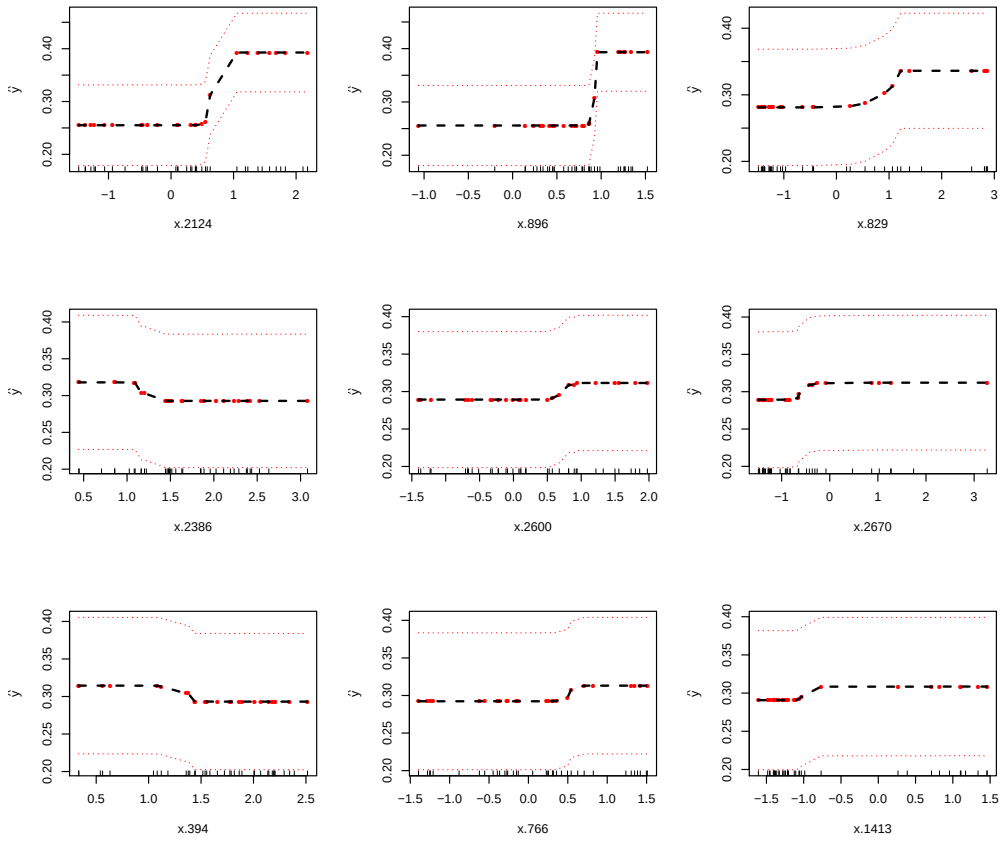
```
sum(md<=md.obj$threshold)
```

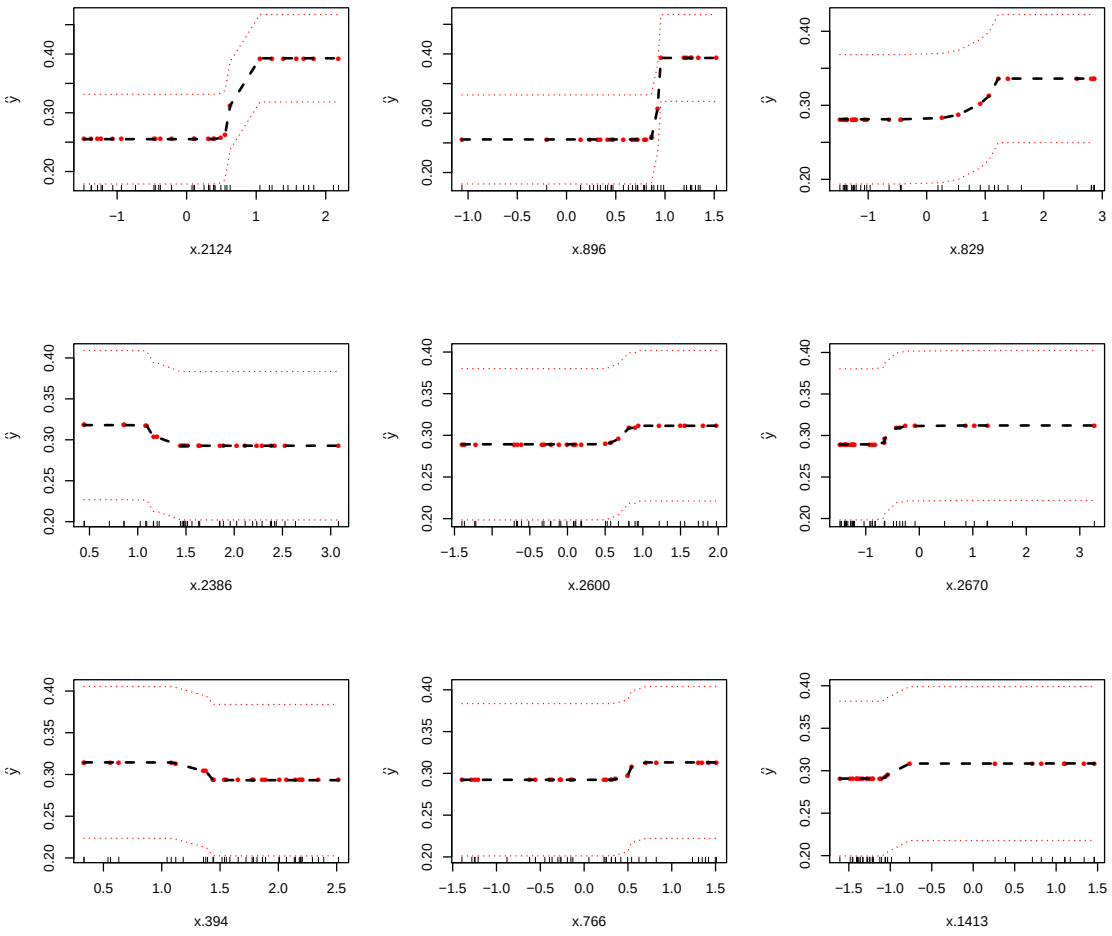
```
[1] 185
```

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:

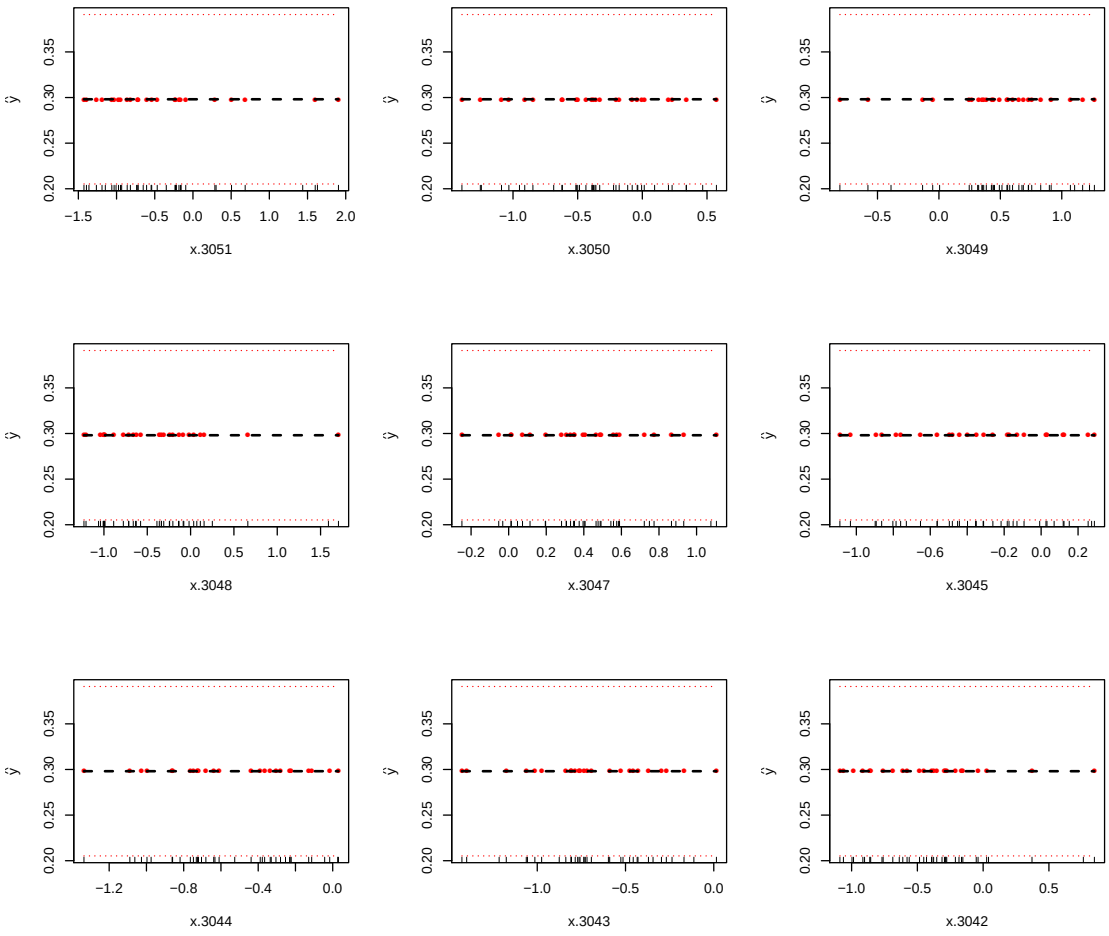
```
plot.variable(tuned.forest.tumor, partial=TRUE, plots.per.page=3,
          xvar.names=names(sort(md))[1:9])
```





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])
```



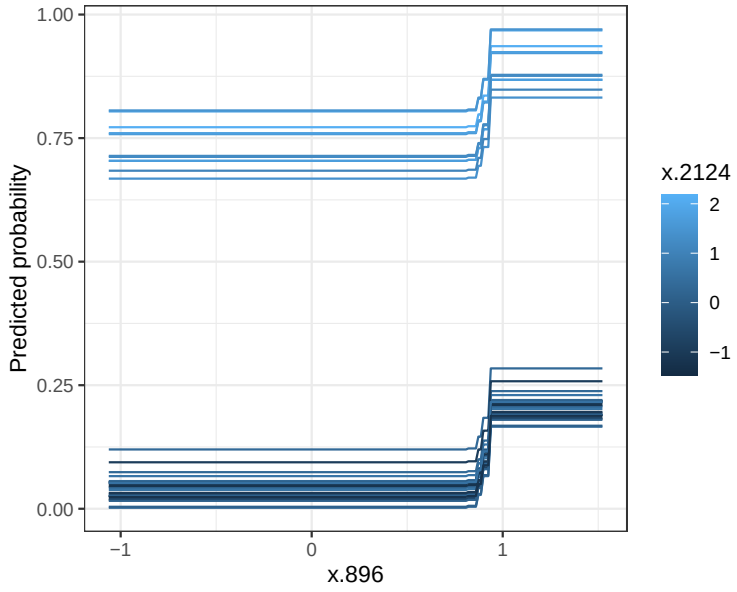
11. We can also look at ICE plots. The code below produces the ICE plot for the variable `x.896`. We want to look for differing patterns across individuals. Note that the plot is colored according to values of the variable `x.2124`.

```
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)
```

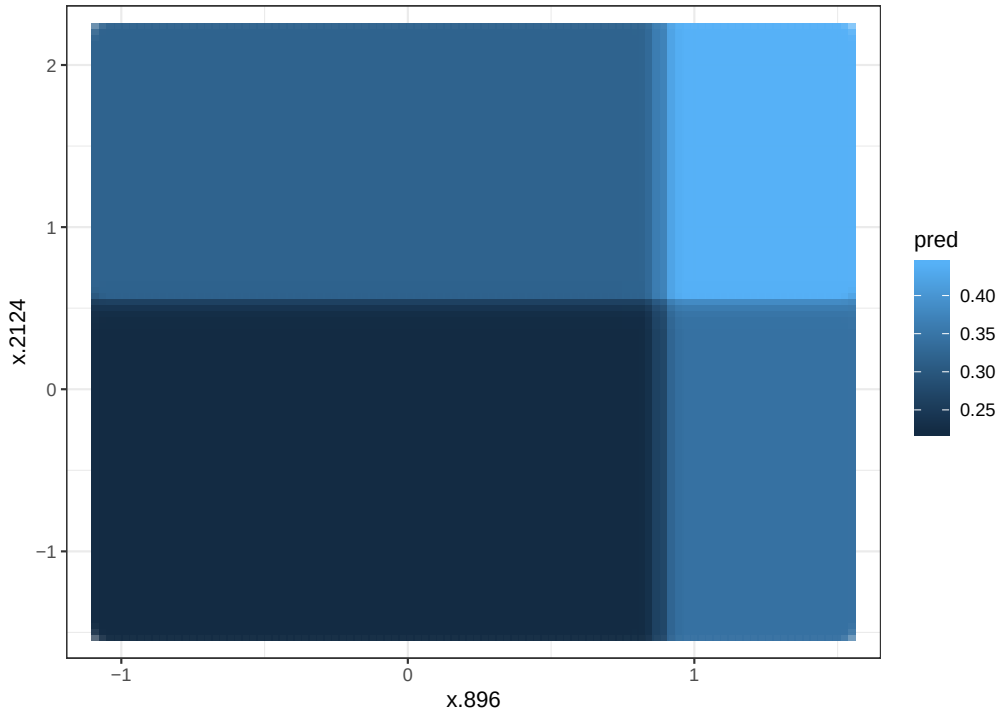
```
}))
```

```
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

References

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, et al. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science 286*(5439), 531–537.