Random forest exercises

Mark Bech Knudsen

2026-10-29

Table of contents

Loading the randomForestSRC package and the Epo data 1
1 Exercise 1: From trees to forests 2
1.1 Code for Exercise 1 3
2 Exercise 2: Get predictions for newpatient 4
2.1 Code for Exercise 2 5
3 Exercise 3: Identifying risk factors 8
4 Exercise 4: Predicting tumor class 9
4.1 Getting forest predictions 11
4.2 Growing a forest for different values of hyperparameters 12
4.3 Tuning the forest 14
4.4 Interpretable machine learning measures 17

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")
```

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 a 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:

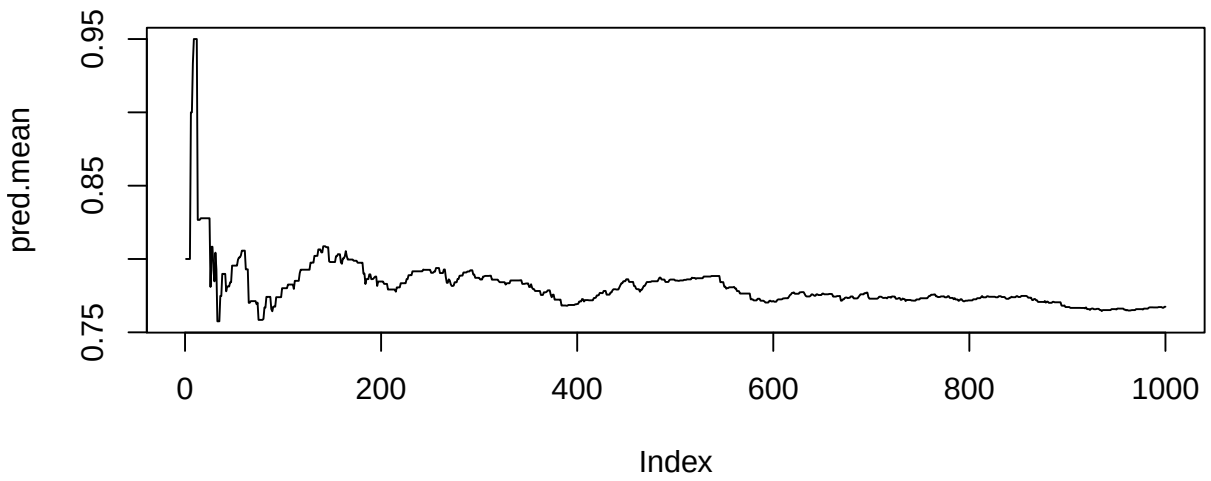
```
pred[1:10]
```

```
 [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, type = "l")
```



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 = 1`	`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 = 2`	`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 hyperparameters, we first define the different combinations of hyperparameters that we are interested in. The following code defines a grid for all possible combinations of the hyperparameters considered in this exercise (which is slight more than what the exercise asks for), and three different seed values:

```
hyper.grid.prediction <- expand.grid(
  mtry = c(1, 2, 3, 6),
  nodesize = c(1, 3, 5, 10),
  ntree = c(1, 50, 100, 1000),
  seed = c(1, 5, 12),
  prediction.newpatient = NA
)
```

The following loop fits a new random forest for each combination of hyperparameters and gets the prediction for `newpatient`:

```
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 needed for the tables are as follows:

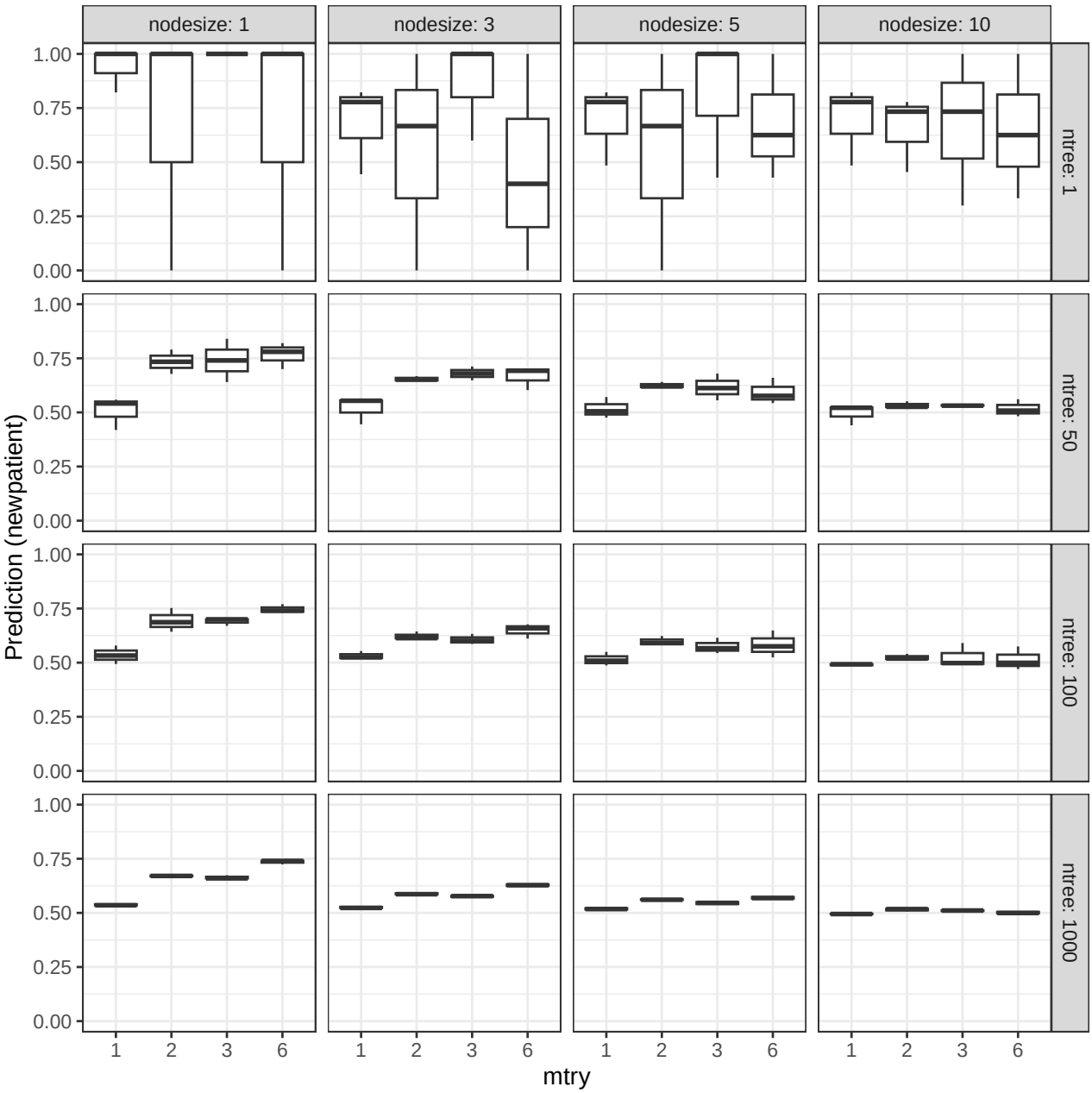
```
subset(
  hyper.grid.prediction,
  (nodesize == 1 & mtry == 3) |
    (ntree == 1000 & mtry == 3) |
    (ntree == 1000 & nodesize == 1)
)
```

```
    mtry nodesize ntree seed prediction.newpatient
3      3        1     1    1             1.0000000
19     3        1    50    1             0.7400000
35     3        1   100    1             0.6700000
49     1        1  1000    1             0.5352921
50     2        1  1000    1             0.6742462
51     3        1  1000    1             0.6740625
52     6        1  1000    1             0.7400000
55     3        3  1000    1             0.5773792
59     3        5  1000    1             0.5460466
```

```
63     3       10  1000    1             0.5119986
67     3        1     1    5             1.0000000
83     3        1    50    5             0.6400000
99     3        1   100    5             0.7000000
113    1        1  1000    5             0.5340832
114    2        1  1000    5             0.6706662
115    3        1  1000    5             0.6595000
116    6        1  1000    5             0.7410000
119    3        3  1000    5             0.5718667
123    3        5  1000    5             0.5366944
127    3       10  1000    5             0.5070496
131    3        1     1   12             1.0000000
147    3        1    50   12             0.8400000
163    3        1   100   12             0.7000000
177    1        1  1000   12             0.5437999
178    2        1  1000   12             0.6678137
179    3        1  1000   12             0.6590000
180    6        1  1000   12             0.7240000
183    3        3  1000   12             0.5791500
187    3        5  1000   12             0.5529619
191    3       10  1000   12             0.5108428
```

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

```
library(ggplot2)
hyper.grid.prediction |>
  ggplot(aes(x = factor(mtry), y = prediction.newpatient, group = mtry)) +
  theme_bw() +
  facet_grid(ntree ~ nodesize, labeller = label_both) +
  geom_boxplot() +
  xlab("mtry") + ylab("Prediction (newpatient)")
```


nodesize: 1
nodesize: 3
nodesize: 5
nodesize: 10
ntree: 1
ntree: 50
ntree: 100
ntree: 1000
Prediction (newpatient)
1.00
0.75
0.50
0.25
0.00
1
2
3
6
mtry

3 Exercise 3: Identifying risk factors

In this exercise we consider the paper: doi.org/10.1161/CIRCOUTCOMES.110.939371

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.

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.

```
# --- 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])
```

[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

```
#---  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? (Note that `tuned.forest.tumor` is the name of the tuned forest object)

```
ice.grid <- merge(
  data.frame(id = 1:nrow(tumor.data), tumor.data[, names(tumor.data) != "x.896"]),
  data.frame(x.896 = seq(min(tumor.data$x.896), max(tumor.data$x.896), length = 200))
)
ice.grid$pred <- predict(
  tuned.forest.tumor,
  newdata = ice.grid,
  type = "response"
)$predicted

library(ggplot2)
ice.grid |>
  ggplot(aes(x = x.896, y = pred, group = id, col = x.2124)) +
  geom_line() +
  theme_bw() + ylab("Predicted probability")
```

4.1 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]
```

```
[1] 0.08263242
```

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

```
[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]
```

```
[1] 0.06857844
```

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

```
[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
```

4.2 Growing a forest for different values of hyperparameters

4. Compute the forest prediction for patient 28 for different values of hyperparameters and for different random seeds. Comment on these results.

We define a grid of hyperparameter values and get the oob prediction for patient 28 for each combination:

```
hyper.grid <- expand.grid(
  mtry = floor((ncol(tumor.data) - 1) / c(12, 8, 4, 3, 2)),
  ntree = c(100, 500, 1000, 2500, 5000),
  nodesize = c(1, 3, 5, 10),
  seed = c(2, 1240, 1919133, 111099, 867),
  prediction = NA
)

for (j in 1:nrow(hyper.grid)) {
  tmp.forest <-
```

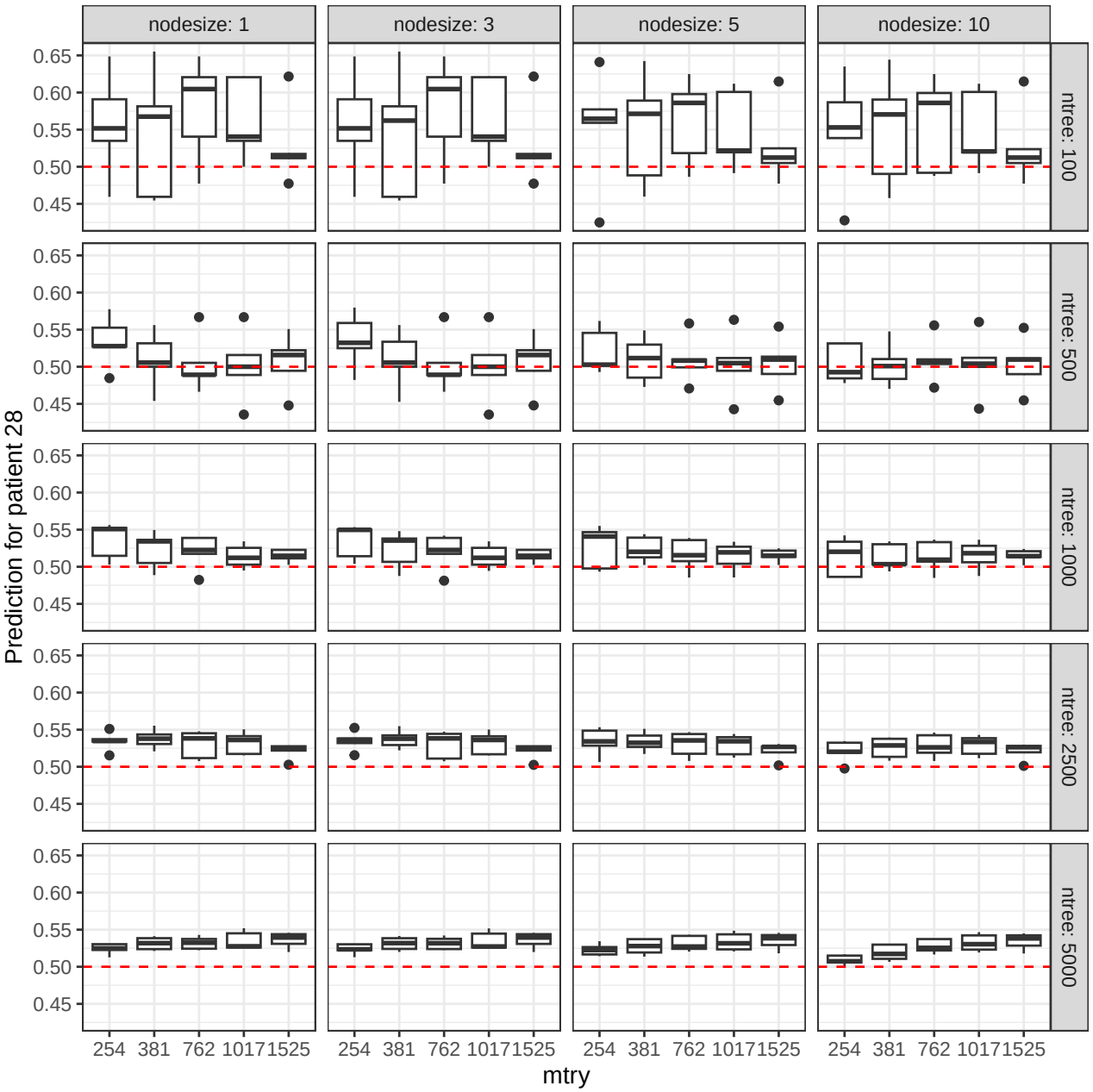
```
    rfsrc(
      y ~ .,
      tumor.data,
      mtry = hyper.grid[j, "mtry"],
      nodesize = hyper.grid[j, "nodesize"],
      ntree = hyper.grid[j, "ntree"],
      seed = hyper.grid[j, "seed"]
    )
  hyper.grid[j, "prediction"] <- tmp.forest$predicted.oob[28]
}
```

We can look at some of the predictions in plain text as follows, or get an overview with a boxplot as shown further below.

```
hyper.grid[order(hyper.grid$ntree), ][1:20, ]
```

```
   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
26  254   100        3    2  0.5517241
27  381   100        3    2  0.6551724
28  762   100        3    2  0.6206897
29 1017   100        3    2  0.6206897
30 1525   100        3    2  0.5172414
51  254   100        5    2  0.5772441
52  381   100        5    2  0.6423782
53  762   100        5    2  0.5860016
54 1017   100        5    2  0.6007800
55 1525   100        5    2  0.5123153
76  254   100       10    2  0.5868464
77  381   100       10    2  0.6442939
78  762   100       10    2  0.5860016
79 1017   100       10    2  0.6007800
80 1525   100       10    2  0.5123153
```

```
library(ggplot2)
hyper.grid |>
  ggplot(aes(x = factor(mtry), y = prediction)) +
  theme_bw() +
  facet_grid(ntree ~ nodesize, labeller = label_both) +
  geom_boxplot(width = 0.75) +
  geom_hline(yintercept = 0.5, color = "red", linetype = "dashed") +
  xlab("mtry") + ylab("Prediction for patient 28")
```



4.3 Tuning the forest

5. Compute (or extract) the estimated error rates of the forest models for different values of hyperparameters. Comment on these results.

We define the loss function:

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

We use the same grid of hyperparameter values as in the previous question. We choose to fix `ntree = 1000` since the predictions for patient 28 have more or less stabilized at that point (see results above).

```
hyper.grid.mse <- hyper.grid[hyper.grid$ntree == 1000, ]
```

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:

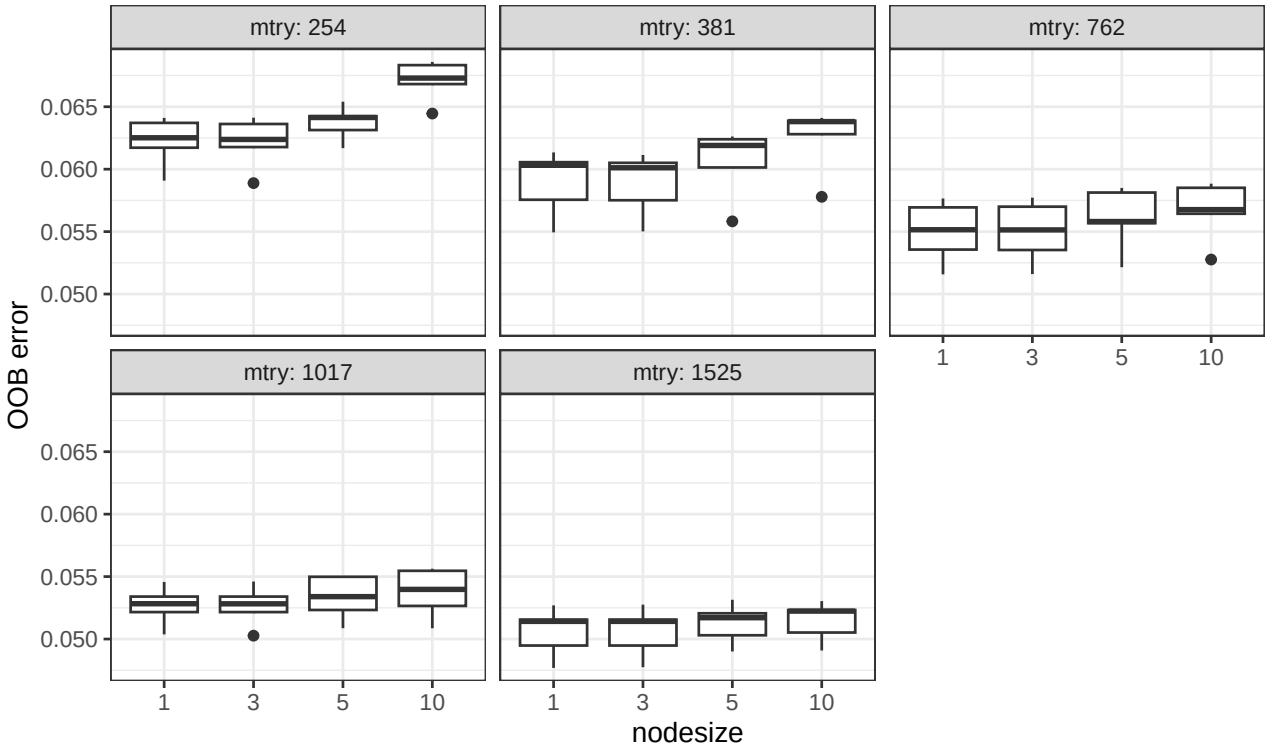
```
for (j in 1:nrow(hyper.grid.mse)) {
  tmp.forest <- rfsrc(
    y ~ .,
    tumor.data,
    mtry = hyper.grid.mse[j, "mtry"],
    nodesize = hyper.grid.mse[j, "nodesize"],
    ntree = hyper.grid.mse[j, "ntree"],
    seed = hyper.grid.mse[j, "seed"]
  )
  hyper.grid.mse[j, "oob.error"] <- loss.fun(tumor.data$y,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

hyper.grid.mse[order(hyper.grid.mse$oob.error), ][1:10, ]
```

```
    mtry ntree nodesize    seed prediction  oob.error
415 1525  1000        1     867  0.5227882 0.04768460
440 1525  1000        3     867  0.5227882 0.04774320
465 1525  1000        5     867  0.5217509 0.04900490
490 1525  1000       10     867  0.5209466 0.04908228
215 1525  1000        1 1919133  0.5147453 0.04947867
240 1525  1000        3 1919133  0.5147453 0.04947867
439 1017  1000        3     867  0.5254692 0.05026842
265 1525  1000        5 1919133  0.5148570 0.05030030
414 1017  1000        1     867  0.5254692 0.05037069
290 1525  1000       10 1919133  0.5143208 0.05051985
```

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

```
library(ggplot2)
hyper.grid.mse |>
  ggplot(aes(x = factor(nodesize), y = oob.error)) +
  theme_bw() +
  facet_wrap(vars(mtry), labeller = label_both) +
  geom_boxplot() +
  xlab("nodesize") + ylab("OOB error")
```



6. Continue with the (tuned) random forest model, corresponding to the one minimizing the estimated error rate.

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

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

This corresponds to the following values of hyperparameters:

```
opt.hypers
```

```
  mtry nodesize ntree  oob.error
5 1525        1  1000 0.05056447
```

```
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] 0.05102041
```

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

```
[1] 0.5014164
```

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
```

4.4 Interpretable machine learning measures

8. Compute the VIMP measures for the predictor variables of the tuned forest. Which variables seem important with this method?

We compute 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
)
```

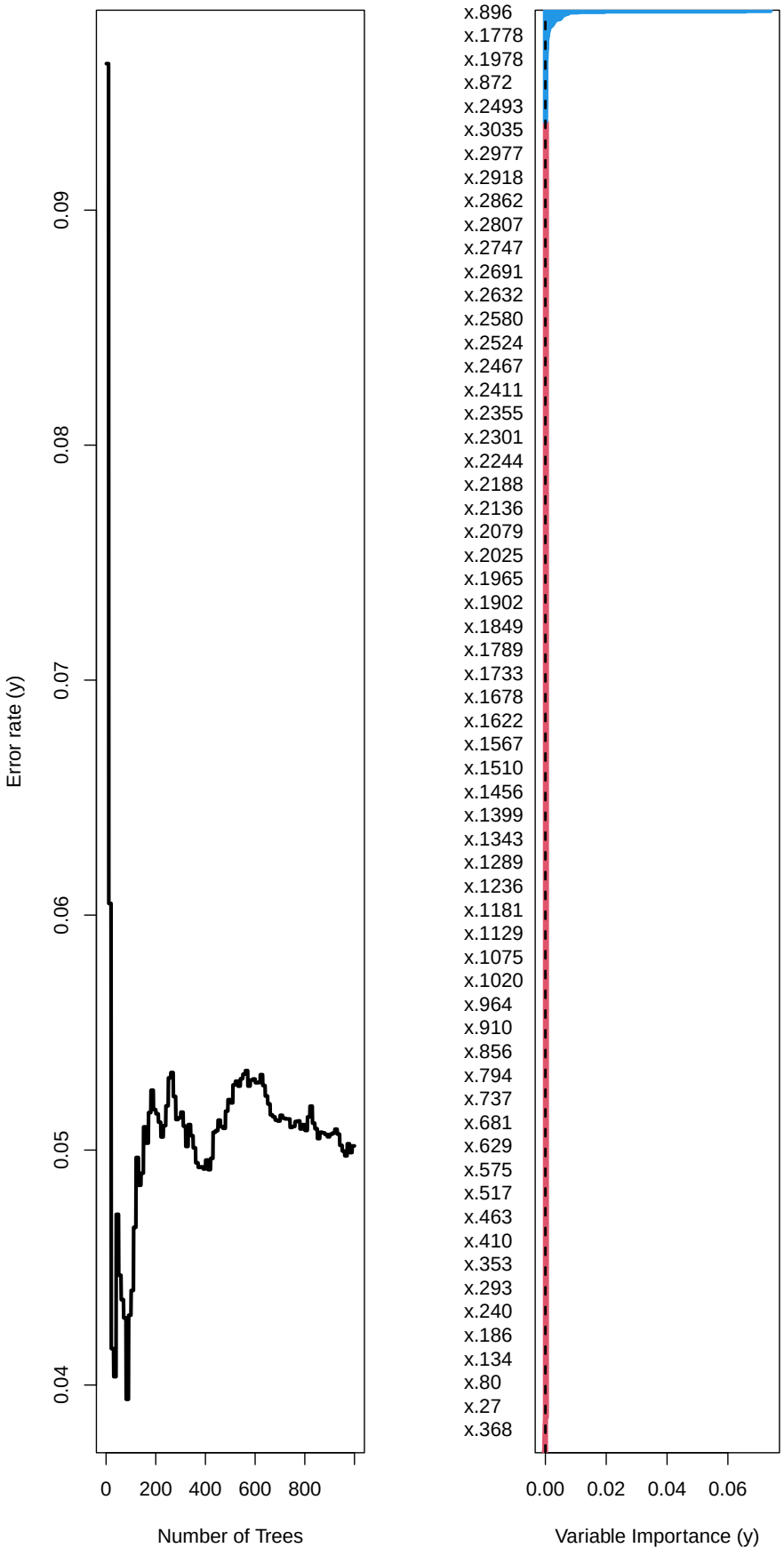
There are many variables. Let's look at the 25 with the highest value:

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

```
       ranked.no        vimp
x.2124         1 0.073984091
x.896          2 0.065885967
x.829          3 0.019401691
x.2386         4 0.010858275
x.2670         5 0.007581097
x.394          6 0.007564473
x.1413         7 0.007351798
x.808          8 0.006722768
x.2002         9 0.006381904
x.2939        10 0.006020706
x.766         11 0.005693903
x.1037        12 0.005534397
x.1907        13 0.005254604
x.2600        14 0.005146564
x.786         15 0.005135253
x.937         16 0.005121526
x.1448        17 0.004889751
x.1834        18 0.004818054
x.894         19 0.004291645
x.108         20 0.003607825
x.849         21 0.003155101
x.283         22 0.003140005
x.523         23 0.002966260
x.1995        24 0.002880564
x.378         25 0.002772662
```

We can also plot the VIMP:

```
plot(tuned.forest.tumor.vimp, verbose = FALSE)
```


Error rate (y)
0.04
0.05
0.06
0.07
0.08
0.09
0
200
400
600
800
Number of Trees
x.896
x.1778
x.1978
x.872
x.2493
x.3035
x.2977
x.2918
x.2862
x.2807
x.2747
x.2691
x.2632
x.2580
x.2524
x.2467
x.2411
x.2355
x.2301
x.2244
x.2188
x.2136
x.2079
x.2025
x.1965
x.1902
x.1849
x.1789
x.1733
x.1678
x.1622
x.1567
x.1510
x.1456
x.1399
x.1343
x.1289
x.1236
x.1181
x.1129
x.1075
x.1020
x.964
x.910
x.856
x.794
x.737
x.681
x.629
x.575
x.517
x.463
x.410
x.353
x.293
x.240
x.186
x.134
x.80
x.27
x.368
0.00
0.02
0.04
0.06
Variable Importance (y)

9. Compute 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         1.045
x.896          2         1.055
x.829          3         1.127
x.2670         4         1.152
x.2386         5         1.153
x.1834         6         1.154
x.2002         7         1.154
x.2939         8         1.154
x.808          9         1.156
x.394         10         1.157
x.1413        11         1.158
x.1037        12         1.159
x.2600        13         1.161
x.766         14         1.163
x.894         15         1.164
x.937         16         1.165
x.1907        17         1.165
x.1995        18         1.165
x.786         19         1.166
x.1448        20         1.167
x.283         21         1.168
x.108         22         1.169
x.2198        23         1.169
x.2489        24         1.170
x.515         25         1.171
```

The threshold computed by the function is:

```
md.obj$threshold
```

```
[1] 1.182552
```

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

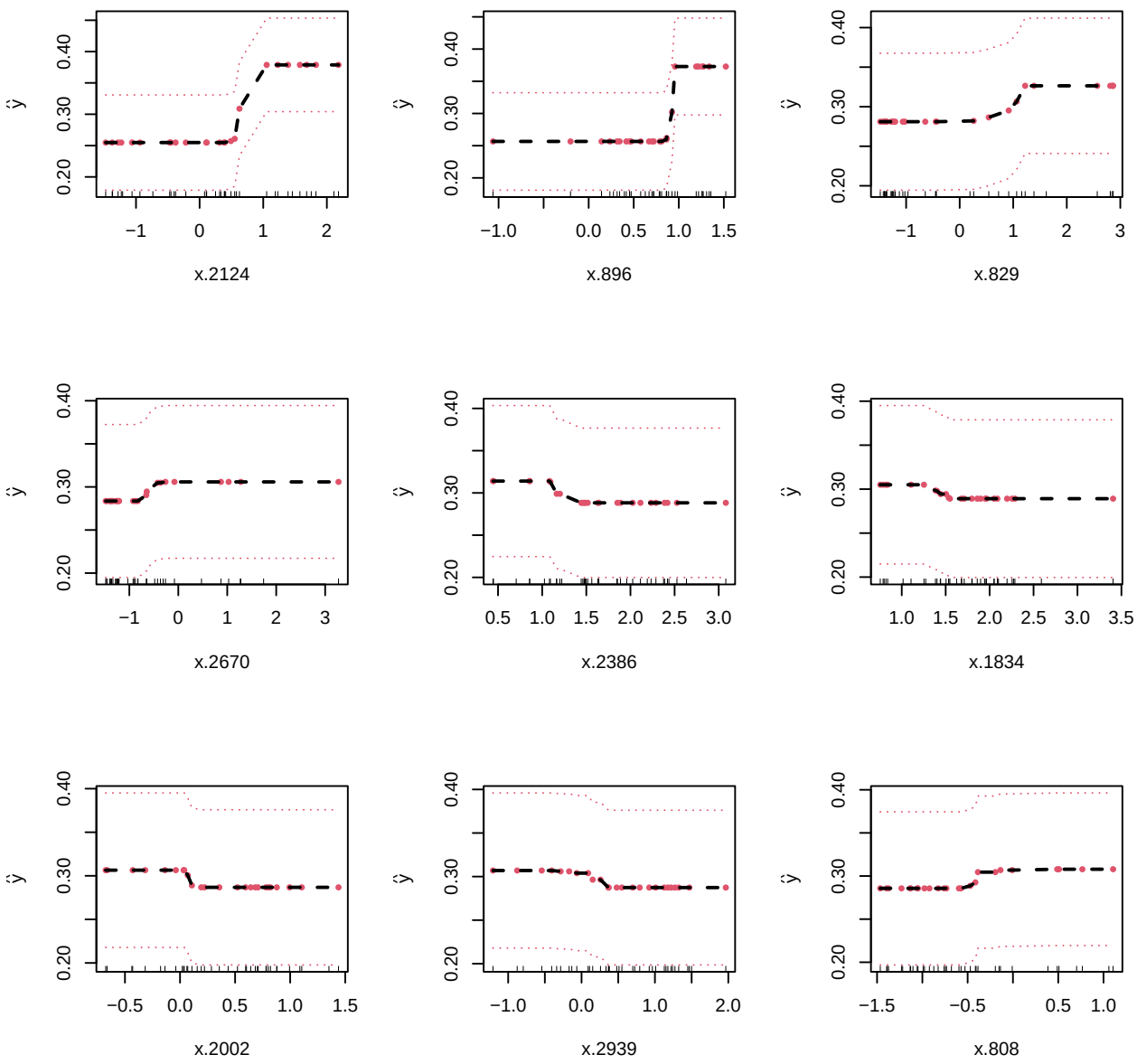
```
sum(md <= md.obj$threshold)
```

```
[1] 313
```

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.

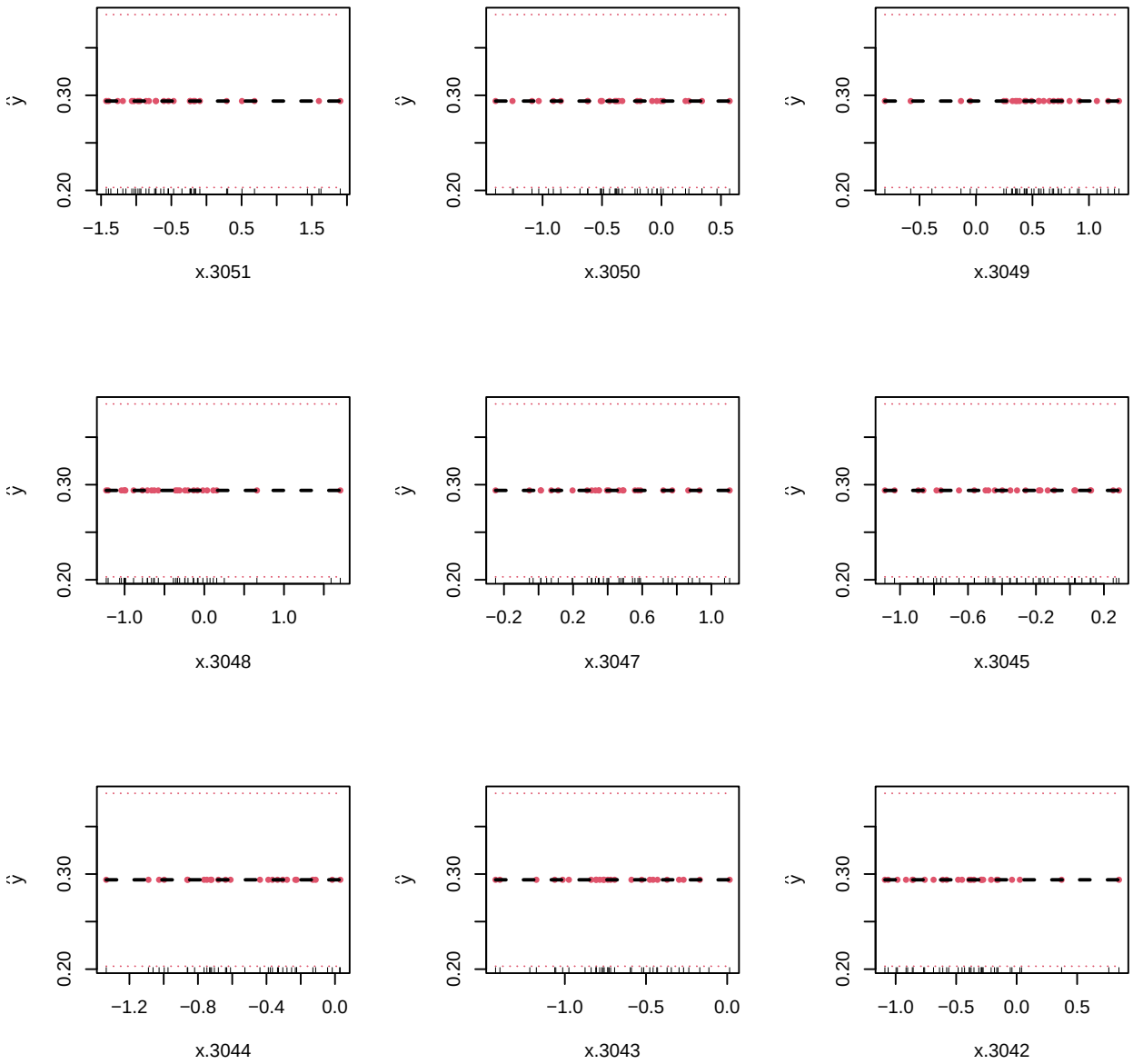
It takes too long to make the PDP plot for all variables, so we will simply consider first the 9 with lowest minimal depth:

```
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]
)
```

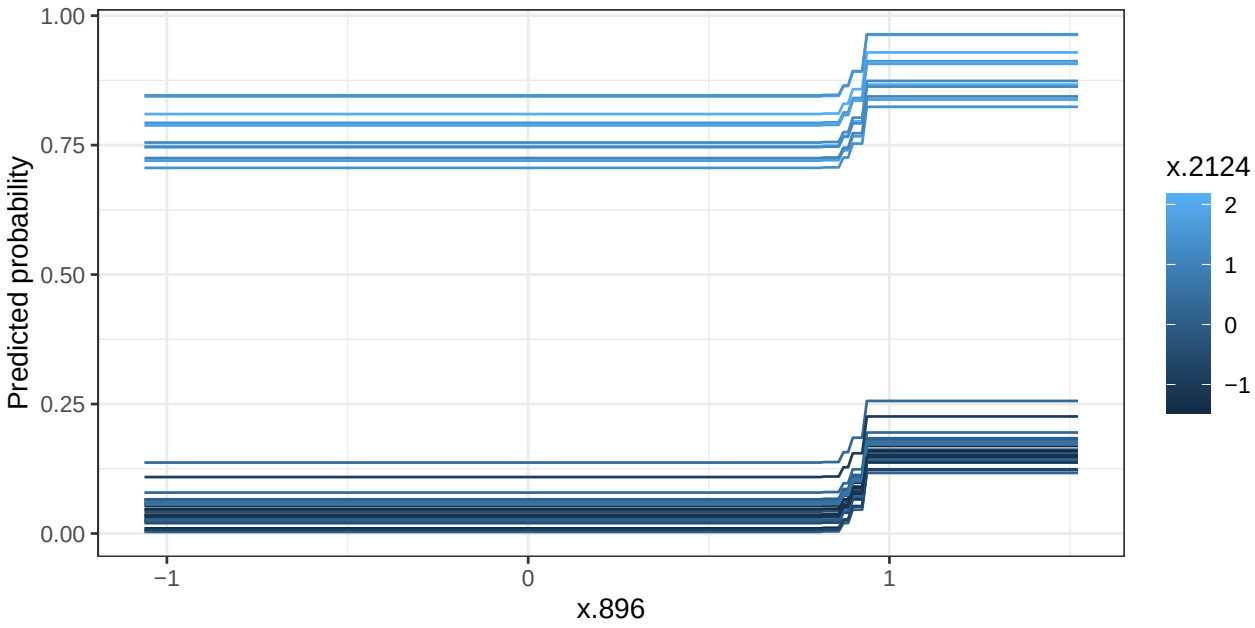


11. Produce the ICE plot for the variable `x.896`. Can you see differing patterns across individuals?

The code below produces the ICE plot for the variable `x.896`. Note that the plot is colored according to values of the variable `x.2124`.

```
ice.grid <- merge(
  data.frame(id = 1:nrow(tumor.data), tumor.data[, names(tumor.data) != "x.896"]),
  data.frame(x.896 = seq(min(tumor.data$x.896), max(tumor.data$x.896), length = 200))
)
ice.grid$pred <- predict(
  tuned.forest.tumor,
  newdata = ice.grid,
  type = "response"
)$predicted
```

```
library(ggplot2)
ice.grid |>
  ggplot(aes(x = x.896, y = pred, group = id, col = x.2124)) +
  geom_line() +
  theme_bw() + ylab("Predicted probability")
```



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

