

Sensitivity Analysis: Random Survival Forest

Mingcheng Hu

Table of contents

Load Data	2
Random Survival Forest (RSF)	6
Variable Selection	6
Cross Validation to Select the Best Number of Features	9
Model Fitting	11

```
library(tidyverse)
library(survival)
library(randomForestSRC)
library(caret)
library(survcomp)
library(parallel)
library(doParallel)
library(mcprogress) # wrap mclapply with progress bar.
library(kableExtra) # include knitr automatically

source("/work/users/y/u/youkias/BIOS-Material/BIOS992/utlis/csv_utils.r")
# * Don't use setwd() for Quarto documents!
# setwd("/work/users/y/u/youkias/BIOS-Material/BIOS992/data")

adjust_type <- ifelse(exists("params"), params$adjust_type, "full") #
  ↳ options: "minimal", "partial", "full"
impute_type <- ifelse(exists("params"), params$impute_type, "unimputed") #
  ↳ options: "unimputed", "imputed"
include_statin <- ifelse(exists("params"), params$include_statin, "no") #
  ↳ options: "yes", "no"

n_folds <- 10
set.seed(1234)
```

```
# string of parameters
adjust_type_str <- switch(adjust_type,
  minimal = "minimal",
  partial = "partial",
  full = "full"
)
print(paste0("Model Adjustment Type: ", adjust_type_str))
```

```
[1] "Model Adjustment Type: full"
```

```
impute_type_str <- switch(impute_type,
  unimputed = "unimputed",
  imputed = "imputed"
)
print(paste0("Data Imputation Type: ", impute_type_str))
```

```
[1] "Data Imputation Type: unimputed"
```

Load Data

```
if (include_statin == "yes") {
  data_train <-
  ↪ read.csv(paste0("/work/users/y/u/youkias/BIOS-Material/BIOS992/data/train_data_",
  ↪ impute_type_str, "_statin.csv"),
    header = TRUE
  )
} else {
  data_train <-
  ↪ read.csv(paste0("/work/users/y/u/youkias/BIOS-Material/BIOS992/data/train_data_",
  ↪ impute_type_str, ".csv"),
    header = TRUE
  )
}

data_train <- data_train[, -1] # the first column is the index generated by
  ↪ sklearn
(dim(data_train))
```

```
[1] 28127    100
```

```
data <- select_subset(data_train, type = adjust_type)
(dim(data))
```

```
[1] 28127    89
```

```
colnames(data)
```

```
[1] "event"           "time"
[3] "age"             "sex"
[5] "ethnicity"       "BMI"
[7] "smoking"         "diabetes"
[9] "systolic_bp"     "hypertension_treatment"
[11] "total_chol"      "hdl_chol"
[13] "education"       "activity"
[15] "max_workload"    "max_heart_rate"
[17] "HRV_MeanNN"      "HRV_SDNN"
[19] "HRV_RMSSD"       "HRV_SSD"
[21] "HRV_CVNN"        "HRV_CVSD"
[23] "HRV_MedianNN"    "HRV_MadNN"
[25] "HRV_MCVNN"       "HRV_IQRNN"
[27] "HRV_SDRMSSD"     "HRV_Prc20NN"
[29] "HRV_Prc80NN"     "HRV_pNN50"
[31] "HRV_pNN20"       "HRV_MinNN"
[33] "HRV_MaxNN"       "HRV_HTI"
[35] "HRV_TINN"        "HRV_LF"
[37] "HRV_HF"          "HRV_VHF"
[39] "HRV_TP"          "HRV_LFHF"
[41] "HRV_LFn"         "HRV_HFn"
[43] "HRV_LnHF"        "HRV_SD1"
[45] "HRV_SD2"         "HRV_SD1SD2"
[47] "HRV_S"           "HRV_CSI"
[49] "HRV_CVI"         "HRV_CSI_Modified"
[51] "HRV_PIP"         "HRV_IALS"
[53] "HRV_PSS"         "HRV_PAS"
[55] "HRV_GI"          "HRV_SI"
[57] "HRV_AI"          "HRV_PI"
[59] "HRV_C1d"         "HRV_C1a"
[61] "HRV_SD1d"        "HRV_SD1a"
```

[63] "HRV_C2d"	"HRV_C2a"
[65] "HRV_SD2d"	"HRV_SD2a"
[67] "HRV_Cd"	"HRV_Ca"
[69] "HRV_SDNNd"	"HRV_SDNNa"
[71] "HRV_ApEn"	"HRV_ShanEn"
[73] "HRV_FuzzyEn"	"HRV_MSEn"
[75] "HRV_CMSEn"	"HRV_RCMSEn"
[77] "HRV_CD"	"HRV_HFD"
[79] "HRV_KFD"	"HRV_LZC"
[81] "HRV_DFA_alpha1"	"HRV_MFDFA_alpha1_Width"
[83] "HRV_MFDFA_alpha1_Peak"	"HRV_MFDFA_alpha1_Mean"
[85] "HRV_MFDFA_alpha1_Max"	"HRV_MFDFA_alpha1_Delta"
[87] "HRV_MFDFA_alpha1_Asymmetry"	"HRV_MFDFA_alpha1_Fluctuation"
[89] "HRV_MFDFA_alpha1_Increment"	

```
data <- tibble::as_tibble(data)
```

```
# * There are some imputed ethnicity set to "e". We will exclude them at this
  ↪ time.
```

```
data <- data %>%
  filter(ethnicity != "e")
```

```
# * We also need to manually relevel the categorical variables
```

```
data <- data %>%
  mutate(
    # Set "Never" (0) as baseline for smoking
    smoking = factor(smoking,
      levels = c("0", "1", "2", "-3"),
      labels = c("Never", "Previous", "Current", "Prefer not to
        ↪ answer")
    ),

    # Set "No" (0) as baseline for diabetes
    diabetes = factor(diabetes,
      levels = c("0", "1", "-1", "-3"),
      labels = c("No", "Yes", "Do not know", "Prefer not to answer")
    ),

    # Ensure other categorical variables are properly factored
    ethnicity = factor(ethnicity,
      levels = c("1", "2", "3", "4", "5", "6"),
      labels = c("White", "Mixed", "Asian/Asian British", "Black/Black
        ↪ British", "Chinese", "Other")
    )
  )
```

```

    ),
    education = factor(education,
      levels = c("1", "2", "3", "4", "5", "6", "-7", "-3"),
      labels = c(
        "College/University degree", "A levels/AS levels",
        "0 levels/GCSEs", "CSEs", "NVQ/HND/HNC",
        "Other professional", "None of the above",
        "Prefer not to answer"
      )
    ),
    activity = factor(activity,
      levels = c("0", "1", "2"),
      labels = c("Low", "Moderate", "High")
    ),
    sex = factor(sex,
      levels = c("0", "1"),
      labels = c("Female", "Male")
    ),
    hypertension_treatment = factor(hypertension_treatment,
      levels = c("0", "1"),
      labels = c("No", "Yes")
    )
  )
)

```

```

# * It is very hard to compare the HR as different predictors are on
↳ different magnitudes, so we need to normalize them.
time_col <- data$time
event_col <- data$event
data <- data %>%
  select(-c(time, event)) %>%
  mutate(across(where(is.numeric), scale)) %>%
  mutate(
    time = time_col,
    event = event_col
  )

```

Note now the interpretation of HR is different! For example, if $HR=1.16$ for the predictor in the univariate model fitted using scaled data, it means that each standard deviation increase is associated with 16% higher risk of event.

```
# For RSF model, we don't need to exclude the missing values
```

Random Survival Forest (RSF)

Variable Selection

The `method` argument can be set to `vh` instead for variable hunting, which should be used for problems where the number of variables is substantially larger than the sample size.

```
n_cores <- min(parallel::detectCores() - 1, 32)
cl <- makeCluster(n_cores)
registerDoParallel(cl)
```

```
rsf_var_select <- var.select.rfsrc(Surv(time, event) ~ .,
  data = data,
  method = "md",
  seed = 1234,
  ntree = 200,
  parallel = TRUE
) # minimal depth variable selection
```

```
running forests ...
minimal depth variable selection ...
```

```
-----
family           : surv
var. selection    : Minimal Depth
conservativeness  : medium
x-weighting used? : TRUE
dimension         : 87
sample size      : 19577
ntree            : 200
nsplit           : 10
mtry             : 30
nodesize         : 2
refitted forest   : FALSE
model size       : 87
depth threshold   : 13.917
```

PE (true OOB) : 32.1211

Top variables:

	depth	vimp
age	1.105	0.116
hypertension_treatment	1.950	0.024
sex	2.335	0.016
systolic_bp	2.940	0.003
diabetes	3.285	0.045
max_heart_rate	3.530	-0.014
BMI	3.545	0.003
max_workload	3.605	-0.006
hdl_chol	3.865	-0.002
total_chol	4.735	0.002
education	4.740	0.004
HRV_Prc20NN	5.010	-0.011
HRV_ApEn	5.505	-0.001
smoking	5.545	-0.007
HRV_IALS	5.610	0.000
HRV_PIP	5.670	0.002
HRV_KFD	5.700	-0.002
HRV_CD	5.750	-0.006
HRV_MinNN	5.865	0.000
HRV_TINN	5.925	0.012
HRV_HTI	5.950	-0.011
HRV_PAS	5.990	-0.003
HRV_LZC	6.040	-0.006
HRV_ShanEn	6.130	-0.002
HRV_AI	6.140	-0.008
HRV_PI	6.160	-0.002
ethnicity	6.215	-0.017
HRV_pNN50	6.215	-0.003
HRV_MadNN	6.240	-0.003
HRV_MFDFA_alpha1_Delta	6.245	-0.004
HRV_GI	6.260	-0.005
HRV_MFDFA_alpha1_Asymmetry	6.290	-0.006
HRV_MedianNN	6.310	-0.001
HRV_SI	6.380	-0.006
HRV_MFDFA_alpha1_Peak	6.380	-0.009
HRV_MFDFA_alpha1_Max	6.410	-0.004
HRV_Prc80NN	6.450	-0.006
HRV_HFn	6.500	-0.006

HRV_FuzzyEn	6.535	-0.007
HRV_MCVNN	6.560	-0.001
HRV_MFDFA_alpha1_Mean	6.560	-0.005
HRV_Cd	6.590	0.000
HRV_Ca	6.605	-0.002
HRV_C2a	6.615	-0.003
HRV_CMSEn	6.615	-0.005
HRV_MFDFA_alpha1_Increment	6.645	-0.004
HRV_MFDFA_alpha1_Width	6.670	-0.003
HRV_LF	6.680	-0.002
HRV_C2d	6.685	-0.002
HRV_C1d	6.700	-0.004
HRV_MFDFA_alpha1_Fluctuation	6.705	-0.006
HRV_IQRNN	6.720	0.000
HRV_SDRMSSD	6.725	-0.004
HRV_HF	6.740	-0.007
HRV_SD1SD2	6.740	-0.006
HRV_MSEn	6.740	-0.005
HRV_MeanNN	6.750	-0.004
HRV_PSS	6.805	-0.003
HRV_TP	6.810	-0.005
HRV_C1a	6.810	-0.004
HRV_DFA_alpha1	6.810	-0.005
HRV_VHF	6.825	-0.001
HRV_MaxNN	6.840	-0.005
HRV_RCMSEn	6.865	-0.005
HRV_pNN20	6.875	-0.002
HRV_HFD	6.905	-0.002
HRV_LnHF	6.945	-0.007
HRV_LFHF	6.950	-0.004
HRV_CSI	6.990	-0.002
HRV_CVSD	7.025	-0.003
HRV_LFn	7.050	-0.003
HRV_CSI_Modified	7.140	-0.001
HRV_CVNN	7.480	-0.001
HRV_SDNN	7.535	0.001
HRV_RMSSD	7.560	-0.001
HRV_SD2a	7.560	-0.002
HRV_SDNNa	7.565	0.001
HRV_SD1	7.635	-0.005
HRV_SD2	7.650	0.000
HRV_SD1d	7.695	-0.002
HRV_SDNNd	7.715	0.001

HRV_SDSD	7.750	-0.001
HRV_SD2d	7.755	-0.001
HRV_SD1a	7.825	-0.001
HRV_CVI	7.900	-0.001
HRV_S	8.025	0.000
activity	8.050	0.000

```
stopCluster(cl)
```

```
vars_ranked <- rsf_var_select$topvars
```

Cross Validation to Select the Best Number of Features

We will use 10-fold cross validation to select the best number of features used in the model.

```
set.seed(1234)
folds <- createFolds(data$event, k = n_folds) # return indices of folds

cv_errors <- pmclapply(seq(1, length(vars_ranked), by = 1),
  ↪ function(num_vars) {
    message(paste0("Calculating the CV error with ", num_vars, " variables"))
    selected_vars <- vars_ranked[1:num_vars]

    fold_errors <- sapply(folds, function(fold_idx) {
      # * We adopt same approach as XGBoost to avoid segmentation fault.
      train_data_fold <- data[-fold_idx, c("time", "event", selected_vars)]
      train_data_fold <- model.frame(~ . - 1, data = train_data_fold,
  ↪ na.action = na.pass)
      train_data_fold <- model.matrix(~ . - 1, data = train_data_fold)
      train_data_fold <- as.data.frame(train_data_fold)
      val_data_fold <- data[fold_idx, c("time", "event", selected_vars)]
      val_data_fold <- model.frame(~ . - 1, data = val_data_fold, na.action
  ↪ = na.pass)
      val_data_fold <- model.matrix(~ . - 1, data = val_data_fold)
      val_data_fold <- as.data.frame(val_data_fold)
      model <- rfsrc.fast(Surv(time, event) ~ .,
        data = train_data_fold,
        ntree = 200,
        forest = TRUE
```

```

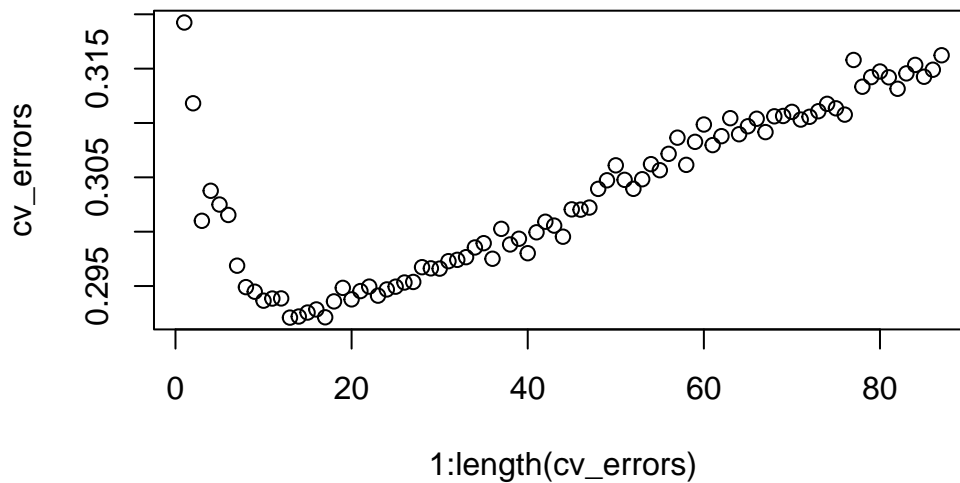
    )
    pred <- predict(model,
      newdata = val_data_fold,
      na.action = "na.impute" # * There may be missing values in the
        ↪ dataset
    )$predicted # define pred has attributes survival(sample_size*time)
    ↪ and predicted(sample_size) for risk
    # Use C-index to measure the performance of the model
    1 - concordance.index(
      pred, # pass risk prediction for first argument
      val_data_fold$time,
      val_data_fold$event
    )$c.index
  })
  mean(fold_errors)
}, title = "Cross Validation to Select the Best Number of Features")

```

```

cv_errors <- as.numeric(cv_errors)
plot(1:length(cv_errors), cv_errors)

```



```
best_num_vars <- which.min(cv_errors)
vars_selected <- vars_ranked[1:best_num_vars]
```

```
print(paste0("The best number of features to retain is ", best_num_vars))
```

```
[1] "The best number of features to retain is 13"
```

Model Fitting

```
data_selected <- data[, c("time", "event", vars_selected)]
data_selected <- model.frame(~ . - 1, data = data_selected, na.action =
  ↪ na.pass)
data_selected <- model.matrix(~ . - 1, data = data_selected)
data_selected <- as.data.frame(data_selected)

# Before formally fitting the model, we can tune the hyperparameters to find:
# 1. optimal mtry (possible split at each node)
# 2. optimal nodesize (minimum size of terminal nodes)
rsf_tuned <- tune.rfsrc(
  Surv(time, event) ~ .,
  data = data_selected,
)

rsf_model <- rfsrc(Surv(time, event) ~ .,
  data = data_selected,
  ntree = 500,
  mtry = rsf_tuned$best.mtry,
  nodesize = rsf_tuned$best.nodesize
)
```

```
data_full <- model.frame(~ . - 1, data = data, na.action = na.pass)
data_full <- model.matrix(~ . - 1, data = data_full)
data_full <- as.data.frame(data_full)
# We also fit the full model
rsf_tuned_full <- tune.rfsrc(
  Surv(time, event) ~ .,
  data = data_full,
)
```

```
rsf_model_full <- rfsrc(Surv(time, event) ~ .,  
  data = data_full,  
  ntree = 1000,  
  mtry = rsf_tuned_full$best.mtry,  
  nodesize = rsf_tuned_full$best.nodesize  
)
```