Build Survival Model: Random Survival Forest

Mingcheng Hu

Table of contents


```
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/yuukias/BIOS-Material/BIOS992/utils/csv_utils.r")
# * Don't use setwd() for Quarto documents!
# setwd("/work/users/y/u/yuukias/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, "imputed") #
 ↪  options: "unimputed", "imputed"
include_statin <- ifelse(exists("params"), params$include_statin, "no") #
 ↪  options: "yes", "no"

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

```r
# 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"
```

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

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

Load Data

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

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

```r
# * 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",
                "O 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

stopCluster(cl)
```

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

Cross Validation to Select the Best Number of Features

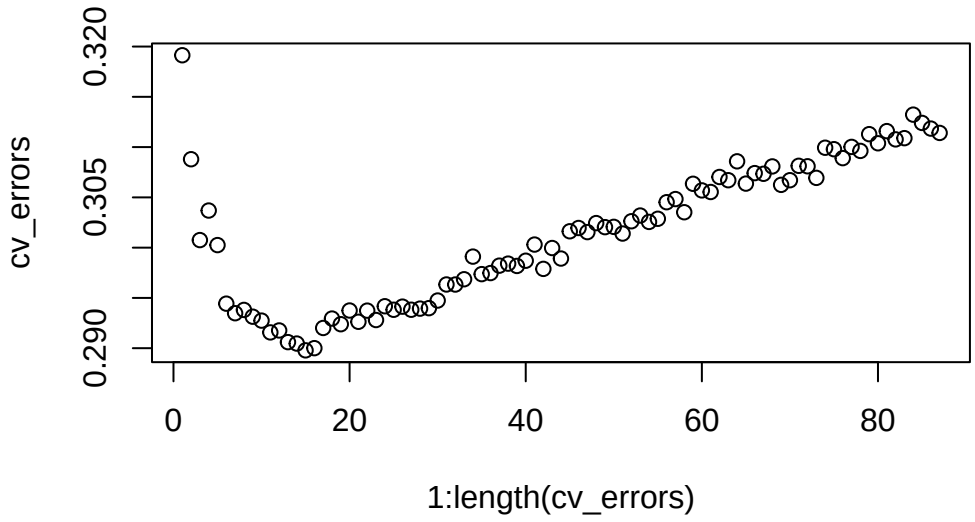
We will use 10-fold cross validation to select the best number of features used in the model.

```
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 15"
```

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
)
```

```
# plot.rfsrc?
# plot.variable.rfsrc?
# print.rfsrc?
```