randomForestRHF: getting started with randomForestRHF vignette

Hemant Ishwaran (hemant.ishwaran@gmail.com) Min Lu (luminwin@gmail.com)

Udaya B. Kogalur (ubk@kogalur.com)

2026-05-19

Contents

Overview

Event-time analysis usually starts with a simple question. Given what is known about a subject now, what is the subject's risk of an event later? In many applications the word ``now" matters. Patient physiology, medications, laboratory values, device measurements, exposures, and symptoms can all evolve during follow-up. A baseline-only model uses one snapshot of this information. A dynamic hazard model tries to use the trajectory.

Random Hazard Forests are designed for this dynamic setting. The input data are arranged in counting-process format, with one row per interval over which recorded covariates are treated as current. The response is written

```
Surv(id, start, stop, event)
```

where `id` identifies the subject, `start` and `stop` define the interval, and `event` marks whether the event occurred at the end of that interval. Static covariates appear once per subject and are carried through the repeated rows. Time-dependent covariates may change from interval to interval.

RHF differs from a standard Random Survival Forest (RSF) in two central ways. First, RHF directly estimates the hazard function rather than using only a terminal-node summary for a time-static survival curve. Second, RHF naturally handles time-dependent covariates by routing interval-level records through trees. It retains the spirit of random forests [1] and random survival forests [2], but the splitting rule is derived from a hazard-based likelihood for time-varying data [3–5].

> **What RHF returns**
>
> For each subject and each time point in the working time grid (stored in `time.interest`), a fitted `rhf()` object returns an estimated hazard and cumulative hazard. RHF uses bootstrap sampling (bagging), thus for inference and model checking on the training sample, always use the out-of-bag matrices `hazard.oob` and `chf.oob`.

The statistical target

For subject $i = 1, \ldots, n$, let T_i denote the observed follow-up time and $\delta_i \in \{0, 1\}$ the event indicator. Let $\{{\bf X}_i(t): t \ge 0\}$ denote the time-dependent covariate process, with

$$\mathbf{X}_i(t) = \left(X_i^{(1)}(t), \ldots, X_i^{(p)}(t)\right)^T$$

its value at time t. Define the at-risk process and event counting process by

$$Y_i(t) = \mathbf{1}_{\{T_i \ge t\}}, \qquad N_i(t) = \mathbf{1}_{\{T_i \le t,\, \delta_i = 1\}}.$$

The continuous-time hazard is

$$\lambda(t, \mathbf{X}_i(t)) = \lim_{\Delta t \downarrow 0} \frac{\mathbb{P}\{T_i \in [t, t+\Delta t) \mid T_i \ge t,\ \text{history through } t\}}{\Delta t}.$$

This is the natural prediction target when risk is allowed to change over time. It records the instantaneous event rate among subjects who are still at risk and whose current history is known.

Writing the log-hazard as

$$F(t, \mathbf{x}) = \log\{\lambda(t, \mathbf{x})\},$$

the counting-process log-likelihood has the familiar form

$$\ell(F) = \sum_{i=1}^{n} \left\{ \int F(t, \mathbf{X}_i(t))\, dN_i(t) - \int Y_i(t) \exp\{F(t, \mathbf{X}_i(t))\}\, dt \right\}.$$

RHF grows trees by reducing the empirical risk

$$R_n(F) = -\frac{1}{n}\ell(F).$$

This is the key modeling step. Rather than measuring separation between daughter nodes using log-rank statistics, as in standard random survival forests, RHF scores each candidate split by the gain it produces in the nodewise hazard likelihood. Thus tree induction directly targets local estimation of the conditional hazard given the observed time-dependent covariate process. This results in a likelihood-based forest constructed by empirical hazard risk reduction, not a log-rank procedure adapted to time-dependent covariates.

How RHF trees are grown

A RHF tree partitions the covariate space into terminal node regions. During splitting, RHF uses a simple working model in which the log-hazard is constant within a node. If A_j is a node, define

$$U_j = \frac{1}{n}\sum_{i=1}^{n} \int Y_i(t) \mathbf{1}_{\{\mathbf{X}_i(t) \in A_j\}}\, dt, \qquad V_j = \frac{1}{n}\sum_{i=1}^{n} \delta_i \mathbf{1}_{\{\mathbf{X}_i(T_i) \in A_j\}}.$$

Here U_j is the normalized amount of at-risk time spent in node A_j, and V_j is the normalized number of events whose covariates fall in that node at the event time. For a fixed node, the best constant log-hazard is

$$c_j^* = \log\left(\frac{V_j}{U_j}\right),$$

whenever the ratio is well-defined. This is the log of the empirical hazard in that node.

For a candidate split of node A_j into left and right children $A_{j,1}$ and $A_{j,2}$, let $(U_{j,1}, V_{j,1})$ and $(U_{j,2}, V_{j,2})$ be the same statistics computed in the two children. The empirical risk reduction is

$$\Delta_n(A_{j,1}, A_{j,2}) = V_{j,1}\log\left(\frac{V_{j,1}}{U_{j,1}}\right) + V_{j,2}\log\left(\frac{V_{j,2}}{U_{j,2}}\right) - (V_{j,1}+V_{j,2})\log\left(\frac{V_{j,1}+V_{j,2}}{U_{j,1}+U_{j,2}}\right).$$

RHF selects the split that gives the largest risk reduction, subject to node size, event count, and at-risk-time constraints. In practice, the argument `nsplit` controls how many random split points are tried for each variable. Setting `nsplit = 0` evaluates all possible split points, which is more deterministic but usually slower.

Pseudo-individuals and predictable covariates

Time-dependent covariates are represented by pseudo-individuals. Suppose subject i contributes intervals

$$(S_{i,1}, T_{i,1}], \ldots, (S_{i,n_i}, T_{i,n_i}],$$

where intervals end at covariate updates, events, censoring times, or changes in at-risk status. The interval carries the predictable covariate value

$$\mathbf{Z}_{i,r} = \mathbf{X}_i(T_{i,r}-),$$

the last value available just before the end of the interval. A split rule is therefore applied to $\mathbf{Z}_{i,r}$, not to information that would only become available after the interval. This is what lets RHF use longitudinal data without violating the time order of prediction.

For users, the practical interpretation is straightforward. Each row in the counting-process data is an interval-level record. If a subject's covariates change over time, the subject has multiple rows. RHF treats those rows as the units that travel through each tree, but predictions are stitched back together at the subject level.

Terminal-node hazards and ensemble prediction

The constant-hazard model above is used to choose splits efficiently. After the tree structure is fixed, each terminal node receives a fully time-varying Nelson–Aalen cumulative hazard estimator [3, 6]. On a working grid

$$\mathcal{T} = \{0 = t_0 < t_1 < \cdots < t_q\},$$

the node-level hazard can be represented as a piecewise-constant increment

$$\widehat{\lambda}_A(t) = \frac{\widehat{\Lambda}_A(t_{k+1}) - \widehat{\Lambda}_A(t_k)}{t_{k+1} - t_k}, \qquad t \in [t_k, t_{k+1}).$$

A subject can move across terminal nodes as its covariates change. Therefore, RHF obtains a subject-specific hazard by stitching together the terminal-node hazards visited by the subject's interval-level records. For each tree b, RHF estimates both a case specific hazard $\widehat{\lambda}_{i,b}$ and a case specific cumulative hazard $\widehat{\Lambda}_{i,b}$. These are then averaged over trees. Thus, for a working time point t_j,

$$\widehat{\lambda}_i^{\mathrm{RHF}}(t_j) = \frac{1}{B}\sum_{b=1}^{B} \widehat{\lambda}_{i,b}(t_j), \qquad \widehat{\Lambda}_i^{\mathrm{RHF}}(t_j) = \frac{1}{B}\sum_{b=1}^{B} \widehat{\Lambda}_{i,b}(t_j).$$

The package returns these quantities on the working time grid stored in `time.interest`.

Counting-process data layout

The canonical RHF formula is
`Surv(id, start, stop, event) ~ predictors`

The four response columns have the following meaning.

Column	Meaning
`id`	Subject identifier. Repeated rows correspond to the same subject.
`start`	Beginning of the interval.
`stop`	End of the interval. Must be larger than `start`.
`event`	Event indicator at the end of the interval. Use 1 for event and 0 otherwise.

Counting-process layout is used throughout RHF. Genuine TDC data are usually entered directly in this format. Standard time-static survival data are converted to the same layout using `convert.counting()`, as described next. The software works internally on a normalized time scale, while the output grid is reported back on the data scale used by the analyst.

Data preparation checklist

1. Use one row per interval, not one row per measurement if several measurements belong to the same unchanged interval.
2. Keep `id`, `start`, `stop`, and `event` in the data frame and use the same names in the formula.
3. Make sure `stop > start`. Very short or invalid intervals are not useful for estimating at-risk time.
4. Encode character variables as factors or numeric variables before fitting.
5. RHF currently uses complete interval records for the variables in the formula; impute or remove missing predictor values before fitting.

Time-static survival data and `convert.counting()`

RHF also handles ordinary time-static survival problems. In that setting each subject has a single event or censoring time and the predictors are fixed at baseline or are otherwise treated as fixed for the analysis. The helper `convert.counting()` converts such data into the counting-process format required by `rhf()`. In this way, users can fit RHF to familiar right-censored data.

For a time-static problem, the cumulative hazard returned by RHF can be mapped to the usual survival curve by

$$\widehat{S}_i(t) = \exp\{-\widehat{\Lambda}_i(t)\}.$$

For genuine TDC problems, there is technically no survival function, therefore RHF deliberately focuses on hazards and cumulative hazards as the target because these quantities are always well defined and exist.

Quick installation

Install the release version from CRAN.

```
install.packages("randomForestRHF")

library(randomForestRHF)
library(survival)
```

Users who want beta features under active development can install from GitHub.

```
install.packages("remotes")
remotes::install_github("kogalur/randomForestRHF")
```

The examples below also use data sets from `randomForestSRC`. If that package is not already installed, install it before running the examples.

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

Example 1. A first RHF fit on standard survival data

We begin with the primary biliary cholangitis (PBC) data set from `randomForestSRC`, a simple example with no time-dependent covariates. The goal here is to show how RHF handles standard survival data. The helper `convert.counting()` converts the data into a start–stop format with one interval per subject as required by RHF.

```
library(randomForestRHF)
library(survival)

data(pbc, package = "randomForestSRC")
pbc0 <- na.omit(pbc)

d <- convert.counting(Surv(days, status) ~ ., pbc0)
head(d[, c("id", "start", "stop", "event")])

f <- "Surv(id, start, stop, event) ~ ."
fit <- rhf(f, d, ntree = 250, seed = -20260422)
print(fit)
```

The formula tells RHF that the response is a counting-process survival object. The right-hand side `~ .` means that all remaining columns are predictors. The printed output summarizes the forest size, the number of records and subjects, the splitting rule, and the out-of-bag risk.

The object contains a working time grid and subject-level hazard and cumulative hazard estimates.

```
str(fit$time.interest)
dim(fit$hazard.oob)
dim(fit$chf.oob)
```

A row of `hazard.oob` is a subject-specific out-of-bag hazard curve evaluated on `fit\$time.interest`. A row of `chf.oob` is the corresponding out-of-bag cumulative hazard curve. Because this is a time-static survival example, the usual survival curve is obtained by exponentiating the negative cumulative hazard.

```
case.index <- 1:10
S.oob <- exp(-fit$chf.oob)

matplot(fit$time.interest,
        t(S.oob[case.index, , drop = FALSE]),
        type = "l", lty = 1,
        xlab = "Time", ylab = "OOB survival")
```

The hazard information can be displayed using the package plotting interface.

```
plot(fit, idx = c(1, 5, 10))
plot(fit, idx = c(1, 5, 10), hazard.only = TRUE)
```

What to look for

In the PBC example, each subject has one row, so the model behaves like a hazard-focused forest for standard survival data. The important object to watch is the hazard. If two subjects have different covariate profiles, RHF can assign different hazard trajectories over the same time grid. The cumulative hazard then integrates those trajectories. For this time-static example, the survival curve follows from $\widehat{S}(t) = \exp\{-\widehat{\Lambda}(t)\}$.

Example 2. Choosing the time grid

The argument `ntime` controls the grid on which hazards and cumulative hazards are returned. The default requests a moderate grid chosen from observed event times. The companion argument `min.events.per.gap` avoids creating many grid intervals with too few events.

```
fit50 <- rhf(f, d, ntree = 250, ntime = 50,
             min.events.per.gap = 10,
             seed = -1)

fit100 <- rhf(f, d, ntree = 250, ntime = 100,
              min.events.per.gap = 5,
              seed = -1)

length(fit50$time.interest)
length(fit100$time.interest)
```

A larger grid can show more time detail, but it can also make tail estimates noisier when few subjects remain at risk. In applied work, it is often useful to begin with the default grid, inspect the fitted hazard curves, and then increase `ntime` only when the additional time resolution answers a scientific question.

You may also pass a numeric vector of requested time points. The software aligns those times to observed event times.

```
requested.time <- quantile(d$stop, probs = seq(0.1, 0.9, by = 0.1))
fit.grid <- rhf(f, d, ntree = 250, ntime = requested.time,
                seed = -2)
fit.grid$time.interest
```

Example 3. Time-dependent covariates

The built-in simulation function creates counting-process data with interval-level covariates. This is the setting RHF is designed for. Subjects in the simulated data can contribute several rows, and covariate values can change across rows.

```
sim <- hazard.simulation(1)
d.tdc <- sim$dta

head(d.tdc)
with(d.tdc, table(id))[1:5]

f.tdc <- "Surv(id, start, stop, event) ~ ."
fit.tdc <- rhf(f.tdc, d.tdc,
               ntree = 500,
               treesize = 30,
               nodesize = 15,
               seed = -3)

print(fit.tdc)
plot(fit.tdc, idx = 1:3, hazard.only = TRUE)
```

The object `d.tdc` already contains the pseudo-individual representation. For example, if subject 12 has five rows, RHF can route those five records to different nodes as the subject's covariates evolve. The final prediction for subject 12 is not five unrelated predictions. It is one stitched hazard curve assembled from the nodes visited by those interval records.

This point is important for interpretation. A high hazard at a later time need not mean that the subject had high baseline risk. It may mean that the subject entered a high-risk covariate state later in follow-up.

Inspecting a single subject

The fitted RHF object stores the subject identifiers in `ensemble.id`. These are the case identifiers, in the order used by the fitted object. This is useful because the original data set is in counting process form, so one subject can occupy several rows. The plotting method selects subjects by their identifier, not by the row number of the input data.

Here we select the first fitted subject, inspect the rows of the counting process data for that subject, and then display its fitted case specific hazard.

```
one.id <- fit.tdc$ensemble.id[1]

subj.rows <- d.tdc[d.tdc$id == one.id, ]
subj.rows

xcols <- setdiff(names(d.tdc), c("id", "start", "stop", "event"))
subj.rows[, c("start", "stop", "event", xcols[1:min(3, length(xcols))])]

plot(fit.tdc, idx = one.id, hazard.only = TRUE)
```

The call to `plot` uses `idx = one.id` because `idx` is a subject identifier. It is matched to the identifiers in `fit.tdc$ensemble.id`. The displayed curve is the out of bag case specific hazard estimate for that subject, evaluated on the working time grid `fit.tdc$time.interest`. The rows printed above show the observed start–stop history that contributes to the fitted hazard for this case.

Example 4. Comparing RHF and RSF on a common time grid

When covariates are time-static, RHF and RSF can be compared on the same data. The goal is not to show that one method must dominate the other. Rather, the comparison helps users understand that RHF is estimating a hazard, while RSF returns survival curves from terminal-node survival estimators.

```
data(pbc, package = "randomForestSRC")
pbc0 <- na.omit(pbc)
d <- convert.counting(Surv(days, status) ~ ., pbc0)
f <- "Surv(id, start, stop, event) ~ ."

fit.rhf <- rhf(f, d, ntree = 500, seed = -4)
time <- fit.rhf$time.interest
S.rhf <- exp(-fit.rhf$chf.oob)

fit.rsf <- randomForestSRC::rfsrc(Surv(days, status) ~ ., pbc0,
                                  ntime = time)
S.rsf <- fit.rsf$survival.oob

case.index <- 1:10
matplot(time, t(S.rhf[case.index, ]), type = "l", lty = 1,
        xlab = "Time", ylab = "Survival",
        main = "RHF OOB survival")

matplot(time, t(S.rsf[case.index, ]), type = "l", lty = 1,
        xlab = "Time", ylab = "Survival",
        main = "RSF OOB survival")
```

For time-static data, both methods can produce subject-specific survival curves. The conceptual difference becomes more important when predictors are time-dependent. RHF targets tge continuous-time hazard for the fitting and prediction process, and it uses the interval-level covariate state to route a subject through the forest over time.

Example 5. Time-varying prediction accuracy

RHF supports time-dependent AUC summaries based on out-of-bag prediction. These are useful when the question is not only whether a model predicts well overall, but also when during follow-up the model is most discriminating. The function `auct.rhf()` computes AUC over time using either cumulative hazard or hazard as the marker. Time-dependent ROC and AUC summaries are standard tools for censored event-time prediction [7–10].

```
data(peakVO2, package = "randomForestSRC")
d.vo2 <- convert.counting(Surv(ttodead, died) ~ ., peakVO2)
f.vo2 <- "Surv(id, start, stop, event) ~ ."

fit.n1  <- rhf(f.vo2, d.vo2, ntree = 500, nodesize = 1,
               seed = -5)
fit.n15 <- rhf(f.vo2, d.vo2, ntree = 500, nodesize = 15,
               seed = -5)

auc.n1.chf  <- auct.rhf(fit.n1)                # marker = "chf"
auc.n1.haz  <- auct.rhf(fit.n1,  marker = "haz")
auc.n15.chf <- auct.rhf(fit.n15)               # marker = "chf"
auc.n15.haz <- auct.rhf(fit.n15, marker = "haz")

print(auc.n1.chf)
print(auc.n15.chf)

oldpar <- par(mfrow = c(2, 2))
plot(auc.n1.chf,  main = "nodesize 1, CHF marker")
plot(auc.n1.haz,  main = "nodesize 1, hazard marker")
plot(auc.n15.chf, main = "nodesize 15, CHF marker")
plot(auc.n15.haz, main = "nodesize 15, hazard marker")
par(oldpar)
```

The CHF marker assesses whether accumulated risk separates subjects who experience the event from those who do not. The hazard marker assesses whether instantaneous risk separates them at a given time. Both can be useful. For chronic disease, CHF-based summaries may be more stable. For acute deterioration or monitoring applications, hazard-based summaries may be closer to the scientific question.

Example 6. Tuning tree size

The argument controls the number of terminal nodes requested for each tree. Very small trees can underfit. Very large trees can produce noisy hazard estimates, especially late in follow-up when the at-risk set is small. The tuning helper searches over tree sizes using out-of-bag empirical risk by default, or time-dependent iAUC if requested.

```
d.sim <- hazard.simulation(1)$dta
f.sim <- "Surv(id, start, stop, event) ~ ."

tune.risk <- rhf.tune(f.sim, d.sim,
                      ntree = 300,
                      perf = "risk",
                      lower = 2,
                      upper = 80,
                      seed = -6)

print(tune.risk$best.size)
plot(tune.risk)

fit.tuned <- tune.risk$forest
print(fit.tuned)
```

To tune by integrated time-dependent AUC, use `perf = "iAUC"`. This is slower because each candidate tree size requires a forest fit and an AUC calculation, but it can be appropriate when discrimination is the primary goal.

```
tune.iauc <- rhf.tune(f.sim, d.sim,
                      ntree = 300,
                      perf = "iAUC",
                      auct.args = list(marker = "chf"),
                      lower = 2,
                      upper = 80,
                      seed = -7)

plot(tune.iauc)
```

Example 7. VarPro-guided feature weighting and RHF variable priority

High-dimensional TDC data raise two related questions. Which variables drive risk, and which variables should be emphasized during forest growth? RHF uses the `xvar.wt` argument to control the probability that variables are selected as candidate split variables. Uniform weights are the default. Larger weights make a variable more likely to be considered for splitting.

The package includes a VarPro-based helper, `rhf.xvar.wt()`, for constructing such weights. VarPro is a model-independent, rule-based variable priority method [11–14]. In the RHF workflow, the helper converts the counting-process data to a standard survival snapshot, runs VarPro, and maps the resulting variable priority scores into split weights.

```
d.sim <- hazard.simulation(1)$dta
f.sim <- "Surv(id, start, stop, event) ~ ."

w <- rhf.xvar.wt(f.sim, d.sim, scale = 4, parallel = TRUE)
sort(w, decreasing = TRUE)[1:10]

fit.uniform <- rhf(f.sim, d.sim,
                   ntree = 500,
                   seed = -8)

fit.guided <- rhf(f.sim, d.sim,
                  ntree = 500,
                  xvar.wt = w,
                  seed = -8)

print(fit.uniform)
print(fit.guided)
```

The `scale` argument sharpens or flattens the VarPro-derived weights. The value `scale = 4` strongly emphasizes the highest-priority variables. A smaller value is more exploratory. In exploratory analyses, it is often useful to compare uniform and guided forests rather than replacing one with the other.

The RHF-VarPro idea

VarPro works at the level of decision rules extracted from the fitted forest. A rule ζ is a root to leaf path in a tree. Let $V(\zeta)$ denote the set of variables that appear in the splits along this path. For each $k \in V(\zeta)$, let $\mathcal{I}_k(\zeta)$ denote the set of values of variable k that satisfy all split constraints on k along the path. Thus $\mathcal{I}_k(\zeta)$ is the pathwise admissible set for variable k. For a continuous variable this is usually an interval, while for a categorical variable it is a subset of categories.

The rule defines the region

$$r(\zeta) = \bigcap_{k \in V(\zeta)} \{\mathbf{x} : x^{(k)} \in \mathcal{I}_k(\zeta)\}.$$

A variable can appear more than once along a path. In that case, $\mathcal{I}_k(\zeta)$ is the intersection of all restrictions involving variable k along that path.

To evaluate a variable set S, VarPro releases the rule by dropping the constraints involving variables in S. The released rule is

$$r\left(\zeta^S\right) = \bigcap_{k \in V(\zeta) \setminus S} \{\mathbf{x} : x^{(k)} \in \mathcal{I}_k(\zeta)\}.$$

If releasing S changes the local RHF risk summary, then the variables in S were important for that rule. The RHF implementation uses this idea to measure variable importance in a way that is tied to the hazard estimation problem, rather than to a generic prediction score.

For RHF, the natural units are pseudo-individuals (i, r). A scalar pseudo-outcome can be attached to each interval, for example the integrated RHF hazard over that interval,

$$W_{i,r} = \int_{S_{i,r}}^{T_{i,r}} \hat{\lambda}_i^{\mathrm{RHF}}(t)\, dt,$$

or over the extended interval used by the stitching rule. For a rule ζ and release set S, define a local rule average and a released, near-miss average.

$$\hat{\theta}(\zeta) = \frac{\sum_{i,r} \mathbf{1}_{\{\mathbf{Z}_{i,r} \in r(\zeta)\}} W_{i,r}}{\sum_{i,r} \mathbf{1}_{\{\mathbf{Z}_{i,r} \in r(\zeta)\}}},$$

$$\hat{\theta}_{\mathrm{miss}}(\zeta; S) = \frac{\sum_{i,r} \mathbf{1}_{\{\mathbf{Z}_{i,r} \in r(\zeta^S) \setminus r(\zeta)\}} W_{i,r}}{\sum_{i,r} \mathbf{1}_{\{\mathbf{Z}_{i,r} \in r(\zeta^S) \setminus r(\zeta)\}}}.$$

A rule-level RHF-VarPro contribution is

$$\Delta_S(\zeta) = \left|\hat{\theta}(\zeta) - \hat{\theta}_{\mathrm{miss}}(\zeta; S)\right|.$$

Aggregating $\Delta_S(\zeta)$ over many rules gives a variable-priority score. The same idea can be localized in time by restricting the sums to intervals active at a grid point t_k and using the pointwise pseudo-outcome

$$W_{i,r}(t_k) = \log\{\hat{\lambda}_i^{\mathrm{oob}}(t_k)\}.$$

This produces a curve $t_k \mapsto \mathrm{VIMP}(S; t_k)$ that describes when a variable or group of variables is driving dynamic risk.

For an applied user, the practical interpretation is that RHF-VarPro compares nearby rule-defined neighborhoods. It is not a permutation based method and it does not generate synthetic covariates. Instead, it uses observed data and calculates how much the local hazard-based risk summary changes when the rule stops conditioning on the variable being assessed.

Predicting on new data

A fitted RHF forest can be applied to new counting process data with `predict.rhf()`. The new data should contain the same predictor columns used for training and the same start–stop representation. When response columns are present, they are used to align the counting process intervals. For prospective prediction, the event column will usually be zero over the observed intervals.

The train and test split should be made at the subject level. This is important because a single subject can contribute several counting process rows. Splitting rows directly would allow part of a subject's history to appear in the training data and another part to appear in the test data, which would create leakage.

```
set.seed(17)

id.vec <- unique(d.tdc$id)

train.id <- sample(id.vec, size = floor(length(id.vec) / 2))
test.id  <- setdiff(id.vec, train.id)

train <- d.tdc[d.tdc$id %in% train.id, ]
test  <- d.tdc[d.tdc$id %in% test.id, ]

fit.train <- rhf(f.tdc, train,
                 ntree = 500,
                 treesize = 30,
                 seed = -9)

pred <- predict(fit.train, newdata = test)

str(pred$hazard.test)
str(pred$chf.test)
```

The objects `hazard.test` and `chf.test` contain case specific test set estimates on the working time grid used by the fitted forest. As with the training ensemble, the test set hazard and cumulative hazard are computed as RHF ensemble quantities for each subject.

In production or external validation settings, the most important rule is to represent new records in the same way as training records. The model cannot repair a mismatch between the training definition of an interval and the test or deployment definition. If the training rows represent one hour ICU windows, then prospective rows should represent comparable one hour windows or another well defined interval construction chosen before validation.

Common choices and practical guidance

Argument	Practical role
`ntree`	Number of trees. Increase for smoother ensemble estimates and more stable OOB summaries.
`nsplit`	Number of random split points per variable. Larger values search more thoroughly but cost more. Use `0` for exhaustive split search.
`treesize`	Target tree size. Tune this when hazard curves look too rough or too flat.
`nodesize`	Minimum terminal-node size. Larger values regularize terminal-node hazards.
`ntime`	Number of requested time-grid points, a vector of requested times, or `0` / `NULL` for all event times.
`min.events.per.gap`	Event-balanced grid control that avoids overly sparse time intervals.
`xvar.wt`	Variable selection weights for candidate split variables. Use `rhf.xvar.wt()` for VarPro-guided weights.

A good first analysis is usually simple. Use the default `nodesize`, a moderate `ntree`, and the default time grid. Then inspect individual hazard curves, OOB risk, and time-varying AUC. If the curves are too noisy, increase `nodesize` or reduce `treesize`. If the model appears too coarse, increase `treesize` or tune it. If there are many irrelevant variables, compare a uniform forest with a VarPro-guided forest.

Interpreting the RHF estimand

RHF does not split on time itself. This may seem surprising at first. Time enters through at-risk time, event counts, terminal-node Nelson–Aalen estimators, and stitched subject-level hazards. A split on time itself would mix the question **which covariate state is risky?** with the question **when are events common?** RHF keeps those roles separate. Covariates define neighborhoods, and time is handled by the node-level hazard estimators.

This distinction is also why the primary RHF outputs are hazard and cumulative hazard. For time-static data, such as the PBC and peakVO2 examples above, cumulative hazard can be transformed into the usual survival curve. For genuine TDC data, interpretation should remain attached to the hazard process and to the covariate history used to construct it.

Function map

The main RHF routines are summarized in the following table.

Common RHF functions and fitted object components.

Function or object	Purpose	Typical use
`convert.counting()`	Converts a standard time static survival data set into counting process form.	Use this when the data are given as one row per subject, but the analyst wants to fit RHF using start and stop times.
`rhf()`	Fits a random hazard forest.	Use this as the main model fitting function for time dependent covariate data or for time static data after conversion to counting process form.
`hazard.oob` and `chf.oob`	Out of bag case specific hazard and cumulative hazard estimates.	Use these for training sample inference. Since bootstrap sampling is always used in RHF, out of bag summaries are the preferred inferential quantities.
`predict.rhf()`	Applies a fitted RHF forest to new data.	Use this for external validation, test set prediction, or prospective prediction on new counting process records.

Function or object	Purpose	Typical use
`hazard.test` and `chf.test`	Test set case specific hazard and cumulative hazard estimates returned by `predict.rhf()`.	Use these to summarize fitted hazards and cumulative hazards for new subjects supplied through `newdata`. These are the prediction analogues of `hazard.oob` and `chf.oob`.
`plot.rhf()`	Displays fitted case specific hazard and cumulative hazard estimates.	Use this to inspect subject level fitted values on the working time grid.
`auct.rhf()`	Computes time dependent AUC summaries.	Use this to assess discrimination over time, preferably using out of bag quantities for training sample assessment.
`rhf.tune()`	Tunes RHF fitting parameters.	Use this to search over model settings before fitting a final forest.
`rhf.tune.treesize()`	Tunes terminal node size.	Use this when the main tuning target is the size of terminal nodes.
`rhf.smoothed.hazard()`	Produces smoothed hazard estimates.	Use this when a smoother display or downstream summary of the estimated hazard is needed.
`rhf.xvar.wt()`	Computes VarPro guided split weights.	Use this to adapt splitting probabilities using variable importance information derived from released rules.

Cite this vignette as

H. Ishwaran, M. Lu, and U. B. Kogalur. 2026. “randomForestRHF: getting started with randomForestRHF vignette.” http://www.randomforestrhf.org/articles/getstarted.html (http://www.randomforestrhf.org/articles/getstarted.html).

```
@misc{LuGettingStarted,
    author = "Hemant Ishwaran and Min Lu and Udaya B. Kogalur",
    title = {{randomForestRHF}: getting started with {randomForestRHF} vignette},
    year = {2026},
    url = {http://www.randomforestrhf.org/articles/getstarted.html},
    howpublished = "\url{http://www.randomforestrhf.org/articles/getstarted.html}",
    note = "[accessed date]"
}
```


1. Breiman L. Random forests. Machine Learning. 2001;45:5–32.
2. Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. Annals of Applied Statistics. 2008;2:841–60.
3. Aalen OO. Nonparametric inference for a family of counting processes. Annals of Statistics. 1978;6:701–26.
4. Andersen PK, Borgan O, Gill RD, Keiding N. Statistical models based on counting processes. Springer Science & Business Media; 1993.
5. Lee DK, Chen N, Ishwaran H. Boosted nonparametric hazards with time-dependent covariates. Annals of Statistics. 2021;49:2101–28.
6. Gill RD, Johansen S. A product integral with applications to survival analysis. The Annals of Statistics. 1990;18:909–30.
7. Heagerty PJ, Lumley T, Pepe MS. Time-dependent ROC curves for censored survival data and a diagnostic marker. Biometrics. 2000;56:337–44.
8. Heagerty PJ, Zheng Y. Survival model predictive accuracy and ROC curves. Biometrics. 2005;61:92–105.
9. Uno H, Cai T, Pencina MJ, D'Agostino RB, Wei L-J. On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. Statistics in Medicine. 2011;30:1105–17.
10. Uno H, Tian L, Cai T, Kohane IS, Wei L. A unified inference procedure for a class of measures to assess improvement in risk prediction systems with survival data. Statistics in Medicine. 2013;32:2430–42.
11. Lu M, Ishwaran H. Model-independent variable selection via the rule-based variable priority. arXiv preprint arXiv:240909003. 2024.
12. Ishwaran H. Multivariate statistics: Classical foundations and modern machine learning. 1st edition. Boca Raton, FL: Chapman; Hall/CRC; 2025.
13. Lu M, Ishwaran H. Individual variable priority: A model-independent local gradient method for variable importance. Artificial Intelligence Review. 2025;58:407.
14. Zhou L, Lu M, Ishwaran H. Variable priority for unsupervised variable selection (https://doi.org/10.1016/j.patcog.2025.112727). Pattern Recognition. 2026;172:112727.