Package ‘hdnom’

June 23, 2019

Type Package
Title Benchmarking and Visualization Toolkit for Penalized Cox Models
Version 6.0.0
Maintainer Nan Xiao <me@nanx.me>
Description Creates nomogram visualizations for penalized Cox regression models, with the support of reproducible survival model building, validation, calibration, and comparison for high-dimensional data.
License GPL-3 | file LICENSE
LazyData TRUE
VignetteBuilder knitr
BugReports https://github.com/nanxstats/hdnom/issues
Depends R (>= 3.5.0)
Imports survival, glmnet, penalized, ncvreg, foreach, survAUC, ggplot2, gridExtra
Suggests knitr, rmarkdown, doParallel
Encoding UTF-8
RoxygenNote 6.1.1
NeedsCompilation no
Author Nan Xiao [aut, cre] (<https://orcid.org/0000-0002-0250-5673>), Qing-Song Xu [aut], Miao-Zhu Li [aut], Frank Harrell [ctb]
Repository CRAN
Date/Publication 2019-06-23 05:10:03 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_nomogram</td>
<td>3</td>
</tr>
<tr>
<td>calibrate</td>
<td>4</td>
</tr>
<tr>
<td>calibrate_external</td>
<td>6</td>
</tr>
<tr>
<td>compare_by_calibrate</td>
<td>9</td>
</tr>
<tr>
<td>compare_by_validate</td>
<td>10</td>
</tr>
<tr>
<td>fit_aenet</td>
<td>11</td>
</tr>
<tr>
<td>fit_alasso</td>
<td>12</td>
</tr>
<tr>
<td>fit_enet</td>
<td>13</td>
</tr>
<tr>
<td>fit_flasso</td>
<td>14</td>
</tr>
<tr>
<td>fit_lasso</td>
<td>16</td>
</tr>
<tr>
<td>fit_mcp</td>
<td>17</td>
</tr>
<tr>
<td>fit_mnet</td>
<td>18</td>
</tr>
<tr>
<td>fit_snet</td>
<td>19</td>
</tr>
<tr>
<td>glmnet_basesurv</td>
<td>20</td>
</tr>
<tr>
<td>glmnet_survcurve</td>
<td>21</td>
</tr>
<tr>
<td>infer_variable_type</td>
<td>22</td>
</tr>
<tr>
<td>kmplot</td>
<td>23</td>
</tr>
<tr>
<td>logrank_test</td>
<td>24</td>
</tr>
<tr>
<td>ncvreg_basesurv</td>
<td>25</td>
</tr>
<tr>
<td>ncvreg_survcurve</td>
<td>26</td>
</tr>
<tr>
<td>penalized_basesurv</td>
<td>27</td>
</tr>
<tr>
<td>penalized_survcurve</td>
<td>27</td>
</tr>
<tr>
<td>plot.hdnom.calibrate</td>
<td>28</td>
</tr>
<tr>
<td>plot.hdnom.calibrate.external</td>
<td>28</td>
</tr>
<tr>
<td>plot.hdnom.compare.calibrate</td>
<td>29</td>
</tr>
<tr>
<td>plot.hdnom.compare.validate</td>
<td>30</td>
</tr>
<tr>
<td>plot.hdnom.nomogram</td>
<td>30</td>
</tr>
<tr>
<td>plot.hdnom.validate</td>
<td>31</td>
</tr>
<tr>
<td>plot.hdnom.validate.external</td>
<td>31</td>
</tr>
<tr>
<td>predict.hdnom.model</td>
<td>32</td>
</tr>
<tr>
<td>print.hdnom.calibrate</td>
<td>33</td>
</tr>
<tr>
<td>print.hdnom.calibrate.external</td>
<td>33</td>
</tr>
<tr>
<td>print.hdnom.compare.calibrate</td>
<td>34</td>
</tr>
<tr>
<td>print.hdnom.compare.validate</td>
<td>34</td>
</tr>
<tr>
<td>print.hdnom.model</td>
<td>35</td>
</tr>
<tr>
<td>print.hdnom.nomogram</td>
<td>35</td>
</tr>
<tr>
<td>print.hdnom.validate</td>
<td>36</td>
</tr>
<tr>
<td>print.hdnom.validate.external</td>
<td>36</td>
</tr>
<tr>
<td>smart</td>
<td>37</td>
</tr>
<tr>
<td>smarto</td>
<td>38</td>
</tr>
<tr>
<td>summary.hdnom.calibrate</td>
<td>39</td>
</tr>
<tr>
<td>summary.hdnom.calibrate.external</td>
<td>40</td>
</tr>
<tr>
<td>summary.hdnom.compare.calibrate</td>
<td>40</td>
</tr>
<tr>
<td>summary.hdnom.compare.validate</td>
<td>41</td>
</tr>
<tr>
<td>summary.hdnom.validate</td>
<td>41</td>
</tr>
</tbody>
</table>
as_nomogram

Construct nomogram objects for high-dimensional Cox models

Usage

as_nomogram(object, x, time, event, pred.at = NULL, fun.at = NULL, funlabel = NULL)

Arguments

object Model object fitted by `hdnom::fit_*()` functions.
x Matrix of training data used for fitting the model.
time Survival time. Must be of the same length with the number of rows as x.
event Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as x.
pred.at Time point at which to plot nomogram prediction axis.
fun.at Function values to label on axis.
funlabel Label for fun axis.

Note

The nomogram visualizes the model under the automatically selected "optimal" hyperparameters (e.g. lambda, alpha, gamma).

Examples

data(smart)
x <- as.matrix(smart[, -c(1,2)])
time <- smart$TEVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

fit <- fit_lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

nom <- as_nomogram(
  fit, x, time, event, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability"
)
Calibrate high-dimensional Cox models

Description

Calibrate high-dimensional Cox models

Usage

```r
calibrate(x, time, event, model.type = c("lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", "snet"), alpha, lambda, pen.factor = NULL, gamma, lambda1, lambda2, method = c("fitting", "bootstrap", "cv", "repeated.cv"), boot.times = NULL, nfolds = NULL, rep.times = NULL, pred.at, ngroup = 5, seed = 1001, trace = TRUE)
```

Arguments

- `x`: Matrix of training data used for fitting the model; on which to run the calibration.
- `time`: Survival time. Must be of the same length with the number of rows as `x`.
- `event`: Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as `x`.
- `model.type`: Model type to calibrate. Could be one of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
- `alpha`: Value of the elastic-net mixing parameter alpha for enet, aenet, mnet, and snet models. For lasso, alasso, mcp, and scad models, please set alpha = 1. alpha=1: lasso (l1) penalty; alpha=0: ridge (l2) penalty. Note that for mnet and snet models, alpha can be set to very close to 0 but not 0 exactly.
- `lambda`: Value of the penalty parameter lambda to use in the model fits on the resampled data. From the Cox model you have built.
- `pen.factor`: Penalty factors to apply to each coefficient. From the built adaptive lasso or adaptive elastic-net model.
- `gamma`: Value of the model parameter gamma for MCP/SCAD/Mnet/Snet models.
- `lambda1`: Value of the penalty parameter lambda1 for fused lasso model.
- `lambda2`: Value of the penalty parameter lambda2 for fused lasso model.
- `method`: Calibration method. Options including "fitting", "bootstrap", "cv", and "repeated.cv".
- `boot.times`: Number of repetitions for bootstrap.
- `nfolds`: Number of folds for cross-validation and repeated cross-validation.
rep.times  Number of repeated times for repeated cross-validation.
pred.at    Time point at which calibration should take place.
group      Number of groups to be formed for calibration.
seed       A random seed for resampling.
trace      Logical. Output the calibration progress or not. Default is TRUE.

Examples

data("smart")
x <- as.matrix_smart[, -c(1, 2)]
time <- smart$EVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

# Fit Cox model with lasso penalty
fit <- lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

# Model calibration by fitting the original data directly
fit.cal <- calibrate(
  x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "fitting",
  pred.at = 365 * 9, ngroup = 5,
  seed = 1010
)

# Model calibration by 5-fold cross-validation
fit.cal.cv <- calibrate(
  x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "cv", nfolds = 5,
  pred.at = 365 * 9, ngroup = 5,
  seed = 1010
)

print(fit.cal)
summary(fit.cal)
plot(fit.cal)

print(fit.cal.cv)
summary(fit.cal.cv)
plot(fit.cal.cv)

# # Test fused lasso, SCAD, and Mnet models
# data(smart)
# x = as.matrix(smart[, -c(1, 2)])[1:500, ]
# time = smart$EVENT[1:500]
# event = smart$EVENT[1:500]
# y = survival::Surv(time, event)
# set.seed(1010)
# cal.fitting = calibrate(
#   x, time, event, model.type = "flasso",
#   lambda1 = 5, lambda2 = 2,
#   method = "fitting",
#   pred.at = 365 * 9, ngroup = 5,
#   seed = 1010)
#
# cal.boot = calibrate(
#   x, time, event, model.type = "scad",
#   gamma = 3.7, alpha = 1, lambda = 0.03,
#   method = "bootstrap", boot.times = 10,
#   pred.at = 365 * 9, ngroup = 5,
#   seed = 1010)
#
# cal.cv = calibrate(
#   x, time, event, model.type = "mnet",
#   gamma = 3, alpha = 0.3, lambda = 0.03,
#   method = "cv", nfolds = 5,
#   pred.at = 365 * 9, ngroup = 5,
#   seed = 1010)
#
# cal.repcv = calibrate(
#   x, time, event, model.type = "flasso",
#   lambda1 = 5, lambda2 = 2,
#   method = "repeated.cv", nfolds = 5, rep.times = 3,
#   pred.at = 365 * 9, ngroup = 5,
#   seed = 1010)
#
# print(cal.fitting)
# summary(cal.fitting)
# plot(cal.fitting)
#
# print(cal.boot)
# summary(cal.boot)
# plot(cal.boot)
#
# print(cal.cv)
# summary(cal.cv)
# plot(cal.cv)
#
# print(cal.repcv)
# summary(cal.repcv)
# plot(cal.repcv)

---
calibrate_external

Externally calibrate high-dimensional Cox models
**Description**

Externally calibrate high-dimensional Cox models

**Usage**

```
calibrate_external(object, x, time, event, x_new, time_new, event_new, pred.at, ngroup = 5)
```

**Arguments**

- `object`: Model object fitted by `hdnom::fit_*()`.
- `x`: Matrix of training data used for fitting the model.
- `time`: Survival time of the training data. Must be of the same length with the number of rows as `x`.
- `event`: Status indicator of the training data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as `x`.
- `x_new`: Matrix of predictors for the external validation data.
- `time_new`: Survival time of the external validation data. Must be of the same length with the number of rows as `x_new`.
- `event_new`: Status indicator of the external validation data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as `x_new`.
- `pred.at`: Time point at which external calibration should take place.
- `ngroup`: Number of groups to be formed for external calibration.

**Examples**

```r
library("survival")

# Load imputed SMART data
data(smart)
# Use the first 1000 samples as training data
# (the data used for internal validation)
x <- as.matrix(smart[, -c(1, 2)])[1:1000, ]
time <- smart$EVENT[1:1000]
event <- smart$EVENT[1:1000]

# Take the next 1000 samples as external calibration data
# In practice, usually use data collected in other studies
x_new <- as.matrix(smart[, -c(1, 2)][1001:2000, ]
time_new <- smart$EVENT[1001:2000]
event_new <- smart$EVENT[1001:2000]

# Fit Cox model with lasso penalty
fit <- fit.lasso(
  x, Surv(time, event),
  nfold = 5, rule = "lambda.1se", seed = 11)
```
# External calibration

cal.ext <- calibrate_external(
  fit, x, time, event,
  x_new, time_new, event_new,
  pred.at = 365 * 5, ngroup = 5
)

print(cal.ext)
summary(cal.ext)
plot(cal.ext, xlab = c(0.6, 1), ylab = c(0.6, 1))
# # Test fused lasso, MCP, and Snet models
# data(smart)
# # Use first 500 samples as training data
# (the data used for internal validation)
# x <- as.matrix(smart[, -c(1, 2)])[1:500, ]
# time <- smart$EVENT[1:500]
# event <- smart$EVENT[1:500]
# # # Take 1000 samples as external validation data.
# # In practice, usually use data collected in other studies.
# x_new <- as.matrix(smart[, -c(1, 2)])[1001:2000, ]
# time_new <- smart$EVENT[1001:2000]
# event_new <- smart$EVENT[1001:2000]
# # # flassofit <- fit_flas(x, survival::Surv(time, event), nfolds = 5, seed = 11)
# # scadfit <- fit_mcp(x, survival::Surv(time, event), nfolds = 5, seed = 11)
# # mnetfit <- fit_snet(x, survival::Surv(time, event), nfolds = 5, seed = 11)
# # cal.ext1 <- calibrate_external(
# # flassofit, x, time, event,
# # x_new, time_new, event_new,
# # pred.at = 365 * 5, ngroup = 5)
# # cal.ext2 <- calibrate_external(
# # scadfit, x, time, event,
# # x_new, time_new, event_new,
# # pred.at = 365 * 5, ngroup = 5)
# # cal.ext3 <- calibrate_external(
# # mnetfit, x, time, event,
# # x_new, time_new, event_new,
# # pred.at = 365 * 5, ngroup = 5)
# # print(cal.ext1)
# # summary(cal.ext1)
# # plot(cal.ext1)
# # print(cal.ext2)
# # summary(cal.ext2)
# # plot(cal.ext2)
# # print(cal.ext3)
# # summary(cal.ext3)
compare_by_calibrate

# plot(cal.ext3)

---

**compare_by_calibrate**  
*Compare high-dimensional Cox models by model calibration*

**Description**

Compare high-dimensional Cox models by model calibration

**Usage**

```r
compare_by_calibrate(x, time, event, model.type = c("lasso", "alasso",  
                    "flasso", "enet", "aenet", "mcp", "mnet", "scad", "snet"),  
                    method = c("fitting", "bootstrap", "cv", "repeated.cv"),  
                    boot.times = NULL, nfolds = NULL, rep.times = NULL, pred.at,  
                    ngroup = 5, seed = 1001, trace = TRUE)
```

**Arguments**

- **x**: Matrix of training data used for fitting the model; on which to run the calibration.
- **time**: Survival time. Must be of the same length with the number of rows as `x`.
- **event**: Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as `x`.
- **model.type**: Model types to compare. Could be at least two of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
- **method**: Calibration method. Could be "bootstrap", "cv", or "repeated.cv".
- **boot.times**: Number of repetitions for bootstrap.
- **nfolds**: Number of folds for cross-validation and repeated cross-validation.
- **rep.times**: Number of repeated times for repeated cross-validation.
- **pred.at**: Time point at which calibration should take place.
- **ngroup**: Number of groups to be formed for calibration.
- **seed**: A random seed for cross-validation fold division.
- **trace**: Logical. Output the calibration progress or not. Default is TRUE.

**Examples**

```r
data(smart)
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$EVENT
event <- smart$EVENT

# Compare lasso and adaptive lasso by 5-fold cross-validation
cmp.cal.cv <- compare_by_calibrate(
  x, time, event,  
  model.type = c("lasso", "alasso"),
```

compare_by_validate

Description

Compare high-dimensional Cox models by model validation

Usage

compare_by_validate(x, time, event, model.type = c("lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", "snet"), method = c("bootstrap", "cv", "repeated.cv"), boot.times = NULL, nfolds = NULL, rep.times = NULL, tauc.type = c("CD", "SZ", "UNO"), tauc.time, seed = 1001, trace = TRUE)

Arguments

x Matrix of training data used for fitting the model; on which to run the validation.
time Survival time. Must be of the same length with the number of rows as x.
event Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as x.
model.type Model types to compare. Could be at least two of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
method Validation method. Could be "bootstrap", "cv", or "repeated.cv".
boot.times Number of repetitions for bootstrap.
nfolds Number of folds for cross-validation and repeated cross-validation.
rep.times Number of repeated times for repeated cross-validation.
tauc.type Type of time-dependent AUC. Including "CD" proposed by Chambless and Diao (2006), "SZ" proposed by Song and Zhou (2008), "UNO" proposed by Uno et al. (2007).
tauc.time Numeric vector. Time points at which to evaluate the time-dependent AUC.
seed A random seed for cross-validation fold division.
trace Logical. Output the validation progress or not. Default is TRUE.
References


Examples

data(smart)
x <- as.matrix(smart[, -c(1, 2)])[1:1000, ]
time <- smart$EVENT[1:1000]
event <- smart$EVENT[1:1000]

# Compare lasso and adaptive lasso by 5-fold cross-validation
cmp.val.cv <- compare_by_validate(
  x, time, event,
  model.type = c("lasso", "alasso"),
  method = "cv", nfolds = 5, tauc.type = "UNO",
  tauc.time = seq(0.25, 2, 0.25) * 365, seed = 1001
)

print(cmp.val.cv)
summary(cmp.val.cv)
plot(cmp.val.cv)
plot(cmp.val.cv, interval = TRUE)

fit_aenet

**Model selection for high-dimensional Cox models with adaptive elastic-net penalty**

Description

Automatic model selection for high-dimensional Cox models with adaptive elastic-net penalty, evaluated by penalized partial-likelihood.

Usage

```
fit_aenet(x, y, nfolds = 5L, alphas = seq(0.05, 0.95, 0.05),
  rule = c("lambda.min", "lambda.1se"), seed = c(1001, 1002),
  parallel = FALSE)
```
**Arguments**

- **x**: Data matrix.
- **y**: Response matrix made with `Surv`.
- **nfolds**: Fold numbers of cross-validation.
- **alphas**: Alphas to tune in `cv.glmnet`.
- **rule**: Model selection criterion, "lambda.min" or "lambda.1se". See `cv.glmnet` for details.
- **seed**: Two random seeds for cross-validation fold division in two estimation steps.
- **parallel**: Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the `doParallel` package and run `registerDoParallel()` with the number of CPU cores before calling this function.

**Examples**

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$TIMEVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

# To enable parallel parameter tuning, first run:
# library("doParallel")
# registerDoParallel(detectCores())
# then set fit_aenet(..., parallel = TRUE).

fit <- fit_aenet(
  x, y,
  nfolds = 3, alphas = c(0.3, 0.7),
  rule = "lambda.1se", seed = c(5, 7)
)

nom <- as_nomogram(
  fit, x, time, event,
  pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability"
)

plot(nom)
```

---

**fit_lasso**  
*Model selection for high-dimensional Cox models with adaptive lasso penalty*

**Description**

Automatic model selection for high-dimensional Cox models with adaptive lasso penalty, evaluated by penalized partial-likelihood.
Usage

```r
fit_alasso(x, y, nfolds = 5L, rule = c("lambda.min", "lambda.1se"),
    seed = c(1001, 1002))
```

Arguments

- **x**: Data matrix.
- **y**: Response matrix made by `Surv`.
- **nfolds**: Fold numbers of cross-validation.
- **rule**: Model selection criterion, "lambda.min" or "lambda.1se". See `cv.glmnet` for details.
- **seed**: Two random seeds for cross-validation fold division in two estimation steps.

Examples

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$EVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

fit <- fit_alasso(x, y, nfolds = 3, rule = "lambda.1se", seed = c(7, 11))

nom <- as_nomogram(
    fit, x, time, event,
    pred.at = 365 * 2,
    funlabel = "2-Year Overall Survival Probability"
)

plot(nom)
```

---

**fit_enet**

*Model selection for high-dimensional Cox models with elastic-net penalty*

Description

Automatic model selection for high-dimensional Cox models with elastic-net penalty, evaluated by penalized partial-likelihood.

Usage

```r
fit_enet(x, y, nfolds = 5L, alphas = seq(0.05, 0.95, 0.05),
    rule = c("lambda.min", "lambda.1se"), seed = 1001,
    parallel = FALSE)
```
Arguments

\begin{itemize}
\item \textbf{x} \hspace{1cm} Data matrix.
\item \textbf{y} \hspace{1cm} Response matrix made by \texttt{Surv}.
\item \textbf{nlogs} \hspace{1cm} Fold numbers of cross-validation.
\item \textbf{alphas} \hspace{1cm} Alphas to tune in \texttt{cv.glmnet}.
\item \textbf{rule} \hspace{1cm} Model selection criterion, "\texttt{lambda.min}" or "\texttt{lambda.1se}". See \texttt{cv.glmnet} for details.
\item \textbf{seed} \hspace{1cm} A random seed for cross-validation fold division.
\item \textbf{parallel} \hspace{1cm} Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the \texttt{doparallel} package and run \texttt{registerDoParallel()} with the number of CPU cores before calling this function.
\end{itemize}

Examples

\begin{verbatim}
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$TEVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

# To enable parallel parameter tuning, first run:
# library("doParallel")
# registerDoParallel(detectCores())
# then set fit_enet(..., parallel = TRUE).

fit <- fit_enet(
  x, y,
  nfolds = 3, alphas = c(0.3, 0.7),
  rule = "lambda.1se", seed = 11
)

nom <- as_nomogram(
  fit, x, time, event,
  pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability"
)

plot(nom)
\end{verbatim}

---

\textbf{Model selection for high-dimensional Cox models with fused lasso penalty}

Description

Automatic model selection for high-dimensional Cox models with fused lasso penalty, evaluated by cross-validated likelihood.
fit_flasso

Usage

```
fit_flasso(x, y, nfolds = 5L, lambda1 = c(0.001, 0.05, 0.5, 1, 5),
lambda2 = c(0.001, 0.01, 0.5), maxiter = 25, epsilon = 0.001,
seed = 1001, trace = FALSE, parallel = FALSE, ...)
```

Arguments

- `x`: Data matrix.
- `y`: Response matrix made by `Surv`.
- `nfolds`: Fold numbers of cross-validation.
- `lambda1`: Vector of lambda1 candidates. Default is `c(0.001, 0.05, 0.5, 1, 5)`.
- `lambda2`: Vector of lambda2 candidates. Default is `c(0.001, 0.01, 0.5)`.
- `maxiter`: The maximum number of iterations allowed. Default is 25.
- `epsilon`: The convergence criterion. Default is `1e-3`.
- `seed`: A random seed for cross-validation fold division.
- `trace`: Output the cross-validation parameter tuning progress or not. Default is `FALSE`.
- `parallel`: Logical. Enable parallel parameter tuning or not, default is `FALSE`. To enable parallel tuning, load the `doparallel` package and run `registerDoParalell()` with the number of CPU cores before calling this function.

Note

The cross-validation procedure used in this function is the approximated cross-validation provided by the penalized package. Be careful dealing with the results since they might be more optimistic than a traditional CV procedure. This cross-validation method is more suitable for datasets with larger number of observations, and a higher number of cross-validation folds.

Examples

```
data("smart")
x <- as.matrix(smart[, -c(1, 2)][1:120, ]
time <- smart$EVENT[1:120]
event <- smart$EVENT[1:120]
y <- survival::Surv(time, event)

fit <- fit_flasso(
x, y,
lambda1 = c(1, 10), lambda2 = c(0.01),
nfolds = 3, seed = 11
)

nom <- as_nomogram(
fit, x, time, event,
pred.at = 365 * 2,
funlabel = "2-Year Overall Survival Probability"
)
```
fit_lasso

Model selection for high-dimensional Cox models with lasso penalty

Description

Automatic model selection for high-dimensional Cox models with lasso penalty, evaluated by penalized partial-likelihood.

Usage

```r
fit_lasso(x, y, nfolds = 5L, rule = c("lambda.min", "lambda.1se"),
seed = 1001)
```

Arguments

- `x` Data matrix.
- `y` Response matrix made by `Surv`.
- `nfolds` Fold numbers of cross-validation.
- `rule` Model selection criterion, "lambda.min" or "lambda.1se". See `cv.glmnet` for details.
- `seed` A random seed for cross-validation fold division.

Examples

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$TEVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

fit <- fit_lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

nom <- as_nomogram(
  fit, x, time, event,
  pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability"
)

plot(nom)
```
**Description**

Automatic model selection for high-dimensional Cox models with MCP penalty, evaluated by penalized partial-likelihood.

**Usage**

```r
fit_mcp(x, y, nfolds = 5L, gammas = c(1.01, 1.7, 3, 100),
        eps = 1e-04, max.iter = 10000L, seed = 1001L, trace = FALSE,
        parallel = FALSE)
```

**Arguments**

- `x`: Data matrix.
- `y`: Response matrix made by `Surv`.
- `nfolds`: Fold numbers of cross-validation.
- `gammas`: Gammas to tune in `cv.ncvsurv`.
- `eps`: Convergence threshold.
- `max.iter`: Maximum number of iterations.
- `seed`: A random seed for cross-validation fold division.
- `trace`: Output the cross-validation parameter tuning progress or not. Default is `FALSE`.
- `parallel`: Logical. Enable parallel parameter tuning or not, default is `FALSE`. To enable parallel tuning, load the `doParallel` package and run `registerDoParallel()` with the number of CPU cores before calling this function.

**Examples**

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)][1:150, ]
time <- smart$EVENT[1:150]
event <- smart$EVENT[1:150]
y <- survival::Surv(time, event)

fit <- fit_mcp(x, y, nfolds = 3, gammas = c(2.1, 3), seed = 1001)

nom <- as_nomogram(
    fit, x, time, event,
    pred.at = 365 * 2,
    funlabel = "2-Year Overall Survival Probability"
)

plot(nom)
```
Model selection for high-dimensional Cox models with Mnet penalty

**Description**

Automatic model selection for high-dimensional Cox models with Mnet penalty, evaluated by penalized partial-likelihood.

**Usage**

```r
fit_mnet(x, y, nfolds = 5L, gammas = c(1.01, 1.7, 3, 100),
         alphas = seq(0.05, 0.95, 0.05), eps = 1e-04, max.iter = 10000L,
         seed = 1001, trace = FALSE, parallel = FALSE)
```

**Arguments**

- **x**: Data matrix.
- **y**: Response matrix made by `Surv`.
- **nfolds**: Fold numbers of cross-validation.
- **gammas**: Gammas to tune in `cv.ncvsurv`.
- **alphas**: Alphas to tune in `cv.ncvsurv`.
- **eps**: Convergence threshold.
- **max.iter**: Maximum number of iterations.
- **seed**: A random seed for cross-validation fold division.
- **trace**: Output the cross-validation parameter tuning progress or not. Default is `FALSE`.
- **parallel**: Logical. Enable parallel parameter tuning or not, default is `FALSE`. To enable parallel tuning, load the `doParallel` package and run `registerDoParallel()` with the number of CPU cores before calling this function.

**Examples**

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)))[1:120, ]
time <- smart$EVENT[1:120]
event <- smart$EVENT[1:120]
y <- survival::Surv(time, event)

fit <- fit_mnet(
  x, y,
  nfolds = 3,
  gammas = 3, alphas = c(0.3, 0.8),
  max.iter = 15000, seed = 1010
)

nom <- as_nomogram(
  fit, x, time, event,
)```
fit_scad

Model selection for high-dimensional Cox models with SCAD penalty

Description

Automatic model selection for high-dimensional Cox models with SCAD penalty, evaluated by penalized partial-likelihood.

Usage

fit_scad(x, y, nfolds = 5, gammas = c(2.01, 2.3, 3.7, 200), eps = 1e-04, max.iter = 10000L, seed = 1001, trace = FALSE, parallel = FALSE)

Arguments

x Data matrix.
y Response matrix made by Surv.
nfolds Fold numbers of cross-validation.
gammas Gammas to tune in cv.ncvsurv.
eps Convergence threshold.
max.iter Maximum number of iterations.
seed A random seed for cross-validation fold division.
trace Output the cross-validation parameter tuning progress or not. Default is FALSE.
parallel Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the doParallel package and run registerDoParallel() with the number of CPU cores before calling this function.

Examples

data("smart")
x <- as.matrix(smart[, -c(1, 2)][1:120, ]
time <- smart$EVENT[1:120]
event <- smart$EVENT[1:120]
y <- survival::Surv(time, event)

fit <- fit_scad(
x, y,
nfolds = 3, gammas = c(3.7, 5),
max.iter = 15000, seed = 1010
)
fit_snet

Model selection for high-dimensional Cox models with Snet penalty

Description

Automatic model selection for high-dimensional Cox models with Snet penalty, evaluated by penalized partial-likelihood.

Usage

```r
fit_snet(x, y, nfolds = 5L, gammas = c(2.01, 2.3, 3.7, 200),
          alphas = seq(0.05, 0.95, 0.05), eps = 1e-04, max.iter = 10000L,
          seed = 1001, trace = FALSE, parallel = FALSE)
```

Arguments

- `x`  
  - Data matrix.
- `y`  
  - Response matrix made by `Surv`.
- `nfolds`  
  - Fold numbers of cross-validation.
- `gammas`  
  - Gammas to tune in `cv.ncvsurv`.
- `alphas`  
  - Alphas to tune in `cv.ncvsurv`.
- `eps`  
  - Convergence threshold.
- `max.iter`  
  - Maximum number of iterations.
- `seed`  
  - A random seed for cross-validation fold division.
- `trace`  
  - Output the cross-validation parameter tuning progress or not. Default is `FALSE`.
- `parallel`  
  - Logical. Enable parallel parameter tuning or not, default is `FALSE`. To enable parallel tuning, load the `doParallel` package and run `registerDoParallel()` with the number of CPU cores before calling this function.
Examples

data("smart")
x <- as.matrix(smart[, -c(1, 2)][1:120, ])
time <- smart$EVENT[1:120]
event <- smart$EVENT[1:120]
y <- survival::Surv(time, event)

fit <- fit_snet(
x, y,
nfolds = 3,
gammas = 3.7, alphas = c(0.3, 0.8),
max.iter = 15000, seed = 1010
)

nom <- as_nomogram(
fit, x, time, event,
pred.at = 365 * 2,
funlabel = "2-Year Overall Survival Probability"
)

plot(nom)

---

glmnet_basesurv  
Breslow baseline hazard estimator for glmnet objects

Description

Derived from peperr::basesurv and gbm::basehaz.gbm.

Usage

glmnet_basesurv(time, event, lp, times.eval = NULL, centered = FALSE)

Arguments

time  
Survival time

event  
Status indicator

lp  
Linear predictors

times.eval  
Survival time to evaluate

centered  
Should we center the survival curve? See basehaz for details.

Value

list containing cumulative base hazard

Examples

NULL
**glmnet_survcurve**  
*Survival curve prediction for glmnet objects*

**Description**

Derived from c060::predictProb.coxnet

**Usage**

```r
glmnet_survcurve(object, time, event, x, survtime)
```

**Arguments**

- `object`: glmnet model object
- `time`: Survival time
- `event`: Status indicator
- `x`: Predictor matrix
- `survtime`: Survival time to evaluate

**Value**

list containing predicted survival probabilities and linear predictors for all samples

**Examples**

```r
NULL
```

---

**infer_variable_type**  
*Extract information of selected variables from high-dimensional Cox models*

**Description**

Extract the names and type of selected variables from fitted high-dimensional Cox models.

**Usage**

```r
infer_variable_type(object, x)
```

**Arguments**

- `object`: Model object.
- `x`: Data matrix used to fit the model.
kmplot

Value

A list containing the index, name, type and range of the selected variables.

Examples

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$EVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

fit <- fit_lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
infer_variable_type(fit, x)
```

kmplot

Kaplan-Meier plot with number at risk table for internal calibration
and external calibration results

Description

Kaplan-Meier plot with number at risk table for internal calibration and external calibration results

Usage

```r
kmplot(object, group.name = NULL, time.at = NULL, col.pal = c("JCO", "Lancet", "NPG", "AAAS"))
```

Arguments

- `object`: An object returned by `calibrate` or `calibrate_external`.
- `group.name`: Risk group labels. Default is Group 1, Group 2, ..., Group k.
- `time.at`: Time points to evaluate the number at risk.
- `col.pal`: Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".

Examples

```r
data("smart")
# Use the first 1000 samples as training data
# (the data used for internal validation)
x <- as.matrix(smart[, -c(1, 2)][1:1000, ]
time <- smart$EVENT[1:1000]
event <- smart$EVENT[1:1000]

# Take the next 1000 samples as external calibration data
# In practice, usually use data collected in other studies
x_new <- as.matrix(smart[, -c(1, 2)][1001:2000, ]
time_new <- smart$EVENT[1001:2000]
```
event_new <- smart$EVENT[1001:2000]

# Fit Cox model with lasso penalty
fit <- fit_lasso(x, survival::Surv(time, event), nfolds = 5, rule = "lambda.1se", seed = 11)

# Internal calibration
cal.int <- calibrate(
  x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "cv", nfolds = 5,
  pred.at = 365 * 9, ngroup = 3
)

kmplot(  
cal.int,
  group.name = c("High risk", "Medium risk", "Low risk"),
  time.at = 1:6 * 365
)

# External calibration
cal.ext <- calibrate_external(
  fit, x, time, event,
  x_new, time_new, event_new,
  pred.at = 365 * 5, ngroup = 3
)

kmplot(  
cal.ext,
  group.name = c("High risk", "Medium risk", "Low risk"),
  time.at = 1:6 * 365
)

---

logrank_test  

Log-rank test for internal calibration and external calibration results

Description

Log-rank test for internal calibration and external calibration results

Usage

logrank_test(object)

Arguments

object  
An object returned by calibrate or calibrate_external.
Examples

data("smart")
# Use the first 1000 samples as training data
# (the data used for internal validation)
x <- as.matrix(smart[, -c(1, 2)])[1:1000, ]
time <- smart$EVENT[1:1000]
event <- smart$EVENT[1:1000]

# Take the next 1000 samples as external calibration data
# In practice, usually use data collected in other studies
x_new <- as.matrix(smart[, -c(1, 2)])[1001:2000, ]
time_new <- smart$EVENT[1001:2000]
event_new <- smart$EVENT[1001:2000]

# Fit Cox model with lasso penalty
fit <- fit_lasso(  
x, survival::Surv(time, event),
  nfolds = 5, rule = "lambda.1se", seed = 11
)

# Internal calibration
cal.int <- calibrate(  
x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "cv", nfolds = 5,
  pred.at = 365 * 9, ngroup = 3
)
logrank_test(cal.int)

# External calibration
cal.ext <- calibrate_external(  
  fit, x, time, event,
  x_new, time_new, event_new,
  pred.at = 365 * 5, ngroup = 3
)
logrank_test(cal.ext)

ncvreg_basesurv Breslow baseline hazard estimator for ncvreg objects

Description

Derived from peperr::basesurv and gbm::basehaz.gbm.

Usage

ncvreg_basesurv(time, event, lp, times.eval = NULL, centered = FALSE)
**ncvreg_survcurve**

**Arguments**

- **time**: Survival time
- **event**: Status indicator
- **lp**: Linear predictors
- **times.eval**: Survival time to evaluate
- **centered**: Should we center the survival curve? See `basehaz` for details.

**Value**

list containing cumulative base hazard

**Examples**

```r
NULL
```

---

**ncvreg_survcurve**  
*Survival curve prediction for ncvreg objects*

---

**Description**

Derived from c060::predictProb.coxnet

**Usage**

```r
ncvreg_survcurve(object, time, event, x, survtime)
```

**Arguments**

- **object**: ncvreg model object
- **time**: Survival time
- **event**: Status indicator
- **x**: Predictor matrix
- **survtime**: Survival time to evaluate

**Value**

list containing predicted survival probabilities and linear predictors for all samples

**Examples**

```r
NULL
```
penalized_basesurv  
_Breslow baseline hazard estimator for penfit objects_

**Description**  
Derived from peperr::basesurv and gbm::basehaz.gbm.

**Usage**  
```r  
penalized_basesurv(time, event, lp, times.eval = NULL,  
centered = FALSE)  
```

**Arguments**
- `time`: Survival time
- `event`: Status indicator
- `lp`: Linear predictors
- `times.eval`: Survival time to evaluate
- `centered`: Should we center the survival curve? See `basehaz` for details.

**Value**
- list containing cumulative base hazard

**Examples**
```r
NULL
```

penalized_survcurve  
_Survival curve prediction for penfit objects_

**Description**
Derived from c060::predictProb.coxnet

**Usage**
```r  
penalized_survcurve(object, time, event, x, survtime)  
```

**Arguments**
- `object`: penalized model object
- `time`: Survival time
- `event`: Status indicator
- `x`: Predictor matrix
- `survtime`: Survival time to evaluate
Value

list containing predicted survival probabilities and linear predictors for all samples

Examples

NULL

plot.hdnom.calibrate external

Plot calibration results

Description

Plot calibration results

Usage

## S3 method for class 'hdnom.calibrate'
plot(x, xlim = c(0, 1), ylim = c(0, 1),
    col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ...)

Arguments

x

An object returned by calibrate.

xlim

x axis limits of the plot.

ylim

y axis limits of the plot.

col.pal

Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".

...

Other parameters for plot.

Examples

NULL

plot.hdnom.calibrate.external

Plot external calibration results

Description

Plot external calibration results

Usage

## S3 method for class 'hdnom.calibrate.external'
plot(x, xlim = c(0, 1), ylim = c(0,
    1), col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ...)


plot.hdnom.compare.calibrate

Arguments

x An object returned by `calibrate_external`.
xlim x axis limits of the plot.
ylim y axis limits of the plot.
col.pal Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
... Other parameters for plot.

Examples

NULL

plot.hdnom.compare.calibrate

Plot model comparison by calibration results

Description

Plot model comparison by calibration results

Usage

```r
## S3 method for class 'hdnom.compare.calibrate'
plot(x, xlim = c(0, 1), ylim = c(0, 1), col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ...)
```

Arguments

x An object returned by `compare_by_calibrate`.
xlim x axis limits of the plot.
ylim y axis limits of the plot.
col.pal Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
... Other parameters (not used).

Examples

NULL
plot.hdnom.compare.validate

*Plot model comparison by validation results*

**Description**

Plot model comparison by validation results

**Usage**

```r
## S3 method for class 'hdnom.compare.validate'
plot(x, interval = FALSE,
     col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ylim = NULL, ...)
```

**Arguments**

- `x`: An object returned by `compare_by_validate`.
- `interval`: Show maximum, minimum, 0.25, and 0.75 quantiles of time-dependent AUC as ribbons? Default is `FALSE`.
- `col.pal`: Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
- `ylim`: Range of y coordinates. For example, `c(0.5, 1)`.
- `...`: Other parameters (not used).

**Examples**

```r
NULL
```

plot.hdnom.nomogram

*Plot nomogram objects*

**Description**

Plot nomogram objects

**Usage**

```r
## S3 method for class 'hdnom.nomogram'
plot(x, ...
```

**Arguments**

- `x`: An object returned by `as_nomogram`.
- `...`: Other parameters.
Examples

NULL

---

**plot.hdnom.validate**  
*Plot optimism-corrected time-dependent discrimination curves for validation*

**Description**  
Plot optimism-corrected time-dependent discrimination curves for validation

**Usage**

```r
## S3 method for class 'hdnom.validate'
plot(x, col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ylim = NULL, ...)
```

**Arguments**

- **x**: An object returned by `validate`.
- **col.pal**: Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
- **ylim**: Range of y coordinates. For example, `c(0.5, 1)`.
- **...**: Other parameters (not used).

**Examples**

NULL

---

**plot.hdnom.validate.external**  
*Plot time-dependent discrimination curves for external validation*

**Description**  
Plot time-dependent discrimination curves for external validation

**Usage**

```r
## S3 method for class 'hdnom.validate.external'
plot(x, col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ylim = NULL, ...)
```
predict.hdnom.model

Make predictions from high-dimensional Cox models

Description

Predict overall survival probability at certain time points from fitted Cox models.

Usage

## S3 method for class 'hdnom.model'
predict(object, x, y, newx, pred.at, ...)

Arguments

- **object**: Model object.
- **x**: Data matrix used to fit the model.
- **y**: Response matrix made with `Surv`.
- **newx**: Matrix (with named columns) of new values for x at which predictions are to be made.
- **pred.at**: Time point at which prediction should take place.
- **...**: Other parameters (not used).

Value

A `nrow(newx) x length(pred.at)` matrix containing overall survival probability.

Examples

```r
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$EVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

fit <- fit_lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
predict(fit, x, y, newx = x[101:105, ], pred.at = 1:10 * 365)
```
### `print.hdnom.calibrate`

**Print calibration results**

#### Description

Print calibration results

#### Usage

```r
## S3 method for class 'hdnom.calibrate'
print(x, ...)
```

#### Arguments

- `x`: An object returned by `calibrate`.
- `...`: Other parameters (not used).

#### Examples

```r
NULL
```

### `print.hdnom.calibrate.external`

**Print external calibration results**

#### Description

Print external calibration results

#### Usage

```r
## S3 method for class 'hdnom.calibrate.external'
print(x, ...)
```

#### Arguments

- `x`: An object returned by `calibrate_external`.
- `...`: Other parameters (not used).

#### Examples

```r
NULL
```
**print.hdnom.compare.calibrate**

*Print model comparison by calibration results*

**Description**

Print model comparison by calibration results

**Usage**

```r
## S3 method for class 'hdnom.compare.calibrate'
print(x, ...)
```

**Arguments**

- `x` An object returned by `compare_by_calibrate`.
- `...` Other parameters (not used).

**Examples**

```r
NULL
```

---

**print.hdnom.compare.validate**

*Print model comparison by validation results*

**Description**

Print model comparison by validation results

**Usage**

```r
## S3 method for class 'hdnom.compare.validate'
print(x, ...)
```

**Arguments**

- `x` An object returned by `compare_by_validate`.
- `...` Other parameters (not used).

**Examples**

```r
NULL
```
print.hdnom.model

Print high-dimensional Cox model objects

Description
Print high-dimensional Cox model objects

Usage
## S3 method for class 'hdnom.model'
print(x, ...)

Arguments
x Model object.
...
Other parameters (not used).

Examples
data("smart")
x <- as.matrix(smart[, -c(1, 2)])
time <- smart$EVENT
event <- smart$EVENT
y <- survival::Surv(time, event)

fit <- fit_lasso(x, y, nfolds = 5, rule = "lambda.ise", seed = 11)
print(fit)

print.hdnom.nomogram

Print nomograms objects

Description
Print nomograms objects

Usage
## S3 method for class 'hdnom.nomogram'
print(x, ...)

Arguments
x An object returned by as_nomogram.
...
Other parameters.

Examples
NULL
print.hdnom.validate  
*Print validation results*

**Description**

Print validation results

**Usage**

```r
## S3 method for class 'hdnom.validate'
print(x, ...)
```

**Arguments**

- `x`  
  An object returned by `validate`.
- `...`  
  Other parameters (not used).

**Examples**

```r
NULL
```

---

print.hdnom.validate.external  
*Print external validation results*

**Description**

Print external validation results

**Usage**

```r
## S3 method for class 'hdnom.validate.external'
print(x, ...)
```

**Arguments**

- `x`  
  An object returned by `validate_external`.
- `...`  
  Other parameters (not used).

**Examples**

```r
NULL
```
**Description**

Imputed SMART study data (no missing values).

**Usage**

data(smart)

**Format**

A numeric matrix with 3873 samples, and 29 variables (27 variables + time variable + event variable):

- **Demographics**
  - SEX - gender
  - AGE - age in years
- **Classical risk factors**
  - SMOKING - smoking (never, former, current)
  - PACKYRS - in years
  - ALCOHOL - alcohol use (never, former, current)
  - BMI - Body mass index, in kg/m^2
  - DIABETES
- **Blood pressure**
  - SYSTH - Systolic, by hand, in mm Hg
  - SYSTBP - Systolic, automatic, in mm Hg
  - DIASTH - Diastolic, by hand, in mm Hg
  - DIASTBP - Diastolic, automatic, in mm Hg
- **Lipid levels**
  - CHOL - Total cholesterol, in mmol/L
  - HDL - High-density lipoprotein cholesterol, in mmol/L
  - LDL - Low-density lipoprotein cholesterol, in mmol/L
  - TRIG - Triglycerides, in mmol/L
- **Previous symptomatic atherosclerosis**
  - CEREBRAL - Cerebral
  - CARDIAC - Coronary
  - PERIPH - Peripheral
  - AAA - Abdominal aortic aneurysm
- **Markers of atherosclerosis**
  - HOMOC - Homocysteine, in µmol/L
- GLUT - Glutamine, in $\mu$mol/L
- CREAT - Creatinine clearance, in mL/min
- ALBUMIN - Albumin (no, micro, macro)
- IMT - Intima media thickness, in mm
- STENOSIS - Carotid artery stenosis > 50%

Note

See data-raw/smart.R for the code to generate this data.

References


Examples

data(smart)
dim(smart)

<table>
<thead>
<tr>
<th>smarto</th>
<th>Original SMART study data</th>
</tr>
</thead>
</table>

Description

Original SMART study data (with missing values) from Steyerberg et al. 2008.

Usage

data(smarto)

Format

A numeric matrix with 3873 samples, and 29 variables (27 variables + time variable + event variable):

- Demographics
  - SEX - gender
  - AGE - age in years

- Classical risk factors
  - SMOKING - smoking (never, former, current)
  - PACKYRS - in years
  - ALCOHOL - alcohol use (never, former, current)
  - BMI - Body mass index, in kg/m^2
  - DIABETES

- Blood pressure
SYSTH - Systolic, by hand, in mm Hg
SYSTBP - Systolic, automatic, in mm Hg
DIASTH - Diastolic, by hand, in mm Hg
DIASTBP - Diastolic, automatic, in mm Hg

• Lipid levels
  CHOL - Total cholesterol, in mmol/L
  HDL - High-density lipoprotein cholesterol, in mmol/L
  LDL - Low-density lipoprotein cholesterol, in mmol/L
  TRIG - Triglycerides, in mmol/L

• Previous symptomatic atherosclerosis
  CEREBRAL - Cerebral
  CARDIAC - Coronary
  PERIPH - Peripheral
  AAA - Abdominal aortic aneurysm

• Markers of atherosclerosis
  HOMOC - Homocysteine, in $\mu$mol/L
  GLUT - Glutamine, in $\mu$mol/L
  CREAT - Creatinine clearance, in mL/min
  ALBUMIN - Albumin (no, micro, macro)
  IMT - Intima media thickness, in mm
  STENOSIS - Carotid artery stenosis > 50%

References

Examples
data(smarto)
dim(smarto)
Arguments

object  An object returned by `calibrate`.
...  Other parameters (not used).

Examples

NULL

---

**summary.hdnom.calibrate.external**

*Summary of external calibration results*

---

**Description**

Summary of external calibration results

**Usage**

```r
## S3 method for class 'hdnom.calibrate.external'
summary(object, ...)
```

**Arguments**

object  An object returned by `calibrate_external`.
...  Other parameters (not used).

**Examples**

NULL

---

**summary.hdnom.compare.calibrate**

*Summary of model comparison by calibration results*

---

**Description**

Summary of model comparison by calibration results

**Usage**

```r
## S3 method for class 'hdnom.compare.calibrate'
summary(object, ...)
```
summary.hdnom.compare.validate

Arguments

object An object returned by compare_by_calibrate.
... Other parameters (not used).

Examples

NULL

summary.hdnom.compare.validate

Summary of model comparison by validation results

Description

Summary of model comparison by validation results

Usage

## S3 method for class 'hdnom.compare.validate'
summary(object, silent = FALSE, ...)

Arguments

object An object compare_by_calibrate.
silent Print summary table header or not, default is FALSE.
... Other parameters (not used).

Examples

NULL

summary.hdnom.validate

Summary of validation results

Description

Summary of validation results

Usage

## S3 method for class 'hdnom.validate'
summary(object, silent = FALSE, ...)

Arguments

object An object compare_by_validate.
silent Print summary table header or not, default is FALSE.
... Other parameters (not used).

Examples

NULL
### theme_hdnom

**Plot theme (ggplot2) for hdnom**

**Description**

Plot theme (ggplot2) for hdnom

**Usage**

```r
theme_hdnom(base_size = 14)
```

**Arguments**

- `base_size`  
  base font size

---

### summary.hdnom.validate.external

**Summary of external validation results**

**Description**

Summary of external validation results

**Usage**

```r
## S3 method for class 'hdnom.validate.external'
summary(object, silent = FALSE, ...)
```

**Arguments**

- `object`  
  An object returned by `validate_external`.
- `silent`  
  Print summary table header or not, default is FALSE.
- `...`  
  Other parameters (not used).

**Examples**

```r
NULL
```

---

### theme_hdnom

**Plot theme (ggplot2) for hdnom**

**Description**

Plot theme (ggplot2) for hdnom

**Usage**

```r
theme_hdnom(base_size = 14)
```

**Arguments**

- `base_size`  
  base font size

---

### Arguments

- `object`  
  A `validate` object.
- `silent`  
  Print summary table header or not, default is FALSE.
- `...`  
  Other parameters (not used).

**Examples**

```r
NULL
```
validate high-dimensional Cox models with time-dependent AUC

Description
Validate high-dimensional Cox models with time-dependent AUC

Usage
validate(x, time, event, model.type = c("lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", "snet"), alpha, lambda, pen.factor = NULL, gamma, lambda1, lambda2, method = c("bootstrap", "cv", "repeated.cv"), boot.times = NULL, nfolds = NULL, rep.times = NULL, tauc.type = c("CD", "SZ", "UNO"), tauc.time, seed = 1001, trace = TRUE)

Arguments

x  Matrix of training data used for fitting the model; on which to run the validation.
time Survival time. Must be of the same length with the number of rows as x.
event Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as x.
model.type Model type to validate. Could be one of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
alpha Value of the elastic-net mixing parameter alpha for enet, aenet, mnet, and snet models. For lasso, alasso, mcp, and scad models, please set alpha = 1. alpha=1: lasso (l1) penalty; alpha=0: ridge (l2) penalty. Note that for mnet and snet models, alpha can be set to very close to 0 but not 0 exactly.
lambda Value of the penalty parameter lambda to use in the model fits on the resampled data. From the fitted Cox model.
pen.factor Penalty factors to apply to each coefficient. From the fitted adaptive lasso or adaptive elastic-net model.
gamma Value of the model parameter gamma for MCP/SCAD/Mnet/Snet models.
lambda1 Value of the penalty parameter lambda1 for fused lasso model.
lambda2 Value of the penalty parameter lambda2 for fused lasso model.
method Validation method. Could be "bootstrap", "cv", or "repeated.cv".
boot.times Number of repetitions for bootstrap.
nfolds Number of folds for cross-validation and repeated cross-validation.
rep.times Number of repeated times for repeated cross-validation.
tauc.type Type of time-dependent AUC. Including "CD" proposed by Chambless and Diao (2006), "SZ" proposed by Song and Zhou (2008), "UNO" proposed by Uno et al. (2007).
tauc.time Numeric vector. Time points at which to evaluate the time-dependent AUC.
seed A random seed for resampling.
trace Logical. Output the validation progress or not. Default is TRUE.
References


Examples

data(smart)
x <- as.matrix(smart[, -c(1, 2)])[1:500, ]
time <- smart$EVENT[1:500]
event <- smart$EVENT[1:500]
y <- survival::Surv(time, event)

fit <- fit_lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

# Model validation by bootstrap with time-dependent AUC
# Normally boot.times should be set to 200 or more,
# we set it to 3 here only to save example running time.
val.boot <- validate(
  x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "bootstrap", boot.times = 3,
  tauc.type = "uno", tauc.time = seq(0.25, 2, 0.25) * 365,
  seed = 1010
)

# Model validation by 5-fold cross-validation with time-dependent AUC
val.cv <- validate(
  x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "cv", nfolds = 5,
  tauc.type = "uno", tauc.time = seq(0.25, 2, 0.25) * 365,
  seed = 1010
)

# Model validation by repeated cross-validation with time-dependent AUC
val.repcv <- validate(
  x, time, event,
  model.type = "lasso",
  alpha = 1, lambda = fit$lambda,
  method = "repeated.cv", nfolds = 5, rep.times = 3,
  tauc.type = "uno", tauc.time = seq(0.25, 2, 0.25) * 365,
  seed = 1010
)

# bootstrap-based discrimination curves has a very narrow band
validate

```r
print(val.boot)
summary(val.boot)
plot(val.boot)

# k-fold cv provides a more strict evaluation than bootstrap
print(val.cv)
summary(val.cv)
plot(val.cv)

# repeated cv provides similar results as k-fold cv
# but more robust than k-fold cv
print(val.repcv)
summary(val.repcv)
plot(val.repcv)

# Test fused lasso, SCAD, and Mnet models
#
# data(smart)
# x = as.matrix(smart[, -c(1, 2)])[1:500,]
# time = smart$EVENT[1:500]
# event = smart$EVENT[1:500]
# y = survival::Surv(time, event)
#
# set.seed(1010)
# val.boot = validate(
# x, time, event, model.type = "flasso",
# lambda1 = 5, lambda2 = 2,
# method = "bootstrap", boot.times = 10,
# tauc.type = "UNO", tauc.time = seq(0.25, 2, 0.25) * 365,
# seed = 1010)
#
# val.cv = validate(
# x, time, event, model.type = "scad",
# gamma = 3.7, alpha = 1, lambda = 0.05,
# method = "cv", nfolds = 5,
# tauc.type = "UNO", tauc.time = seq(0.25, 2, 0.25) * 365,
# seed = 1010)
#
# val.repcv = validate(
# x, time, event, model.type = "mnet",
# gamma = 3, alpha = 0.3, lambda = 0.05,
# method = "repeated.cv", nfolds = 5, rep.times = 3,
# tauc.type = "UNO", tauc.time = seq(0.25, 2, 0.25) * 365,
# seed = 1010)
#
# print(val.boot)
# summary(val.boot)
# plot(val.boot)
#
# print(val.cv)
# summary(val.cv)
# plot(val.cv)
#
# print(val.repcv)
```
**validate_external**

Externally validate high-dimensional Cox models with time-dependent AUC

**Description**

Externally validate high-dimensional Cox models with time-dependent AUC

**Usage**

```r
validate_external(object, x, time, event, x_new, time_new, event_new,
                  tauc.type = c("CD", "SZ", "UNO"), tauc.time)
```

**Arguments**

- **object**: Model object fitted by `hdnom::fit_*()`.
- **x**: Matrix of training data used for fitting the model.
- **time**: Survival time of the training data. Must be of the same length with the number of rows as `x`.
- **event**: Status indicator of the training data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as `x`.
- **x_new**: Matrix of predictors for the external validation data.
- **time_new**: Survival time of the external validation data. Must be of the same length with the number of rows as `x_new`.
- **event_new**: Status indicator of the external validation data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as `x_new`.
- **tauc.type**: Type of time-dependent AUC. Including "CD" proposed by Chambless and Diao (2006), "SZ" proposed by Song and Zhou (2008), "UNO" proposed by Uno et al. (2007).
- **tauc.time**: Numeric vector. Time points at which to evaluate the time-dependent AUC.

**References**


Examples

data(smart)
# Use the first 1000 samples as training data
# (the data used for internal validation)
x <- as.matrix(smart[, -c(1, 2)][1:1000, ])
time <- smart$EVENT[1:1000]
event <- smart$EVENT[1:1000]

# Take the next 1000 samples as external validation data
# In practice, usually use data collected in other studies
x_new <- as.matrix(smart[, -c(1, 2)][1001:2000, ])
time_new <- smart$EVENT[1001:2000]
event_new <- smart$EVENT[1001:2000]

# Fit Cox model with lasso penalty
fit <- fit_lasso(x, survival::Surv(time, event),
nfolds = 5, rule = "lambda.1se", seed = 11)

# External validation with time-dependent AUC
val.ext <- validate_external(fit, x, time, event,
x_new, time_new, event_new,
tauc.type = "UNO",
tauc.time = seq(0.25, 2, 0.25) * 365)

print(val.ext)
summary(val.ext)
plot(val.ext)

# # Test fused lasso, MCP, and Snet models
# data(smart)
# # Use first 600 samples as training data
# # (the data used for internal validation)
# x <- as.matrix(smart[, -c(1, 2)][1:600, ])
# time <- smart$EVENT[1:600]
# event <- smart$EVENT[1:600]
#
# # Take 500 samples as external validation data.
# # In practice, usually use data collected in other studies.
# x_new <- as.matrix(smart[, -c(1, 2)][1001:1500, ])
# time_new <- smart$EVENT[1001:1500]
# event_new <- smart$EVENT[1001:1500]
#
# flassofit <- fit_flasso(x, survival::Surv(time, event), nfolds = 5, seed = 11)
# scadfit <- fit_mcp(x, survival::Surv(time, event), nfolds = 5, seed = 11)
# mnetfit <- fit_snet(x, survival::Surv(time, event), nfolds = 5, seed = 11)
#
# val.ext1 <- validate_external(flassofit, x, time, event,
# x_new, time_new, event_new,
# tauc.type = "UNO",
# tauc.time = seq(0.25, 2, 0.25) * 365
#
# val.ext2 <- validate_external(
# scadfit, x, time, event,
# x_new, time_new, event_new,
# tauc.type = "CD",
# tauc.time = seq(0.25, 2, 0.25) * 365)
#
# val.ext3 <- validate_external(
# mnetfit, x, time, event,
# x_new, time_new, event_new,
# tauc.type = "SZ",
# tauc.time = seq(0.25, 2, 0.25) * 365)
#
# print(val.ext1)
# summary(val.ext1)
# plot(val.ext1)
#
# print(val.ext2)
# summary(val.ext2)
# plot(val.ext2)
#
# print(val.ext3)
# summary(val.ext3)
# plot(val.ext3)
Index

as_nomogram, 3, 30, 35
basehaz, 21, 26, 27
calibrate, 4, 23, 24, 28, 33, 40
calibrate_external, 6, 23, 24, 29, 33, 40
compare_by_calibrate, 9, 29, 34, 41
compare_by_validate, 10, 30, 34, 41
cv.glmmnet, 12–14, 16
cv.ncvrsurv, 17–20
cvl, 15
fit.aenet, 11
fit.alasso, 12
fit.enet, 13
fit.flasso, 14
fit.lasso, 16
fit.mcp, 17
fit.mnet, 18
fit.scad, 19
fit.snet, 20
glmnet.basesurv, 21
glmnet.survcurve, 22
infer_variable_type, 22
kmplot, 23
logrank_test, 24
ncvreg.basesurv, 25
ncvreg.survcurve, 26
penalized, 15
penalized.basesurv, 27
penalized.survcurve, 27
plot.hdnom.calibrate, 28
plot.hdnom.evaluate, 28
plot.hdnom.compare.calibrate, 29
plot.hdnom.compare.validate, 30
plot.hdnom.nomogram, 30
plot.hdnom.validate, 31
plot.hdnom.evaluate.externa, 31
predict.hdnom.model, 32
print.hdnom.calibrate, 33
print.hdnom.evaluate.externa, 33
print.hdnom.compare.calibrate, 34
print.hdnom.compare.validate, 34
print.hdnom.model, 35
print.hdnom.nomogram, 35
print.hdnom.evaluate, 36
print.hdnom.evaluate.externa, 36
smart, 37
smarto, 38
summary.hdnom.calibrate, 39
summary.hdnom.evaluate.externa, 40
summary.hdnom.compare.calibrate, 40
summary.hdnom.compare.validate, 41
summary.hdnom.validate, 41
summary.hdnom.evaluate.externa, 42
Surv, 12–20, 32
theme.hdnom, 42
validate, 31, 36, 42, 43
validate.externa, 32, 36, 42, 46

49