Package 'hdnom'

September 29, 2017

Type Package

Title Benchmarking and Visualization Toolkit for Penalized Cox Models

Version 4.9

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

URL https://hdnom.org, https://github.com/road2stat/hdnom, http://hdnom.io

BugReports https://github.com/road2stat/hdnom/issues

Depends R (>= 3.0.2)

Imports survival, glmnet, penalized, ncvreg (>= 3.8-0), rms, foreach, survAUC, ggplot2 (>= 2.2.1), gridExtra

Suggests knitr, rmarkdown, doParallel, Hmisc, mice

Encoding UTF-8

RoxygenNote 6.0.1.9000

NeedsCompilation no

Author Nan Xiao [aut, cre],
Qingsong Xu [aut],
Miaozhu Li [aut]

Repository CRAN

Date/Publication 2017-09-29 03:47:53 UTC

R topics documented:

hdnom-package *Benchmarking and Visualization Toolkit for Penalized Cox Models*

Description

Benchmarking and Visualization Toolkit for Penalized Cox Models

Details

Browse vignettes with `browseVignettes(package = "hdnom")`.

Package:	hdnom
Type:	Package
License:	GPL-3

Author(s)

Nan Xiao <<me@nanx.me>> Qingsong Xu <<qsxu@csu.edu.cn>> Miaozhu Li <<miaozhu.li@duke.edu>>

hdcox.aenet *Adaptive Elastic-Net Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.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

```
library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

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

# Fit Cox model with adaptive elastic-net penalty
fit = hdcox.aenet(
  x, y, nfolds = 3, alphas = c(0.3, 0.7),
  rule = "lambda.1se", seed = c(5, 7))

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$aenet_model, model.type = "aenet",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

---

hdcox.alasso *Adaptive Lasso Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.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

```
library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

# Fit Cox model with adaptive lasso penalty
fit = hdcox.alasso(x, y, nfolds = 3, rule = "lambda.1se", seed = c(7, 11))

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$alasso_model, model.type = "alasso",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

---

hdcox.enet *Elastic-Net Model Selection for High-Dimensional Cox Models*

Description

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

Usage

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

Arguments

x	Data matrix.
y	Response matrix made by 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	A random seed for cross-validation fold division.
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

```
library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

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

# Fit Cox model with elastic-net penalty
fit = hdcox.enet(x, y, nfolds = 3, alphas = c(0.3, 0.7),
                 rule = "lambda.1se", seed = 11)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$enet_model, model.type = "enet",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

`hdcox.flasso` *Fused Lasso Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.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 `0.001, 0.05, 0.5, 1, 5`.
`lambda2`	Vector of lambda2 candidates. Default is `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 `registerDoParallel()` with the number of CPU cores before calling this function.
`...`	other parameters to `cvl` and `penalized`.

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

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
```

```
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model with fused lasso penalty
fit = hdcox.flasso(x, y,
  lambda1 = c(1, 10), lambda2 = c(0.01),
  nfolds = 3, seed = 11)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$flasso_model, model.type = "flasso",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

`hdcox.lasso` *Lasso Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.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

```
library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

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

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$lasso_model, model.type = "lasso",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

---

hdcox.mcp *MCP Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.mcp(x, y, nfolds = 5L, gammas = c(1.01, 1.7, 3, 100), 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 threshhold.

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

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 150 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:150, ]
time = smart$TEVENT[1:150]
event = smart$EVENT[1:150]
y = Surv(time, event)

# Fit Cox model with MCP penalty
fit = hdcox.mcp(x, y, nfolds = 3, gammas = c(2.1, 3), seed = 1001)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$mcp_model, model.type = "mcp",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

---

hdcox.mnet *Mnet Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.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 threshhold.
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

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model with Mnet penalty
fit = hdcox.mnet(
  x, y, nfolds = 3,
  gammas = 3, alphas = c(0.3, 0.8),
  max.iter = 15000, seed = 1010)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$mnet_model, model.type = "mnet",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

hdcox.scad *SCAD Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.scad(x, y, nfolds = 5L, 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 threshhold.
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

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model with SCAD penalty
fit = hdcox.scad(
  x, y, nfolds = 3, gammas = c(3.7, 5),
  max.iter = 15000, seed = 1010)

# Prepare data for hdnom.nomogram
```

```
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$scad_model, model.type = "scad",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

---

hdcox.snet *Snet Model Selection for High-Dimensional Cox Models*

Description

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

Usage

```
hdcox.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 threshhold.
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

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model with Snet penalty
fit = hdcox.snet(
  x, y, nfolds = 3,
  gammas = 3.7, alphas = c(0.3, 0.8),
  max.iter = 15000, seed = 1010)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$snet_model, model.type = "snet",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

---

hdnom.calibrate *Calibrate High-Dimensional Cox Models*

Description

Calibrate High-Dimensional Cox Models

Usage

```
hdnom.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.
ngroup	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

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

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

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

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

print(cal.fitting)
summary(cal.fitting)
plot(cal.fitting)

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

`hdnom.compare.calibrate` *Compare High-Dimensional Cox Models by Model Calibration*

Description

Compare High-Dimensional Cox Models by Model Calibration

Usage

```
hdnom.compare.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

```
# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT

# Compare lasso and adaptive lasso by 5-fold cross-validation
cmp.cal.cv = hdnom.compare.calibrate(
  x, time, event,
  model.type = c("lasso", "alasso"),
  method = "fitting",
  pred.at = 365 * 9, ngroup = 5, seed = 1001)

print(cmp.cal.cv)
summary(cmp.cal.cv)
plot(cmp.cal.cv)
```

---

hdnom.compare.validate

Compare High-Dimensional Cox Models by Model Validation

Description

Compare High-Dimensional Cox Models by Model Validation

Usage

```
hdnom.compare.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

Chambless, L. E. and G. Diao (2006). Estimation of time-dependent area under the ROC curve for long-term risk prediction. *Statistics in Medicine* 25, 3474–3486.

Song, X. and X.-H. Zhou (2008). A semiparametric approach for the covariate specific ROC curve with survival outcome. *Statistica Sinica* 18, 947–965.

Uno, H., T. Cai, L. Tian, and L. J. Wei (2007). Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association* 102, 527–537.

Examples

```
# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])[1:1000, ]
time = smart$TEVENT[1:1000]
event = smart$EVENT[1:1000]

# Compare lasso and adaptive lasso by 5-fold cross-validation
cmp.val.cv = hdnom.compare.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)
```

---

hdnom.external.calibrate

Externally Calibrate High-Dimensional Cox Models

Description

Externally Calibrate High-Dimensional Cox Models

Usage

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

Arguments

object	Model object fitted by hdcox.*() functions.
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

```
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$TEVENT[1:1000]
event = smart$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$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

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

# External calibration
cal.ext = hdnom.external.calibrate(
  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, xlim = c(0.6, 1), ylim = c(0.6, 1))
```

`hdnom.external.validate` *Externally Validate High-Dimensional Cox Models with Time-Dependent AUC*

Description

Externally Validate High-Dimensional Cox Models with Time-Dependent AUC

Usage

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

Arguments

`object`	Model object fitted by `hdcox.*()` functions.
`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

Chambless, L. E. and G. Diao (2006). Estimation of time-dependent area under the ROC curve for long-term risk prediction. *Statistics in Medicine* 25, 3474–3486.

Song, X. and X.-H. Zhou (2008). A semiparametric approach for the covariate specific ROC curve with survival outcome. *Statistica Sinica* 18, 947–965.

Uno, H., T. Cai, L. Tian, and L. J. Wei (2007). Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association* 102, 527–537.

Examples

```
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$TEVENT[1:1000]
event = smart$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$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

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

# External validation with time-dependent AUC
val.ext = hdnom.external.validate(
  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)
```

`hdnom.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

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

Arguments

object	An object returned by hdnom.calibrate or hdnom.external.calibrate.
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

```
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$TEVENT[1:1000]
event = smart$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$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

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

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

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

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

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

hdnom.logrank *Log-Rank Test for Internal Calibration and External Calibration Results*

Description

Log-Rank Test for Internal Calibration and External Calibration Results

Usage

```
hdnom.logrank(object)
```

Arguments

object	An object returned by hdnom.calibrate or hdnom.external.calibrate.

Examples

```
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$TEVENT[1:1000]
event = smart$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$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

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

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

hdnom.logrank(cal.int)

# External calibration
```

```
cal.ext = hdnom.external.calibrate(
  fit, x, time, event,
  x_new, time_new, event_new,
  pred.at = 365 * 5, ngroup = 3)

hdnom.logrank(cal.ext)
```

`hdnom.nomogram` *Nomograms for High-Dimensional Cox Models*

Description

Nomograms for High-Dimensional Cox Models

Usage

```
hdnom.nomogram(object, model.type = c("lasso", "alasso", "flasso", "enet",
  "aenet", "mcp", "mnet", "scad", "snet"), x, time, event, ddist,
  pred.at = NULL, fun.at = NULL, funlabel = NULL)
```

Arguments

`object`	Fitted model object.
`model.type`	Fitted model type. Could be one of `"lasso"`, `"alasso"`, `"flasso"`, `"enet"`, `"aenet"`, `"mcp"`, `"mnet"`, `"scad"`, or `"snet"`.
`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`.
`ddist`	Data frame version of x, made by `datadist`.
`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

We will try to use the value of the selected penalty parameter (e.g. lambda, alpha) in the model object fitted by `glmnet`, `ncvsurv`, or `penalized`. The selected variables under the penalty parameter will be used to build the nomogram and make predictions. This means, if you use routines other than `hdcox.*` functions to fit the model, please make sure that there is only one set of necessary parameters (e.g. only a single lambda should be in glmnet objects intead of a vector of multiple lambdas) in the fitted model object.

Examples

```
library("rms")

# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

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

# Prepare data needed for plotting nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(
  fit$lasso_model, model.type = "lasso",
  x, time, event, x.df, pred.at = 365 * 2,
  funlabel = "2-Year Overall Survival Probability")

print(nom)
plot(nom)
```

`hdnom.validate` *Validate High-Dimensional Cox Models with Time-Dependent AUC*

Description

Validate High-Dimensional Cox Models with Time-Dependent AUC

Usage

```
hdnom.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 built Cox model.
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	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

Chambless, L. E. and G. Diao (2006). Estimation of time-dependent area under the ROC curve for long-term risk prediction. *Statistics in Medicine* 25, 3474–3486.

Song, X. and X.-H. Zhou (2008). A semiparametric approach for the covariate specific ROC curve with survival outcome. *Statistica Sinica* 18, 947–965.

Uno, H., T. Cai, L. Tian, and L. J. Wei (2007). Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association* 102, 527–537.

Examples

```
library("survival")

# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])[1:500, ]
time = smart$TEVENT[1:500]
event = smart$EVENT[1:500]
y = Surv(time, event)

# Fit penalized Cox model with lasso penalty
```

```
fit = hdcox.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 = hdnom.validate(
  x, time, event, model.type = "lasso",
  alpha = 1, lambda = fit$lasso_best_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 = hdnom.validate(
  x, time, event, model.type = "lasso",
  alpha = 1, lambda = fit$lasso_best_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 = hdnom.validate(
  x, time, event, model.type = "lasso",
  alpha = 1, lambda = fit$lasso_best_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
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)
```

`hdnom.varinfo` *Extract Information of Selected Variables from High-Dimensional Cox Models*

Description

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

Usage

```
hdnom.varinfo(object, x)
```

Arguments

object	Model object fitted by hdcox.*() functions.
x	Data matrix used to fit the model.

Value

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

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

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

---

plot.hdnom.calibrate *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 hdnom.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.compare.calibrate`
Plot Model Comparison by Calibration Results

Description

Plot Model Comparison by Calibration Results

Usage

```
## 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 `hdnom.compare.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

```
## 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 hdnom.compare.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

```
NULL
```

---

plot.hdnom.external.calibrate

Plot External Calibration Results

Description

Plot External Calibration Results

Usage

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

Arguments

x	An object returned by hdnom.external.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.external.validate *Plot Time-Dependent Discrimination Curves for External Validation*

Description

Plot Time-Dependent Discrimination Curves for External Validation

Usage

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

Arguments

x	An object returned by hdnom.external.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.nomogram *Plot Nomogram Objects Generated by hdnom.nomogram*

Description

Plot Nomogram Objects Generated by hdnom.nomogram

Usage

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

Arguments

x	An object returned by hdnom.nomogram.
...	Other parameters for nomogram.

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

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

Arguments

x	An object returned by hdnom.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
```

predict.hdcox.model *Make Predictions from High-Dimensional Cox Models*

Description

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

Usage

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

Arguments

object	Model object fitted by hdcox.*() functions.
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 probablity.

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

fit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
predict(fit, x, y, newx = x[101:105, ], pred.at = 1:10 * 365)
```

`print.hdcox.model` *Print High-Dimensional Cox Model Objects*

Description

Print information about high-dimensional Cox models.

Usage

```
## S3 method for class 'hdcox.model'
print(x, ...)
```

Arguments

`x`	Model object fitted by `hdcox.*()` functions.
`...`	Other parameters (not used).

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

fit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
print(fit)
```

print.hdnom.calibrate *Print Calibration Results*

Description

Print Calibration Results

Usage

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

Arguments

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

Examples

```
NULL
```

print.hdnom.compare.calibrate *Print Model Comparison by Calibration Results*

Description

Print Model Comparison by Calibration Results

Usage

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

Arguments

x	An object returned by hdnom.compare.calibrate.
...	Other parameters (not used).

Examples

```
NULL
```

print.hdnom.compare.validate

Print Model Comparison by Validation Results

Description

Print Model Comparison by Validation Results

Usage

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

Arguments

x	An object returned by hdnom.compare.validate.
...	Other parameters (not used).

Examples

```
NULL
```

print.hdnom.external.calibrate

Print External Calibration Results

Description

Print External Calibration Results

Usage

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

Arguments

x	An object returned by hdnom.external.calibrate.
...	Other parameters (not used).

Examples

```
NULL
```

print.hdnom.external.validate *Print External Validation Results*

Description

Print External Validation Results

Usage

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

Arguments

x	An object returned by hdnom.external.validate.
...	Other parameters (not used).

Examples

```
NULL
```

print.hdnom.nomogram *Print Nomograms Objects Generated by hdnom.nomogram*

Description

Print Nomograms Objects Generated by hdnom.nomogram

Usage

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

Arguments

x	An object returned by hdnom.nomogram.
...	Other parameters for nomogram.

Examples

```
NULL
```

print.hdnom.validate *Print Validation Results*

Description

Print Validation Results

Usage

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

Arguments

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

Examples

```
NULL
```

smart *Imputed SMART Study Data*

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 μmol/L
  - CREAT - Creatinine clearance, in mL/min
  - ALBUMIN - Albumin (no, micro, macro)
  - IMT - Intima media thickness, in mm
  - STENOSIS - Carotid artery stenosis > 50%

References

Steyerberg, E. W. (2008). Clinical prediction models: a practical approach to development, validation, and updating. Springer Science & Business Media.

Examples

```
data(smart)
dim(smart)

# # Code for generating the smart dataset by imputing the smarto data
# library("mice")
# library("Hmisc")
# data(smarto)
# set.seed(1000)
# smarti = aregImpute(~ I(TEVENT) + EVENT + SEX + I(AGE) +
#   DIABETES + CEREBRAL + CARDIAC + AAA + PERIPH +
#   STENOSIS + SYSTBP + DIASTBP + SYSTH + DIASTH +
#   I(LENGTH) + I(WEIGHT) + I(BMI) +
#   I(CHOL) + I(HDL) + I(LDL) + I(TRIG) + I(HOMOC) + I(GLUT) + I(CREAT) + I(IMT) +
```

```
#   as.factor(ALBUMIN) + as.factor(SMOKING) + I(PACKYRS) + as.factor(ALCOHOL),
#   n.impute = 10, data = smarto)
# imputed = impute.transcan(smarti, imputation = 1, data = smarto,
#   list.out = TRUE, pr = FALSE, check = FALSE)
# smart = smarto
# smart[names(imputed)] = imputed
# str(smart)
```

`smarto` *Original SMART Study Data*

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 μmol/L
  - GLUT - Glutamine, in μmol/L
  - CREAT - Creatinine clearance, in mL/min
  - ALBUMIN - Albumin (no, micro, macro)
  - IMT - Intima media thickness, in mm
  - STENOSIS - Carotid artery stenosis > 50%

References

Steyerberg, E. W. (2008). Clinical prediction models: a practical approach to development, validation, and updating. Springer Science & Business Media.

Examples

```
data(smarto)
dim(smarto)
```

`summary.hdnom.calibrate`

Summary of Calibration Results

Description

Summary of Calibration Results

Usage

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

Arguments

`object`	An object returned by `hdnom.calibrate`.
`...`	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

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

Arguments

object	An object returned by hdnom.compare.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 hdnom.compare.validate.
silent	Print summary table header or not, default is FALSE.
...	Other parameters (not used).

Examples

```
NULL
```

summary.hdnom.external.calibrate

Summary of External Calibration Results

Description

Summary of External Calibration Results

Usage

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

Arguments

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

Examples

```
NULL
```

summary.hdnom.external.validate

Summary of External Validation Results

Description

Summary of External Validation Results

Usage

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

Arguments

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

Examples

```
NULL
```

Index