Package ‘survivalmodels’

March 24, 2022

Title Models for Survival Analysis

Version 0.1.13


License MIT + file LICENSE

URL https://github.com/RaphaelS1/survivalmodels/

BugReports https://github.com/RaphaelS1/survivalmodels/issues

Imports Rcpp (>= 1.0.5)

Suggests distr6 (>= 1.6.6), keras, pseudo, reticulate, survival, testthat

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation yes

Author Raphael Sonabend [aut, cre] (<https://orcid.org/0000-0001-9225-4654>)

Maintainer Raphael Sonabend <raphaelsonabend@gmail.com>

Repository CRAN

Date/Publication 2022-03-24 08:40:05 UTC
R topics documented:

survivalmodels-package ........................................... 2
akritas ................................................................. 3
build_keras_net ......................................................... 4
build_pytorch_net ....................................................... 6
cindex ................................................................. 7
coxtime ................................................................. 8
deephit ................................................................. 11
depsurv ................................................................. 14
dnnsurv ................................................................. 16
get_keras_optimizer ................................................... 19
get_pycox_activation ................................................ 21
get_pycox_callbacks ............................................... 24
get_pycox_init ......................................................... 25
get_pycox_optim ...................................................... 26
install_keras .......................................................... 28
install_pycox .......................................................... 29
install_torch .......................................................... 30
loghaz ................................................................. 30
pchazard ............................................................... 33
predict.akritas ....................................................... 36
predict.dnnsurv ....................................................... 38
predict.pycox ........................................................ 40
pycox_prepare_train_data ......................................... 41
requireNamespaces .................................................. 43
set_seed ............................................................... 43
simsurvdata .......................................................... 44
surv_to_risk .......................................................... 44

Index 46

survivalmodels-package

survivalmodels: Models for Survival Analysis

Description

survivalmodels implements classical and machine learning models for survival analysis that either
do not already exist in R or for more efficient implementations.

Author(s)

Maintainer: Raphael Sonabend <raphaelsonabend@gmail.com> (ORCID)
akritas

See Also

Useful links:

- https://github.com/RaphaelS1/survivalmodels/
- Report bugs at https://github.com/RaphaelS1/survivalmodels/issues

akritas  

Akritas Conditional Non-Parametric Survival Estimator

Description

The Akritas survival estimator is a conditional nearest-neighbours approach to the more common Kaplan-Meier estimator. Common usage includes IPCW Survival models and measures, which do not assume that censoring is independent of the covariates.

Usage

akritas(
  formula = NULL,
  data = NULL,
  reverse = FALSE,
  time_variable = "time",
  status_variable = "status",
  x = NULL,
  y = NULL,
  ...
)

Arguments

formula (formula(1))
  Object specifying the model fit, left-hand-side of formula should describe a survival::Surv() object.

data (data.frame(1))
  Training data of data.frame like object, internally is coerced with stats::model.matrix().

reverse (logical(1))
  If TRUE fits estimator on censoring distribution, otherwise (default) survival distribution.

time_variable (character(1))
  Alternative method to call the function. Name of the 'time' variable, required if formula or x and Y not given.

status_variable (character(1))
  Alternative method to call the function. Name of the 'status' variable, required if formula or x and Y not given.
build_keras_net

Description
Utility function to build a Keras MLP.

Usage
build_keras_net(
  n_in,
  n_out,
  nodes = c(32L, 32L),
  layer_pars = list(),
  activation = "linear",
  act_pars = list(),

Details
This implementation uses a fit/predict interface to allow estimation on unseen data after fitting on training data. This is achieved by fitting the empirical CDF on the training data and applying this to the new data.

Value
An object inheriting from class akritas.

References

Examples
if (requireNamespaces(c("distr6", "survival"))) {
  library(survival)
  akritas(Surv(time, status) ~ ., data = rats[1:10, ])
}

$build_keras_net

Build a Keras Multilayer Perceptron

Prints

$build_keras_net
### Arguments

- **n_in** (`integer(1)`)
  Number of input features.
- **n_out** (`integer(1)`)
  Number of targets.
- **nodes** (`numeric()`)  
  Hidden nodes in network, each element in vector represents number of hidden nodes in respective layer.
- **layer_pars** (`list()`)  
  Arguments passed to `keras::layer_dense`.
- **activation** (`character(1)`)  
  Activation function passed to `keras::layer_activation`. Default is linear.
- **act_pars** (`list()`)  
  Parameters for activation function, see `keras::layer_activation`.
- **dropout** (`numeric(1)`)  
  Optional dropout layer, if NULL then no dropout layer added otherwise either same dropout will be added to all layers.
- **batch_norm** (`logical(1)`)  
  If TRUE (default) then batch normalisation is applied to all layers.
- **batch_pars** (`list()`)  
  Parameters for batch normalisation, see `keras::layer_batch_normalization`.

### Details

This function is a helper for R users with less Python experience. Currently it is limited to simple MLPs and with identical layers. More advanced networks will require manual creation with `keras`.

### Examples

```r
if (requireNamespaces("keras")) {
  build_keras_net(4L, 2L)

  build_keras_net(n_in = 4L, n_out = 2L, nodes = c(32L, 64L, 32L),
                 activation = "elu", dropout = 0.4)
}
```
Description

Utility function to build an MLP with a choice of activation function and weight initialization with optional dropout and batch normalization.

Usage

build_pytorch_net(
  n_in,  
  n_out,  
  nodes = c(32, 32),  
  activation = "relu",  
  act_pars = list(),  
  dropout = 0.1,  
  bias = TRUE,  
  batch_norm = TRUE,  
  batch_pars = list(eps = 1e-05, momentum = 0.1, affine = TRUE),  
  init = "uniform",  
  init_pars = list()
)

Arguments

n_in (integer(1))
Number of input features.
n_out (integer(1))
Number of targets.
nodes (numeric())
Hidden nodes in network, each element in vector represents number of hidden nodes in respective layer.
activation (character(1)|list())
Activation function, can either be a single character and the same function is used in all layers, or a list of length length(nodes). See get_pycox_activation for options.
act_pars (list())
Passed to get_pycox_activation.
dropout (numeric())
Optional dropout layer, if NULL then no dropout layer added otherwise either a single numeric which will be added to all layers or a vector of differing drop-out amounts.
bias (logical(1))
If TRUE (default) then a bias parameter is added to all linear layers.
batch_norm (logical(1))
If TRUE (default) then batch normalisation is applied to all layers.

batch_pars (list())
Parameters for batch normalisation, see reticulate::py_help(torch$nn$BatchNorm1d).

init (character(1))
Weight initialization method. See get_pycox_init for options.

init_pars (list())
Passed to get_pycox_init.

Details
This function is a helper for R users with less Python experience. Currently it is limited to simple
MLPs. More advanced networks will require manual creation with reticulate.

Examples

if (requireNamespaces("reticulate")) {
  build_pytorch_net(4L, 2L, nodes = c(32, 64, 32), activation = "selu")

  # pass parameters to activation and initializer functions
  build_pytorch_net(4L, 2L, activation = "elu", act_pars = list(alpha = 0.1),
                   init = "kaiming_uniform", init_pars = list(mode = "fan_out"))
}

cindex
Compute Concordance of survivalmodel Risk

Description
A thin wrapper around survival::concordance which essentially just sets reverse = TRUE.

Usage
cindex(risk, truth, ...)

Arguments

risk (numeric())
Vector of risk predictions from a survivalmodel model (so high risk implies
low survival time prediction).

truth (numeric())
Vector of true survival times, must be same length as risk.

... (ANY)
Further parameters passed to survival::concordance.
Examples

```r
if (!requireNamespace("survival", quietly = TRUE)) {
  set.seed(10)
  data <- simsurv(data = data[1:10, ])
  fit <- deepsurv(data = data[1:10, ])
  p <- predict(fit, type = "risk", newdata = data[11:20, ])
  concordance(risk = p, truth = data[11:20, "time"])
}
```

---

**coxtime**

*Cox-Time Survival Neural Network*

**Description**

Cox-Time fits a neural network based on the Cox PH with time-varying effects.

**Usage**

```r
coxtime(
  formula = NULL,
  data = NULL,
  reverse = FALSE,
  time_variable = "time",
  status_variable = "status",
  x = NULL,
  y = NULL,
  frac = 0,
  standardize_time = FALSE,
  log_duration = FALSE,
  with_mean = TRUE,
  with_std = TRUE,
  activation = "relu",
  num_nodes = c(32L, 32L),
  batch_norm = TRUE,
  dropout = NULL,
  device = NULL,
  shrink = 0,
  early_stopping = FALSE,
  best_weights = FALSE,
  min_delta = 0,
  patience = 10L,
  batch_size = 256L,
  epochs = 1L,
  verbose = FALSE,
  num_workers = 0L,
  shuffle = TRUE,
  ...
)
```
Arguments

**formula** *(formula(1))*  
Object specifying the model fit, left-hand-side of formula should describe a `survival::Surv()` object.

**data** *(data.frame(1))*  
Training data of `data.frame` like object, internally is coerced with `stats::model.matrix()`.

**reverse** *(logical(1))*  
If `TRUE` fits estimator on censoring distribution, otherwise (default) survival distribution.

**time_variable** *(character(1))*  
Alternative method to call the function. Name of the 'time' variable, required if formula, or `x` and `Y` not given.

**status_variable** *(character(1))*  
Alternative method to call the function. Name of the 'status' variable, required if formula or `x` and `Y` not given.

**x** *(data.frame(1))*  
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Data frame like object of features which is internally coerced with `model.matrix`.

**y** *(survival::Surv())*  
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Survival outcome of right-censored observations.

**frac** *(numeric(1))*  
Fraction of data to use for validation dataset, default is 0 and therefore no separate validation dataset.

**standardize_time** *(logical(1))*  
If `TRUE`, the time outcome is standardized.

**log_duration** *(logical(1))*  
If `TRUE` and standardize_time is `TRUE` then time variable is log transformed.

**with_mean** *(logical(1))*  
If `TRUE` (default) and standardize_time is `TRUE` then time variable is centered.

**with_std** *(logical(1))*  
If `TRUE` (default) and standardize_time is `TRUE` then time variable is scaled to unit variance.

**activation** *(character(1))*  
See `get_pycox_activation`.

**num_nodes, batch_norm, dropout** *(integer()/logical(1)/numeric(1))*  
See `build_pytorch_net`.

**device** *(integer(1)|character(1))*  
Passed to `pycox.models.Coxtime`, specifies device to compute models on.

**shrink** *(numeric(1))*  
Passed to `pycox.models.Coxtime`, shrinkage parameter for regularization.
early_stopping, best_weights, min_delta, patience
  (logical(1)/logical(1)/numeric(1)/integer(1))
See get_pycox_callbacks.
batch_size (integer(1))
  Passed to pycox.models.Coxtime.fit, elements in each batch.
epochs (integer(1))
  Passed to pycox.models.Coxtime.fit, number of epochs.
verbose (logical(1))
  Passed to pycox.models.Coxtime.fit, should information be displayed during fitting.
num_workers (integer(1))
  Passed to pycox.models.Coxtime.fit, number of workers used in the dataloader.
shuffle (logical(1))
  Passed to pycox.models.Coxtime.fit, should order of dataset be shuffled?
... ANY
  Passed to get_pycox_optim.

Details
Implemented from the pycox Python package via reticulate. Calls pycox.models.Coxtime.

Value
An object inheriting from class coxtime.
An object of class survivalmodel.

References

Examples
if (requireNamespaces("reticulate")) {
  # all defaults
  coxtime(data = simsurvdata(50))

  # common parameters
  coxtime(data = simsurvdata(50), frac = 0.3, activation = "relu",
         num_nodes = c(4L, 8L, 4L, 2L), dropout = 0.1, early_stopping = TRUE, epochs = 100L,
         batch_size = 32L)
}
**DeepHit Survival Neural Network**

**Description**

DeepHit fits a neural network based on the PMF of a discrete Cox model. This is the single (non-competing) event implementation.

**Usage**

```r
deephit(
  formula = NULL,
  data = NULL,
  reverse = FALSE,
  time_variable = "time",
  status_variable = "status",
  x = NULL,
  y = NULL,
  frac = 0,
  cuts = 10,
  cutpoints = NULL,
  scheme = c("equidistant", "quantiles"),
  cut_min = 0,
  activation = "relu",
  custom_net = NULL,
  num_nodes = c(32L, 32L),
  batch_norm = TRUE,
  dropout = NULL,
  device = NULL,
  mod_alpha = 0.2,
  sigma = 0.1,
  early_stopping = FALSE,
  best_weights = FALSE,
  min_delta = 0,
  patience = 10L,
  batch_size = 256L,
  epochs = 1L,
  verbose = FALSE,
  num_workers = 0L,
  shuffle = TRUE,
  ...
)
```

**Arguments**

- **formula** *(formula(1))*
  
  Object specifying the model fit, left-hand-side of formula should describe a `survival::Surv()` object.
data (data.frame(1))
Training data of data.frame like object, internally is coerced with stats::model.matrix().

reverse (logical(1))
If TRUE fits estimator on censoring distribution, otherwise (default) survival distribution.

time_variable (character(1))
Alternative method to call the function. Name of the 'time' variable, required if formula or x and Y not given.

status_variable (character(1))
Alternative method to call the function. Name of the 'status' variable, required if formula or x and Y not given.

x (data.frame(1))
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Data frame like object of features which is internally coerced with model.matrix.

y ([survival::Surv()])
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Survival outcome of right-censored observations.

frac (numeric(1))
Fraction of data to use for validation dataset, default is 0 and therefore no separate validation dataset.

cuts (integer(1))
If discretise is TRUE then determines number of cut-points for discretisation.

cutpoints (numeric())
Alternative to cuts if discretise is true, provide exact cutpoints for discretisation. cuts is ignored if cutpoints is non-NULL.

scheme (character(1))
Method of discretisation, either "equidistant" (default) or "quantiles". See reticulate::py_help(pycox$models$LogisticHazard$label_transform) for more detail.

cut_min (integer(1))
Starting duration for discretisation, see reticulate::py_help(pycox$models$LogisticHazard$label_transform) for more detail.

activation (character(1))
See get_pycox_activation.

custom_net (torch.nn.modules.module.Module(1))
Optional custom network built with build_pytorch_net, otherwise default architecture used. Note that if building a custom network the number of output channels depends on cuts or cutpoints.

num_nodes, batch_norm, dropout (integer()/logical(1)/numeric(1))
See build_pytorch_net.

device (integer(1)|character(1))
Passed to pycox.models.DeepHitSingle, specifies device to compute models on.
### deephit

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_alpha</td>
<td>numeric(1)</td>
<td>Weighting in (0,1) for combining likelihood (L1) and rank loss (L2). See reference and <code>py_help(pycox$models$DeepHitSingle)</code> for more detail.</td>
</tr>
<tr>
<td>sigma</td>
<td>numeric(1)</td>
<td>From eta in rank loss (L2) of ref. See reference and <code>py_help(pycox$models$DeepHitSingle)</code> for more detail.</td>
</tr>
<tr>
<td>early_stopping, best_weights, min_delta, patience</td>
<td>logical(1)/logical(1)/numeric(1)/integer(1)</td>
<td>Passed to <code>get_pycox_callbacks</code>.</td>
</tr>
<tr>
<td>batch_size</td>
<td>integer(1)</td>
<td>Passed to <code>pycox.models.DeepHitSingle.fit</code>, elements in each batch.</td>
</tr>
<tr>
<td>epochs</td>
<td>integer(1)</td>
<td>Passed to <code>pycox.models.DeepHitSingle.fit</code>, number of epochs.</td>
</tr>
<tr>
<td>verbose</td>
<td>logical(1)</td>
<td>Passed to <code>pycox.models.DeepHitSingle.fit</code>, should information be displayed during fitting.</td>
</tr>
<tr>
<td>num_workers</td>
<td>integer(1)</td>
<td>Passed to <code>pycox.models.DeepHitSingle.fit</code>, number of workers used in the dataloader.</td>
</tr>
<tr>
<td>shuffle</td>
<td>logical(1)</td>
<td>Passed to <code>pycox.models.DeepHitSingle.fit</code>, should order of dataset be shuffled?</td>
</tr>
<tr>
<td>...</td>
<td>ANY</td>
<td>Passed to <code>get_pycox_optim</code>.</td>
</tr>
</tbody>
</table>

### Details


### Value

An object inheriting from class `deephit`.

An object of class `survivalmodel`.

### References


### Examples

```r
if (requireNamespaces("reticulate")) {
  # all defaults
  deephit(data = simsurvdata(50))
```
# common parameters
deepsurv(data = simsurvdata(50), frac = 0.3, activation = "relu",
        num_nodes = c(4L, 8L, 4L, 2L), dropout = 0.1, early_stopping = TRUE, epochs = 100L,
        batch_size = 32L)
}

---

deepsurv  

_DeepSurv Survival Neural Network_

**Description**

DeepSurv neural fits a neural network based on the partial likelihood from a Cox PH.

**Usage**

```r
deejsonr(  
  formula = NULL,  
  data = NULL,  
  reverse = FALSE,  
  time_variable = "time",  
  status_variable = "status",  
  x = NULL,  
  y = NULL,  
  frac = 0,  
  activation = "relu",  
  num_nodes = c(32L, 32L),  
  batch_norm = TRUE,  
  dropout = NULL,  
  device = NULL,  
  early_stopping = FALSE,  
  best_weights = FALSE,  
  min_delta = 0,  
  patience = 10L,  
  batch_size = 256L,  
  epochs = 1L,  
  verbose = FALSE,  
  num_workers = 0L,  
  shuffle = TRUE,  
  ...
)
```

**Arguments**

- **formula**  
  Object specifying the model fit, left-hand-side of formula should describe a `survival::Surv()` object.
data (data.frame(1))
Training data of data.frame like object, internally is coerced with stats::model.matrix().
reverse (logical(1))
If TRUE fits estimator on censoring distribution, otherwise (default) survival distribution.
time_variable (character(1))
Alternative method to call the function. Name of the 'time' variable, required if formula or x and Y not given.
status_variable (character(1))
Alternative method to call the function. Name of the 'status' variable, required if formula or x and Y not given.
x (data.frame(1))
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Data frame like object of features which is internally coerced with model.matrix.
y ([survival::Surv()])
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Survival outcome of right-censored observations.
frac (numeric(1))
Fraction of data to use for validation dataset, default is 0 and therefore no separate validation dataset.
activation (character(1))
See get_pycox_activation.
num_nodes, batch_norm, dropout (integer()/logical(1)/numeric(1))
See build_pytorch_net.
device (integer(1)|character(1))
Passed to pycox.models.CoxPH, specifies device to compute models on.
early_stopping, best_weights, min_delta, patience (logical(1)/logical(1)/numeric(1)/integer(1))
See get_pycox_callbacks.
batch_size (integer(1))
Passed to pycox.models.CoxPH.fit, elements in each batch.
epochs (integer(1))
Passed to pycox.models.CoxPH.fit, number of epochs.
verbose (logical(1))
Passed to pycox.models.CoxPH.fit, should information be displayed during fitting.
num_workers (integer(1))
Passed to pycox.models.CoxPH.fit, number of workers used in the dataloader.
shuffle (logical(1))
Passed to pycox.models.CoxPH.fit, should order of dataset be shuffled?
... ANY
Passed to get_pycox_optim.
Details

Implemented from the pycox Python package via reticulate. Calls pycox.models.CoxPH.

Value

An object inheriting from class deepsurv.
An object of class survivalmodel.

References


Examples

```r
if (requireNamespaces("reticulate")) {
  # all defaults
deepsurv(data = simsurvdata(50))

  # common parameters
deepsurv(data = simsurvdata(50), frac = 0.3, activation = "relu",
          num_nodes = c(4L, 8L, 4L, 2L), dropout = 0.1, early_stopping = TRUE, epochs = 100L,
          batch_size = 32L)
}
```

---

**dnnsurv**

**DNNSurv Neural Network for Conditional Survival Probabilities**

Description

DNNSurv neural fits a neural network based on pseudo-conditional survival probabilities.

Usage

```r
dnnsurv(
  formula = NULL,
  data = NULL,
  reverse = FALSE,
  time_variable = "time",
  status_variable = "status",
  x = NULL,
  y = NULL,
  cutpoints = NULL,
  cuts = 5L,
```
custom_model = NULL,
loss_weights = NULL,
weighted_metrics = NULL,
optimizer = "adam",
early_stopping = FALSE,
min_delta = 0,
patience = 0L,
verbose = 0L,
baseline = NULL,
restore_best_weights = FALSE,
batch_size = 32L,
ePOCHS = 10L,
validation_split = 0,
shuffle = TRUE,
sample_weight = NULL,
initial_epoch = 0L,
steps_per_epoch = NULL,
validation_steps = NULL,
...)

Arguments

formula (formula(1))
Object specifying the model fit, left-hand-side of formula should describe a survival::Surv() object.
data (data.frame(1))
Training data of data.frame like object, internally is coerced with stats::model.matrix().
reverse (logical(1))
If TRUE fits estimator on censoring distribution, otherwise (default) survival distribution.
time_variable (character(1))
Alternative method to call the function. Name of the 'time' variable, required if formula or x and Y not given.
status_variable (character(1))
Alternative method to call the function. Name of the 'status' variable, required if formula or x and Y not given.
x (data.frame(1))
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Data frame like object of features which is internally coerced with model.matrix.
y ([survival::Surv()])
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Survival outcome of right-censored observations.
cutpoints (numeric())
Points at which to cut survival time into discrete points.
cuts  (integer(1))
    If cutpoints not provided then number of equally spaced points at which to cut survival time.

custom_model  (keras.engine.training.Model(1))
    Optional custom architecture built with build_keras_net or directly with keras. Output layer should be of length 1 input is number of features plus number of cuts.

loss_weights, weighted_metrics
    See keras::compile.keras.engine.training.Model.

optimizer  (character(1))
    See get_keras_optimizer.

eyear_stopping (logical(1))
    If TRUE then early stopping callback is included.

min_delta, patience, baseline, restore_best_weights
    See keras::callback_early_stopping.

verbose  (integer(1))
    Level of verbosity for printing, 0 or 1.

batch_size, epochs, validation_split, shuffle, sample_weight, initial_epoch, steps_per_epoch, validation_steps
    See keras::fit.keras.engine.training.Model. # nolint

... ANY
    Passed to get_keras_optimizer.

Details

Code for generating the conditional probabilities and pre-processing data is taken from https://github.com/lilizhaoUM/DNNSurv.

Value

An object of class survivalmodel.

References


Examples

```r
if (requireNamespaces(c("keras", "pseudo")))
  # all defaults
dnnsurv(data = simsurvdata(10))

  # setting common parameters
dnnsurv(time_variable = "time", status_variable = "status", data = simsurvdata(10),
   early_stopping = TRUE, epochs = 100L, validation_split = 0.3)

  # custom model
```
library(keras)
cuts <- 10
df <- simsurvdata(50)
# shape = features + cuts
input <- layer_input(shape = c(3L + cuts), name = 'input')
output <- input %>%
  layer_dense(units = 4L, use_bias = TRUE) %>%
  layer_dense(units = 1L, use_bias = TRUE) %>%
  layer_activation(activation = "sigmoid")

model <- keras_model(input, output)
class(model)

dnnsurv(custom_model = model, time_variable = "time",
        status_variable = "status", data = df, cuts = cuts)

get_keras_optimizer
Get Keras Optimizer

Description
Utility function to construct optimiser from keras, primarily for internal use.

Usage
get_keras_optimizer(
  optimizer = "adam",
  lr = 0.02,
  beta_1 = 0.9,
  beta_2 = 0.999,
  epsilon = NULL,
  decay = 0,
  clipnorm = NULL,
  clipvalue = NULL,
  schedule_decay = 0.004,
  momentum = 0,
  nesterov = FALSE
)

Arguments

optimizer (character(1))
  Optimizer to construct, see details for those available. Default is "adam".

lr (numeric(1))
  Passed to all optimizers except adadelta and adagrad.
get_keras_optimizer

beta_1, beta_2, epsilon
   (numeric(1))
   Passed to adamax, adam, and nadam.

decay
   (numeric(1))
   Passed to adamax, adam, and sgd.

clipnorm, clipvalue
   (numeric(1))
   Passed to adamax, adam, nadam, and sgd.

schedule_decay
   (numeric(1))
   Passed to nadam.

momentum
   (numeric(1))
   Passed to sgd.

nesterov
   (logical(1))
   Passed to sgd.

Details

Implemented optimizers are

- "adadelta"
  keras::optimizer_adadelta
- "adagrad"
  keras::optimizer_adagrad
- "adamax"
  keras::optimizer_adamax
- "adam"
  keras::optimizer_adam
- "nadam"
  keras::optimizer_nadam
- "rmsprop"
  keras::optimizer_rmsprop
- "sgd"
  keras::optimizer_sgd

Examples

if (requireNamespaces("keras")) {
  get_keras_optimizer()

  get_keras_optimizer(optimizer = "adamax", decay = 0.1, lr = 0.01)
}
**get_pycox_activation**  

Get Pytorch Activation Function

**Description**

Helper function to return a class or constructed object for pytorch activation function from `torch.nn.modules.activation`.

**Usage**

```r
get_pycox_activation(
  activation = "relu",
  construct = TRUE,
  alpha = 1,
  dim = NULL,
  lambd = 0.5,
  min_val = -1,
  max_val = 1,
  negative_slope = 0.01,
  num_parameters = 1L,
  init = 0.25,
  lower = 1/8,
  upper = 1/3,
  beta = 1,
  threshold = 20,
  value = 20
)
```

**Arguments**

- `activation` (character(1))  
  Activation function method, see details for list of implemented methods.
- `construct` (logical(1))  
  If TRUE (default) returns constructed object, otherwise a class.
- `alpha` (numeric(1))  
  Passed to celu and elu.
- `dim` (integer(1))  
  Passed to glu, logsoftmax, softmax, and
- `lambd` (numeric(1))  
  Passed to hardshrink and softshrink.
- `min_val, max_val` (numeric(1))  
  Passed to hardtanh.
- `negative_slope` (numeric(1))  
  Passed to leakyrelu.
num_parameters (integer(1))
   Passed to prelu.
init (numeric(1))
   Passed to prelu.
lower, upper (numeric(1))
   Passed to rrelu.
beta (numeric(1))
   Passed to softplus.
threshold (numeric(1))
   Passed to softplus and threshold.
value (numeric(1))
   Passed to threshold.

Details

Implemented methods (with help pages) are

- "celu"
  reticulate::py_help(torch$nn$modules$activation$CELU)
- "elu"
  reticulate::py_help(torch$nn$modules$activation$ELU)
- "gelu"
  reticulate::py_help(torch$nn$modules$activation$GELU)
- "glu"
  reticulate::py_help(torch$nn$modules$activation$GLU)
- "hardshrink"
  reticulate::py_help(torch$nn$modules$activation$Hardshrink)
- "hardsigmoid"
  reticulate::py_help(torch$nn$modules$activation$Hardsigmoid)
- "hardswish"
  reticulate::py_help(torch$nn$modules$activation$Hardswish)
- "hardtanh"
  reticulate::py_help(torch$nn$modules$activation$Hardtanh)
- "relu6"
  reticulate::py_help(torch$nn$modules$activation$ReLU6)
- "leakyrelu"
  reticulate::py_help(torch$nn$modules$activation$LeakyReLU)
- "logsigmoid"
  reticulate::py_help(torch$nn$modules$activation$LogSigmoid)
- "logsoftmax"
  reticulate::py_help(torch$nn$modules$activation$LogSoftmax)
- "prelu"
  reticulate::py_help(torch$nn$modules$activation$PReLU)
get_pycox_activation

- "rrelu"
  reticulate::py_help(torch$nn$modules$activation$RReLU)
- "relu"
  reticulate::py_help(torch$nn$modules$activation$ReLU)
- "selu"
  reticulate::py_help(torch$nn$modules$activation$SELU)
- "sigmoid"
  reticulate::py_help(torch$nn$modules$activation$Sigmoid)
- "softmax"
  reticulate::py_help(torch$nn$modules$activation$Softmax)
- "softmax2d"
  reticulate::py_help(torch$nn$modules$activation$Softmax2d)
- "softmin"
  reticulate::py_help(torch$nn$modules$activation$Softmin)
- "softplus"
  reticulate::py_help(torch$nn$modules$activation$Softplus)
- "softshrink"
  reticulate::py_help(torch$nn$modules$activation$Softshrink)
- "softsign"
  reticulate::py_help(torch$nn$modules$activation$Softsign)
- "tanh"
  reticulate::py_help(torch$nn$modules$activation$Tanh)
- "tanhshrink"
  reticulate::py_help(torch$nn$modules$activation$Tanhshrink)
- "threshold"
  reticulate::py_help(torch$nn$modules$activation$Threshold)

Examples

if (requireNamespaces("reticulate")) {
  # returns constructed objects
  get_pycox_activation(activation = "relu", construct = TRUE)
  # returns class
  get_pycox_activation(activation = "selu", construct = FALSE)
}
get_pycox_callbacks  Get Torchtuples Callbacks

Description

Helper function to return torchtuples callbacks from torchtuples.callbacks.

Usage

get_pycox_callbacks(
  early_stopping = FALSE,
  best_weights = FALSE,
  min_delta = 0,
  patience = 10L
)

Arguments

early_stopping  (logical(1))
   If TRUE then constructs torchtuples.callbacks.EarlyStopping.

best_weights  (logical(1))
   If TRUE then returns torchtuples.callbacks.BestWeights. Ignored if early_stopping
   is TRUE.

min_delta  (numeric(1))
   Passed to torchtuples.callbacks.EarlyStopping.

patience  (integer(1))
   Passed to torchtuples.callbacks.EarlyStopping.

Examples

if (requireNamespaces("reticulate")) {
  get_pycox_callbacks(early_stopping = TRUE)

  get_pycox_callbacks(best_weights = TRUE)
}
get_pycox_init  

Get Pytorch Weight Initialization Method

Description
Helper function to return a character string with a populated pytorch weight initializer method from torch.nn.init. Used in build_pytorch_net to define a weighting function.

Usage
get_pycox_init(
  init = "uniform",
  a = 0,
  b = 1,
  mean = 0,
  std = 1,
  val,
  gain = 1,
  mode = c("fan_in", "fan_out"),
  non_linearity = c("leaky_relu", "relu")
)

Arguments

  init (character(1))
  Initialization method, see details for list of implemented methods.

  a (numeric(1))
  Passed to uniform, kaiming_uniform, and kaiming_normal.

  b (numeric(1))
  Passed to uniform.

  mean, std (numeric(1))
  Passed to normal.

  val (numeric(1))
  Passed to constant.

  gain (numeric(1))
  Passed to xavier_uniform, xavier_normal, and orthogonal.

  mode (character(1))
  Passed to kaiming_uniform and kaiming_normal, one of fan_in (default) and fan_out.

  non_linearity (character(1))
  Passed to kaiming_uniform and kaiming_normal, one of leaky_relu (default) and relu.
get_pycox_optim

Details

Implemented methods (with help pages) are

- "uniform"
  \texttt{reticulate::py\_help(torch\$nn\$init\$uniform\_)}
- "normal"
  \texttt{reticulate::py\_help(torch\$nn\$init\$normal\_)}
- "constant"
  \texttt{reticulate::py\_help(torch\$nn\$init\$constant\_)}
- "xavier\_uniform"
  \texttt{reticulate::py\_help(torch\$nn\$init\$xavier\_uniform\_)}
- "xavier\_normal"
  \texttt{reticulate::py\_help(torch\$nn\$init\$xavier\_normal\_)}
- "kaiming\_uniform"
  \texttt{reticulate::py\_help(torch\$nn\$init\$kaiming\_uniform\_)}
- "kaiming\_normal"
  \texttt{reticulate::py\_help(torch\$nn\$init\$kaiming\_normal\_)}
- "orthogonal"
  \texttt{reticulate::py\_help(torch\$nn\$init\$orthogonal\_)}

Examples

```r
if (requireNamespaces("reticulate")) {
  get_pycox_init(init = "uniform")
  get_pycox_init(init = "kaiming_uniform", a = 0, mode = "fan_out")
}
```

---

get_pycox_optim \hspace{1cm} \textit{Get Pytorch Optimizer}

Description

Helper function to return a constructed pytorch optimizer from \texttt{torch.optim}.

Usage

```r
get_pycox_optim(
  optimizer = "adam",
  net,
  rho = 0.9,
  eps = 1e-08,
)```
get_pycox_optim

```r
def get_pycox_optim(
  lr = 1,
  weight_decay = 0,
  learning_rate = 0.01,
  lr_decay = 0,
  betas = c(0.9, 0.999),
  amsgrad = FALSE,
  lambd = 1e-04,
  alpha = 0.75,
  t0 = 1e+06,
  momentum = 0,
  centered = TRUE,
  etas = c(0.5, 1.2),
  step_sizes = c(1e-06, 50),
  dampening = 0,
  nesterov = FALSE
)
```

**Arguments**

- `optimizer` (character(1))
  Optimizer, see details for list of implemented methods.

- `net` (torch.nn.modules.module.Module)
  Network architecture, can be built from `build_pytorch_net`.

- `rho, lr, lr_decay` (numeric(1))
  Passed to adadelta.

- `eps` (numeric(1))
  Passed to all methods except asgd, rprop, and sgd.

- `weight_decay` (numeric(1))
  Passed to all methods except rprop and sparse_adam.

- `learning_rate` (numeric(1))
  Passed to all methods except adadelta.

- `betas` (numeric(2))
  Passed to adam, adamax, adamw, and sparse_adam.

- `amsgrad` (logical(1))
  Passed to adam and adamw.

- `lambd, t0` (numeric(1))
  Passed to asgd.

- `alpha` (numeric(1))
  Passed to asgd and rmsprop.

- `momentum` (numeric(1))
  Passed to rmsprop and sgd.

- `centered` (logical(1))
  Passed to rmsprop.
etas, step_sizes
  (numeric(2))
  Passed to rprop.

dampening  (numeric(1))
  Passed to sgd.

nesterov   (logical(1))
  Passed to sgd.

Details

Implemented methods (with help pages) are

- "adadelta"
  reticulate::py_help(torch$optim$Adadelta)
- "adagrad"
  reticulate::py_help(torch$optim$Adagrad)
- "adam"
  reticulate::py_help(torch$optim$Adam)
- "adamax"
  reticulate::py_help(torch$optim$Adamax)
- "adamw"
  reticulate::py_help(torch$optim$AdamW)
- "asgd"
  reticulate::py_help(torch$optim$ASGD)
- "rmsprop"
  reticulate::py_help(torch$optim$RMSprop)
- "rprop"
  reticulate::py_help(torch$optim$Rprop)
- "sgd"
  reticulate::py_help(torch$optim$SGD)
- "sparse_adam"
  reticulate::py_help(torch$optim$SparseAdam)

install_keras  

Install Keras and Tensorflow

Description

Stripped back version of keras::install_keras. Note the default for pip is changed to TRUE.
install_pycox

Usage

```r
install_keras(
    method = "auto",
    conda = "auto",
    pip = TRUE,
    install_tensorflow = FALSE,
    ...
)
```

Arguments

- `method`, `conda`, `pip`
  See `reticulate::py_install`.
- `install_tensorflow`
  If `TRUE` installs the dependency `tensorflow` package as well.
- `...`
  Passed to `reticulate::py_install`.

install_pycox

Install Pycox With Reticulate

Description

Installs the python 'pycox' package via reticulate. Note the default for `pip` is changed to `TRUE`.

Usage

```r
install_pycox(
    method = "auto",
    conda = "auto",
    pip = TRUE,
    install_torch = FALSE,
    ...
)
```

Arguments

- `method`, `conda`, `pip`
  See `reticulate::py_install`.
- `install_torch`
  If `TRUE` installs the dependency `torch` package as well.
- `...`
  Passed to `reticulate::py_install`. 
install_torch  

**Install Torch With Reticulate**

**Description**
Installs the python `torch` package via reticulate. Note the default for `pip` is changed to `TRUE`.

**Usage**
```
install_torch(method = "auto", conda = "auto", pip = TRUE)
```

**Arguments**
- `method`, `conda`, `pip`  
  See `reticulate::py_install`

loghaz  

**Logistic-Hazard Survival Neural Network**

**Description**
Logistic-Hazard fits a discrete neural network based on a cross-entropy loss and predictions of a discrete hazard function, also known as Nnet-Survival.

**Usage**
```
loghaz(
  formula = NULL,
  data = NULL,
  reverse = FALSE,
  time_variable = "time",
  status_variable = "status",
  x = NULL,
  y = NULL,
  frac = 0,
  cuts = 10,
  cutpoints = NULL,
  scheme = c("equidistant", "quantiles"),
  cut_min = 0,
  activation = "relu",
  custom_net = NULL,
  num_nodes = c(32L, 32L),
  batch_norm = TRUE,
  dropout = NULL,
  device = NULL,
  early_stopping = FALSE,
)```
loghaz

```r
best_weights = FALSE,
min_delta = 0,
patience = 10L,
batch_size = 256L,
ePOCHS = 1L,
verbose = FALSE,
num_workers = 0L,
shuffle = TRUE,
...
```

### Arguments

**formula** (formula(1))
Object specifying the model fit, left-hand-side of formula should describe a `survival::Surv()` object.

**data** (data.frame(1))
Training data of `data.frame` like object, internally is coerced with `stats::model.matrix()`.

**reverse** (logical(1))
If TRUE fits estimator on censoring distribution, otherwise (default) survival distribution.

**time_variable** (character(1))
Alternative method to call the function. Name of the 'time' variable, required if formula, or x and Y not given.

**status_variable** (character(1))
Alternative method to call the function. Name of the 'status' variable, required if formula or x and Y not given.

**x** (data.frame(1))
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Data frame like object of features which is internally coerced with `model.matrix`.

**y** ([`survival::Surv()`])
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Survival outcome of right-censored observations.

**frac** (numeric(1))
Fraction of data to use for validation dataset, default is 0 and therefore no separate validation dataset.

**cuts** (integer(1))
If discretise is TRUE then determines number of cut-points for discretisation.

**cutpoints** (numeric())
Alternative to cuts if discretise is true, provide exact cutpoints for discretisation. cuts is ignored if cutpoints is non-NULL.

**scheme** (character(1))
Method of discretisation, either "equidistant" (default) or "quantiles". See `reticulate::py_help(pycox$models$LogisticHazard$label_transform)` for more detail.
cut_min (integer(1))
Starting duration for discretisation, see reticulate::py_help(pycox$models$LogisticHazard$label_transform) for more detail.

activation (character(1))
See get_pycox_activation.

custom_net (torch.nn.modules.module.Module(1))
Optional custom network built with build_pytorch_net, otherwise default architecture used. Note that if building a custom network the number of output channels depends on cuts or cutpoints.

num_nodes, batch_norm, dropout
(integer()/logical(1)/numeric(1))
See build_pytorch_net.

device (integer(1)|character(1))
Passed to pycox.models.LogisticHazard, specifies device to compute models on.

early_stopping, best_weights, min_delta, patience
(logical(1)/logical(1)/numeric(1)/integer(1)
See get_pycox_callbacks.

batch_size (integer(1))
Passed to pycox.models.LogisticHazard.fit, elements in each batch.

epochs (integer(1))
Passed to pycox.models.LogisticHazard.fit, number of epochs.

verbose (logical(1))
Passed to pycox.models.LogisticHazard.fit, should information be displayed during fitting.

num_workers (integer(1))
Passed to pycox.models.LogisticHazard.fit, number of workers used in the dataloader.

shuffle (logical(1))
Passed to pycox.models.LogisticHazard.fit, should order of dataset be shuffled?

... ANY
Passed to get_pycox_optim.

Details


Value

An object inheriting from class loghaz.

An object of class survivalmodel.
References


Examples

```r
if (requireNamespaces("reticulate")) {
  # all defaults
  loghaz(data = simsurvdata(50))

  # common parameters
  loghaz(data = simsurvdata(50), frac = 0.3, activation = "relu",
         num_nodes = c(4L, 8L, 4L, 2L), dropout = 0.1, early_stopping = TRUE, epochs = 100L,
         batch_size = 32L)
}
```

pchazard

PC-Hazard Survival Neural Network

Description

Logistic-Hazard fits a discrete neural network based on a cross-entropy loss and predictions of a discrete hazard function, also known as Nnet-Survival.

Usage

```r
pchazard(
  formula = NULL,
  data = NULL,
  reverse = FALSE,
  time_variable = "time",
  status_variable = "status",
  x = NULL,
  y = NULL,
  frac = 0,
  cuts = 10,
  cutpoints = NULL,
  scheme = c("equidistant", "quantiles"),
  cut_min = 0,
  activation = "relu",
  custom_net = NULL,
  num_nodes = c(32L, 32L),
)```
Arguments

formula (formula(1))
Object specifying the model fit, left-hand-side of formula should describe a survival::Surv() object.

data (data.frame(1))
Training data of data.frame like object, internally is coerced with stats::model.matrix().

reverse (logical(1))
If TRUE fits estimator on censoring distribution, otherwise (default) survival distribution.

time_variable (character(1))
Alternative method to call the function. Name of the 'time' variable, required if formula or x and Y not given.

status_variable (character(1))
Alternative method to call the function. Name of the 'status' variable, required if formula or x and Y not given.

x (data.frame(1))
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Data frame like object of features which is internally coerced with model.matrix.

y ([survival::Surv()])
Alternative method to call the function. Required if formula, time_variable and status_variable not given. Survival outcome of right-censored observations.

frac (numeric(1))
Fraction of data to use for validation dataset, default is 0 and therefore no separate validation dataset.

cuts (integer(1))
If discretise is TRUE then determines number of cut-points for discretisation.
cutpoints (numeric())
Alternative to cuts if discretise is true, provide exact cutpoints for discretisation. cuts is ignored if cutpoints is non-NULL.

scheme (character(1))
Method of discretisation, either "equidistant" (default) or "quantiles". See reticulate::py_help(pycox$models$LogisticHazard$label_transform) for more detail.

cut_min (integer(1))
Starting duration for discretisation, see reticulate::py_help(pycox$models$LogisticHazard$label_transform) for more detail.

activation (character(1))
See get_pycox_activation.

custom_net (torch.nn.modules.module.Module(1))
Optional custom network built with build_pytorch_net, otherwise default architecture used. Note that if building a custom network the number of output channels depends on cuts or cutpoints.

num_nodes, batch_norm, dropout
(integer()/logical(1)/numeric(1))
See build_pytorch_net.

reduction (character(1))
How to reduce the loss, see to reticulate::py_help(pycox$models$loss$NLLPCHazardLoss).

device (integer(1)|character(1))
Passed to pycox.models.PCHazard, specifies device to compute models on.

early_stopping, best_weights, min_delta, patience
(logical(1)/logical(1)/numeric(1)/integer(1))
See get_pycox_callbacks.

batch_size (integer(1))
Passed to pycox.models.PCHazard.fit, elements in each batch.

epochs (integer(1))
Passed to pycox.models.PCHazard.fit, number of epochs.

verbose (logical(1))
Passed to pycox.models.PCHazard.fit, should information be displayed during fitting.

num_workers (integer(1))
Passed to pycox.models.PCHazard.fit, number of workers used in the dataloader.

shuffle (logical(1))
Passed to pycox.models.PCHazard.fit, should order of dataset be shuffled?

... ANY
Passed to get_pycox_optim.

Details
Implemented from the pycox Python package via reticulate. Calls pycox.models.PCHazard.
Value

An object inheriting from class pchazard.
An object of class survivalmodel.

References


Examples

if (requireNamespaces("reticulate")) {
  # all defaults
  pchazard(data = simsurvdata(50))

  # common parameters
  pchazard(data = simsurvdata(50), frac = 0.3, activation = "relu",
           num_nodes = c(4L, 8L, 4L, 2L), dropout = 0.1, early_stopping = TRUE, epochs = 100L,
           batch_size = 32L)
}

predict.akritas      Predict method for Akritas Estimator

Description

Predicted values from a fitted Akritas estimator.

Usage

## S3 method for class 'akritas'
predict(
  object, 
  newdata, 
  times = NULL, 
  lambda = 0.5, 
  type = c("survival", "risk", "all"), 
  distr6 = FALSE, 
  ...
)
**Arguments**

- **object** (akritas(1))
  Object of class inheriting from "akritas".
- **newdata** (data.frame(1))
  Testing data of data.frame like object, internally is coerced with `stats::model.matrix()`. If missing then training data from fitted object is used.
- **times** (numeric())
  Times at which to evaluate the estimator. If NULL (default) then evaluated at all unique times in the training set.
- **lambda** (numeric(1))
  Bandwidth parameter for uniform smoothing kernel in nearest neighbours estimation. The default value of 0.5 is arbitrary and should be chosen by the user.
- **type** (character(1))
  Type of predicted value. Choices are survival probabilities over all time-points in training data ("survival") or a relative risk ranking ("risk"), which is the negative mean survival time so higher rank implies higher risk of event, or both ("all").
- **distr6** (logical(1))
  If FALSE (default) and type is "survival" or "all" returns matrix of survival probabilities, otherwise returns a `distr6::Matdist()`.
- **...** ANY
  Currently ignored.

**Details**

This implementation uses a fit/predict interface to allow estimation on unseen data after fitting on training data. This is achieved by fitting the empirical CDF on the training data and applying this to the new data.

**Value**

A numeric if type = "risk", a `distr6::Matdist()` (if distr6 = TRUE) and type = "survival"; a matrix if (distr6 = FALSE) and type = "survival" where entries are survival probabilities with rows of observations and columns are time-points; or a list combining above if type = "all".

**References**


**Examples**

```r
if (requireNamespaces(c("distr6", "survival"))) {

library(survival)

train <- 1:10
```
```r
test <- 11:20
fit <- akritas(Surv(time, status) ~ ., data = rats[train, ])
predict(fit, newdata = rats[test, ])

# when lambda = 1, identical to Kaplan-Meier
fit <- akritas(Surv(time, status) ~ ., data = rats[1:100, ])
predictAkritas <- predict(fit, newdata = rats[1:100, ], lambda = 1)[1, ]
predict_km <- survfit(Surv(time, status) ~ 1, data = rats[1:100, ])$surv
all(predictAkritas == predict_km)

# Use distr6 = TRUE to return a distribution
predict_distr <- predict(fit, newdata = rats[test, ], distr6 = TRUE)
predict_distr$survival(100)

# Return a relative risk ranking with type = "risk"
predict(fit, newdata = rats[test, ], type = "risk")

# Or survival probabilities and a rank
predict(fit, newdata = rats[test, ], type = "all", distr6 = TRUE)
```

---

**predict.dnnsurv**  
*Predict Method for DNNSurv*

**Description**

Predicted values from a fitted object of class dnnsurv.

**Usage**

```r
## S3 method for class 'dnnsurv'
predict(
  object,
  newdata,
  batch_size = 32L,
  verbose = 0L,
  steps = NULL,
  callbacks = NULL,
  type = c("survival", "risk", "all"),
  distr6 = FALSE,
  ...
)
```

**Arguments**

- **object** (dnnsurv(1))
  Object of class inheriting from "dnnsurv".
**predict.dnnsurv**

- **newdata**
  
  Testing data of data.frame like object, internally is coerced with `stats::model.matrix()`. If missing then training data from fitted object is used.

- **batch_size**
  
  Passed to `keras::predict.keras.engine.training.Model`, elements in each batch.

- **verbose**
  
  Level of verbosity for printing, 0 or 1.

- **steps**
  
  Number of batches before evaluation finished, see `keras::predict.keras.engine.training.Model`.

- **callbacks**
  
  Optional callbacks to apply during prediction.

- **type**
  
  Type of predicted value. Choices are survival probabilities over all time-points in training data ("survival") or a relative risk ranking ("risk"), which is the negative mean survival time so higher rank implies higher risk of event, or both ("all").

- **distr6**
  
  If FALSE (default) and type is "survival" or "all" returns matrix of survival probabilities, otherwise returns a `distr6::Matdist()`.

... **ANY**

Currently ignored.

**Value**

A numeric if type = "risk", a `distr6::Matdist()` (if distr6 = TRUE) and type = "survival"; a matrix if (distr6 = FALSE) and type = "survival" where entries are survival probabilities with rows of observations and columns are time-points; or a list combining above if type = "all".

**Examples**

```r
if (requireNamespaces(c("keras", "pseudo")))
  fit <- dnnsurv(data = simsurvdata(10))

  # predict survival matrix and relative risks
  predict(fit, simsurvdata(10), type = "all")

  # return as distribution
  if (requireNamespaces("distr6")) {
    predict(fit, simsurvdata(10), distr6 = TRUE)
  }
```
predict.pycox

Predict Method for pycox Neural Networks

Description

Predicted values from a fitted pycox ANN.

Usage

```r
## S3 method for class 'pycox'
predict(
  object,
  newdata,
  batch_size = 256L,
  num_workers = 0L,
  interpolate = FALSE,
  inter_scheme = c("const_hazard", "const_pdf"),
  sub = 10L,
  type = c("survival", "risk", "all"),
  distr6 = FALSE,
  ...
)
```

Arguments

- `object` (pycox(1))
  Object of class inheriting from "pycox".
- `newdata` (data.frame(1))
  Testing data of data.frame like object, internally is coerced with `stats::model.matrix()`. If missing then training data from fitted object is used.
- `batch_size` (integer(1))
  Passed to `pycox.models.X.fit`, elements in each batch.
- `num_workers` (integer(1))
  Passed to `pycox.models.X.fit`, number of workers used in the dataloader.
- `interpolate` (logical(1))
  For models deephit and loghaz, should predictions be linearly interpolated? Ignored for other models.
- `inter_scheme` (character(1))
  If `interpolate` is TRUE then the scheme for interpolation, see `reticulate::py_help(py_help(pycox$models$DeepHitSingle$interpolate))` for further details.
- `sub` (integer(1))
  If `interpolate` is TRUE or model is loghaz, number of sub-divisions for interpolation. See `reticulate::py_help(py_help(pycox$models$DeepHitSingle$interpolate))` for further details.
pycox_prepare_train_data  

**Prepare Data for Pycox Model Training**

**Description**

Utility function to prepare data for training in a Pycox model. Generally used internally only.

**Usage**

```r
pycox_prepare_train_data(
  x_train, 
  y_train, 
  frac = 0,
)```

**Value**

A numeric if `type = "risk"`, a `distr6::Matdist()` (if `distr6 = TRUE`) and `type = "survival"`; a matrix if (`distr6 = FALSE`) and `type = "survival"` where entries are survival probabilities with rows of observations and columns are time-points; or a list combining above if `type = "all"`.

**Examples**

```r
if (requireNamespaces("reticulate")) {
  fit <- coxtime(data = simsurvdata(50))

  # predict survival matrix and relative risks
  predict(fit, simsurvdata(10), type = "all")

  # return as distribution
  if (requireNamespaces("distr6")) {
    predict(fit, simsurvdata(10), distr6 = TRUE)
  }
}
```
standardize_time = FALSE,
log_duration = FALSE,
with_mean = TRUE,
with_std = TRUE,
discretise = FALSE,
cuts = 10L,
cutpoints = NULL,
scheme = c("equidistant", "quantiles"),
cut_min = 0L,
model = c("coxtime", "deepsurv", "deephit", "loghaz", "pchazard")
)

Arguments

x_train (matrix(1))
Training covariates.
y_train (matrix(1))
Training outcomes.
frac (numeric(1))
Fraction of data to use for validation dataset, default is 0 and therefore no separate validation dataset.
standardize_time (logical(1))
If TRUE, the time outcome to be standardized. For use with coxtime.
log_duration (logical(1))
If TRUE and standardize_time is TRUE then time variable is log transformed.
with_mean (logical(1))
If TRUE (default) and standardize_time is TRUE then time variable is centered.
with_std (logical(1))
If TRUE (default) and standardize_time is TRUE then time variable is scaled to unit variance.
discretise (logical(1))
If TRUE then time is discretised. For use with the models deephit, pchazard, and loghaz.
cuts (integer(1))
If discretise is TRUE then determines number of cut-points for discretisation.
cutpoints (numeric(1))
Alternative to cuts if discretise is true, provide exact cutpoints for discretisation. cuts is ignored if cutpoints is non-NULL.
scheme (character(1))
Method of discretisation, either "equidistant" (default) or "quantiles". See reticulate::py_help(pycox$models$LogisticHazard$label_transform).
cut_min (integer(1))
Starting duration for discretisation, see reticulate::py_help(pycox$models$LogisticHazard$label_transform).
model (character(1))
Corresponding pycox model.
**requireNamespaces**  
*Vectorised Logical requireNamespace*

---

### Description

Helper function for internal use. Vectorises the `requireNamespace` function and returns TRUE if all packages, `x`, are available and FALSE otherwise.

### Usage

```r
requireNamespaces(x)
```

### Arguments

- **x**  
  (character())
  string naming the packages/name spaces to load.

---

**set_seed**  
*Set seed in R numpy and torch*

---

### Description

To ensure consistent results, a seed has to be set in R using `set.seed` as usual but also in numpy and torch via reticulate. Therefore this function simplifies the process into one function.

### Usage

```r
set_seed(seed_R, seed_np = seed_R, seed_torch = seed_R)
```

### Arguments

- **seed_R**  
  (integer(1))
  seed passed to `set.seed`.

- **seed_np**  
  (integer(1))
  seed passed to `numpy$random$seed`. Default is same as `seed_R`.

- **seed_torch**  
  (integer(1))
  seed passed to `numpy$random$seed`. Default is same as `seed_R`. 

### simsurvdata

**Simulate Survival Data**

**Description**

Function for simulating survival data.

**Usage**

```r
simsurvdata(n = 100, trt = 2, age = 2, sex = 1.5, cens = 0.3)
```

**Arguments**

- `n` (integer(1))
  
  Number of samples

- `trt, age, sex` (numeric(1))
  
  Coefficients for covariates.

- `cens` (numeric(1))
  
  Proportion of censoring to be generated, cut-off time is then selected as the quantile that results in `cens`.

**Details**

Currently limited to three covariates, Weibull survival times, and Type I censoring. This will be expanded to a flexible simulation function in future updates. For now the function is primarily limited to helping function examples.

**Value**

- `data.frame()`

**Examples**

```r
simsurvdata()
```

---

### surv_to_risk

**Safely convert a survival matrix prediction to a relative risk**

**Description**

Many methods can be used to reduce a discrete survival distribution prediction (i.e. matrix) to a relative risk / ranking prediction. Here we define the predicted relative risk as the sum of the predicted cumulative hazard function - which can be loosely interpreted as the expected number of deaths for patients with similar characteristics.
**Usage**

```r
surv_to_risk(x)
```

**Arguments**

- `x` (matrix())
  
  TxN survival matrix prediction where T is number of time-points and N is number of predicted observations. Column names correspond to predicted time-points and should therefore be coercable to numeric and increasing. Entries are survival predictions and should be (non-strictly) decreasing in each row.

**References**

Index

akritas, 3
build_keras_net, 4, 18
build_pytorch_net, 6, 9, 12, 15, 25, 27, 32, 35
cindex, 7
coxtime, 8, 42
data.frame(), 44
deeplhit, 11, 42
deespsurv, 14
distr6::Matdist(), 37, 39, 41
dnnsurv, 16
get_keras_optimizer, 18, 19
get_pycox_activation, 6, 9, 12, 15, 21, 32, 35
get_pycox_callbacks, 10, 13, 15, 24, 32, 35
get_pycox_init, 7, 25
get_pycox_optim, 10, 13, 15, 26, 32, 35
install_keras, 28
install_pycox, 29
install_torch, 30
keras::callback_early_stopping, 18
keras::compile.keras.engine.training.Model, 18
keras::fit.keras.engine.training.Model, 18
keras::install_keras, 28
keras::layer_activation, 5
keras::layer_batch_normalization, 5
keras::layer_dense, 5
keras::optimizer_adadelta, 20
keras::optimizer_adagrad, 20
keras::optimizer_adam, 20
keras::optimizer_adamax, 20
keras::optimizer_nadam, 20
keras::optimizer_rmsprop, 20
keras::optimizer_sgd, 20
keras::predict.keras.engine.training.Model, 39
loghaz, 30, 42
pchazard, 33, 42
predict.akritas, 36
predict.dnnsurv, 38
predict.pycox, 40
pycox_prepare_train_data, 41
requireNamespace, 43
requireNamespaces, 43
reticulate::py_install, 29, 30
set.seed, 43
set_seed, 43
simsurvdata, 44
stats::model.matrix(), 3, 9, 12, 15, 17, 31, 34, 37, 39, 40
surv_to_risk, 44
survival::concordance, 7
survival::Surv(), 3, 9, 11, 14, 17, 31, 34
survivalmodels
(survivalmodels-package), 2
survivalmodels-package, 2