Package 'glmnetr'

February 9, 2024

Title Nested Cross Validation for the Relaxed Lasso and Other Machine Learning Models

Version 0.4-2

Date 2024-02-02

Depends R (>= 3.4.0)

Suggests R.rsp

VignetteBuilder R.rsp

Imports glmnet, survival, Matrix, xgboost, smoof, mlrMBO, ParamHelpers, randomForestSRC, rpart, torch

ByteCompile Yes

Author Walter K Kremers [aut, cre] (<https://orcid.org/0000-0001-5714-3473>), Nicholas B Larson [ctb]

Maintainer Walter K Kremers <kremers.walter@mayo.edu>

Description Cross validation informed Relaxed LASSO, Artificial Neural Network (ANN), gradient boosting machine ('xgboost'), Random Forest ('RandomForestSRC'), Recursive Partitioning ('RPART') or step wise regression models are fit. Nested cross validation (or analogous for the random forest) to estimate and compare performances between these models is used to describe model performances.
For some datasets, for example when the design matrix is not of full rank, 'glmnet' may have very long run times when fitting the relaxed lasso model, from our experience when fitting Cox models on data with many predictors and many patients, making it difficult to get solutions from either glmnet() or cv.glmnet(). This may be remedied with the 'path=TRUE' options when calling glmnet() and cv.glmnet(). Within the glmnetr package the approach of path=TRUE is taken by default.
When fitting not a relaxed lasso model but an elastic-net model, then the R-packages 'nestedcv' <https://cran.r-project.org/package=nestedcv>, 'glmnetSE' <https://cran.r-project.org/package=glmnetSE> or others may provide greater functionality when performing a nested CV.
As with the 'glmnet' package, this package passes most relevant output to the output object and tabular and graphical summaries can be generated using the summary and plot functions. Use of the 'glmnetr' has many similarities to the 'glmnet' package and it is recommended that the user of 'glmnetr' first become familiar with the 'glmnet' package <https://cran.r-project.org/package=glmnet>, with the ``An Introduction to 'glmnet'" and ``The Relaxed Lasso" being especially helpful in this regard.

License GPL-3

NeedsCompilation no

RoxygenNote 7.2.3

Encoding UTF-8

Repository CRAN

Date/Publication 2024-02-09 00:50:02 UTC

R topics documented:

`aicreg` *Identify model based upon AIC criteria from a stepreg() putput*

Description

Identify model based upon AIC criteria from a stepreg() putput

Usage

```
aicreg(
  xs,
  start,
  y_,
  event,
  steps_n = steps_n,
  family = family,
  object = NULL,
  track = 0
)
```

Arguments

`xs`	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
`start`	start time, Cox model only - class numeric of length same as number of patients (n)
`y_`	output vector: time, or stop time for Cox model, y_ 0 or 1 for binomial (logistic), numeric for gaussian. Must be a vector of length same as number of sample size.
`event`	event indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size.
`steps_n`	maximum number of steps done in stepwise regression fitting
`family`	model family, "cox", "binomial" or "gaussian"

object	A stepreg() output. If NULL it will be derived.
track	Indicate whether or not to update progress in the console. Default of 0 suppresses these updates. The option of 1 provides these updates. In fitting clinical data with non full rank design matrix we have found some R-packages to take a very long time or possibly get caught in infinite loops. Therefore we allow the user to track the package and judge whether things are moving forward or if the process should be stopped.

Value

The identified model in form of a glm() or coxph() output object, with an entry of the stepreg() output object.

Examples

```
set.seed(18306296)
sim.data=glmnetr.simdata(nrows=100, ncols=100, beta=c(0,1,1))
# this gives a more intersting case but takes longer to run
xs=sim.data$xs
# this will work numerically
xs=sim.data$xs[,c(2,3,50:55)]
y_=sim.data$yt
event=sim.data$event
cox.aic.fit = aicreg(xs, NULL, y_, event, family="cox", steps_n=40)
summary(cox.aic.fit)

y_=sim.data$yt
norm.aic.fit = aicreg(xs, NULL, y_, NULL, family="gaussian", steps_n=40)
summary(norm.aic.fit)
```

---

ann_tab_cv *Fit an Artificial Neural Network model on "tabular" provided as a matrix, optionally allowing for an offset term*

Description

Fit an Artificial Neural Network model for analysis of "tabular" data. The model has two hidden layers where the number of terms in each layer is configurable by the user. The activation function can also be switched between relu() (default) gelu() or sigmoid(). Optionally an offset term may be included. Model "family" may be "cox" to fit a generalization of the Cox proportional hazards model, "binomial" to fit a generalization of the logistic regression model and "gaussian" to fit a generalization of linear regression model for a quantitative response. See the corresponding vignette for examples.

Usage

```
ann_tab_cv(
  myxs,
  mystart = NULL,
  myy,
  myevent = NULL,
  myoffset = NULL,
  family = "binomial",
  fold_n = 5,
  epochs = 200,
  eppr = 40,
  lenz1 = 16,
  lenz2 = 8,
  actv = 1,
  drpot = 0,
  mylr = 0.005,
  wd = 0,
  l1 = 0,
  lasso = 0,
  lscale = 5,
  scale = 1,
  resetlw = 1,
  minloss = 1,
  gotoend = 0,
  seed = NULL,
  foldid = NULL
)
```

Arguments

myxs	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
mystart	an optional vector of start times in case of a Cox model. Class numeric of length same as number of patients (n)
myy	dependent variable as a vector: time, or stop time for Cox model, Y_ 0 or 1 for binomial (logistic), numeric for gaussian. Must be a vector of length same as number of sample size.
myevent	event indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size.
myoffset	an offset term to be used when fitting the ANN. Not yet implemented in its pure form. Functionally an offset can be included in the first column of the predictor or feature matrix myxs and indicated as such using the lasso option.
family	model family, "cox", "binomial" or "gaussian" (default)
fold_n	number of folds for each level of cross validation
epochs	number of epochs to run when tuning on number of epochs for fitting final model number of epochs informed by cross validation

eppr	for EPoch PRint. print summary info every eppr epochs. 0 will print first and last epochs, 0 for first and last epoch, -1 for minimal and -2 for none.
lenz1	length of the first hidden layer in the neural network, default 16
lenz2	length of the second hidden layer in the neural network, default 16
actv	for ACTiVation function. Activation function between layers, 1 for relu, 2 for gelu, 3 for sigmoid.
drpot	fraction of weights to randomly zero out. NOT YET implemented.
mylr	learning rate for the optimization step in the neural network model fit
wd	a possible weight decay for the model fit, default 0 for not considered
l1	a possible L1 penalty weight for the model fit, default 0 for not considered
lasso	1 to indicate the first column of the input matrix is an offset term, often derived from a lasso model, else 0 (default)
lscale	Scale used to allow ReLU to exend +/- lscale before capping the inputted linear estimated
scale	Scale used to transform the inital random paramter assingments by dividing by scale
resetlw	1 as default to re-adjust weights to account for the offset every epoch. This is only used in case lasso is set to 1.
minloss	default of 1 for minimizing loss, else maximizing agreement (concordance for Cox and Binomial, R-square for Gaussian), as function of epochs by cross validaition
gotoend	fit to the end of epochs. Good for plotting and exploration
seed	an optional a numerical/integer vector of length 2, for R and torch random generators, default NULL to generate these. Integers should be positive and not more than 2147483647.
foldid	a vector of integers to associate each record to a fold. Should be integers from 1 and fold_n.

Value

an artificial neural network model fit

See Also

ann_tab_cv_best , summary.nested.glmnetr , glmnetr.compcv , glmnetr.simdata

ann_tab_cv_best *Fit multiple Artificial Neural Network models on "tabular" provided as a matrix, and keep the best one.*

Description

Fit an multiple Artificial Neural Network models for analysis of "tabular" data using ann_tab_cv() and select the best fitting model according to cross validaiton.

Usage

```
ann_tab_cv_best(
  myxs,
  mystart = NULL,
  myy,
  myevent = NULL,
  myoffset = NULL,
  family = "binomial",
  fold_n = 5,
  epochs = 200,
  eppr = 40,
  lenz1 = 32,
  lenz2 = 8,
  actv = 1,
  drpot = 0,
  mylr = 0.005,
  wd = 0,
  l1 = 0,
  lasso = 0,
  lscale = 5,
  scale = 1,
  resetlw = 1,
  minloss = 1,
  gotoend = 0,
  bestof = 10,
  seed = NULL,
  foldid = NULL
)
```

Arguments

myxs	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
mystart	an optional vector of start times in case of a Cox model. Class numeric of length same as number of patients (n)

myy	dependent variable as a vector: time, or stop time for Cox model, Y_ 0 or 1 for binomial (logistic), numeric for gaussian. Must be a vector of length same as number of sample size.
myevent	event indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size.
myoffset	an offset term to be ues when fitting the ANN. Not yet implemented.
family	model family, "cox", "binomial" or "gaussian" (default)
fold_n	number of folds for each level of cross validation
epochs	number of epochs to run when tuning on number of epochs for fitting final model number of epochs informed by cross validation
eppr	for EPoch PRint. print summry info every eppr epochs. 0 will print first and last epochs, -1 nothing.
lenz1	length of the first hidden layer in the neural network, default 16
lenz2	length of the second hidden layer in the neural network, default 16
actv	for ACTiVation function. Activation function between layers, 1 for relu, 2 for gelu, 3 for sigmoid.
drpot	fraction of weights to randomly zero out. NOT YET implemented.
mylr	learning rate for the optimzaiton step in teh neural network model fit
wd	weight decay for the model fit.
l1	a possible L1 penalty weight for the model fit, default 0 for not considered
lasso	1 to indicate the first collumn of the input matrix is an offset term, often derived from a lasso model
lscale	Scale used to allow ReLU to exend +/- lscale before capping the inputted linear estimated
scale	Scale used to transform the inital random paramter assingments by dividing by scale
resetlw	1 as default to re-adjust weights to account for the offset every epoch. This is only used in case lasso is set to 1
minloss	default of 1 for minimizing loss, else maximizing agreement (concordance for Cox and Binomial, R-square for Gaussian), as function of epochs by cross validaition
gotoend	fit to the end of epochs. Good for plotting and exploration
bestof	how many models to run, from which the best fitting model will be selected.
seed	an optional a numerical/integer vector of length 2, for R and torch random generators, default NULL to generate these. Integers should be positive and not more than 2147483647.
foldid	a vector of integers to associate each record to a fold. Should be integers from 1 and fold_n.

Value

an artifical neural network model fit

See Also

ann_tab_cv , nested.glmnetr , summary.nested.glmnetr , glmnetr.compcv , glmnetr.simdata

---

`best.preds` *Get the best models for the steps of a stepreg() fit*

Description

Get the best models for the steps of a stepreg() fit

Usage

```
best.preds(modsum, risklist)
```

Arguments

`modsum`	model summmary
`risklist`	riskset list

Value

best predictors at each step of a stepwise regerssion

`bsint` *Construct the bias terms for going from model layer to layer to carry forward an offset to mimic a linear model*

Description

Construct the bias terms for going from model layer to layer to carry forward an offset to mimic a linear model

Usage

```
bsint(tnsr, lasso = 0, rreturn = 1)
```

Arguments

`tnsr`	an input tensor which is to be modified to mimic the linear term of a generalized linear model, e.g a Cox or logistic regression model
`lasso`	1 if the first column is the linear estimate from a linear model, often a lasso model
`rreturn`	1 (default) to return an R (numeric) vector, 0 to return a torch tensor

Value

a weight matrix in tensor format

calceloss *calculate cross-entry for multinomial outcomes*

Description

calculate cross-entry for multinomial outcomes

Usage

```
calceloss(xx, yy)
```

Arguments

xx	the sigmoid of the link, i.e, the estimated probabilities, i.e. xx = 1/(1+exp(-xb))
yy	the observed data as 0's and 1's

Value

the cross-entropy on a per observation basis

cox.sat.dev *Calculate the CoxPH saturated log-likelihood*

Description

Calculate the saturated log-likelihood for the Cox model using both the Efron and Breslow approximations for the case where all ties at a common event time have the same weights (exp(X*B)). For the simple case without ties the saturated log-likelihood is 0 as the contribution to the log-likelihood at each event time point can be made arbitrarily close to 1 by assigning a much larger weight to the record with an event. Similarly, in the case of ties one can assign a much larger weight to be associated with one of the event times such that the associated record contributes a 1 to the likelihood. Next one can assign a very large weight to a second tie, but smaller than the first tie considered, and this too will contribute a 1 to the likelihood. Continuing in this way for this and all time points with ties, the partial log-likelihood is 0, just like for the no-ties case. Note, this is the same argument with which we derive the log-likelihood of 0 for the no ties case. Still, to be consistent with others we derive the saturated log-likelihood with ties under the constraint that all ties at each event time carry the same weights.

Usage

```
cox.sat.dev(y_, e_)
```

Arguments

y_	Time variable for a survival analysis, whether or not there is a start time
e_	Event indicator with 1 for event 0 otherwise.

Value

Saturated log likelihood for the Efron and Breslow approximations.

`cv.glmnetr` *Get a cross validation informed relaxed lasso model fit.*

Description

Derive a relaxed lasso model and identifies hyperparameters, i.e. lambda and gamma, which give the best bit using cross validation. It is analogous to the cv.glmnet() function of the 'glmnet' package, but handles cases where glmnet() may run slowly when using the relaxed=TRUE option.

Usage

```
cv.glmnetr(
  xs,
  start = NULL,
  y_,
  event = NULL,
  family = "gaussian",
  lambda = NULL,
  gamma = c(0, 0.25, 0.5, 0.75, 1),
  folds_n = 10,
  limit = 2,
  fine = 0,
  track = 0,
  seed = NULL,
  foldid = NULL,
  ties = "efron",
  stratified = 1,
  time = NULL,
  ...
)
```

Arguments

`xs`	predictor matrix
`start`	vector of start times or the Cox model. Should be NULL for other models.
`y_`	outcome vector
`event`	event vector in case of the Cox model. May be NULL for other models.
`family`	model family, "cox", "binomial" or "gaussian" (default)
`lambda`	the lambda vector. May be NULL.
`gamma`	the gamma vector. Default is c(0,0.25,0.50,0.75,1).
`folds_n`	number of folds for cross validation. Default and generally recommended is 10.

limit	limit the small values for lambda after the initial fit. This will eliminate calculations that have small or minimal impact on the cross validation. Default is 2 for moderate limitation, 1 for less limitation, 0 for none.
fine	use a finer step in determining lambda. Of little value unless one repeats the cross validation many times to more finely tune the hyperparameters. See the 'glmnet' package documentation.
track	indicate whether or not to update progress in the console. Default of 0 suppresses these updates. The option of 1 provides these updates. In fitting clinical data with non full rank design matrix we have found some R-packages to take a vary long time or seemingly be caught in infinite loops. Therefore we allow the user to track the program progress and judge whether things are moving forward or if the process should be stopped.
seed	a seed for set.seed() so one can reproduce the model fit. If NULL the program will generate a random seed. Whether specified or NULL, the seed is stored in the output object for future reference. Note, for the default this randomly generated seed depends on the seed in memory at that time so will depend on any calls of set.seed prior to the call of this function.
foldid	a vector of integers to associate each record to a fold. The integers should be between 1 and folds_n.
ties	method for handling ties in Cox model for relaxed model component. Default is "efron", optionally "breslow". For penalized fits "breslow" is always used as in the 'glmnet' package.
stratified	folds are to be constructed stratified on an indicator outcome 1 (default) for yes, 0 for no. Pertains to event variable for "cox" and y_ for "binomial" family.
time	track progress by printing to console elapsed and split times. Suggested to use track option instead as time options will be eliminated.
...	Additional arguments that can be passed to glmnet()

Details

This is the main program for model derivation. As currently implemented the package requires the data to be input as vectors and matrices with no missing values (NA). All data vectors and matrices must be numerical. For factors (categorical variables) one should first construct corresponding numerical variables to represent the factor levels. To take advantage of the lasso model, one can use one hot coding assigning an indicator for each level of each categorical variable, or creating as well other contrasts variables suggested by the subject matter.

Value

A cross validation informed relaxed lasso model fit.

Author(s)

Walter Kremers (kremers.walter@mayo.edu)

See Also

glmnetr , nested.glmnetr , glmnetr.simdata

Examples

```
# set seed for random numbers, optionally, to get reproducible results
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=100, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$y_
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
cv.glmnetr.fit = cv.glmnetr(xs, NULL, y_, NULL, family="gaussian", folds_n=3, limit=2)
plot(cv.glmnetr.fit)
plot(cv.glmnetr.fit, coefs=1)
summary(cv.glmnetr.fit)
```

---

cv.stepreg *Cross validation informed stepwise regression model fit.*

Description

Cross validation informed stepwise regression model fit.

Usage

```
cv.stepreg(
  xs_cv,
  start_cv = NULL,
  y_cv,
  event_cv,
  family = "cox",
  steps_n = 0,
  folds_n = 10,
  method = "loglik",
  seed = NULL,
  foldid = NULL,
  stratified = 1,
  track = 0
)
```

Arguments

xs_cv	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
start_cv	start time, Cox model only - class numeric of length same as number of patients (n)
y_cv	output vector: time, or stop time for Cox model, Y_ 0 or 1 for binomal (logistic), numeric for gaussian. #' Must be a vector of length same as number of sample size.

event_cv	event indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size.
family	model family, "cox", "binomial" or "gaussian"
steps_n	Maximun number of steps done in stepwise regression fitting. If 0, then takes the value rank(xs_cv).
folds_n	number of folds for cross validation
method	method for choosing model in stepwise procedure, "loglik" or "concordance". Other procedures use the "loglik".
seed	a seed for set.seed() to assure one can get the same results twice. If NULL the program will generate a random seed. Whether specified or NULL, the seed is stored in the output object for future reference.
foldid	a vector of integers to associate each record to a fold. The integers should be between 1 and folds_n.
stratified	folds are to be constructed stratified on an indicator outcome 1 (default) for yes, 0 for no. Pertains to event variable for "cox" and y_ for "binomial" family.
track	indicate whether or not to update progress in the console. Default of 0 suppresses these updates. The option of 1 provides these updates. In fitting clinical data with non full rank design matrix we have found some R-packages to take a very long time. Therefore we allow the user to track the program progress and judge whether things are moving forward or if the process should be stopped.

Value

cross validation infomred stepwise regression model fit tuned by number of model terms or p-value for inclusion.

Examples

```
set.seed(955702213)
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=c(0,1,1))
# this gives a more interesting case but takes longer to run
xs=sim.data$xs
# this will work numerically as an example
xs=sim.data$xs[,c(2,3,50:55)]
dim(xs)
y_=sim.data$yt
event=sim.data$event
# for this example we use small numbers for steps_n and folds_n to shorten run time
cv.stepreg.fit = cv.stepreg(xs, NULL, y_, event, steps_n=10, folds_n=3, track=0)
summary(cv.stepreg.fit)
```

diff_time *Output to console the elapsed and split times*

Description

Output to console the elapsed and split times

Usage

```
diff_time(time_start = NULL, time_last = NULL)
```

Arguments

time_start	beginning time for printing elapsed time
time_last	last time for calculating split time

Value

Time of program invocation

Examples

```
time_start = diff_time()
time_last = diff_time(time_start)
time_last = diff_time(time_start,time_last)
time_last = diff_time(time_start,time_last)
```

diff_time1 *Get elapsed time in c(hour, minute, secs)*

Description

Get elapsed time in c(hour, minute, secs)

Usage

```
diff_time1(time1, time2)
```

Arguments

time1	start time
time2	stop time

Value

Returns a vector of elapsed time in (hour, minute, secs)

dtstndrz *Standardize a data set*

Description

Standardize a data set

Usage

```
dtstndrz(datain, lasso = 0)
```

Arguments

datain	The data matrix set to be standardized
lasso	1 to not standardize the first column, 0 (default) to not

Value

a standarized data matrix

factor.foldid *Generate foldid's by factor levels*

Description

Generate foldid's by factor levels

Usage

```
factor.foldid(event, fold_n = 10)
```

Arguments

event	the outcome variable in a vector identifying the different potential levels of the outcome
fold_n	the numbe of folds to be constructed

Value

foldid's in a vector the same length as event

get.foldid *Get foldid's with branchning for cox, binomial and gaussian models*

Description

Get foldid's with branchning for cox, binomial and gaussian models

Usage

```
get.foldid(y_, event, family, folds_n, stratified = 1)
```

Arguments

y_	see help for cv.glmnetr() or nested.glmnetr()
event	y_ see help for cv.glmnetr() or nested.glmnetr()
family	y_ see help for cv.glmnetr() or nested.glmnetr()
folds_n	y_ see help for cv.glmnetr() or nested.glmnetr()
stratified	y_ see help for cv.glmnetr() or nested.glmnetr()

Value

A numeric vector with foldid's for use in a cross validation

Author(s)

Walter Kremers (kremers.walter@mayo.edu)

See Also

cv.glmnetr , nested.glmnetr , factor.foldid

getlamgam *get numerical values for lam and gam*

Description

This functin derives the numerical values for lam and gam (lambda and gamma) for usage in plot and predict() functions. If the input variables lam and gam are unspecified then the cross validation informed lambda and gamma values ('lambda.min' and 'gamma.mim) which minimize the cross validation deviance, are returned. One may also give lam='lambda.1se' and gam='gamma.1se' to identify the corresponding numerical values. Importantly one may also simply specify gam=1 (and lam=NULL) to get the best lasso fit from the unrelaxed lasso model (gamma=1), or gam=0 (and lam=NULL) to get the best lasso fit from the fully relaxed lasso model (gamma=0).

Usage

```
getlamgam(object, lam, gam, comment)
```

Arguments

object	glmnetr object as input
lam	value for lam, may be NULL, typically NULL.
gam	value for gam, may be MULL, typically NULL, 0 or 1.
comment	Default of TRUE to write to console information on lam and gam selected for output. FALSE will suppress this write to console.

Value

numerical values for lam and gam for use in plot(), predict() and summary() functions

---

glmnetr *Fit relaxed part of lasso model*

Description

Derive the relaxed lasso fits and optionally calls glmnet() to derive the fully penalized lasso fit.

Usage

```
glmnetr(
  xs_tmp,
  start_tmp,
  y_tmp,
  event_tmp,
  family = "cox",
  lambda = NULL,
  gamma = c(0, 0.25, 0.5, 0.75, 1),
  object = NULL,
  track = 0,
  ties = "efron",
  time = NULL,
  ...
)
```

Arguments

xs_tmp	predictor (X) matrix
start_tmp	start time in case Cox model and (Start, Stop) time for use in model
y_tmp	outcome (Y) variable, in case of Cox model (stop) time
event_tmp	event variable in case of Cox model

family	model family, "cox", "binomial" or "gaussian" (default)
lambda	lambda vector, as in glmnet(), default is NULL
gamma	gamma vector, as with glmnet(), default c(0,0.25,0.50,0.75,1)
object	an output object from glmnet() using relax=FALSE with the model fits for the fully penalized lasso models, i.e. gamma=1. Default is NULL in which case these are derived within the function.
track	Indicate whether or not to update progress in the console. Default of 0 suppresses these updates. The option of 1 provides these updates. In fitting clinical data with non full rank design matrix we have found some R-packages to take a vary long time or possibly get caught in infinite loops. Therefore we allow the user to track the package and judge whether things are moving forward or if the process should be stopped.
ties	method for handling ties in Cox model for relaxed model component. Default is "efron", optionally "breslow". For penalized fits "breslow" is always used as in the 'glmnet' package.
time	track progress by printing to console elapsed and split times. Suggested to use track option instead as time options will be eliminated.
...	Additional arguments that can be passed to glmnet()

Value

A list with two matrices, one for the model coefficients with gamma=1 and the other with gamma=0.

See Also

cv.glmnetr , nested.glmnetr , glmnetr.simdata

Examples

```
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=200, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
glmnetr.fit = glmnetr( xs, NULL, y_, event, family="cox")
plot(glmnetr.fit)
```

`glmnetr.compcv` *Compare cross validation fits from a nested.glmnetr output.*

Description

Compare cross-validation model fits in terms of average concordance from the nested cross validaiton fits.

Usage

```
glmnetr.compcv(object, digits = 4, pow = 1)
```

Arguments

`object`	A nested.glmnetr output object.
`digits`	digits for printing of z-scores, p-values, etc. with default of 4
`pow`	the power to which the average of correlations is to be raised. Only applies to the "gaussian" model. Default is 2 to yield R-square but can be on to show correlations. pow is ignored for the family of "cox" and "binomial".

Value

A printout to the R console.

See Also

`summary.nested.glmnetr`

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
fit3 = nested.glmnetr(xs, NULL, y_, event, family="cox", folds_n=3)
glmnetr.compcv(fit3)
```

glmnetr.compcv0 *Calculate agreement differences with CI and p*

Description

Perform a paired t-test as called from glmnetr.compcv().

Usage

```
glmnetr.compcv0(a, b, digits = 4, txt = 0, pow = 1)
```

Arguments

a	One term
b	A second term
digits	digits for printing of z-scores, p-values, etc. with default of 4
txt	1 (default) to include inline text for estimated, 95 percent CI and p
pow	Power to which the average of correlations is to be raised. Only applies to the "gaussian" model. Default is 2 to yield R-square but can be on to show correlations. Pow is ignored for the family of "cox" and "binomial".

Value

An estimate, 95

glmnetr.simdata *Generate example data*

Description

Generate an example data set with specified number of observations, and predictors. The first column in the design matrix is identically equal to 1 for an intercept. Columns 2 to 5 are for the 4 levels of a character variable, 6 to 11 for the 6 levels of another character variable. Columns 12 to 17 are for 3 binomial predictors, again over parameterized. Such over parameterization can cause difficulties with the glmnet() of the 'glmnet' package.

Usage

```
glmnetr.simdata(nrows = 1000, ncols = 100, beta = NULL, intr = NULL)
```

Arguments

nrows	Sample size (>=100) for simulated data, default=1000.
ncols	Number of columns (>=17) in design matrix, i.e. predictors, default=100.
beta	Vector of length <= ncols for "left most" coefficients. If beta has length < ncols, then the values at length(beta)+1 to ncols are set to 0. Default=NULL, where a beta of length 25 is assigned standard normal values.
intr	either NULL for no interactions or a vector of length 3 to impose a product effect as decribed by intr[1]*xs[,3]*xs[,8] + intr[2]*xs[,4]*xs[,16] + intr[3]*xs[,18]*xs[,19] + intr[4]*xs[,21]*xs[,22]

Value

A list with elements xs for desing matrix, y_ for a quantitative outcome, yt for a survival time, event for an indicator of event (1) or censoring (0), in the Cox proportional hazards survival model setting, yb for yes/no (binomial) outcome data, and beta the beta used in random number generation.

See Also

glmnetr , cv.glmnetr , nested.glmnetr

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
# for Cox PH survial model data
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
# for linear regression model data
xs=sim.data$xs
y_=sim.data$y_
# for logistic regression model data
xs=sim.data$xs
y_=sim.data$yb
```

---

glmnetrll_1fold *Evaluate fit of leave out fold*

Description

Derive the log likelihood for a leave out based upon the fit of the input object.

Usage

```
glmnetrll_1fold(
  object,
  xs_new,
  start_new,
  y_new,
  event_new,
  family = "cox",
  lambda_n = NULL,
  gamma = c(0, 0.25, 0.5, 0.75, 1),
  ties = "efron"
)
```

Arguments

object	an output object from _cv.glmnetr_
xs_new	A new predictor matrix
start_new	A new vector of start times or the Cox model. May be NULL.
y_new	a new outcome vector.
event_new	event vector in case of the Cox model. May be NULL for other models.
family	Model family, one of "cox", "gaussian" or "binomial".
lambda_n	length of the lambda vector.
gamma	The gamma vector.
ties	method for handling ties in Cox model for relaxed model component. Default is "efron", optionally "breslow". For penalized fits "breslow" is always used as in the 'glmnet' package.

Value

Returns the log likelihood of object fit using new data.

---

glmnetr_devratio *Get Deviance ratio.*

Description

fit models to derive the devaince ratios.

Usage

```
glmnetr_devratio(
  object,
  object2,
  xs_new,
  start_new,
  y_new,
  event_new,
  family,
  ties = "efron"
)
```

Arguments

object	a glmnet() output object with relax=FALSE, i.e model fit for gamma=1.
object2	a glmnetr() output object with relaxed fits, i.e model fit for gamma=0.
xs_new	predictor matrix
start_new	start times in case of usage in Cox model. Else should be NULL.
y_new	outcome vector.
event_new	event indicator in case of Cox model. Else should be NULL.
family	model family, one of "cox", "gaussian" or "binomial".
ties	method for handling ties in Cox model for relaxed model component. Default is "efron", optionally "breslow". For penalized fits "breslow" is always used as in the 'glmnet' package.

Value

- Deviance ratios.

---

nested.glmnetr *Using nested cross validation, describe and compare fits of various cross validation informed machine learning models.*

Description

Performs a nested cross validation for cross validation informed relaxed lasso, (artificial) Neural Network (ANN) with two hidden layers, Gradient Boosting Machine (GBM), Recursive Partitioning (RPART) and step wise regression. That is hyper parameters for all these models are informed by cross validation (CV), and a second layer of CV is used to evaluate the performance of these CV informed model fits. For step wise regression CV is used to inform either ap value for entry or degress of freedom (df) for the final model shoice. For input we require predictors (features) to be in numeric matrix format with no missing values. This is similar to how the glmnet package expects predictors. For survival data we allow input of start time as an option, and require stop time, and an event indicator, 1 for event and 0 for censoring, as separate terms. This may seem

unorthodox as it might seem simpler to accept a Surv() object as input. However the XGBoost routines require a different data format with only a "stop time" variable, taking a positive value to indicate being associated with an event, and the negative of the time when associated with a censoring. Further with XGBoost we combine both the predictors and outcome (or label) in an xgb.DMatrix() object. Further the evaluation of the loss function when fitting a neural network model also requires the individual vectors for time to event and the event indicator and not a Surv() object. Instead of translating between these formats we take as input the individual data elements and construct whatever object is needed for the particular model.

Usage

```
nested.glmnetr(
  xs,
  start = NULL,
  y_,
  event = NULL,
  family = "gaussian",
  do_ncv = 1,
  folds_n = 10,
  stratified = 1,
  dolasso = 1,
  doxgb = 0,
  dorf = 0,
  dorpart = 0,
  doann = 0,
  dostep = 0,
  doaic = 0,
  ensemble = 0,
  method = "loglik",
  lambda = NULL,
  gamma = NULL,
  relax = TRUE,
  steps_n = 0,
  seed = NULL,
  foldid = NULL,
  limit = 1,
  fine = 0,
  ties = "efron",
  track = 0,
  ...
)
```

Arguments

xs	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
start	optional start times in case of a Cox model. A numeric (vector) of length same as number of patients (n). Optionally start may be specified as a column matrix

in which case the colname value is used when outputing summaries.

`y_` dependent variable as a vector: time, or stop time for Cox model, Y_ 0 or 1 for binomal (logistic), numeric for gaussian. Must be a vector of length same as number of sample size. Optionally y_ may be specified as a column matrix in which case the colname value is used when outputing summaries.

`event` event indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size. Optionally event may be specified as a column matrix in which case the colname value is used when outputing summaries.

`family` model family, "cox", "binomial" or "gaussian" (default)

`do_ncv` 1 by default to do the Nested Cross Validation, or 0 to only fit the various models without doing the Nested part. In this case the nested.glmnetr() function will only derive the models based upon the full data set. This may be useful when exploring various models without having to the Nested Cross Validation for assessing model performance. for example when wanting to examine a "lasso informed" extreme gradient boosting models (GBM) or Artificial Neural Network (ANN) models which are based upon both a lasso fit and a GBM or ANN fit. See the predict_ann_tab() function regarding getting predicteds for this "lasso informed" models from a nested.glmnetr() output.

`folds_n` the number of folds for the outer loop of the nested cross validation, and if not overridden by the individual model specifications, also the number of folds for the inner loop of the nested cross validation, i.e. the number of folds used in model derivation.

`stratified` 1 to generate fold IDs stratified on outcome or event indicators for the binomial or Cox model.

`dolasso` fit and do cross validation for lasso model, 0 or 1

`doxgb` fit and evaluate a cross validation informed XGBoost (GBM) model. 1 for yes, 0 for no (default). By default the number of folds used when training the GBM model will be the same as the number of folds used in the outer loop of the nested cross validation, and the maximum number of rounds when training the GBM model is set to 1000. To control these values one may specify a list for the doxgb argument. The list can have elements $nfold, $folds and $nrounds which specify the number of folds, a list with fold ids and the maximum number of rounds when training the GBM model. Here we use nomenclature nomenclature used elsewhere in the package to be able to use terms those used in the 'xgboost' package, e.g. nfold instead of folds_n and folds instead of foldid. See xgb.cv() help for more information on these three options. If to shorten run time the user sets nfold to a value other than folds_n we recommend that nfold = folds_n/2 or folds_n/3. Then the folds will be formed by collapsing the folds_n folds allowing a better comparisons of model performances between the different machine learning models.

`dorf` fit and evaluate a random forest (RF) model. 1 for yes, 0 for no (default). Also, if dorf is specified by a list, then RF models will be fit. The randomForestSRC package is used. This list can have three elements. One is the vector mtryc, and contains values for mtry. The program searches over the different values to find a better fir for the final model. If not specified mtryc is set to round(

sqrt(dim(xs)[2]) * c(0.67 , 1, 1.5, 2.25, 3.375)). The second list element the vector ntreec. The first item (ntreec[1]) specifies the number of trees to fit in evaluating the models specified by the different mtry values. The second item (ntreec[2]) specifies the number of trees to fit in the final model. The default is ntreec = c(25,250). The third element in the list is the numeric varaible keep, with the value 1 to store the models fits from the search or the value 0 (default) to not store these model fits. Random forests use the out-of-bag (OOB) data elements fro assessing model fit and hyperparameter tuning and so cross validation is not used for tuning. Still, because of the number of trees in the forest random forest can take long to run.

`dorpart` fit and do a nested cross validation for an RPART model. As rpart() does its own approximation for cross validation there is no new functions for cross validation.

`doann` fit and evaluate a cross validation informed Artificial Neural Network (ANN) model with two hidden levels. 1 for yes, 0 for no (default). By default the number of folds used when training the ANN model will be the same as the number of folds used in the outer loop of the nested cross validation. To override this, for example to shrtn run time, one may specify a list for the doann argument where the element $folds_ann_n gives the number of folds used when training the ANN. To shorten run we recommend folds_ann_n = folds_n/2 or folds_n/3, and at least 3. Then the folds will be formed by collapsing the folds_n folds using in fitting other models allowing a better comparisons of model performances between the different machine learning models. The list can also have elements $epochs, $epochs2, $myler, $myler2, $eppr, $eppr2, $lenv1, $lenz2, $actv, $drpot, $wd, wd2, l1, l12, $lscale, $scale, $minloss and $gotoend. These arguments are then passed to the ann_tab_cv_best() function, with the meanings described in the help for that function, with some exception. When there are two similar values like $epoch and $epoch2 the first applies to the ANN models trained without transfer learning and the second to the models trained with transfer learning from the lasso model. Elements of this list unspecified will take default values. The user may also specify the element $bestof (a positive integer) to fit bestof models with different random starting weights and biases while taking the best performing of the different fits based upon CV as the final model. The default value for bestof is 1.

`dostep` fit and do cross validation for stepwise regression fit, 0 or 1, as discussed in James, Witten, Hastie and Tibshirani, 2nd edition.

`doaic` fit and do cross validation for AIC fit, 0 or 1. This is provided primarily as a reference.

`ensemble` This is a vector 8 characters long and specifies a set of ensemble like model to be fit based upon the predicteds form a relaxed lasso model fit, by either inlcuding the predicteds as an additional term (feature) in the machine learning model, or including the predicteds similar to an offset. For XGBoost, the offset is specified in the model with the "base_margin" in the XGBoost call. For the Artificial Neural Network models fit using the ann_tab_cv_best() function, one can initialize model weights (parameters) to account for the predicteds in prediction and either let these weights by modified each epoch or update and maintain these weights during the fitting process. For ensemble[1] = 1 a model is fit ignoring these predicteds, ensemble[2]=1 a model is fit including the predicteds as an additional feature. For ensemble[3]=1 a model is fit using the

	predicteds as an offset when running the xgboost model, or a model is fit including the predicteds with initial weights corresponding to an offset, but then weights are allowed to be tuned over the epochs. For i >= 4 ensemble[i] only applies to the neural network models. For ensemble[4]=1 a model is fit like for ensemble[3]=1 but the weights are reassigned to correspond to an offset after each epoch. For i in (5,6,7,8) ensemble[i] is similar to ensemble[i-4] except the original predictor (feature) set is replaced by the set of non-zero terms in the relaxed lasso model fit. If ensemble is specified as 0 or NULL, then ensemble is assigned c(1,0,0,0, 0,0,0,0). If ensemble is specified as 1, then ensemble is assigned c(1,1,1,1, 1,1,1,1).
`method`	method for choosing model in stepwise procedure, "loglik" or "concordance". Other procedures use the "loglik".
`lambda`	lambda vector for the lasso fit
`gamma`	gamma vector for the relaxed lasso fit, default is c(0,0.25,0.5,0.75,1)
`relax`	fit the relaxed lasso model when fitting a lasso model
`steps_n`	number of steps done in stepwise regression fitting
`seed`	optional, either NULL, or a numerical/integer vector of length 2, for R and torch random generators, or a list with two two vectors, each of length folds_n+1, for generation of random folds of the outer cross validation loop, and the remaining folds_n terms for the random generation of the folds or the bootstrap samples for the model fits of the inner loops. This can be used to replicate model fits. Whether specified or NULL, the seed is stored in the output object for future reference. The stored seed is a list with two vectors seedr for the seeds used in generating the random fold splits, and seedt for generating the random initial weights and biases in the torch neural network models. The first element in each of these vectors is for the all data fits and remaining elements for the folds of the inner cross validation. The integers assigned to seed should be positive and not more than 2147483647.
`foldid`	a vector of integers to associate each record to a fold. Should be integers from 1 and folds_n. These will only be used in the outer folds.
`limit`	limit the small values for lambda after the initial fit. This will have minimal impact on the cross validation. Default is 2 for moderate limitation, 1 for less limitation, 0 for none.
`fine`	use a finer step in determining lambda. Of little value unless one repeats the cross validation many times to more finely tune the hyper paramters. See the 'glmnet' package documentation
`ties`	method for handling ties in Cox model for relaxed model component. Default is "efron", optionally "breslow". For penalized fits "breslow" is always used as derived form to 'glmnet' package.
`track`	track progress by printing to console elapsed and split times
`...`	additional arguments that can be passed to glmnet()

Value

- Cross validation informed LASSO, GBM, RPART or STEPWISE model fits, together with estimates of model performance derived using nested cross validation.

See Also

glmnetr , cv.glmnetr , glmnetr.simdata , summary.nested.glmnetr , glmnetr.compcv , plot.nested.glmnetr , predict_ann_tab

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$y_
# for this example we use a small number for folds_n to shorten run time
nested.glmnetr.fit = nested.glmnetr( xs, NULL, y_, NULL, family="gaussian", folds_n=3)
plot(nested.glmnetr.fit)
plot(nested.glmnetr.fit, coefs=TRUE)
summary(nested.glmnetr.fit)
summary(nested.glmnetr.fit, cvfit=TRUE)
```

---

plot.cv.glmnetr *Plot cross-validation deviances, or model coefficients.*

Description

By default, with coefs=FALSE, plots the average deviances as function of lam (lambda) and gam (gamma), and also indicates the gam and lam which minimize deviance based upon a cv.glmnetr() output object. Optionally, with coefs=TRUE, plots the relaxed lasso coefficients.

Usage

```
## S3 method for class 'cv.glmnetr'
plot(
  x,
  gam = NULL,
  lambda.lo = NULL,
  plup = 0,
  title = NULL,
  coefs = FALSE,
  comment = TRUE,
  ...
)
```

Arguments

x	a cv.glmnetr() output object.
gam	a specific level of gamma for plotting. By default gamma.min will be used.
lambda.lo	a lower limit of lambda when plotting.

plup	an indicator to plot the upper 95 percent two-sided confidence limits.
title	a title for the plot.
coefs	default of FALSE plots deviances, option of TRUE plots coefficients.
comment	default of TRUE to write to console information on lam and gam selected for output. FALSE will suppress this write to console.
...	Additional arguments passed to the plot function.

Value

This program returns a plot to the graphics window, and may provide some numerical information to the R Console. If gam is not specified, then then the gamma.min from the deviance minimizing (lambda.min, gamma.min) pair will be used, and the corresponding lambda.min will be indicated by a vertical line, and the lambda minimizing deviance under the restricted set of models where gamma=0 will be indicated by a second vertical line.

See Also

plot.glmnetr , plot.nested.glmnetr , cv.glmnetr

Examples

```
# set seed for random numbers, optionally, to get reproducible results
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=100, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$y_
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
cv_glmnetr_fit = cv.glmnetr(xs, NULL, y_, NULL, family="gaussian", folds_n=3, limit=2)
plot(cv_glmnetr_fit)
plot(cv_glmnetr_fit, coefs=1)
```

---

plot.glmnetr *Plot the relaxed lasso coefficients.*

Description

Plot the relaxed lasso coefficients from either a glmnetr(), cv.glmnetr() or nested.glmnetr() output object. One may specify gam, single value for gamma. If gam is unspecified (NULL), then cv.glmnetr and nested.glmnetr() will use the gam which minimizes loss, and glmentr() will use gam=1.

Usage

```
## S3 method for class 'glmnetr'
plot(x, gam = NULL, lambda.lo = NULL, title = NULL, comment = TRUE, ...)
```

Arguments

x	Either a glmnetr, cv.glmnetr or a nested.glmnetr output object.
gam	A specific level of gamma for plotting. By default gamma.min from the deviance minimizing (lambda.min, gamma.min) pair will be used.
lambda.lo	A lower limit of lambda for plotting.
title	A title for the plot
comment	Default of TRUE to write to console information on lam and gam selected for output. FALSE will suppress this write to console.
...	Additional arguments passed to the plot function.

Value

This program returns a plot to the graphics window, and may provide some numerical information to the R Console. If the input object is from a nested.glmnetr or cv.glmnetr object, and gamma is not specified, then the gamma.min from the deviance minimizing (lambda.min, gamma.min) pair will be used, and the minimizing lambda.min will be indicated by a vertical line. Also, if one specifies gam=0, the lambda which minimizes deviance for the restricted set of models where gamma=0 will indicated by a vertical line.

See Also

plot.cv.glmnetr , plot.nested.glmnetr , glmnetr

Examples

```
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=200, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
glmnetr.fit = glmnetr( xs, NULL, y_, event, family="cox")
plot(glmnetr.fit)
```

---

plot.nested.glmnetr *Plot the cross validated relaxed lasso deviances or coefficients from a nested.glmnetr call. See plot.cv.glmnetr().*

Description

Plot the cross validated relaxed lasso deviances or coefficients from a nested.glmnetr call. See plot.cv.glmnetr().

Usage

```
## S3 method for class 'nested.glmnetr'
plot(
  x,
  gam = NULL,
  lambda.lo = NULL,
  title = NULL,
  plup = 0,
  coefs = FALSE,
  comment = TRUE,
  ...
)
```

Arguments

x	A nested.glmnetr output object
gam	A specific level of gamma for plotting. By default gamma.min will be used.
lambda.lo	A lower limit of lambda when plotting.
title	A title
plup	Plot upper 95 percent two-sided confidence intervals for the deviance plots.
coefs	Default is FALSE to plot deviances. Option of TRUE to plot coefficients.
comment	Default of TRUE to write to console information on lam and gam selected for output. FALSE will suppress this write to console.
...	Additional arguments passed to the plot function.

Value

This program returns a plot to the graphics window, and may provide some numerical information to the R Console.

Author(s)

Walter Kremers (kremers.walter@mayo.edu)

See Also

plot.glmnetr , plot.cv.glmnetr , nested.glmnetr

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
fit3 = nested.glmnetr(xs, NULL, y_, event, family="cox", folds_n=3)
```

```
plot(fit3)
plot(fit3, coefs=TRUE)
```

`predict.cv.glmnetr` *Give predicteds based upon a cv.glmnetr() output object.*

Description

Give predicteds based upon a cv.glmnetr() output object. By default lambda and gamma are chosen as the minimizing values for the relaxed lasso model. If gam=1 and lam=NULL then the best unrelaxed lasso model is chosen and if gam=0 and lam=NULL then the best fully relaxed lasso model is selected.

Usage

```
## S3 method for class 'cv.glmnetr'
predict(object, xs_new = NULL, lam = NULL, gam = NULL, comment = TRUE, ...)
```

Arguments

`object`	A cv.glmnetr (or nested.glmnetr) output object.
`xs_new`	The predictor matrix. If NULL, then betas are provided.
`lam`	The lambda value for choice of beta. If NULL, then lambda.min is used from the cross validated tuned relaxed model. We use the term lam instead of lambda as lambda usually denotes a vector in the package.
`gam`	The gamma value for choice of beta. If NULL, then gamma.min is used from the cross validated tuned relaxed model. We use the term gam instead of gamma as gamma usually denotes a vector in the package.
`comment`	Default of TRUE to write to console information on lam and gam selected for output. FALSE will suppress this write to console.
`...`	Additional arguments passed to the predict function.

Value

Either predicteds (xs_new*beta estimates based upon the predictor matrix xs_new) or model coefficients, based upon a cv.glmnetr() output object. When outputting coefficients (beta), creates a list with the first element, beta_, including 0 and non-0 terms and the second element, beta, including only non 0 terms.

See Also

`predict.glmnetr` , `cv.glmnetr` , `nested.glmnetr`

Examples

```
# set seed for random numbers, optionally, to get reproducible results
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=200, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$y_
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
cv.glmnetr.fit = cv.glmnetr(xs, NULL, y_, NULL, family="gaussian", folds_n=3, limit=2)
predict(cv.glmnetr.fit)
```

---

predict.cv.stepreg *Beta's or predicteds based upon a cv.stepreg() output object.*

Description

Give predicteds or Beta's based upon a cv.stepreg() output object. If an input data matrix is specified the X*Beta's are output. If an input data matrix is not specified then the Beta's are output. In the first column values are given based upon df as a tuning parameter and in the second column values based upon p as a tuning parameter.

Usage

```
## S3 method for class 'cv.stepreg'
predict(object, xs = NULL, ...)
```

Arguments

object	cv.stepreg() output object
xs	dataset for predictions. Must have the same columns as the input predictor matrix in the call to cv.stepreg().
...	pass through parameters

Value

a matrix of beta's or predicteds

predict.glmnetr *Get predicteds or coefficients using a glmnetr output object*

Description

Give predicteds based upon a glmnetr() output object. Because the glmnetr() function has no cross validation information, lambda and gamma must be specified. To choose lambda and gamma based upon cross validation one may use the cv.glmnetr() or nested.glmnetr() and the corresponding predict() functions.

Usage

```
## S3 method for class 'glmnetr'
predict(object, xs_new = NULL, lam = NULL, gam = NULL, ...)
```

Arguments

object	A glmnetr output object
xs_new	A desing matrix for predictions
lam	The value for lambda for determining the lasso fit. Required.
gam	The value for gamma for determining the lasso fit. Required.
...	Additional arguments passed to the predict function.

Value

Coefficients or predictions using a glmnetr output object. When outputting coefficients (beta), creates a list with the first element, beta_, including 0 and non-0 terms and the second element, beta, including only non 0 terms.

See Also

glmnetr , cv.glmnetr , nested.glmnetr

Examples

```
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=200, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
glmnetr.fit = glmnetr( xs, NULL, y_, event, family="cox")
betas = predict(glmnetr.fit,NULL,exp(-2),0.5 )
betas$beta
```

predict.nested.glmnetr *Give predicteds based upon the cv.glmnet output object contained in the nested.glmnetr output object.*

Description

This is essentially a redirect to the summary.cv.glmnetr function for nested.glmnetr output objects, based uopn the cv.glmnetr output object contained in the nested.glmnetr output object.

Usage

```
## S3 method for class 'nested.glmnetr'
predict(object, xs_new = NULL, lam = NULL, gam = NULL, comment = TRUE, ...)
```

Arguments

object	A nested.glmnetr output object.
xs_new	The predictor matrix. If NULL, then betas are provided.
lam	The lambda value for choice of beta. If NULL, then lambda.min is used from the cross validation informed relaxed model. We use the term lam instead of lambda as lambda usually denotes a vector in the package.
gam	The gamma value for choice of beta. If NULL, then gamma.min is used from the cross validation informed relaxed model. We use the term gam instead of gamma as gamma usually denotes a vector in the package.
comment	Default of TRUE to write to console information on lam and gam selected for output. FALSE will suppress this write to console.
...	Additional arguments passed to the predict function.

Value

Either the xs_new*Beta estimates based upon the predictor matrix, or model coefficients.

See Also

predict.cv.glmnetr

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
fit3 = nested.glmnetr(xs, NULL, y_, event, family="cox", folds_n=3)
betas = predict(fit3)
```

```
betas$beta
```

`predict_ann_tab` *Get predicteds for an Artificial Neural Network model fit in nested.glmnetr()*

Description

All but one of the Artificial Neural Network (ANNs) fit by nested.glmnetr() are based upon a neural network model and input from a lasso model. Thus a simple model(xs) statement will not give the proper predicted values. This function process information form the lasso and ANN model fits to give the correct predicteds. Whereas the ann_tab_cv() function ca be used to fit a model based upon an input data set it does not fit a lasso model to allow an informed starting point for the ANN fit. The pieces fo this are in nested.glmnetr(). To fit a cross validation (CV) informed ANN model fit one can run nested.glmnetr() with folds_n = 0 to derive the full data models without doing a cross validation.

Usage

```
predict_ann_tab(object, xs, modl = NULL)
```

Arguments

`object`	a output object from the nested.glmnetr() function
`xs`	new data of the same form used as input to nested.glmnetr()
`modl`	ANN model entry an integer from 1 to 5 indicating which "lasso informed" ANN is to be used for calculations. The number corresponds to the position of the ensemble input from the nested.glmnetr() call. The model must already be fit to calculate predicteds: 1 for ensemble[1] = 1, for model based upon raw data ; 2 for ensemble[2] = 1, raw data plus lasso predicteds as a predictor variable (features) ; 4 for ensemble[3] = 1, raw data plus lasso predicteds and initial weights corresponding to offset and allowed to update ; 5 for ensemble[4] = 1, raw data plus lasso predicteds and initial weights corresponding to offset and not allowed to updated ; 6 for ensemble[5] = 1, nonzero relaxed lasso terms ; 7 for ensemble[6] = 1, nonzero relaxed lasso terms plus lasso predicteds as a predictor variable (features) ; 8 for ensemble[7] = 1, nonzero relaxed lasso terms plus lasso predicteds with initial weights corresponding to offset and allowed to update ; 9 for ensemble[8] = 1, nonzero relaxed lasso terms plus lasso predicteds with initial weights corresponding to offset and not allowed to update.

Value

a vector of predicteds

prednn_tl *predicted values from an ann_tab_cv output object based upon the model and its lasso model used for generating an offset*

Description

predicted values from an ann_tab_cv output object based upon the model and its lasso model used for generating an offset

Usage

```
prednn_tl(lassomod, nnmodel, datain, lasso = 1)
```

Arguments

lassomod	a lasso model from a glmnetr() call used to genrte an offset
nnmodel	a ann_tab_cv() output object for
datain	new data
lasso	1 if an offset is to be added as column 1 for calculations (default), 0 to subset to the terms significant in a lasso model without adding the offset

Value

predictions from an nerual network model accounting from a lasso model

preds_1 *Get predictors form a stepwise regression model.*

Description

Get predictors form a stepwise regression model.

Usage

```
preds_1(modsumbest, k_, risklist, risklistl)
```

Arguments

modsumbest	matrix with best predictors based upon number of model terms
k_	Value for number of predictors in model
risklist	Riskset list
risklistl	Number of terms (length) in the riskset

Value

input to best.preds()

print.nested.glmnetr *Print an abbreviated summary of a nested.glmnetr() output object*

Description

Print an abbreviated summary of a nested.glmnetr() output object

Usage

```
## S3 method for class 'nested.glmnetr'
print(x, ...)
```

Arguments

x	a nested.glmnetr() output object.
...	additional pass through inputs for the print function.

Value

- a nested cross validation fit summary, or a cross validation model summary.

See Also

nested.glmnetr, glmnetr.compcv , summary.nested.glmnetr

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
fit3 = nested.glmnetr(xs, NULL, y_, event, family="cox", folds_n=3)
print(fit3)
```

print.rf_tune *Print output from rf_tune() function*

Description

Print output from rf_tune() function

Usage

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

Arguments

x	output from an rf_tune() function
...	optional pass through parameters to pass to print.rfsrc()

Value

summary to console

rf_tune *Fit a Random Forest model on data provided in matrix and vector formats.*

Description

Fit an Random Forest model using the rfsrc() function of the randomForestSRC package.

Usage

```
rf_tune(
  xs,
  start = NULL,
  y_,
  event = NULL,
  family = NULL,
  mtryc = NULL,
  ntreec = NULL,
  seed = NULL,
  keep = 0,
  track = 0
)
```

Arguments

xs	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
start	an optional vector of start times in case of a Cox model. Class numeric of length same as number of patients (n)
y_	dependent variable as a vector: time, or stop time for Cox model, Y_ 0 or 1 for binomial (logistic), numeric for gaussian. Must be a vector of length same as number of sample size.
event	event indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size.
family	model family, "cox", "binomial" or "gaussian" (default)
mtryc	a vector (numeric) of values to search over for optimization of the Random Forest fit. This if for the mtry input variable of the rfsrc() program specifying the number of terms to consider in each step of teh Random Forest fit.
ntreec	a vector (numeric) of 2 values, the first for the number of forests (ntree from rfsrc()) to use when searhcing for a better bit and the second to use when fitting the final model. More trees should give a better fit but require more computations and storage for the final. model.
seed	a seed for set.seed() so one can reproduce the model fit. If NULL the program will generate a random seed. Whether specified or NULL, the seed is stored in the output object for future reference. Note, for the default this randomly generated seed depends on the seed in memory at that time so will depend on any calls of set.seed prior to the call of this function.
keep	1 to keep the model fits used to select the value for mtry, or 0 (default) to not keep these initial model fits.
track	1 to output a brief summary of the final selected model, 2 to output a brief summary on each model fit in search of a better model or 0 (default) to not output this information.

Value

a Random Forest model fit

---

stepreg *Fit the steps of a stepwise regression.*

Description

Fit the steps of a stepwise regression.

Usage

```
stepreg(
  xs_st,
  start_time_st = NULL,
  y_st,
  event_st,
  steps_n = 0,
  method = "loglik",
  family = NULL,
  track = 0
)
```

Arguments

xs_st	predictor input - an n by p matrix, where n (rows) is sample size, and p (columns) the number of predictors. Must be in matrix form for complete data, no NA's, no Inf's, etc., and not a data frame.
start_time_st	start time, Cox model only - class numeric of length same as number of patients (n)
y_st	output vector: time, or stop time for Cox model, y_st 0 or 1 for binomal (logistic), numeric for gaussian. Must be a vector of length same as number of sample size.
event_st	event_st indicator, 1 for event, 0 for census, Cox model only. Must be a numeric vector of length same as sample size.
steps_n	number of steps done in stepwise regression fitting
method	method for choosing model in stepwise procedure, "loglik" or "concordance". Other procedures use the "loglik".
family	model family, "cox", "binomial" or "gaussian"
track	1 to output stepwise fit program, 0 (default) to suppress

Value

does a stepwise regression of depth maximum depth steps_n

Examples

```
set.seed(18306296)
sim.data=glmnetr.simdata(nrows=100, ncols=100, beta=c(0,1,1))
# this gives a more intersting case but takes longer to run
xs=sim.data$xs
# this will work numerically
xs=sim.data$xs[,c(2,3,50:55)]
y_=sim.data$yt
event=sim.data$event
# for a Cox model
cox.step.fit = stepreg(xs, NULL, y_, event, family="cox", steps_n=40)
# ... and for a linear model
```

```
y_=sim.data$yt
norm.step.fit = stepreg(xs, NULL, y_, NULL, family="gaussian", steps_n=40)
```

---

summary.cv.glmnetr *Output summary of a cv.glmnetr() output object.*

Description

Summarize the cross-validation informed model fit. The fully penalized (gamma=1) beta estimate will not be given by default but can too be output using printg1=TRUE.

Usage

```
## S3 method for class 'cv.glmnetr'
summary(object, printg1 = "FALSE", orderall = FALSE, ...)
```

Arguments

object	a cv.glmnetr() output object.
printg1	TRUE to also print out the fully penalized lasso beta, else FALSE to suppress.
orderall	By default (orderall=FALSE) the order terms enter into the lasso model is given for the number of terms that enter in lasso minimizing loss model. If orderall=TRUE then all terms that are included in any lasso fit are described.
...	Additional arguments passed to the summary function.

Value

Coefficient estimates (beta)

See Also

cv.glmnetr , nested.glmnetr

Examples

```
# set seed for random numbers, optionally, to get reproducible results
set.seed(82545037)
sim.data=glmnetr.simdata(nrows=100, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$y_
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
cv.glmnetr.fit = cv.glmnetr(xs, NULL, y_, NULL, family="gaussian", folds_n=3, limit=2)
summary(cv.glmnetr.fit)
```

summary.cv.stepreg *Summarize results from a cv.stepreg() output object.*

Description

Summarize results from a cv.stepreg() output object.

Usage

```
## S3 method for class 'cv.stepreg'
summary(object, ...)
```

Arguments

object	A cv.stepreg() output object
...	Additional arguments passed to the summary function.

Value

Summary of a stepreg() (stepwise regression) output object.

Examples

```
set.seed(955702213)
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=c(0,1,1))
# this gives a more interesting case but takes longer to run
xs=sim.data$xs
# this will work numerically as an example
xs=sim.data$xs[,c(2,3,50:55)]
dim(xs)
y_=sim.data$yt
event=sim.data$event
# for this example we use small numbers for steps_n and folds_n to shorten run time
cv.stepreg.fit = cv.stepreg(xs, NULL, y_, event, steps_n=10, folds_n=3, track=0)
summary(cv.stepreg.fit)
```

summary.nested.glmnetr *Summarize a nested.glmnetr() output object*

Description

Summarize the model fit from a nested.glmnetr() output object, i.e. the fit of a cross-validation informed relaxed lasso model fit, inferred by nested cross validation. Else summarize the cross-validated model fit.

Usage

```
## S3 method for class 'nested.glmnetr'
summary(
  object,
  cvfit = FALSE,
  pow = 2,
  printg1 = FALSE,
  short = 0,
  digits = 3,
  Call = NULL,
  ...
)
```

Arguments

object	a nested.glmnetr() output object.
cvfit	default of FALSE to summarize fit of a cross validation informed relaxed lasso model fit, inferred by nested cross validation. Option of TRUE will describe the cross validation informed relaxed lasso model itself.
pow	the power to which the average of correlations is to be raised. Only applies to the "gaussian" model. Default is 2 to yield R-square but can be on to show correlations. Pow is ignored for the family of "cox" and "binomial".
printg1	TRUE to also print out the fully penalized lasso beta, else to suppress. Only applies to cvfit=TRUE.
short	optionally print just the CV agreement summaries (short=1)
digits	digits for printing of deviances, linear calibration coefficients and agreement (concordances and R-squares).
Call	1 to print call used in generation of the object, 0 or NULL to not print
...	Additional arguments passed to the summary function.

Value

- a nested cross validation fit summary, or a cross validation model summary.

See Also

glmnetr.compcv , summary.cv.stepreg , nested.glmnetr

Examples

```
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
xs=sim.data$xs
y_=sim.data$yt
event=sim.data$event
# for this example we use a small number for folds_n to shorten run time
fit3 = nested.glmnetr(xs, NULL, y_, event, family="cox", folds_n=3)
```

```
summary(fit3)
```

`summary.rf_tune` *Summarize output from rf_tune() function*

Description

Summarize output from rf_tune() function

Usage

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

Arguments

`object`	output from an rf_tune() function
`...`	optional pass through parameters to pass to summary.rfsrc()

Value

summary to console

`summary.stepreg` *Briefly summarize steps in a stepreg() output object, i.e. a stepwise regression fit*

Description

Briefly summarize steps in a stepreg() output object, i.e. a stepwise regression fit

Usage

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

Arguments

`object`	A stepreg() output object
`...`	Additional arguments passed to the summary function.

Value

Summarize a stepreg() object

`wtlast` *Construct the weights for going from the last hidden layer to the last layer of the model, not counting any activation, to carry forward an offset to mimic a linear model*

Description

Construct the weights for going from the last hidden layer to the last layer of the model, not counting any activation, to carry forward an offset to mimic a linear model

Usage

```
wtlast(tnsr, lasso = 0, lscale = 5, scale = 1, rreturn = 1, trnspose = 0)
```

Arguments

`tnsr`	an input tensor which is to be modified to mimic the linear term of a generalized linear model, e.g a Cox or logistic regression model
`lasso`	1 if the first column is the linear estimate from a linear model, often a lasso model
`lscale`	Scale used to allow ReLU to exend +/- lscale before capping the inputted linear estimated
`scale`	Scale used to transform the inital random paramter assingments by dividing by scale
`rreturn`	1 (default) to return an R (numeric) vector, 0 to return a torch tensor
`trnspose`	1 to transpose the matrix before returning, 0 to not.

Value

a weight matrix in tensor format

`wtmiddle` *Construct the weights for going between two hidden layers, carrying forward an offset term to mimic a linear model*

Description

Construct the weights for going between two hidden layers, carrying forward an offset term to mimic a linear model

Usage

```
wtmiddle(tnsr, lasso = 0, rreturn = 1, trnspose = 0)
```

Arguments

tnsr	an input tensor wihc is to be modified to mimic the linear term of a generalized linear model, e.g a Cox or logistic regression model
lasso	1 if the first column is the lienar estimate from a linear model, often a lasso model
rreturn	1 (default) to return an R (numeric) vector, 0 to return a torch tensor
trnspose	1 to transpose the matrix before returning, 0 to not.

Value

a weight matrix in tensor format

---

wtzero *Construct the weights for going from the observed data with an offset in column 1 to the first hidden layer*

Description

Construct the weights for going from the observed data with an offset in column 1 to the first hidden layer

Usage

```
wtzero(tnsr, lasso = 0, lscale = 5, scale = 1, rreturn = 1, trnspose = 0)
```

Arguments

tnsr	an input tensor which is to be modified to mimic the linear term of a generalized linear model, e.g a Cox or logistic regression model
lasso	1 if the first column is the linear estimate from a linear model, often a lasso model
lscale	Scale used to allow ReLU to exend +/- lscale before capping the inputted linear estimated
scale	Scale used to transform the inital random paramter assingments by dividing by scale
rreturn	1 (default) to return an R (numeric) vector, 0 to return a torch tensor
trnspose	1 to transpose the matrix before returning, 0 to not.

Value

a weight matrix in tensor format

`xgb.simple` *Get a simple XGBoost model fit (no tuning)*

Description

This fits a gradient boosting machine model using the XGBoost platform. If uses a single set of hyperparameters that have sometimes been reasonable so runs very fast. For a better fit one can use xgb.tuned() which searches for a set of hyperparameters using the mlrMBO package which will generally provide a better fit but take much longer. See xgb.tuned() for a description of the data format required for input.

Usage

```
xgb.simple(
  train.xgb.dat,
  booster = "gbtree",
  objective = "survival:cox",
  eval_metric = NULL,
  nfold = 5,
  seed = NULL,
  folds = NULL,
  minimize = NULL,
  nrounds = 1000
)
```

Arguments

`train.xgb.dat`	The data to be used for training the XGBoost model
`booster`	for now just "gbtree" (default)
`objective`	one of "survival:cox" (default), "binary:logistic" or "reg:squarederror"
`eval_metric`	one of "cox-nloglik" (default), "auc", "rmse" or NULL. Default of NULL will select an appropriate value based upon the objective value.
`nfold`	number of folds used in xgb.cv() call
`seed`	a seed for set.seed() to assure one can get the same results twice. If NULL the program will generate a random seed. Whether specified or NULL, the seed is stored in the output object for future reference.
`folds`	an optional list where each element is a vector of indeces for a test fold. Default is NULL. If specified then nfold is ignored a la xgb.cv().
`minimize`	whether the eval_metric is to be minimized or maximized
`nrounds`	max number of iterations

Value

an XGBoost model fit

Author(s)

Walter K Kremers with contributions from Nicholas B Larson

Examples

```
# Simulate some data for a Cox model
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
Surv.xgb = ifelse( sim.data$event==1, sim.data$yt, -sim.data$yt )
data.full <- xgboost::xgb.DMatrix(data = sim.data$xs, label = Surv.xgb)
# for this example we use a small number for folds_n and nrounds to shorten run time
xgbfit = xgb.simple( data.full, objective = "survival:cox", nfold=5, nrounds=20)
preds = predict(xgbfit, sim.data$xs)
summary( preds )
preds[1:8]
```

---

xgb.tuned *Get a tuned XGBoost model fit*

Description

This fits a gradient boosting machine model using the XGBoost platform. It uses the mlrMBO mlrMBO package to search for a well fitting set of hyperparameters and will generally provide a better fit than xgb.simple(). Both this program and xgb.simple() require data to be provided in a xgb.DMatrix() object. This object can be constructed with a command like data.full <- xgb.DMatrix(data=myxs, label=mylabel), where myxs object contains the predictors (features) in a numerical matrix format with no missing values, and mylabel is the outcome or dependent variable. For logistic regression this would typically be a vector of 0's and 1's. For linear regression this would be vector of numerical values. For a Cox proportional hazards model this would be in a format required for XGBoost, which is different than for the survival package or glmnet package. For the Cox model a vector is used where observations associated with an event are assigned the time of event, and observations associated with censoring are assigned the NEGATIVE of the time of censoring. In this way information about time and status are communicated in a single vector instead of two vectors. The xgb.tuned() function does not handle (start,stop) time, i.e. interval, data. To tune the xgboost model we use the mlrMBO package which "suggests" the DiceKriging and rgenoud packages, but doe not install these. Still, for xgb.tuned() to run it seems that one should install the DiceKriging and rgenoud packages.

Usage

```
xgb.tuned(
  train.xgb.dat,
  booster = "gbtree",
  objective = "survival:cox",
  eval_metric = NULL,
  nfold = 5,
```

```
  seed = NULL,
  folds = NULL,
  minimize = NULL,
  nrounds = 1000
)
```

Arguments

train.xgb.dat	The data to be used for training the XGBoost model
booster	for now just "gbtree" (default)
objective	one of "survival:cox" (default), "binary:logistic" or "reg:squarederror"
eval_metric	one of "cox-nloglik" (default), "auc" or "rmse",
nfold	number of folds used in xgb.cv() call
seed	a seed for set.seed() to assure one can get the same results twice. If NULL the program will generate a random seed. Whether specified or NULL, the seed is stored in the output object for future reference.
folds	an optional list where each element is a vector of indeces for a test fold. Default is NULL. If specified then nfold is ignored a la xgb.cv().
minimize	whether the eval_metiric is to be minimized or maximized
nrounds	max number of iterations

Value

an XGBoost model fit

Author(s)

Walter K Kremers with contributions from Nicholas B Larson

Examples

```
# Simulate some data for a Cox model
sim.data=glmnetr.simdata(nrows=1000, ncols=100, beta=NULL)
Surv.xgb = ifelse( sim.data$event==1, sim.data$yt, -sim.data$yt )
data.full <- xgboost::xgb.DMatrix(data = sim.data$xs, label = Surv.xgb)
# for this example we use a small number for folds_n and nrounds to shorten
# run time.  This may still take a minute or so.
# xgbfit=xgb.tuned(data.full,objective="survival:cox",nfold=5,nrounds=20)
# preds = predict(xgbfit, sim.data$xs)
# summary( preds )
```

Index