Package: cvwrapr (via r-universe)

September 6, 2024

Type Package

Title Tools for Cross Validation

Version 1.0

Description Tools for performing cross-validation (CV). The main function is a general purpose wrapper that performs k-fold CV for any tuning parameter in any supervised learning method. The package also has a function that computes the loss incurred by a set of predictions for a variety of loss functions and model families.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.1

Suggests doParallel, gbm, glmnet, knitr, Matrix, parallel, pls, rmarkdown, testthat

Imports survival, foreach

VignetteBuilder knitr

Repository https://kjytay.r-universe.dev

RemoteUrl https://github.com/kjytay/cvwrapr

RemoteRef HEAD

RemoteSha 9188a9e0b5fb94f227bdd19a8b9e4f46b08a8723

Contents

`availableTypeMeasures` *Display the names of the measures used in CV for different families*

Description

Produces a list of names of measures that can be used in CV for different families. Note, however, that the package does not check if the measure the user specifies is appropriate for the family.

Usage

```
availableTypeMeasures(
  family = c("all", "gaussian", "binomial", "poisson", "multinomial", "cox",
    "mgaussian", "GLM")
)
```

Arguments

`family`	If a family is supplied, a list of the names of measures available for that family are produced. Default is "all", in which case the names of measures for all families are produced.

Value

If ‘family = "all"‘, a list of names of measures that can be used in CV for each family; otherwise, a vector of names of measures that can be used for the family passed as the parameter.

`buildPredMat` *Build a prediction matrix from CV model fits*

Description

Build a matrix of predictions from CV model fits.

Usage

```
buildPredMat(
  cvfitlist,
  y,
  lambda,
  family,
  foldid,
  predict_fun,
  predict_params,
  predict_row_params = c(),
  type.measure = NULL,
  weights = NULL,
  grouped = NULL
)
```

Arguments

`cvfitlist`	A list of length ‘nfolds‘, with each element being the model fit for each fold.
`y`	Response. It is only used to determine what dimensions the prediction array needs to have.
`lambda`	Lambda values for which we want predictions.
`family`	Model family; one of "gaussian", "binomial", "poisson", "cox", "multinomial", "mgaussian", or a class "family" object.
`foldid`	Vector of values identifying which fold each observation is in.
`predict_fun`	The prediction function; see ‘kfoldcv()‘ documentation for details.
`predict_params`	Any other parameters that should be passed tp ‘predict_fun‘ to get predictions (other than ‘object‘ and ‘newx‘); see ‘kfoldcv()‘ documentation for details.
`predict_row_params`	A vector which is a subset of ‘names(predict_params)‘, indicating which parameters have to be subsetted in the CV loop (other than ‘newx‘); see ‘kfoldcv()‘ documentation for details.
`type.measure`	Loss function to use for cross-validation. Only required for ‘family = "cox"‘.
`weights`	Observation weights. Only required for ‘family = "cox"‘.
`grouped`	Experimental argument; see ‘kfoldcv()‘ documentation for details. Only required for ‘family = "cox"‘.

Value

A matrix of predictions.

checkValidTypeMeasure *Check if loss function is valid for a given family*

Description

Also throws error if family is invalid.

Usage

```
checkValidTypeMeasure(type.measure, family)
```

Arguments

`type.measure`	Loss function to use for cross-validation.
`family`	Model family.

Value

No return value; called for side effects. (If the function returns instead of throwing an error, it means the loss function is valid for that family.)

computeError *Compute CV statistics from a prediction matrix*

Description

Compute CV statistics from a matrix of predictions.

Usage

```
computeError(
  predmat,
  y,
  lambda,
  foldid,
  type.measure,
  family,
  weights = rep(1, dim(predmat)[1]),
  grouped = TRUE
)
```

Arguments

`predmat` Array of predictions. If ‘y‘ is univariate, this has dimensions ‘c(nobs, nlambda)‘. If ‘y‘ is multivariate with ‘nc‘ levels/columns (e.g. for ‘family = "multionmial"‘ or ‘family = "mgaussian"‘), this has dimensions ‘c(nobs, nc, nlambda)‘. Note that these should be on the same scale as ‘y‘ (unlike in the glmnet package where it is the linear predictor).

`y` Response variable. Either a vector or a matrix, depending on the type of model.

`lambda` Lambda values associated with the errors in ‘predmat‘.

`foldid` Vector of values identifying which fold each observation is in.

`type.measure` Loss function to use for cross-validation. See ‘availableTypeMeasures()‘ for possible values for ‘type.measure‘. Note that the package does not check if the user-specified measure is appropriate for the family.

`family` Model family; used to determine the correct loss function.

`weights` Observation weights.

`grouped` This is an experimental argument, with default ‘TRUE‘, and can be ignored by most users. For all models except ‘family = "cox"‘, this refers to computing ‘nfolds‘ separate statistics, and then using their mean and estimated standard error to describe the CV curve. If ‘FALSE‘, an error matrix is built up at the observation level from the predictions from the ‘nfolds‘ fits, and then summarized (does not apply to ‘type.measure="auc"‘). For the "cox" family, ‘grouped=TRUE‘ obtains the CV partial likelihood for the Kth fold by *subtraction*; by subtracting the log partial likelihood evaluated on the full dataset from that evaluated on the on the (K-1)/K dataset. This makes more efficient use of risk sets. With ‘grouped=FALSE‘ the log partial likelihood is computed only on the Kth fold.

Details

Note that for the setting where ‘family = "cox"‘ and ‘type.measure = "deviance"‘ and ‘grouped = TRUE‘, ‘predmat‘ needs to have a ‘cvraw‘ attribute as computed by ‘buildPredMat()‘. This is because the usual matrix of pre-validated fits does not contain all the information needed to compute the model deviance for this setting.

Value

An object of class "cvobj".

`lambda` The values of lambda used in the fits.

`cvm` The mean cross-validated error: a vector of length ‘length(lambda)‘.

`cvsd` Estimate of standard error of ‘cvm‘.

`cvup` Upper curve = ‘cvm + cvsd‘.

`cvlo` Lower curve = ‘cvm - cvsd‘.

`lambda.min` Value of ‘lambda‘ that gives minimum ‘cvm‘.

`lambda.1se` Largest value of ‘lambda‘ such that the error is within 1 standard error of the minimum.

index	A one-column matrix with the indices of ‘lambda.min‘ and ‘lambda.1se‘ in the sequence of coefficients, fits etc.
name	A text string indicating the loss function used (for plotting purposes).

Examples

```
set.seed(1)
x <- matrix(rnorm(500), nrow = 50)
y <- rnorm(50)
cv_fit <- kfoldcv(x, y, train_fun = glmnet::glmnet,
                  predict_fun = predict, keep = TRUE)
mae_err <- computeError(cv_fit$fit.preval, y, cv_fit$lambda,
                        cv_fit$foldid, type.measure = "mae",
                        family = "gaussian")
```

`computeRawError` *Compute the nobs by nlambda matrix of errors*

Description

Computes the nobs by nlambda matrix of errors corresponding to the error measure provided. Only works for "gaussian" and "poisson" families right now.

Usage

```
computeRawError(predmat, y, type.measure, family, weights, foldid, grouped)
```

Arguments

predmat	Array of predictions. If ‘y‘ is univariate, this has dimensions ‘c(nobs, nlambda)‘. If ‘y‘ is multivariate with ‘nc‘ levels/columns (e.g. for ‘family = "multionmial"‘ or ‘family = "mgaussian"‘), this has dimensions ‘c(nobs, nc, nlambda)‘. Note that these should be on the same scale as ‘y‘ (unlike in the glmnet package where it is the linear predictor).
y	Response variable.
type.measure	Loss function to use for cross-validation. See ‘availableTypeMeasures()‘ for possible values for ‘type.measure‘. Note that the package does not check if the user-specified measure is appropriate for the family.
family	Model family; used to determine the correct loss function.
weights	Observation weights.
foldid	Vector of values identifying which fold each observation is in.
grouped	Experimental argument; see ‘kfoldcv()‘ documentation for details.

Value

A list with the following elements:

`cvraw`	An nobs by nlambda matrix of raw error values.
`weights`	Observation weights.
`N`	A vector of length nlambda representing the number of non-NA predictions associated with each lambda value.
`type.measure`	Loss function used for CV.

`computeStats` *Compute CV statistics*

Description

Use the returned output from ‘computeRawError()‘ to compute CV statistics.

Usage

```
computeStats(cvstuff, foldid, lambda, grouped)
```

Arguments

`cvstuff`	Output from a call to ‘computeRawError()‘.
`foldid`	Vector of values identifying which fold each observation is in.
`lambda`	Lambda values associated with the errors in ‘cvstuff‘.
`grouped`	Experimental argument; see ‘kfoldcv()‘ documentation for details.

Value

A list with the following elements:

`lambda`	The values of lambda used in the fits.
`cvm`	The mean cross-validated error: a vector of length ‘length(lambda)‘.
`cvsd`	Estimate of standard error of ‘cvm‘.
`cvup`	Upper curve = ‘cvm + cvsd‘.
`cvlo`	Lower curve = ‘cvm - cvsd‘.

`coxnet.deviance` *Compute deviance for Cox model*

Description

Compute the deviance (-2 log partial likelihood) for Cox model. This is a pared down version of ‘glmnet‘’s ‘coxnet.deviance‘ with one big difference: here, ‘pred‘ is on the scale of ‘y‘ (‘mu‘) while in ‘glmnet‘, ‘pred‘ is the linear predictor (‘eta‘).

Usage

```
coxnet.deviance(pred = NULL, y, weights = NULL, std.weights = TRUE)
```

Arguments

`pred`	Fit vector or matrix. If ‘NULL‘, it is set to all ones.
`y`	Survival response variable, must be a `Surv` or `stratifySurv` object.
`weights`	Observation weights (default is all equal to 1).
`std.weights`	If TRUE (default), observation weights are standardized to sum to 1.

Details

Computes the deviance for a single set of predictions, or for a matrix of predictions. Uses the Breslow approach to ties.

`coxnet.deviance()` is a wrapper: it calls the appropriate internal routine based on whether the response is right-censored data or (start, stop] survival data.

Value

A vector of deviances, one for each column of predictions.

Examples

```
set.seed(1)
eta <- rnorm(10)
time <- runif(10, min = 1, max = 10)
d <- ifelse(rnorm(10) > 0, 1, 0)
y <- survival::Surv(time, d)
coxnet.deviance(pred = exp(eta), y = y)

# if pred not provided, it is set to ones vector
coxnet.deviance(y = y)

# example with (start, stop] data
y2 <- survival::Surv(time, time + runif(10), d)
coxnet.deviance(pred = exp(eta), y = y2)
```

getCindex — *Compute C index for a Cox model*

Description

Computes Harrel's C (concordance) index for predictions, taking censoring into account.

Usage

```
getCindex(pred, y, weights = rep(1, nrow(y)))
```

Arguments

`pred`	A vector of predictions.
`y`	Survival response variable, must be a `Surv` or `stratifySurv` object.
`weights`	Observation weights (default is all equal to 1).

Value

The C index for the predictions (a single numeric value).

Examples

```
set.seed(1)
pred <- rep(1:2, length.out = 10)
y <- survival::Surv(exp(rnorm(10)), rbinom(10, 1, 0.5))
getCindex(pred, y)
```

getOptLambda — *Get lambda.min and lambda.1se values*

Description

Get lambda.min and lambda.1se values and indices.

Usage

```
getOptLambda(lambda, cvm, cvsd, type.measure)
```

Arguments

`lambda`	The values of lambda used in the fits.
`cvm`	The mean cross-validated error: a vector of length 'length(lambda)'.
`cvsd`	Estimate of standard error of 'cvm'.
`type.measure`	Loss function used for CV.

Value

A list with the following elements:

`lambda.min`	Value of ‘lambda‘ that gives minimum ‘cvm‘.
`lambda.1se`	Largest value of ‘lambda‘ such that the error is within 1 standard error of the minimum.
`index`	A one-column matrix with the indices of ‘lambda.min‘ and ‘lambda.1se‘ in the sequence of coefficients, fits etc.

`getTypeMeasureName` *Get full name of loss function*

Description

Get the full name of the loss function from ‘type.measure‘ and ‘family‘.

Usage

```
getTypeMeasureName(type.measure, family)
```

Arguments

`type.measure`	Loss function to use for cross-validation.
`family`	Model family.

Value

A named vector of length 1. The vector’s value is the full name of the loss function, while the name of that element is the short name of the loss function.

`kfoldcv` *K-fold cross-validation wrapper*

Description

Does k-fold cross-validation for a given model training function and prediction function. The hyperparameter to be cross-validated is assumed to be ‘lambda‘. The training and prediction functions are assumed to be able to fit/predict for multiple ‘lambda‘ values at the same time.

Usage

```
kfoldcv(
  x,
  y,
  train_fun,
  predict_fun,
  type.measure = "deviance",
  family = "gaussian",
  lambda = NULL,
  train_params = list(),
  predict_params = list(),
  train_row_params = c(),
  predict_row_params = c(),
  nfolds = 10,
  foldid = NULL,
  parallel = FALSE,
  grouped = TRUE,
  keep = FALSE,
  save_cvfits = FALSE
)
```

Arguments

`x`	Input matrix of dimension ‘nobs‘ by ‘nvars‘; each row is an observation vector.
`y`	Response variable. Either a vector or a matrix, depending on the type of model.
`train_fun`	The model training function. This needs to take in an input matrix as ‘x‘ and a response variable as ‘y‘.
`predict_fun`	The prediction function. This needs to take in the output of ‘train_fun‘ as ‘object‘ and new input matrix as ‘newx‘.
`type.measure`	Loss function to use for cross-validation. See ‘availableTypeMeasures()‘ for possible values for ‘type.measure‘. Note that the package does not check if the user-specified measure is appropriate for the family.
`family`	Model family; used to determine the correct loss function. One of "gaussian", "binomial", "poisson", "cox", "multinomial", "mgaussian", or a class "family" object.
`lambda`	Option user-supplied sequence representing the values of the hyperparameter to be cross-validated.
`train_params`	Any parameters that should be passed to ‘train_fun‘ to fit the model (other than ‘x‘ and ‘y‘). Default is the empty list.
`predict_params`	Any other parameters that should be passed tp ‘predict_fun‘ to get predictions (other than ‘object‘ and ‘newx‘). Default is the empty list.
`train_row_params`	A vector which is a subset of ‘names(train_params)‘, indicating which parameters have to be subsetted in the CV loop (other than ‘x‘ and ‘y‘. Default is ‘c()‘. Other parameters which should probably be included here are "weights" (for observation weights) and "offset".

predict_row_params
: A vector which is a subset of ‘names(predict_params)‘, indicating which parameters have to be subsetted in the CV loop (other than ‘newx‘). Default is ‘c()‘. Other parameters which should probably be included here are "newoffset".

nfolds
: Number of folds (default is 10). Smallest allowable value is 3.

foldid
: An optional vector of values between ‘1‘ and ‘nfolds‘ (inclusive) identifying which fold each observation is in. If supplied, ‘nfolds‘ can be missing.

parallel
: If ‘TRUE‘, use parallel ‘foreach‘ to fit each fold. Must register parallel backend before hand. Default is ‘FALSE‘.

grouped
: This is an experimental argument, with default ‘TRUE‘, and can be ignored by most users. For all models except ‘family = "cox"‘, this refers to computing ‘nfolds‘ separate statistics, and then using their mean and estimated standard error to describe the CV curve. If ‘FALSE‘, an error matrix is built up at the observation level from the predictions from the ‘nfolds‘ fits, and then summarized (does not apply to ‘type.measure="auc"‘). For the "cox" family, ‘grouped=TRUE‘ obtains the CV partial likelihood for the Kth fold by *subtraction*; by subtracting the log partial likelihood evaluated on the full dataset from that evaluated on the on the (K-1)/K dataset. This makes more efficient use of risk sets. With ‘grouped=FALSE‘ the log partial likelihood is computed only on the Kth fold.

keep
: If ‘keep = TRUE‘, a prevalidated array is returned containing fitted values for each observation and each value of lambda. This means these fits are computed with this observation and the rest of its fold omitted. The ‘foldid‘ vector is also returned. Default is ‘keep = FALSE‘.

save_cvfits
: If ‘TRUE‘, the model fits for each CV fold are returned as a list. Default is ‘FALSE‘.

Details

The model training function is assumed to take in the data matrix as ‘x‘, the response as ‘y‘, and the hyperparameter to be cross-validated as ‘lambda‘. It is assumed that in its returned output, the hyperparameter values actually used are stored as ‘lambda‘. The prediction function is assumed to take in the new data matrix as ‘newx‘, and a ‘lambda‘ sequence as ‘s‘.

Value

An object of class "cvobj".

lambda
: The values of lambda used in the fits.

cvm
: The mean cross-validated error: a vector of length ‘length(lambda)‘.

cvsd
: Estimate of standard error of ‘cvm‘.

cvup
: Upper curve = ‘cvm + cvsd‘.

cvlo
: Lower curve = ‘cvm - cvsd‘.

lambda.min
: Value of ‘lambda‘ that gives minimum ‘cvm‘.

lambda.1se
: Largest value of ‘lambda‘ such that the error is within 1 standard error of the minimum.

index	A one-column matrix with the indices of ‘lambda.min‘ and ‘lambda.1se‘ in the sequence of coefficients, fits etc.
name	A text string indicating the loss function used (for plotting purposes).
fit.preval	If ‘keep=TRUE‘, this is the array of prevalidated fits. Some entries can be ‘NA‘, if that and subsequent values of ‘lambda‘ are not reached for that fold.
foldid	If ‘keep=TRUE‘, the fold assignments used.
overallfit	Model fit for the entire dataset.
cvfitlist	If ‘save_cvfits=TRUE‘, a list containing the model fits for each CV fold.

Examples

```
set.seed(1)
x <- matrix(rnorm(500), nrow = 50)
y <- rnorm(50)
cv_fit <- kfoldcv(x, y, train_fun = glmnet::glmnet,
                  predict_fun = predict)
```

---

plot.cvobj — *Plot the cross-validation curve from a class ‘cvobj‘ object*

Description

Plots the cross-validation curve, and upper and lower standard deviation curves, as a function of the ‘lambda‘ values used.

Usage

```
## S3 method for class 'cvobj'
plot(x, sign.lambda = 1, log.lambda = TRUE, ...)
```

Arguments

x	A ‘"cvobj"‘ object.
sign.lambda	Either plot against ‘log(lambda)‘ (default) or its negative if ‘sign.lambda = -1‘.
log.lambda	If ‘TRUE‘ (default), x-axis is ‘log(lambda)‘ instead of ‘lambda‘ (‘log.lambda = FALSE‘).
...	Other graphical parameters to plot.

Value

A plot is produced, and nothing is returned.

`print.cvobj` *Print a class ‘cvobj‘ object*

Description

Print a summary of results of cross-validation for a class ‘cvobj‘ object.

Usage

```
## S3 method for class 'cvobj'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x`	A ‘"cvobj"‘ object.
`digits`	Significant digits in printout.
`...`	Other print arguments.

Value

A summary is printed, and nothing is returned.

Index