Package ‘eml’

February 19, 2015

Encoding UTF-8
Type Package
Title Evaluation of Modeling without Information Leakage
Version 1.1-6
Date 2014-08-01
Author Christofer Backlin, Mats Gustafsson
Maintainer Christofer Backlin <christofer.backlin@medsci.uu.se>
Description A toolbox for designing and evaluating predictive models with resampling methods. The aim of this package is to provide a simple and efficient general framework for working with any type of prediction problem, be it classification, regression or survival analysis, that is easy to extend and adapt to your specific setting. Some commonly used methods for classification, regression and survival analysis are included.
Suggests caret, cmprsk, glmnet, Hmisc, MASS, party, pamr, randomForest, RColorBrewer, survival, testthat (>= 0.8.1)
License GPL (>= 2)
LazyLoad yes
LazyData yes
NeedsCompilation no
Repository CRAN
Date/Publication 2014-08-01 14:19:06

R topics documented:

as.character.outcome ...................................................... 3
as.data.frame.outcome .................................................... 4
as.matrix.outcome ........................................................... 4
as.outcome ................................................................. 5
as.outcome.Surv ............................................................. 5
as.Surv ................................................................. 6
as.Surv.outcome .............................................................. 6
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.Surv.Surv</td>
</tr>
<tr>
<td>batch.model</td>
</tr>
<tr>
<td>dim.outcome</td>
</tr>
<tr>
<td>emil</td>
</tr>
<tr>
<td>emil.extensions</td>
</tr>
<tr>
<td>emil.fit.caret</td>
</tr>
<tr>
<td>emil.fit.cforest</td>
</tr>
<tr>
<td>emil.fit.glmmnet</td>
</tr>
<tr>
<td>emil.fit.lda</td>
</tr>
<tr>
<td>emil.fit.lm</td>
</tr>
<tr>
<td>emil.fit.pamr</td>
</tr>
<tr>
<td>emil.fit.qda</td>
</tr>
<tr>
<td>emil.fit.randomForest</td>
</tr>
<tr>
<td>emil.predict.caret</td>
</tr>
<tr>
<td>emil.predict.cforest</td>
</tr>
<tr>
<td>emil.predict.glmmnet</td>
</tr>
<tr>
<td>emil.predict.lda</td>
</tr>
<tr>
<td>emil.predict.lm</td>
</tr>
<tr>
<td>emil.predict.pamr</td>
</tr>
<tr>
<td>emil.predict.qda</td>
</tr>
<tr>
<td>emil.predict.randomForest</td>
</tr>
<tr>
<td>emil.vimp.pamr</td>
</tr>
<tr>
<td>emil.vimp.randomForest</td>
</tr>
<tr>
<td>error.fun</td>
</tr>
<tr>
<td>evaluate.modeling</td>
</tr>
<tr>
<td>factor.events</td>
</tr>
<tr>
<td>fill</td>
</tr>
<tr>
<td>fit</td>
</tr>
<tr>
<td>image.resample</td>
</tr>
<tr>
<td>impute</td>
</tr>
<tr>
<td>index.fit</td>
</tr>
<tr>
<td>integer.events</td>
</tr>
<tr>
<td>is.blank</td>
</tr>
<tr>
<td>is.na.outcome</td>
</tr>
<tr>
<td>is.outcome</td>
</tr>
<tr>
<td>length.outcome</td>
</tr>
<tr>
<td>modeling.procedure</td>
</tr>
<tr>
<td>neg.gmpa</td>
</tr>
<tr>
<td>nice.require</td>
</tr>
<tr>
<td>outcome</td>
</tr>
<tr>
<td>p.value</td>
</tr>
<tr>
<td>p.value.coxph</td>
</tr>
<tr>
<td>p.value.crr</td>
</tr>
<tr>
<td>p.value.cuminc</td>
</tr>
<tr>
<td>p.value.survdiff</td>
</tr>
<tr>
<td>plot.outcome</td>
</tr>
<tr>
<td>pre.impute.knn</td>
</tr>
<tr>
<td>pre.pamr</td>
</tr>
</tbody>
</table>
as.character.outcome

```r
as.character.outcome
```

Convert outcome vector to character vector

**Description**

Convert outcome vector to character vector

**Usage**

```r
## S3 method for class 'outcome'
as.character(x, ...)
```

**Arguments**

- `x`  
  Outcome vector.

- `...`  
  Ignored, kept for S3 consistency.

**Value**

A character vector.

**Author(s)**

Christofer Bäcklin
as.data.frame.outcome  \textit{Convert outcome vector to data frame}

\textbf{Description}

Convert outcome vector to data frame

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'outcome'
as.data.frame(x, ...)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
\item \textbf{x} \hspace{1cm} Outcome vector.
\item \textbf{...} \hspace{1cm} Ignored, kept for S3 consistency.
\end{itemize}

\textbf{Value}

The data frame underlying the outcome vector.

\textbf{Author(s)}

Christofer Bäcklin

\section*{as.matrix.outcome  \textit{Convert outcome vector to matrix}}

\textbf{Description}

Convert outcome vector to matrix

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'outcome'
as.matrix(x, ...)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
\item \textbf{x} \hspace{1cm} Outcome vector.
\item \textbf{...} \hspace{1cm} Ignored, kept for S3 consistency.
\end{itemize}

\textbf{Value}

The data frame underlying the outcome vector converted to a matrix. Event types are converted to integers.
as.outcome

Author(s)

Christofer Bäcklin

Description

Converts objects to class outcome.

Usage

as.outcome(x, ...)

Arguments

x Generic object.
...

Sent to class methods.

Value

A vector of class outcome.

Author(s)

Christofer Bäcklin

as.outcome.Surv

Convert Surv vector to outcome vector

Description

Convert Surv vector to outcome vector

Usage

## S3 method for class 'Surv'
as.outcome(x, ...)

Arguments

x Surv vector.
...

Ignored, kept for S3 consistency.
as.Surv

Convert object to Surv vector

Description

Converts objects to class Surv.

Usage

as.Surv(x, ...)

Arguments

x    Generic object.
...

Value

A vector of class Surv.

Author(s)

Christofer Bäcklin

as.Surv.outcome

Convert outcome vector to Surv vector

Description

Convert outcome vector to Surv vector

Usage

## S3 method for class 'outcome'
as.Surv(x, main = 1, censor = NA, ...)

### as.Surv.Surv

**Arguments**

- **x**
  - Outcome vector.
- **main**
  - `Surv` only supports one event type. This argument controls which type that will be kept, all others are considered censoring.
- **censor**
  - A vector of event types to consider censoring.
- **...**
  - Ignored, kept for S3 consistency.

**Value**

A vector of class `Surv`. All event types other than the main event are considered censoring.

**Author(s)**

Christofer Bäcklin

---

**as.Surv.Surv**

**Description**

Defined in case `as.Surv` is called on a `Surv` object.

**Usage**

```r
## S3 method for class 'Surv'
as.Surv(x, ...)
```

**Arguments**

- **x**
  - Object.
- **...**
  - Ignored.

**Value**

A vector of class `Surv`.

**Author(s)**

Christofer Bäcklin
batch.model  

Perform modeling

Description

This function is the core of the framework, carrying out most of the work. It fits and evaluates models according to a resampling scheme, and extracts variable importance scores. Note that the typical user does not have to call this function directly, but should use `fit`, `tune` or `evaluate.modeling` instead.

Usage

```
batch.model(proc, x, y, resample = emil::resample("crossval", y, nfold = 2, 
nrep = 2), pre.process = pre.split, .save = list(fit = FALSE, pred = 
FALSE, vimp = FALSE, tuning = FALSE), .parallel.cores = 1, 
.checkpoint.dir = NULL, .return.errors = .parallel.cores > 1, 
.verbose = FALSE)
```

Arguments

- **proc**: modeling procedure, or list of modeling procedures, as produced by `modeling.procedure`.
- **x**: Dataset, observations as rows and descriptors as columns.
- **y**: Response vector.
- **resample**: The test subsets used for parameter tuning. Leave blank to randomly generate a resampling scheme of the same kind as is used by `batch.model` to assess the performance of the whole modeling procedure.
- **pre.process**: Function that performs pre-processing and splits dataset into fitting and test subsets.
- **.save**: What aspects of the modeling to perform and return to the user.
- **.parallel.cores**: Number of CPU-cores to use for parallel computation. The current implementation is based on `mcMap`, which unfortunately do not work on Windows systems. It can however be re-implemented by the user fairly easily by setting up a PSOCK cluster and calling `parLapply` as in the example below. This solution might be included in future versions of the package, after further investigation.
- **.checkpoint.dir**: Directory to save intermediate results to. If set the computation can be restarted with minimal loss of results.
- **.return.errors**: If FALSE the entire modeling is aborted upon an error. If TRUE the modeling of the particular fold is aborted and the error message is returned instead of its results.
- **.verbose**: Whether to print an activity log. Set to -1 to also suppress output generated from the procedure’s functions.
**Value**

A list tree where the top level corresponds to folds (in case of multiple folds), the next level corresponds to the modeling procedures (in case of multiple procedures), and the final level is specified by the `.save` parameter. It typically contains a subset of the following elements:

- **error** Performance estimate of the fitted model. See `error.fun` for more information.
- **fit** Fitted model.
- **pred** Predictions given by the model.
- **vimp** Variable importance scores.
- **tune** Results from the parameter tuning. See `tune` for details.

**Author(s)**

Christofer Bäcklin

**See Also**

`emil`, `modeling.procedure`

**Examples**

```r
x <- iris[-5]
y <- iris$Species
proc <- modeling.procedure("lda")
cv <- resample("crossval", y, 4, 4)
perf <- batch.model(proc, x, y, cv, .save=list(pred=TRUE))

# Parallelization on windows
require(parallel)
cl <- makePSOCKcluster(2)
clusterEvalQ(cl, library(emil))
clusterExport(cl, c("proc", "x", "y"))
perf <- parLapply(cl, cv, function(fold)
  batch.model(proc, x, y, resample=fold))
```

---

**dim.outcome**

*Dimension of an outcome vector*

**Description**

Dimension of an outcome vector

**Usage**

```r
## S3 method for class 'outcome'
dim(x)
```
Arguments

x  Outcome vector.

Value

A vector with number of observations and number of variables, of which there is always two, time and event.

Author(s)

Christofer Bäcklin

Description

The emil package implements a framework for working with predictive modeling problems without information leakage. For an overview of its functionality please read the original publication included as the package’s vignette (to be added).

Central topics and functions

Setting up modeling problems:

resample  Functions for generating and resampling schemes and information on how to implement custom resampling methods.
pre.process  Data pre-processing functions.
modeling.procedure  Manages algorithms used for fitting models, making predictions, and extracting variable importance scores.
error.fun  Performance estimation functions used to tune variables and evaluate performance of modeling procedures.

Solving modeling problems:

evaluate.modeling  Evaluate the performance of a procedure using resampling.
fit  Fit a model (according to a procedure).
tune  Tune parameters of a procedure.
predict  Use a fitted model to predict the response of observations.
vimp  Extract variable importance scores of a fitted model.
batch.model  A semi-internal function that carries out most of the work. Parameters passed along from the above functions to this function are documented here.

Managing the results of modeling problems:

subtree  Extracts results from the output of evaluate.modeling. It is essentially a recursive version of lapply and sapply.
subframe  Extracts results and organize them per observation. Useful when comparing predictions and properties of individual observations across folds of a resampling scheme.

No functions for plotting is included in the current version of the package, except for image.resample.
Methods included in the package

**Resampling methods:** See `resample` for information on usage and implementation of custom methods.
- `resample.holdout` Repeated holdout.
- `resample.crossval` Cross validation.

**Data pre-processing methods:** See `pre.process` for information on usage and implementation of custom methods. The imputation functions can also be used outside of the resampling scheme, see `impute`.
- `pre.split` Only split, no transformation.
- `pre.center` Center data to have mean 0 of each feature.
- `pre.scale` Center and scale data to have mean 0 and standard deviation 1.
- `pre.impute.median` Impute missing values with feature medians.
- `pre.impute.knn` Impute missing values with k-NN, see `pre.impute.knn` for details on how to set parameters.

**Modeling methods:** See `modeling.procedure` for information on usage and `emil.extensions` for information on implementation of custom methods.
- `cforest` Conditional inference forest.
- `glmnet` Elastic net.
- `lda` Linear discriminant.
- `lm` Linear model.
- `pamr` Nearest shrunken centroids.
- `qda` Quadratic discriminant.
- `randomForest` Random forest.

**Performance estimation methods:** See `error.fun` for information on usage and implementation of custom methods. Since the framework is designed to minimize the error when tuning variables, some measures are negated, e.g. `neg.auc`.

For classification problems:
- `error.rate` Fraction of predictions that were incorrect.
- `weighted.error.rate` See its own documentation.
- `neg.auc` Negative area under ROC curve.
- `neg.gmpa` Negative geometric mean of class-specific prediction accuracy. Good for problems with imbalanced class sizes.

For regression problems:
- `mse` Mean square error.
- `rmse` Root mean square error.

For survival analysis problem:
- `neg.harrell.C` Negative Harrell’s concordance index.

**Author(s)**

Christofer Bäcklin
Description

This page describes how to implement custom methods compatible with the functions of the emil framework, most notably fit, tune, and evaluate.modeling. Pre-processing and resampling is not covered here, but in the entries pre.process and resample.

Fitting models

To write and use custom model fitting functions with the emil framework, it must take the the following inputs:

\[
\text{function}(x, y, \ldots)
\]

- \(x\) The descriptors (or variables) of the observations you want to train the model on. This is typically a matrix or data frame where each row corresponds to an observation. In case it is more natural to characterize your observations some other way, maybe as character vectors of varying length for some document classification method, \(x\) can be of any form you like as long as the fitting function knows how to handle it. In that case you will also need supply you own pre-processing function (see pre.process that can extract training and test sets from the entire data set.

See the functions pre.pamr and emil.fit.pamr for an example of a function that does not take its descriptors in the default way.

- \(y\) A response vector. This is the outcome you want to model, e.g. the variable of interest in a regression, class label in a classification problem, or anything else that a fitted model will produce when given data to make predictions from.

- \(\ldots\) Model parameters. These will all be tunable with the tune and evaluate.modeling functions.

The function must return everything necessary to make future predictions, but it can take any form you like. In the simplest case it is just a number of fitted parameter values, like in a least squares regression, but it could also be some big and complex structure holding an ensemble of multiple sub-models.

Making predictions

Once a model is fitted it can be used to make predictions with a prediction function, defined as such:

\[
\text{function}(\text{object}, x, \ldots)
\]

- \(\text{object}\) A fitted model produced by the model fitting function described above.

The output of the prediction function must be an object that can be compared to the true response, by an error function (see below). It is typically a list with elements named "pred" for "predictions" or "risk" for estimated risks. It can also be on an arbitrary form as long as a compatible error function is used.
Calculating performance

See `error.fun`.

Calculating variable importance scores

Estimating the importance of each descriptor (or variable) can often be as important as making predictions. Functions for calculating or extracting variable importance scores from fitted models should be defined as follows:

```r
definition(object, ...)
```

- `object` A fitted model produced by the model fitting function described above.
- `...` Parameters to the prediction functions. These are ignored by `tune` and `evaluate.modeling`, but could be convenient if the user wants to work with it manually.

The function should return a vector of length p or a p-by-c data frame where p is the number of descriptors in the data set and c is the number of classes.

Author(s)

Christofer Bäcklin

See Also

`emil, error.fun, pre.process, resample`

---

**emil.fit.caret**

*Fit a model using the caret package*

---

**Description**

Fit a model using the caret package

**Usage**

```r
definition(x, y, ...)
```

**Arguments**

- `x` Descriptors.
- `y` Response.
- `...` Sent to `train`.

**Author(s)**

Christofer Bäcklin
emil.fit.cforest  

*Fit conditional inference forest*

**Description**

A *cforest* is a random forest based on conditional inference trees, using the implementation in the *party* package. These trees can be used for classification, regression or survival analysis, but only the survival part has been properly tested so far.

**Usage**

emil.fit.cforest(x, y, formula = y ~ ., ctrl.fun = cforest_unbiased, ...)

**Arguments**

- **x**  
  Dataset, observations as rows and descriptors as columns.
- **y**  
  Responses.
- **formula**  
  Formula linking response to descriptors.
- **ctrl.fun**  
  Which control function to use, see *cforest_control*.
- **...**  
  Sent to the function specified by *ctrl.fun*.

**Details**

The parameters to *cforest* are set using a *cforest_control* object. You should read the documentation as the default values are chosen for technical reasons, not predictive performance! Pay special attention to *mtry* which is set very low by default.

**Value**

A fitted *cforest* model.

**Author(s)**

Christofer Bäcklin

**See Also**

*emil*, *emil.predict.cforest*, *modeling.procedure*
emil.fit.glmnet

Fit GLM with LASSO, Ridge or elastic net regularization.

Description

Fits generalized linear models with regularization using the glmnet package implementation.

Usage

emil.fit.glmnet(x, y, family, nfolds, foldid, alpha = 1, lambda = NULL, ...)

Arguments

x
Dataset.
y
Response vector. Can be of many different types for solving different problems, see glmnet.
family
Determines the the type of problem to solve. Auto detected if y is numeric or survival. See family for details.
nfolds
See cv.glmnet.
foldid
See cv.glmnet.
alpha
Regularization parameter, see glmnet.
lambda
Regularization parameter, see glmnet.
...
Sent to cv.glmnet.

Details

The alpha parameter of glmnet controls the type of penalty. Use 0 (default) for lasso only, 1 for ridge only, or an intermediate for a combination. This is typically the variable to tune on. The shrinkage, controlled by the lambda parameter, can be left unspecified for internal tuning (works the same way as emil.fit.glmnet).

Value

Fitted GLM.

Author(s)

Christofer Bäcklin

See Also

emil, emil.predict.glmnet, modeling.procedure
emil.fit.lda  
*Fit linear discriminant*

**Description**

Wrapper for the MASS package implementation.

**Usage**

emil.fit.lda(x, y, ...)

**Arguments**

- `x`  
  Dataset, numerical matrix with observations as rows.

- `y`  
  Class labels, factor.

- `...`  
  Sent to `lda`.

**Value**

Fitted linear discriminant.

**Author(s)**

Christofer Bäcklin

**See Also**

emil, emil.predict.lda, modeling.procedure

emil.fit.lm  
*Fit a linear model fitted with ordinary least squares*

**Description**

Based on `lm`.

**Usage**

emil.fit.lm(x, y, formula = y ~ ., ...)

**Arguments**

- `x`  
  Descriptors.

- `y`  
  Response, numeric.

- `formula`  
  See `lm`.

- `...`  
  Sent to `lm`.
Value

Fitted linear model.

Author(s)

Christofer Bäcklin

See Also

emil, emil.predict.lm, modeling.procedure

emil.fit.pamr

Fit nearest shrunken centroids model.

Description

Wrapped version of the pamr package implementation. Note that this function uses internal cross-validation for determining the value of the shrinkage threshold.

Usage

emil.fit.pamr(x, y, error.fun, cv, threshold = NULL, ...
  thres.fun = function(thr, err) median(thr[err == min(err)]),
  slim.fit = FALSE)

Arguments

x
 Dataset, numerical matrix with observations as rows.
y
 Class labels, factor.
error.fun
 Error function for tuning.
slim.fit
 Set to TRUE if you want to return the fitted classifier but discard pamr's cv.objects, which can be large. memory efficient. This means that the element cv$cv.objects containing the cross-validated fits will be dropped from the returned classifier.
cv
 Cross-validation scheme for shrinkage tuning. It should be supplied on one of the following forms:
  • Resampling scheme produced with resample or resample.holdout.
  • List with elements named nrep and nfold
  • NA, NULL or FALSE to suppress shrinkage tuning.
threshold
 Shrinkage thresholds to try (referred to as 'lambda' in the literature). Chosen and tuned automatically by default, but must be given by the user if not tuned (see the cv argument) if you wish to use it with batch.model.
...
 Sent to pamr.train.
thres.fun
 Threshold selection function. Note that it is not uncommon that several thresholds will result in the same tuning error.
Value

Fitted pamr classifier.

Author(s)

Christofer Bäcklin

See Also

emil, emil.predict.pamr, emil.vimp.pamr, modeling.procedure

Description

Wrapper for the MASS package implementation.

Usage

emil.fit.qda(x, y, ...)

Arguments

x Dataset, numerical matrix with observations as rows.
y Class labels, factor.
... Sent to qda.

Value

Fitted QDA.

Author(s)

Christofer Bäcklin

See Also

emil, emil.predict.qda, modeling.procedure
emil.fit.randomForest  Fit random forest.

Description
Directly calling the randomForest package implementation. See randomForest for parameter specification.

Usage
emil.fit.randomForest(x, y, importance = FALSE, ...)

Arguments
x  Dataset, numerical matrix with observations as rows.
y  Class labels, factor.
importance  Whether to calculate permuted OOB error as a variable importance measure, see Importance.randomForest. Set to FALSE to speed up computation.
...  Sent to randomForest.

Value
Fitted random forest.

Author(s)
Christofer Bäcklin

See Also
emil, emil.predict.randomForest, emil.vimp.randomForest, modeling.procedure

emil.predict.caret  Predict using a caret method

Description
This is not guaranteed to work with all caret methods. If it doesn’t work for a particular method, the user will need to rewrite it.

Usage
emil.predict.caret(...)

Arguments

Object... Sent to predict that forwards it to the appropriate predict function in the caret package.

Author(s)

Christofer Bäcklin

emil.predict.cforest  Predict with conditional inference forest

Description

Prediction function for models fitted with emil.fit.cforest.

Usage

emil.predict.cforest(object, x, at, ...)

Arguments

object  Fitted cforest classifier, as returned by emil.fit.cforest.

x      New data to be used for predictions.

at     Time point to evaluate survival curves at. If omitted it is set to the last observed time point.

...    Sent to treeresponse.

Value

The predicted chance of survival.

Author(s)

Christofer Bäcklin

See Also

emil, emil.fit.cforest, modeling.procedure
emil.predict.glmnet  
*Predict using generalized linear model with elastic net regularization*

**Description**

Due to the way `glmnet` is implemented, the regularization alpha cannot be modified after the model is fitted.

**Usage**

`emil.predict.glmnet(object, x, s, ...)`

**Arguments**

- `object`: Fitted model.
- `x`: New data to be predicted.
- `s`: Regularization parameter lambda.
- `...`: Sent to `predict.glmnet`.

**Value**

A list with elements:

- `pred`: Factor of predicted class memberships.
- `prob`: Data frame of predicted class probabilities.

**Author(s)**

Christofer Bäcklin

**See Also**

`emil, emil.fit.glmnet, modeling.procedure`

---

emil.predict.lda  
*Prediction using already trained prediction model*

**Description**

Wrapper for the MASS package implementation.

**Usage**

`emil.predict.lda(object, x, ...)`
emil.predict.lm

Arguments

object
Fitted classifier as produced by `batch.model`.

x
Dataset of observations to be classified.

Value
A list with elements:

• pred: Factor of predicted class memberships.
• prob: Data frame of predicted class probabilities.

Author(s)

Christofer Bäcklin

See Also

`emil`, `emil.fit.lda`, `modeling.procedure`

Description
Prediction using linear model

Usage

emil.predict.lm(object, x, ...)

Arguments

object
Fitted classifier produced by `emil.fit.lm`.

x
Dataset to be predicted upon.

... Sent to `predict.lda`.

Value
A list with elements:

• pred: Vector of predicted response.

Author(s)

Christofer Bäcklin
Prediction using nearest shrunken centroids.

Description

In case multiple thresholds give the same error the largest one is chosen (i.e. the one keeping the fewest features).

Usage

emil.predict.pamr(object, x, threshold, thres.fun, ...)

Arguments

- **object**: Fitted classifier.
- **x**: Dataset of observations to be classified.
- **threshold**: Threshold to use for classification. This argument is only needed if you want to override the value set during model fitting.
- **thres.fun**: Threshold selection function. Only needed if you want to override the function set during model fitting.
- **...**: Sent to `pamr.predict`.

Value

A list with elements:

- **pred**: Factor of predicted class memberships.
- **prob**: Data frame of predicted class probabilities.

Author(s)

Christofer Bäcklin

See Also

emil, emil.fit.pamr, emil.vimp.pamr, modeling.procedure
emil.predict.qda  Prediction using already trained classifier.

Description
Wrapper for the MASS package implementation.

Usage
emil.predict.qda(object, x, ...)

Arguments
- object: Fitted classifier as produced by `batch.model`.
- x: Dataset of observations to be classified.
- ...: Sent to `predict.qda`.

Value
A list with elements:
- pred: Factor of predicted class memberships.
- prob: Data frame of predicted class probabilities.

Author(s)
Christofer Bäcklin

See Also
emil, emil.fit.qda, modeling.procedure

emil.predict.randomForest  Prediction using random forest.

Description
Prediction using random forest.

Usage
emil.predict.randomForest(object, x, ...)
emil.vimp.pamr

Arguments

object  Fitted model.
\( x \)  Dataset of observations to be classified.
...  Ignored

Value

When used for classification, a list with elements:

- \( \text{pred} \): Factor of predicted class memberships.
- \( \text{prob} \): Data frame of predicted class probabilities.

When used for regression, a list with the element:

- \( \text{pred} \): Vector of predicted response.

Author(s)

Christofer Bäcklin

See Also

emil, emil.fit.randomForest, emil.vimp.randomForest, modeling.procedure

Description

Calculated as the absolute difference between the overall centroid and a class-wise shrunken centroid (which is the same for both classes except sign).

Usage

emil.vimp.pamr(object, threshold, thres.fun, ...)

Arguments

object  Fitted pamr classifier
threshold  Threshold to use for classification. This argument is only needed if you want to override the value set during model fitting.
thres.fun  Threshold selection function. Only needed if you want to override the function set during model fitting.
...  Sent to \texttt{pamr.predict}.  

---
Details
In case multiple thresholds give the same error the largest one is chosen (i.e. the one keeping the fewest features).

Value
A matrix of variable importance scores where the rows represent variables and the columns represent classes.

Author(s)
Christofer Bäcklin

See Also
emil, emil.fit.pamr, emil.predict.pamr, modeling.procedure

Description
Variable importance of random forest.

Usage
emil.vimp.randomForest(object, type = 1, ...)

Arguments
object Fitted randomForest classifier
type Importance can be assessed in two ways:
1. Permuted out-of-bag prediction error (default). This can only be used if the classifier was fitted with argument importance=TRUE which is default.
2. Total decrease in node impurity.
... Ignored.

Value
An importance vector with elements corresponding to variables.

Author(s)
Christofer Bäcklin
See Also

emil, emil.fit.randomForest, emil.predict.randomForest, modeling.procedure

error.fun

Performance estimation functions

Description

These functions determine the performance of fitted model based on its predictions. They are used both for evaluating whole modeling procedures and to tune model parameters, i.e. find the parameter values with the best performance. The parameter tuning routine is designed to minimize its error function (or optimization criteria), which is why functions that are to be maximized must have their sign changed, like neg.auc.

Usage

error.rate(true, pred)

neg.auc(true, pred)

rmse(true, pred)

mse(true, pred)

neg.harrell.C(true, pred)

Arguments

ture  The true response values, be it class labels, numeric values or survival outcomes.

pred  A prediction object.

Details

Custom performance estimation functions should be implemented as follows:

function(true, pred)

ture  A vector of true responses.

pred  Prediction returned from the prediction function.

In most cases the true response and the predictions are of the same type, e.g. true and fitted values in a regression or class labels in a classification problem, but it is not a requirement. An example of different types could be if the prediction function produce class probabilities for all classes rather than one label, or the risks that the observations will experience the event of interest, to be compared to the actual outcome that it did occur or has not yet occurred at a specific time point. See neg.harrell.C for an example of the latter.
Author(s)
Christofer Bäcklin

See Also
emil, neg.gmpa, modeling.procedure, emil.extensions

evaluate.modeling  Performance estimation of modeling procedures

Description
This function performs the important task of evaluating the performance of a modeling procedure with resampling, including tuning and pre-processing to not bias the results by information leakage.

Usage
evaluate.modeling(proc, x, y, ..., .save = list(fit = FALSE, pred = TRUE, vimp
   = FALSE, tuning = TRUE), .verbose = TRUE)

Arguments
proc  modeling procedure, or list of modeling procedures, as produced by modeling.procedure.
x   Dataset, observations as rows and descriptors as columns.
y   Response vector.
   ...  Sent to tune and batch.model.
   .save  See batch.model.
   .verbose  Whether to print an activity log.

Value
A list of fitted models.

Author(s)
Christofer Bäcklin

See Also
emil, modeling.procedure, fit, tune predict, vimp

Examples
proc <- modeling.procedure("lda")
cv <- resample("crossval", y=iris$Species, nfold=5, nrep=3)
perf <- evaluate.modeling(proc, x=iris[-5], y=iris$Species, resample=cv)
err <- subtree(perf, TRUE, "error")
factor.events

**Get events on factor form**

**Description**

By default all events are returned regardless of when they occurred. By setting `time` only the events that has occurred up until then will be returned, and cases with shorter follow-up times and no event will be marked as censored.

**Usage**

```r
factor.events(x, time, censor.label = "no event", keep)
```

**Arguments**

- `x`: Outcome vector.
- `time`: Time point to evaluate at.
- `censor.label`: What to label the absence of an event with.
- `keep`: Event types to keep, defaults to all.

**Value**

A factor of events.

**Author(s)**

Christofer Bäcklin

---

fill

**Replace values with something else**

**Description**

Replace values with something else

**Usage**

```r
fill(x, pattern, replacement, invert = FALSE)
```

```r
na.fill(x, replacement)
```
Arguments

x Variable containing NAs.
pattern The values in x to be replaced. Can also be a function.
replacement The value which is to replace the values matching pattern.
invert Whether to fill all values except the ones matching pattern.

Value

An imputed version of x.

Author(s)

Christofer Bäcklin

Examples

fill(1:10, function(x) x %% 2 == 1, 0)
na.fill(c(1,2,NA,4,5), 3)

Description

Fits a model according to a modeling procedure. If the procedure contains untuned variables they it will automatically be tuned prior to fitting. Note however that the tuning statistics will not be return.

Usage

fit(proc, x, y, ..., .verbose)

Arguments

proc modeling procedure, or list of modeling procedures, as produced by modeling.procedure.
x Dataset, observations as rows and descriptors as columns.
y Response vector.
... Sent to tune, in case tuning is required, which will pass them on to batch.model.
.verbose Whether to print an activity log.

Value

A list of fitted models.

Author(s)

Christofer Bäcklin
See Also

emil, modeling.procedure, evaluate.modeling, tune.predict, vimp

Examples

```r
proc <- modeling.procedure("lda")
mod <- fit(proc, x=iris[-5], y=iris$Species)
```

```r
image(resample)
```

**Description**

Class specific extension to `image`.

**Usage**

```r
## S3 method for class 'resample'
image(x, col, ...)
```

```r
## S3 method for class 'crossval'
image(x, col, ...)
```

**Arguments**

- `x` Resampling scheme, as returned by `resample`.
- `col` Color palette matching the values of `x`. Can also be the response vector used to create the scheme for automatic coloring.
- `...` Sent to `plot`.

**Value**

Nothing, produces a plot.

**Author(s)**

Christofer Bäcklin

**See Also**

emil, resample

**Examples**

```r
image(resample("holdout", 60, frac=1/3, nfold=20))
```

```r
y <- gl(2, 30)
image(resample("crossval", y, nfold=3, nrep=8), col=y)
```
impute

Regular imputation

Description
If you want to impute, build model and predict you should use `pre.impute.median` or `pre.impute.knn`. This function imputes using all observations without caring about cross-validation folds.

Usage
```r
impute.knn(x, k = 0.05, distmat = "auto")
impute.median(x)
```

Arguments
- **x**: Dataset.
- **k**: Number of nearest neighbors to use.
- **distmat**: Distance matrix.

Details
For additional information on the parameters see `pre.impute.knn` and `pre.impute.median`.

Value
An imputed matrix.

Author(s)
Christofer Bäcklin

See Also
`emil, pre.process, pre.impute.knn, pre.impute.median`
index.fit

Convert a fold to row indexes of fitting or test set

Description

Convert a fold to row indexes of fitting or test set

Usage

index.fit(fold, allow.oversample = TRUE)

index.test(fold)

Arguments

fold
A fold of a resampling scheme.
allow.oversample
Whether or not to allow individual observation to exist in multiple copies in the training set. This is typically not the case, but can be used when a class is underrepresented in the data set.

Value

An integer vector of row indexes.

Author(s)

Christofer Bäcklin

See Also

e Emil, resample

integer.events

Return events in integer form

Description

Basically calls factor.events and converts to integer. No event is coded with 0 and the other event types with > 0.

Usage

integer.events(x, ...)


is.blank

Arguments

x Outcome vector.
...

Sent to factor.events.

Value

Integer vector.

Author(s)

Christofer Bäcklin

Description

This is mainly an internal function but as other dependent packages also use it sometimes and it
generally is quite handy to have it is exported for public use.

Usage

is.blank(x, false.triggers = FALSE)

Arguments

x A variable.
false.triggers Whether FALSE should be considered as empty.

Value

Logical telling if variable is blank.

Author(s)

Christofer Bäcklin

Examples

is.blank(NULL)
**is.na.outcome**

*Check for missing values*

---

**Description**

Check for missing values

**Usage**

```r
## S3 method for class 'outcome'
is.na(x)
```

**Arguments**

- `x` : Outcome vector.

**Value**

A logical vector with `TRUE` where an outcome is missing.

**Author(s)**

Christofer Bäcklin

---

**is.outcome**

*Test if object is of class outcome*

---

**Description**

Test if object is of class outcome

**Usage**

```r
is.outcome(x)
```

**Arguments**

- `x` : Object.

**Value**

`TRUE` if `x` is of class `outcome`. `FALSE` otherwise.

**Author(s)**

Christofer Bäcklin
length.outcome  

Length of an outcome vector

Description

Length of an outcome vector

Usage

```r
## S3 method for class 'outcome'
length(x)
```

Arguments

- `x`: Outcome vector.

Value

Length.

Author(s)

Christofer Bäcklin

modeling.procedure  

Setup a modeling procedure

Description

A modeling procedure is an object containing all information necessary to carry out and evaluate the performance of a predictive modeling task with `fit`, `tune`, or `evaluate.modeling`. To use an out-of-the-box algorithm with default values, only the `method` argument needs to be set. See `emil` for a list of available methods. To deviate from the defaults, e.g. by tuning variables or using a custom function for model fitting, set the appropriate parameters as described below. For a guide on how to implement a custom method see the documentation page `emil.extensions`.

Usage

```r
modeling.procedure(method, param = list(), error.fun = NULL, fit.fun, predict.fun, vimp.fun)
```
Arguments

- **method**: The name of the modeling method. Only needed to identify plug-in functions, i.e. if you supply them yourself there is no need to set method.
- **param**: A list of model parameters. These will be fed to the fitting function after the dataset (x and y parameters). To tune a parameter, supply the candidate values in a vector or list.

When tuning more than one parameter, all combinations of parameter values will be tested, if the elements of param are named. To manually specify which parameter value combinations to try, leave the the elements unnamed (see example 3 and 4).

Parameters that should have vectors or lists as values, e.g. `trControl` when using `emil.fit.caret` to train caret models, must be wrapped in an additional list. That is, to set a parameter value to a list, but not tune it, make it a list of length 1 containing the list to be used (see example 6).

- **fit.fun**: The function to be used for model fitting.
- **predict.fun**: The function to be used for model prediction.
- **vimp.fun**: The function to be used for calculating or extracting variable importance scores.
- **error.fun**: Performance measure used to evaluate modeling procedures and to tune parameters. See `error.fun` for details.

Value

An object of class `modeling.procedure`.

Author(s)

Christofer Bäcklin

See Also

`emil`, `evaluate.modeling`, `fit`, `tune`, `predict`, `vimp`

Examples

```r
# 1: Fit linear discriminants without tuning any parameter, # since it has none modeling.procedure("lda")

# 2: Tune random forest's 'mtry' parameter, with 3 possible values modeling.procedure("randomForest", list(mtry = list(100, 250, 1000)))

# 3: Tune random forest's 'mtry' and 'maxnodes' parameters simultaneously, # with 3 values each, testing all 9 possible combinations modeling.procedure("randomForest", list(mtry = list(100, 250, 1000), maxnodes = list(5, 10, 25)))

# 4: Tune random forest's 'mtry' and 'maxnodes' parameters simultaneously, # but only test 3 manually specified combinations of the two
```
modeling.procedure("randomForest", list(mtry = 100, maxnodes = 5),
                   list(mtry = 250, maxnodes = 10),
                   list(mtry = 1000, maxnodes = 25)))

# 5: Tune elastic net's 'alpha' and 'lambda' parameters. Since elastic net's
# fitting function can tune 'lambda' internally in a more efficient way
# than the general framework is able to do, only tune 'alpha' and pass all
# 'lambda' values as a single argument.
modeling.procedure("glmnet", list(alpha = seq(0, 1, length.out=6),
                                 lambda = list(seq(0, 5, length.out=30))))

# 6: Train elastic nets using the caret package's model fitting framework
library(caret)
modeling.procedure("caret", list(method = "glmnet",
                                 trainControl = list(trainControl(verboselter = TRUE, classProbs = TRUE))))

---

neg.gmpa  

**Negative geometric mean of class specific predictive accuracy**

**Description**

When dealing with imbalanced classification problem, i.e. where the class sizes are very different, small classes tend to be overlooked when tuning parameters by optimizing error rate. Blagus and Lusa (2013) suggested to remedy the problem by using this performance measure instead.

**Usage**

neg.gmpa(true, pred)

**Arguments**

- **true**  
  See error.fun.
- **pred**  
  See error.fun.

**Value**

A numeric scalar.

**Author(s)**

Christofer Bäcklin

**References**

Blagus, R., & Lusa, L. (2013). *Improved shrunken centroid classifiers for high-dimensional class-imbalanced data*. BMC bioinformatics, 14, 64. doi:10.1186/1471-2105-14-64

**See Also**

error.fun
nice.require

Load a package and offer to install if missing

Description

If running R in interactive mode, the user is prompted.

Usage

nice.require(pkg, reason = "is required")

Arguments

pkg          Package name.
reason       A status message that informs the user why the package is needed.

Value

Nothing

Author(s)

Christofer Bäcklin

Examples

nice.require("base", "is required to do anything at all")

outcome

Create a vector of outcomes

Description

Heavily modeled after the `Surv` class in the ‘survival’ package. Objects of this class are internally stored as data frames but should be thought of as vectors and can be treated as such through out.

Usage

outcome(time, event, levels, censor)

Arguments

time       Time points at which an event occurred.
event      The type of event that occurred. `NA` codes for no event.
levels     Which levels of event to keep. Defaults to all unique values of event.
censor     What values of event should be considered censoring. Defaults to `NA` and all values in event not present in levels.
Value
A vector of outcomes.

Author(s)
Christofer Bäcklin

See Also
factor.events, integer.events, plot.outcome.

Examples
outcome(runif(15), sample(c(NA, "Mechanical failure", "Out of fuel"), 15, TRUE))

| p.value | Extraction of p-value from a statistical test |

Description
These calculations are written in such a way that they avoid rounding off errors that plague the survival and cmprsk packages.

Usage
p.value(x, log.p = FALSE, ...)

Arguments
x Test, i.e. a fitted object of a supported type.
log.p Whether to return the logarithm of the p-value.
... Sent to class method.

Value
p-value.

Author(s)
Christofer Bäcklin

See Also
p.value.crr, p.value.survdiff, p.value.cuminc
p.value.coxph

---

**p.value.coxph**

*Extract p-value from a Cox proportional hazards model*

### Description

Based on `summary.coxph`.

### Usage

```r
## S3 method for class 'coxph'
p.value(x, log.p = FALSE, test = c("logrank", "wald", "likelihood"), ...)
```

### Arguments

- `x` Fitted `coxph` model.
- `log.p` Whether to return the logarithm of the p-value.
- `test` What test to calculate. "likelihood" is short for means likelihood ratio test.
- `...` Ignored. Kept for S3 consistency.

### Value

p-value.

### Author(s)

Christofer Bäcklin

### See Also

- `p.value`

---

p.value.crr

---

**p.value.crr**

*Extracts p-value from a competing risk model*

### Description

Extracts p-value from a competing risk model

### Usage

```r
## S3 method for class 'crr'
p.value(x, log.p = FALSE, ...)  
```
p.value.cuminc

Arguments

x   Fitted crr model, as returned by crr.
log.p   Whether to return the logarithm of the p-value.
...   Ignored. Kept for S3 consistency.

Value

Two-sided p-value.

Author(s)

Christofer Bäcklin

See Also

p.value

Examples

library(cmpsk)
time <- 1:20
event <- c(rep(0, 9), rep(2, 3), rep(1, 8))
data <- rep(0:1, each=10)
x <- crr(time, event, data)

# Compare p-values of implementations
print(x)
p.value(x)

--------------------------------------------------

Extract p-value from a cumulative incidence estimation

Description

This is also known as Gray’s test.

Usage

## S3 method for class 'cuminc'
p.value(x, log.p = FALSE, ...)

Arguments

x   Fitted cuminc estimate.
log.p   Whether to return the logarithm of the p-value.
...   Ignored. Kept for S3 consistency.
**p.value.survdiff**

**Value**

p-value.

**Author(s)**

Christofer Bäcklin

**See Also**

p.value

---

**p.value.survdiff**  
*Extracts p-value from a logrank test*

**Description**

Extracts p-value from a logrank test

**Usage**

```r
## S3 method for class 'survdiff'
p.value(x, log.p = FALSE, ...)
```

**Arguments**

- `x`  
  Logrank test result, as returned by `survdiff`.  
- `log.p`  
  Whether to return the logarithm of the p-value.  
- `...`  
  Ignored. Kept for S3 consistency.

**Value**

p-value.  

```r
library(survival)
y <- Surv(time=1:100, event=rep(1:0, each=50))
groups <- rep(1:2, each=50)
x <- survdiff(y ~ groups)
# Compare p-values of implementations
print(x) p.value(x)
```

**Author(s)**

Christofer Bäcklin

**See Also**

p.value
### plot.outcome

*Plot outcome vector*

**Description**

Plot outcome vector

**Usage**

```r
## S3 method for class 'outcome'
plot(x, y, segments = TRUE, flip = FALSE,
     legendpos = "topright", ...)
```

**Arguments**

- `x`: outcome vector.
- `y`: Y-values.
- `segments`: Whether to draw horizontal segments.
- `flip`: Flip the plot to show time on y.
- `legendpos`: Position of legend, see `legend`. Set to NA or NULL to suppress legend.
- `...`: Sent to `plot`.

**Author(s)**

Christofer Bäcklin

### pre.impute.knn

*kNN imputation*

**Description**

Nearest neighbor methods needs to have a distance matrix of the dataset it works on. When doing repeated model fittings on subsets of the entire dataset it is unnecessary to recalculate it every time, therefore this function requires the user to manually calculate it prior to resampling and supply it in a wrapper function.

**Usage**

```r
pre.impute.knn(x, y, fold, k = 0.05, distmat)
```
Arguments

x      Dataset.
y      Response vector.
fold   A logical vector with FALSE for fitting observations, TRUE for test observations and NA for observations not to be included.
k      Number of nearest neighbors to calculate mean from. Set to < 1 to specify a fraction.
distmat Distance matrix. A matrix, dist object or "auto". Notice that "auto" will recalculate the distance matrix in each fold, which is only meaningful in case the features of x vary between folds. Otherwise you are just wasting time.

Author(s)

Christofer Bäcklin

Examples

x <- iris[-5]
x[sample(nrow(x), 30), ] <- NA
my.dist <- dist(x)
evaluate.modeling(modeling.procedure("lda"), x=x, y=iris$Species,
                   pre.process=function(...) pre.impute.knn(..., k=4, my.dist))

---

pre.pamr  PAMR adapted dataset pre-processing

Description

The predict framework is designed to work with dataset where rows correspond to observations and columns to descriptors. PAMR wants it the other way, and also to have the fitting set response vector supplied in a list with the descriptors. This function applies a standard pre-processing function and then reformats the result to satisfy PAMR.

Usage

pre.pamr(x, y, fold, pre.process = pre.split, ...)

Arguments

x      Dataset.
y      Response vector.
fold   A logical vector with FALSE for fitting observations, TRUE for test observations and NA for observations not to be included.
pre.process A pre-processing function to be wrapped.
...     Sent to pre.process.
Value

A list with fitting and testing sets, formatted the way pamr wants them.

Author(s)

Christofer Bäcklin

See Also

emil, pre.process

Description

These functions are run in batch.model just prior to model fitting and serve two purposes. 1) They extract fitting and test sets from the entire dataset and 2) they can at the same time apply a transformation to pre-process the data for handling missing values, scaling, compression etc. They can also be used to modify the form of the data, if required by the fitting function, e.g. pre.pamr that transposes the dataset to make it compatible with the pamr classification method.

Usage

pre.split(x, y, fold)
pre.center(x, y, fold)
pre.scale(x, y, fold, scale = TRUE)
pre.impute.median(x, y, fold)

Arguments

x        Dataset.
y        Response vector.
fold     A logical vector with FALSE for fitting observations, TRUE for test observations and NA for observations not to be included.
scale    Whether to scale each feature to have standard deviation = 1.

Details

Note that all transformations are defined based on the fitting data only and then applied to both fitting set and test set. It is important to not let the test data in any way be part of the model fitting, including the preprocessing, to not risk information leakage and biased results!

The imputation functions can also be used outside of the resampling scheme, see impute.
**Value**

A list with the following components

- **fit** Fitting set.
- **test** Test set.
- **features** Logical vector indicating which features were kept (TRUE) and discarded (FALSE).
  This is only set in case of variable selection.

**Author(s)**

Christofer Bäcklin

**See Also**

-emil-, pre.impute.knn

**Examples**

```r
# A splitter that only keeps variables with a class-wise mean difference > 'd'
my.split <- function(x, y, fold, d=2)
  fit.idx <- index.fit(fold)
  test.idx <- index.test(fold)
  class.means <- sapply(
    split(x[fit.idx,, drop=FALSE], y[fit.idx]),
    sapply, mean, na.rm=TRUE)
  diff.feats <- apply(class.means, 1, function(x) diff(range(x))) > d
  return(list(
    fit = list(x = x[fit.idx, diff.feats, drop=FALSE],
               y = y[fit.idx]),
    test = list(x = x[test.idx, diff.feats, drop=FALSE],
                y = y[test.idx]),
    features = diff.feats))

# Use it during modeling
proc <- modeling.procedure("lda")
perf <- evaluate.modeling(proc, x = iris[-5], y = iris$Species,
                           pre.process = my.split)

# Example of how the end user can change the 'd' parameter,
# without redefining the function
perf <- evaluate.modeling(proc, x = iris[-5], y = iris$Species,
                           pre.process = function(...) my.split(..., d = 1.3))
```
**predict.modeling.procedure**

*Predict the response of unknown observations*

### Description

Predict the response of unknown observations

### Usage

```r
## S3 method for class 'modeling.procedure'
predict(object, model, x, ...)
```

### Arguments

- `object` Modeling procedure.
- `model` Fitted model.
- `x` Data set with observations whose response is to be predicted.
- `...` Sent to the procedure’s prediction function.

### Value

See the documentation of procedure’s method.

### Author(s)

Christofer Bäcklin

### See Also

`emil`, `modeling.procedure`, `evaluate.modeling`, `fit`, `tune`, `vimp`

### Examples

```r
proc <- modeling.procedure("lda")
mod <- fit(proc, x=iris[-5], y=iris$Species)
pred <- predict(proc, mod, iris[-5])
```
print.outcome

Print outcome vector

Description

Print outcome vector

Usage

## S3 method for class 'outcome'
print(x, quote = FALSE, ...)

Arguments

x 
Outcome vector.

quote 
Logical, whether to print quotation marks.

... 
Ignored, kept for S3 consistency.

Value

Nothing, only prints the vector to stdout.

Author(s)

Christofer Bäcklin

resample

Resampling schemes

Description

Performance evaluation and variable tuning use resampling methods to estimate the performance of models. These are defined by resampling schemes, which are data frames where each column corresponds to a division of the data set into mutually exclusive training and test sets. Repeated hold out and cross-validation are two methods to create such schemes.

Usage

resample(method, y, ..., subset = TRUE)

resample.holdout(y = NULL, frac = 0.5, nfold = 5,
    balanced = is.factor(y), subset)

resample.crossval(y, nfold = 5, nrep = 5, balanced = is.factor(y), subset)
Arguments

**method**
The resampling method to use, e.g. "holdout" or "crossval".

**y**
Observations to be divided. Can either be supplied as the response of the observations themselves, or as a scalar which is interpreted as the number of objects.

... 
Sent to the method specific function, e.g. "resample.holdout".

**nfold**
Number of folds.

**balanced**
Whether the sets should be balanced or not, i.e. if the class ratio over the sets should be kept constant (as far as possible).

**subset**
Which objects in y that are to be divided and which that are not to be part of neither set. If subset is a resampling scheme, a list of inner cross-validation schemes will be returned.

**frac**
Fraction of objects to hold out (0 < frac < 1).

**nrep**
Number of fold sets to generate.

Details

Note that when setting up analyzes, the user should not call resample.holdout or resample.crossval directly, as resample performs additional necessary processing of the scheme.

Resampling scheme can be visualized in a human digestible form with the `image` function.

Functions for generating custom resampling schemes should be implemented as follows and then called by `resample("myMethod", ...):

```r
resample.myMethod <- function(y, ..., subset)
```

**y** Response vector.

... Method specific attributes.

**subset** Indexes of observations to be excluded for the resampling.

The function should return a list of the following elements:

**folds** A data frame with the folds of the scheme that conforms to the description in the 'Value' section below.

**param** A list with the parameters necessary to generate such a resampling scheme. These are needed when creating subschemes needed for parameter tuning, see `subresample`.

Value

A data frame defining a resampling scheme. TRUE or a positive integer codes for training set and FALSE or 0 codes for test set. Positive integers > 1 code for multiple copies of an observation in the training set. NA codes for neither training nor test set and is used to exclude observations from the analysis altogether.

Author(s)

Christofer Bäcklin
resample.mapply

See Also

emil, subresample, image.resample.index.fit

Examples

```r
resample("holdout", 50, frac=1/3)
resample("holdout", factor(runif(60) >= .5))
y <- factor(runif(60) >= .5)
cv <- resample("crossval", y)
image(cv, main="Cross-validation scheme")
```

---

resample.mapply \hspace{1cm} Compare true response to resampled predictions

Description

This function can be used to compare a true response vector to predictions returned from `evaluate.modeling`. For each fold, the correct subset of the true response vector is extracted and fed to a given function together with the matching predictions.

Usage

```r
resample.mapply(fun, resample, true, pred, ...)
```

Arguments

- `fun` Function to run.
- `resample` Resampling scheme (see `resample`).
- `true` True response vector.
- `pred` Predictions, as returned from `evaluate.modeling`.
- `...` Sent to `mapply`.

Author(s)

Christofer Bäcklin

Examples

```r
proc <- modeling.procedure("lda")
ho <- resample("holdout", iris$Species, frac=1/3, nfold=4)
perf <- evaluate.modeling(proc, iris[-5], iris$Species, resample=ho)
confusion.tables <- resample.mapply(
  function(truth, prediction) table(truth, prediction$pred$pred),
  ho, iris$Species, pred=perf, SIMPLIFY=FALSE)
Reduce("+", confusion.tables)
```
subframe

**Extract and organize predictions according to a resampling scheme**

**Description**
This function arranges predictions of a performance evaluation in a data frame where the rows correspond do observations and the columns to folds, to make it easy to study the variability of each observation with respect to the resampling.

**Usage**
```
subframe(x, ..., resample)
```

**Arguments**
- **x**: Performance evaluation results.
- **...**: Indexes specify what to extract, sent to `subtree`.
- **resample**: Resampling scheme used to carry out a performance evaluation.

**Author(s)**
Christofer Bäcklin

**See Also**
- `subtree`

**Examples**
```
proc <- modeling.procedure("lda")
cv <- resample("crossval", y=iris$Species, nfold=5, nrep=3)
perf <- evaluate.modeling(proc, x=iris[-5], y=iris$Species, resample=cv)
subframe(perf, TRUE, "pred", "prob", 1, resample=cv)
```

---

**subresample**

**Generate resampling subschemes**

**Description**
A subscheme is a resampling scheme that only includes observations in the training set of an original scheme. This function automatically fetches the type and parameters of the prototype and use them to generate the subscheme.

**Usage**
```
subresample(fold, y)
```
Arguments

fold A resampling scheme or fold to use to define the sub scheme(s).
y The observations used to create the resampling scheme. See resample for details.

Value

A resampling scheme.

Author(s)

Christofer Bäcklin

See Also

emil, resample

Examples

cv <- resample("holdout", y=12, frac=1/4, nfold=3)
inner.cv <- subresample(cv, y=12)

subtree Extract a subset of a tree of nested lists

Description

Many functions of the package produce results on the form of nested list, like evaluate.modeling. Use this function to extract only a subset of the tree. Also note the similar function subframe.

Usage

subtree(x, i, ..., error = NULL, simplify = TRUE)

Arguments

x List of lists.
i Indexes to extract on the first level of the tree. Can also be a function that will be applied to the downstream result of the function.
... Indexes to extract on subsequent levels.
error A function to be called if there is an error when parsing x, or a value to replace erroneous elements with, see the examples.
simplify Whether to collapse lists of length one (TRUE) or preserve the original tree structure (FALSE).
Details

This function can only be used to extract data, not to assign.

Value

A subset of the list tree.

Author(s)

Christofer Bäcklin

See Also

subframe

Examples

\[
\begin{align*}
&l \leftarrow \text{list}(A=\text{list}(a=0:2, b=3:4, c=023-22030), \\
&B=\text{list}(a=5:7, b=8:9)) \\
&\text{subtree}(l, 1:2, "b") \\
&\text{subtree}(l, \text{TRUE}, \text{mean}, "a") \\
&\text{subtree}(l, \text{TRUE}, \text{exp}, "c", \text{error}=\text{browser}) \\
&\text{subtree}(l, \text{TRUE}, \text{exp}, "c", \text{error}=\text{NA})
\end{align*}
\]

---

trace.msg  

Print a timestamped and indented log message

Description

Print a timestamped and indented log message

Usage

trace.msg(level = 1, \ldots, time = \text{TRUE}, linebreak = \text{TRUE}, file = "")

Arguments

- \texttt{level}  
  Indentation level.
- \texttt{\ldots}  
  Sent to \texttt{sprintf}.
- \texttt{time}  
  Whether or not to print timestamp.
- \texttt{linebreak}  
  Whether to finish the message with a linebreak or not.
- \texttt{file}  
  Sent to \texttt{cat}.

Author(s)

Christofer Bäcklin
tune

Tune parameters of modeling procedures

Description

These functions are rarely needed to be called manually as they are automatically called by fit and evaluate.modeling when needed.

Usage

tune(proc, ..., .retune = FALSE, .verbose = FALSE)

is.tuned(proc)

is.tunable(proc)

detune(proc)

Arguments

proc      modeling procedure, or list of modeling procedures, as produced by modeling.procedure.
...       Sent to batch.model.
.retune   Whether to retune already tuned processes.
.verbose   Whether to print an activity log.

Value

A tuned modeling procedures or a list of such.
Logical indicating if the procedure(s) are tuned.
Logical indicating if the has tunable parameters.
A list of untuned modeling procedures.

Author(s)

Christofer Bäcklin
See Also

emil, modeling.procedure, evaluate.modeling, fit, predict.vimp

Examples

```r
proc <- modeling.procedure("randomForest", param=list(mtry=1:4))
tuned.proc <- tune(proc, x=iris[-5], y=iris$Species)
mod <- fit(tuned.proc, x=iris[-5], y=iris$Species)
```

vimp

Variable importance of a fitted model

Description

Note that different methods calculates variable importance in different ways and that they are not directly comparable.

Usage

vimp(object, model, ...)

Arguments

- `object` Modeling procedure.
- `model` Fitted model.
- `...` Sent on to the procedure’s variable importance scoring function.

Value

A vector of length p or an p-x-c matrix of variable importance scores where p is the number of descriptors and c is the number of classes.

Author(s)

Christofer Bäcklin

See Also

emil

Examples

```r
proc <- modeling.procedure("randomForest")
mod <- fit(proc, x=iris[-5], y=iris$Species)
vimp(proc, mod)
```
warn.once  

Print a warning message if not printed earlier

Description
To avoid flooding the user with identical warning messages, this function keeps track of which have already been shown.

Usage

warn.once(id, ...)  
reset.warn.once()

Arguments

id Warning message id. This is used internally to refer to the message.  
... Sent to `warning`.

Author(s)

Christofer Bäcklin

weighted.error.rate  
Weighted error rate

Description
If different types of errors are associated with different costs a weighted error function might be more appropriate than the standard.

Usage

weighted.error.rate(x)

Arguments

x Cost matrix or factor response vector.

Details
This function is not in itself an error function, but used to generate error functions. Either supply a predefined cost matrix or a response vector for a classification problem to define it automatically. The automatically generated cost matrix will generate an error of 0 if all predictions are correct, 1 if all predictions are incorrect and 0.5 if all predictions are the same (regardless of class, i.e. if one class is smaller it will be given a higher misclassification cost).
Value
An error function.

Author(s)
Christofer Bäcklin

[.outcome

Description
Extract

Usage
### S3 method for class 'outcome'
x[i, j, drop = FALSE]

Arguments
- x: Outcome vector.
- i: Index.
- j: Column index, if given x is treated as a data frame.
- drop: See Extract

Value
A subset of x.

Author(s)
Christofer Bäcklin
# Index

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>.outcome, 58</td>
<td></td>
</tr>
<tr>
<td>as.character.outcome, 3</td>
<td></td>
</tr>
<tr>
<td>as.data.frame.outcome, 4</td>
<td></td>
</tr>
<tr>
<td>as.matrix.outcome, 4</td>
<td></td>
</tr>
<tr>
<td>as.outcome, 5</td>
<td></td>
</tr>
<tr>
<td>as.outcome.Surv, 5</td>
<td></td>
</tr>
<tr>
<td>as.Surv, 6, 7</td>
<td></td>
</tr>
<tr>
<td>as.Surv.outcome, 6</td>
<td></td>
</tr>
<tr>
<td>as.Surv.Surv, 7</td>
<td></td>
</tr>
<tr>
<td>batch.model, 8, 8, 10, 17, 22, 24, 28, 30, 46, 55</td>
<td></td>
</tr>
<tr>
<td>cat, 54</td>
<td></td>
</tr>
<tr>
<td>cforest, 11, 14</td>
<td></td>
</tr>
<tr>
<td>cforest_control, 14</td>
<td></td>
</tr>
<tr>
<td>coxph, 41</td>
<td></td>
</tr>
<tr>
<td>crr, 42</td>
<td></td>
</tr>
<tr>
<td>cuminc, 42</td>
<td></td>
</tr>
<tr>
<td>cv.glmmnet, 15</td>
<td></td>
</tr>
<tr>
<td>detune (tune), 55</td>
<td></td>
</tr>
<tr>
<td>dim.outcome, 9</td>
<td></td>
</tr>
<tr>
<td>dist, 45</td>
<td></td>
</tr>
<tr>
<td>emil, 9, 10, 13–28, 31–33, 36, 37, 46–48, 51, 53, 56</td>
<td></td>
</tr>
<tr>
<td>emil.extensions, 11, 12, 28, 36</td>
<td></td>
</tr>
<tr>
<td>emil.fit.caret, 13, 37</td>
<td></td>
</tr>
<tr>
<td>emil.fit.cforest, 14, 20</td>
<td></td>
</tr>
<tr>
<td>emil.fit.glmmnet, 15, 15, 21</td>
<td></td>
</tr>
<tr>
<td>emil.fit.lda, 16, 22</td>
<td></td>
</tr>
<tr>
<td>emil.fit.lm, 16, 22, 23</td>
<td></td>
</tr>
<tr>
<td>emil.fit.pamr, 12, 17, 23, 26</td>
<td></td>
</tr>
<tr>
<td>emil.fit.qda, 18, 24</td>
<td></td>
</tr>
<tr>
<td>emil.fit.randomForest, 19, 25, 27</td>
<td></td>
</tr>
<tr>
<td>emil.predict.caret, 19</td>
<td></td>
</tr>
<tr>
<td>emil.predict.cforest, 14, 20</td>
<td></td>
</tr>
<tr>
<td>emil.predict.glmmnet, 15, 21</td>
<td></td>
</tr>
<tr>
<td>emil.predict.lda, 16, 21</td>
<td></td>
</tr>
<tr>
<td>emil.predict.lm, 17, 22</td>
<td></td>
</tr>
<tr>
<td>emil.predict.pamr, 18, 23, 26</td>
<td></td>
</tr>
<tr>
<td>emil.predict.qda, 18, 24</td>
<td></td>
</tr>
<tr>
<td>emil.predict.randomForest, 19, 24, 27</td>
<td></td>
</tr>
<tr>
<td>emil.vimp.pamr, 18, 23, 25</td>
<td></td>
</tr>
<tr>
<td>emil.vimp.randomForest, 19, 25, 26</td>
<td></td>
</tr>
<tr>
<td>error.fun, 9–11, 13, 27, 37, 38</td>
<td></td>
</tr>
<tr>
<td>error.rate, 11</td>
<td></td>
</tr>
<tr>
<td>error.rate (error.fun), 27</td>
<td></td>
</tr>
<tr>
<td>evaluate.modeling, 8, 10, 12, 13, 28, 31, 36, 37, 48, 51, 53, 55, 56</td>
<td></td>
</tr>
<tr>
<td>Extract, 58</td>
<td></td>
</tr>
<tr>
<td>factor.events, 29, 33, 34, 40</td>
<td></td>
</tr>
<tr>
<td>family, 15</td>
<td></td>
</tr>
<tr>
<td>fill, 29</td>
<td></td>
</tr>
<tr>
<td>fit, 8, 10, 12, 28, 30, 36, 37, 48, 55, 56</td>
<td></td>
</tr>
<tr>
<td>glmnet, 11, 15, 21</td>
<td></td>
</tr>
<tr>
<td>image, 31, 50</td>
<td></td>
</tr>
<tr>
<td>image.crossval (image.resample), 31</td>
<td></td>
</tr>
<tr>
<td>image.resample, 10, 31, 51</td>
<td></td>
</tr>
<tr>
<td>importance.randomForest, 19</td>
<td></td>
</tr>
<tr>
<td>impute, 11, 32, 46</td>
<td></td>
</tr>
<tr>
<td>index.fit, 33, 51</td>
<td></td>
</tr>
<tr>
<td>index.test (index.fit), 33</td>
<td></td>
</tr>
<tr>
<td>integer.test (index.fit), 33</td>
<td></td>
</tr>
<tr>
<td>integer.events, 33, 40</td>
<td></td>
</tr>
<tr>
<td>is.blank, 34</td>
<td></td>
</tr>
<tr>
<td>is.na.outcome, 35</td>
<td></td>
</tr>
<tr>
<td>is.outcome, 35</td>
<td></td>
</tr>
<tr>
<td>is.tunable (tune), 55</td>
<td></td>
</tr>
<tr>
<td>is.tuned (tune), 55</td>
<td></td>
</tr>
<tr>
<td>lapply, 10</td>
<td></td>
</tr>
<tr>
<td>lda, 11, 16</td>
<td></td>
</tr>
<tr>
<td>legend, 44</td>
<td></td>
</tr>
<tr>
<td>length.outcome, 36</td>
<td></td>
</tr>
<tr>
<td>lm, 11, 16</td>
<td></td>
</tr>
<tr>
<td>mapply, 51</td>
<td></td>
</tr>
</tbody>
</table>
mcMap, 8
modeling.procedure, 8–11, 14–28, 30, 31, 36, 48, 55, 56
mse, 11
mse (error.fun), 27
na.fill (fill), 29
neg.auc, 11
neg.auc (error.fun), 27
neg.gmpa, 11, 28, 38
neg.harrell.C, 11
neg.harrell.C (error.fun), 27
nice.require, 39
outcome, 5, 6, 39
p.value, 40, 41–43
p.value.coxph, 41
p.value.crr, 40, 41
p.value.cuminc, 40, 42
p.value.survdiff, 40, 43
pamr, 11
pamr.predict, 23, 25
pamr.train, 17
parLapply, 8
plot, 31, 44
plot.outcome, 40, 44
pre.center, 11
pre.center (pre.process), 46
pre.impute.knn, 11, 32, 44, 47
pre.impute.median, 11, 32
pre.impute.median (pre.process), 46
pre.pamr, 12, 45, 46
pre.process, 10–13, 32, 46, 46
pre.scale, 11
pre.scale (pre.process), 46
pre.split, 11
pre.split (pre.process), 46
predict, 10, 20, 28, 31, 37, 56
predict.glmnet, 21
predict.lm, 22
predict.modeling.procedure, 48
predict.qda, 24
print.outcome, 49
qda, 11, 18
randomForest, 11, 19
resample, 10–13, 17, 31, 33, 49, 51, 53
resample.crossval, 11
resample.holdout, 11, 17
resample.maply, 51
reset.warn.once (warn.once), 57
rmse, 11
rmse (error.fun), 27
sapply, 10
sprintf, 54
subframe, 10, 52, 53, 54
subresample, 50, 51, 52
subtree, 10, 52, 53
summary.coxph, 41
Surv, 6, 7, 39
survdiff, 43
trace.msg, 54
tune, 8–10, 12, 13, 28, 30, 31, 36, 37, 48, 55
vimp, 10, 28, 31, 37, 48, 56, 56
warn.once, 57
warning, 57
weighted.error.rate, 11, 57