

Package ‘emil’

July 9, 2015

Encoding UTF-8

Type Package

Title Evaluation of Modeling without Information Leakage

Version 2.1.1

Date 2015-07-09

Author Christofer Backlin [aut, cre],
Mats Gustafsson [aut]

Maintainer Christofer Backlin <emil@christofer.backlin.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.

Depends dplyr

Imports data.table, ggplot2, graphics, grDevices, lazyeval, magrittr,
methods, stats, tidy, utils

Suggests caret, cmprsk, glmnet, Hmisc, MASS, parallel, party, pamr,
randomForest, RColorBrewer, rpart, survival, testthat (>=
0.9.1)

License GPL (>= 2)

LazyLoad yes

LazyData yes

URL <http://molmed.github.io/emil>

BugReports <https://github.com/Molmed/emil/issues>

NeedsCompilation no

Repository CRAN

Date/Publication 2015-07-09 14:47:42

R topics documented:

as.modeling_procedure	3
dichotomize	4
emil	5
error_fun	7
evaluate	8
extension	10
fill	12
fit	13
fit_caret	14
fit_cforest	14
fit_coxph	15
fit_glmnet	16
fit_lda	17
fit_lm	18
fit_pamr	18
fit_qda	19
fit_randomForest	20
fit_rpart	21
get_color	21
get_importance	22
get_performance	23
get_prediction	24
get_response	24
get_tuning	25
image.resample	26
importance_glmnet	27
importance_pamr	27
importance_randomForest	28
impute	29
indent	30
index_fit	30
is_blank	31
is_multi_procedure	32
learning_curve	32
list_method	33
log_message	34
modeling_procedure	35
name_procedure	36
na_index	37
neg_gmpa	37
nice_axis	38
nice_box	39
nice_require	39
notify_once	40
plot.learning_curve	40
plot.Surv	41

predict.model	42
predict_caret	43
predict_cforest	43
predict_coxph	44
predict_glmnet	44
predict_lda	45
predict_lm	46
predict_pamr	47
predict_qda	48
predict_randomForest	48
predict_rpart	49
pre_impute	50
pre_impute_knn	50
pre_pamr	51
pre_process	52
print.preprocessed_data	54
pvalue	55
pvalue.coxph	56
pvalue.crr	56
pvalue.cuminc	57
pvalue.survdiff	58
resample	59
roc_curve	60
select_list	62
subresample	63
subtree	64
tune	66
vlines	67
weighted_error_rate	68

Index 69

as.modeling_procedure *Coerce to modeling procedure*

Description

Coerce to modeling procedure

Usage

```
as.modeling_procedure(x, ...)
```

Arguments

x	modeling procedure.
...	Ignored (kept for S3 consistency).

Value

Modeling procedure

Author(s)

Christofer Bäcklin

Examples

```
as.modeling_procedure("lda")
```

dichotomize

Dichotomize time-to-event data

Description

Convert time-to-event data (typically created with the [Surv](#) function) to factor or integer.

Usage

```
dichotomize(x, time, to_factor)
```

Arguments

x	Surv vector.
time	Time point to dichotomize at.
to_factor	Depending on the type of x the return value may be integer or factor. Set this argument to explicitly state the return type.

Details

If no time point is given the observation times will be stripped, leaving only the event types. If a time point is given observations with events occurring before time will be labelled by their event type, observations with events occurring after time will be labelled as “no event”, and observations censored before time will be considered as missing information.

Value

Integer vector or factor.

Author(s)

Christofer Bäcklin

See Also

[Surv](#)

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 feature importance scores.

[error_fun](#) Performance estimation functions used to tune parameters and evaluate performance of modeling procedures.

Solving modeling problems:

[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.

[evaluate](#) Evaluate the performance of a procedure using resampling.

[learning_curve](#) Learning curve analysis.

Managing the results of modeling problems:

[get_prediction](#) Extract predictions from resampled modeling results.

[get_tuning](#) Extract feature importance scores of a fitted model or resampled modeling results.

[get_importance](#) Extract feature importance scores of a fitted model or resampled modeling results.

[subtree](#) Extracts results from the output of [evaluate](#). It is essentially a recursive version of [lapply](#) and [sapply](#).

[select](#) Interface between emil and the [dplyr](#) package for data manipulation. Can be used to subset modeling results, reorganize or summarize to help interpretation or prepare for plotting.

Methods included in the package

Resampling methods: See [resample](#) for information on usage and implementation of custom methods.

[resample_holdout](#) Repeated holdout.

[resample_crossvalidation](#) 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: The following modeling methods are included in the emil package. For a complete list of available methods in both the emil package and other loaded packages, please use [list_method](#). See [modeling_procedure](#) for information on usage and [extension](#) for information on implementation of custom methods.

[cforest](#) Conditional inference forest.

[coxph](#) Cox proportional hazards model.

[glmnet](#) Elastic net.

[lasso](#) LASSO.

[lda](#) Linear discriminant.

[lm](#) Linear model.

[pamr](#) Nearest shrunken centroids.

[qda](#) Quadratic discriminant.

[randomForest](#) Random forest.

[ridge_regression](#) Ridge regression.

[rpart](#) Decision trees.

It is also possible to incorporate any method from the ‘caret’ package by using the function [fit_caret](#).

To search for emil compatible methods in all attached packages use the [list_method](#) function.

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 parameters, 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. To plot the ROC curves see [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.

Plotting: Plotting is not the one of the main aims of the package and the methods that do exist mainly serves as examples for how to write your own. These exists for:

- [Learning curve analyses.](#)
- [Resampling schemes.](#)
- [ROC-curves.](#)

Author(s)

Christofer Bäcklin

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(truth, prediction, allow_rejection = !missing(rejection_cost),
           rejection_cost)
```

```
neg_auc(truth, prediction)
```

```
rmse(truth, prediction, na.rm = FALSE)
```

```
mse(truth, prediction, na.rm = FALSE)
```

```
neg_harrell_c(truth, prediction, na.rm = FALSE)
```

Arguments

truth	The true response values, be it class labels, numeric values or survival outcomes.
prediction	A prediction object.
allow_rejection	If FALSE missing prediction values will produce an error. If TRUE missing values will be given a cost specified by the rejection_cost argument.
rejection_cost	See the argument allow_rejection. If missing a rejection cost equivalent to the error rate obtained when assigning all test observations to the most common class will be used.
na.rm	Whether to remove missing values or not.

Details

Custom performance estimation functions should be implemented as follows:

```
function(truth, prediction)
```

truth A vector of true responses.

prediction 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](#), [extension](#)

evaluate

Evaluate a modeling procedure

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(procedure, x, y, resample, pre_process = pre_split, .save = c(model
  = TRUE, prediction = TRUE, error = TRUE, importance = FALSE), .cores = 1,
  .checkpoint_dir = NULL, .return_error = .cores > 1,
  .verbose = getOption("emil_verbose", TRUE))
```

Arguments

procedure	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 evaluate to assess the performance of the whole modeling_procedure.

<code>pre_process</code>	Function that performs pre-processing and splits dataset into fitting and test subsets.
<code>.save</code>	What parts of the modeling results to return to the user. If <code>importance</code> is <code>FALSE</code> variable importance calculation will be skipped.
<code>.cores</code>	Number of CPU-cores to use for parallel computation. The current implementation is based on <code>mCMap</code> , which unfortunately do not work on Windows systems. It can however be re-implemented by the user fairly easily by setting up a <code>PSOCK</code> cluster and calling <code>parLapply</code> as in the example below. This solution might be included in future versions of the package, after further investigation.
<code>.checkpoint_dir</code>	Directory to save intermediate results to, after every completed fold. The directory will be created if it doesn't exist, but not recursively.
<code>.return_error</code>	If <code>FALSE</code> the entire modeling is aborted upon an error. If <code>TRUE</code> the modeling of the particular fold is aborted and the error message is returned instead of its results.
<code>.verbose</code>	Whether to print an activity log.

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.

`prediction` Predictions given by the model.

`importance` Feature importance scores.

`tune` Results from the parameter tuning. See [tune](#) for details.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [modeling_procedure](#)

Examples

```
x <- iris[-5]
y <- iris$Species
cv <- resample("crossvalidation", y, nfold = 4, nrepeat = 4)
result <- evaluate("lda", x, y, resample=cv)

# Multiple procedures fitted and tested simultaneously.
# This is useful when the dataset is large and the splitting takes a long time.
# If you name the elements of the list emil will also name the elements of the
# results object in the same way.
```

```

result <- evaluate(c(Linear = "lda", Quadratic = "qda"), x, y, resample=cv)

# Multicore parallelization (on a single computer)
result <- evaluate("lda", x, y, resample=cv, .cores=2)

# Parallelization using a cluster (not limited to a single computer)
# PSOCK is supported on windows too!
require(parallel)
cl <- makePSOCKcluster(2)
clusterEvalQ(cl, library(emil))
clusterExport(cl, c("x", "y"))
result <- parLapply(cl, cv, function(fold)
  evaluate("lda", x, y, resample=fold))

```

extension

Extending the emil framework with user-defined methods

Description

This page describes how to implement custom methods compatible with the functions of the emil framework, most notably [fit](#), [tune](#), and [evaluate](#). 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. Optional

```
function(x, y, p1, p2, p3, ..., .verbose)
```

x The features (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 [fit_pamr](#) for an example of a function that does not take its data in the default way.

y A response vector. This is the outcome you want to model, e.g. the feature 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.

p1, p2, p3, ... (Optional) Method-specific model parameters. These will all be tunable with the [tune](#) and [evaluate](#) functions. Note that you can give them any name you want, the names used here are just an example.

.verbose (Optional) Indentation level of log messages. Feed this to [log_message](#).

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

```
function(object, x, ...)
```

`object` A fitted model produced by the model fitting function described above.

`x` Observations to make predictions on (describing features only).

`...` Parameters to the prediction functions. These are ignored by [tune](#) and [evaluate](#), but could be convenient if the user wants to work with it manually.

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 feature importance scores

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

```
function(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](#), 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 features in the data set and c is the number of classes.

Calculating performance

See [error_fun](#).

Resampling schemes

See [resample](#).

Pre-processing functions

See [pre_process](#).

Code style guidelines

Names of functions, arguments and variables should be written in underscore separated lower case, singular form, unabbreviated, and American English. Users are encouraged to also follow use this style when wrining extensions. However, the guidelines may be violated in cases where they break the consistency with an incorporated well established package, see for example `fit_randomForest` which according to the guidelines should be `fit_randomforest` or `fit_random_forest`.

A few exceptions to the rule against abbreviations exists, namely ‘fun’ and ‘function’ and ‘dir’ for ‘directory’. These are only used for arguments to indicate the type of value that is accepted.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [error_fun](#), [pre_process](#), [resample](#)

fill

Replace values with something else

Description

Replace values with something else

Usage

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

```
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)
```

fit*Fit a model*

Description

Fits a model according to a modeling procedure. If the procedure contains untuned parameters they will automatically be tuned prior to fitting.

Usage

```
fit(procedure, x, y, ..., .verbose = getOption("emil_verbose", FALSE))
```

Arguments

procedure	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 evaluate .
.verbose	Whether to print an activity log. Set to -1 to suppress all messages.

Value

A list of fitted models.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [modeling_procedure](#), [evaluate](#), [tune](#), [predict](#), [get_importance](#)

Examples

```
mod <- fit("lda", x=iris[-5], y=iris$Species)
```

`fit_caret`*Fit a model using the **caret** package*

Description

Fit a model using the **caret** package

Usage

```
fit_caret(x, y, ...)
```

Arguments

<code>x</code>	Descriptors.
<code>y</code>	Response.
<code>...</code>	Sent to train .

Author(s)

Christofer Bäcklin

`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

```
fit_cforest(x, y, formula = y ~ ., ctrl_fun = party::cforest_unbiased, ...)
```

Arguments

<code>x</code>	Dataset, observations as rows and descriptors as columns.
<code>y</code>	Responses.
<code>formula</code>	Formula linking response to descriptors.
<code>ctrl_fun</code>	Which control function to use, see cforest_control .
<code>...</code>	Sent to the function specified by <code>ctrl_fun</code> .

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](#), [predict_cforest](#), [modeling_procedure](#)

`fit_coxph`*Fit Cox proportional hazards model*

Description

Fit Cox proportional hazards model

Usage

```
fit_coxph(x, y, formula = y ~ ., ...)
```

Arguments

<code>x</code>	Dataset.
<code>y</code>	Response. Required if formula is missing.
<code>formula</code>	See coxph .
<code>...</code>	Sent to coxph .

Value

Fitted Cox proportional hazards model.

Author(s)

Christofer Bäcklin

See Also

[predict_coxph](#)

Examples

```

require(survival)
data(ovarian)
model <- fit(
  modeling_procedure(
    method = "coxph",
    parameter = list(formula = list(Surv(futime, fustat) ~ age)),
    x = ovarian, y = NULL
  )
)
predict(model, ovarian[11:16,])

```

fit_glmnet

Fit elastic net, LASSO or ridge regression model

Description

Using the **glmnet** package implementation.

Usage

```
fit_glmnet(x, y, family, nfolds, foldid, alpha = 1, lambda = NULL, ...)
```

```
fit_ridge_regression(...)
```

```
fit_lasso(...)
```

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 fit_glmnet or cv.glmnet .

Details

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

Value

Fitted elastic net model.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [predict_glmnet](#), [importance_glmnet](#), [modeling_procedure](#)

fit_lda

Fit linear discriminant

Description

Wrapper for the MASS package implementation.

Usage

```
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](#), [predict_lda](#), [modeling_procedure](#)

fit_lm	<i>Fit a linear model fitted with ordinary least squares</i>
--------	--

Description

Based on [lm](#).

Usage

```
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](#), [predict_lm](#), [modeling_procedure](#)

fit_pamr	<i>Fit nearest shrunken centroids model.</i>
----------	--

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

```
fit_pamr(x, y, error_fun, cv, nfold, threshold = NULL, ...,  
        thres_fun = function(thr, err) median(thr[err == min(err)]), slim = FALSE)
```

Arguments

x	Dataset, numerical matrix with observations as rows.
y	Class labels, factor.
error_fun	Error function for tuning.
cv	Cross-validation scheme for shrinkage tuning. It should be supplied on one of the following forms: <ul style="list-style-type: none">• Resampling scheme produced with resample or resample_holdout.• List with elements named <code>nrepeat</code> and <code>nfold</code>• NA, NULL or FALSE to suppress shrinkage tuning.
nfold	Sent to pamr.cv . Only used if <code>cv</code> is missing.
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 <code>cv</code> argument) if you wish to use it with evaluate .
...	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.
slim	Set to TRUE if you want to return the fitted classifier but discard <code>pamr</code> 's <code>cv</code> objects, which can be large. memory efficient. This means that the element <code>cv\$cv</code> objects containing the cross-validated fits will be dropped from the returned classifier.

Value

Fitted `pamr` classifier.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [predict_pamr](#), [importance_pamr](#), [modeling_procedure](#)

fit_qda

Fit quadratic discriminant.

Description

Wrapper for the **MASS** package implementation.

Usage

```
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](#), [predict_qda](#), [modeling_procedure](#)

fit_randomForest *Fit random forest.*

Description

Directly calling the **randomForest** package implementation. See [randomForest](#) for parameter specification.

Usage

```
fit_randomForest(x, y, ...)
```

Arguments

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

Value

Fitted random forest.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [predict_randomForest](#), [importance_randomForest](#), [modeling_procedure](#)

fit_rpart	<i>Fit a decision tree</i>
-----------	----------------------------

Description

Fit a decision tree

Usage

```
fit_rpart(x, y, ...)
```

Arguments

x	Data set (features).
y	Response.
...	Sent to rpart .

Value

A fitted decision tree.

Author(s)

Christofer Bäcklin

get_color	<i>Get color palettes</i>
-----------	---------------------------

Description

Can be used to modify an existing palette, e.g. change brightness, or to generate a palette for a response vector.

Usage

```
get_color(x, ...)  
  
## Default S3 method:  
get_color(x, s, v, alpha, ...)  
  
## S3 method for class 'factor'  
get_color(x, levels = FALSE, col = "Set1", ...)
```

Arguments

x	Character vector of colors or factor of class memberships to generate colors for.
...	Sent to <code>get_color.default</code> .
s	Saturation. $s = 0$ leaves it unchanged, $0 < s \leq 1$ increases, and $-1 \leq s < 0$ decreases.
v	Value. $s = 0$ leaves it unchanged, $0 < s \leq 1$ increases, and $-1 \leq s < 0$ decreases.
alpha	Transparency.
levels	If TRUE a palette with one color per level of x is returned. If FALSE one color per element in x is returned.
col	Color palette with one color per class or the name of the color brewer palette to use, see name argument of <code>brewer.pal</code> for a list of possible values.

Value

A character vector of hex colors.

Author(s)

Christofer Bäcklin

get_importance	<i>Feature (variable) importance of a fitted model</i>
----------------	--

Description

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

Usage

```
get_importance(object, format, ...)
```

Arguments

object	Fitted model.
format	Table format of the output. See http://en.wikipedia.org/wiki/Wide_and_narrow_data for more info.
...	Sent on to the procedure's feature importance scoring function.

Details

When extending the **emil** framework with your own method, the importance function should return a data frame where one column is called "feature" and the remaining columns are named after the classes.

Value

A vector of length p or an $p \times c$ matrix of feature importance scores where p is the number of descriptors and c is the number of classes.

Author(s)

Christofer Bäcklin

See Also

[emil](#)

Examples

```
procedure <- modeling_procedure("pamr")
model <- fit("pamr", x=iris[-5], y=iris$Species)
get_importance(model)

cv <- resample("crossvalidation", iris$Species, nrepeat=2, nfold=3)
result <- evaluate("pamr", iris[-5], iris$Species, resample=cv,
  .save=c(importance=TRUE))
get_importance(result)
```

get_performance	<i>Extract prediction performance</i>
-----------------	---------------------------------------

Description

Extract prediction performance

Usage

```
get_performance(result, format = c("wide", "long"))
```

Arguments

result	Modeling result, as returned by evaluate .
format	Table format of the output. See http://en.wikipedia.org/wiki/Wide_and_narrow_data for more info.

Value

Data frame.

Author(s)

Christofer Bäcklin

get_prediction	<i>Extract predictions from modeling results</i>
----------------	--

Description

Extract predictions from modeling results

Usage

```
get_prediction(result, resample, type = "prediction", format = c("long",  
  "wide"))
```

Arguments

result	Modeling result, as returned by evaluate and evaluate .
resample	Resampling scheme used to create the results.
type	The type of prediction to return. The possible types vary between modeling procedure.
format	Table format of the output. See http://en.wikipedia.org/wiki/Wide_and_narrow_data for more info.

Value

A data frame where the id column refers to the observations.

Author(s)

Christofer Bäcklin

get_response	<i>Extract the response from a data set</i>
--------------	---

Description

Extract the response from a data set

Usage

```
get_response(x, y)
```

Arguments

x	Data set features.
y	Response vector or any other type of objects that describe how to extract the response vector from x.

Value

A response vector.

Author(s)

Christofer Bäcklin

Examples

```
identical(iris$Species, get_response(iris, "Species"))
identical(iris$Sepal.Length, get_response(iris, Sepal.Length ~ .))
```

get_tuning

Extract parameter tuning statistics

Description

Extract parameter tuning statistics

Usage

```
get_tuning(object)
```

Arguments

object Fitted model or modeling procedure

Value

A data frame of tuning statistics in long format.

Author(s)

Christofer Bäcklin

Examples

```
procedure <- modeling_procedure("randomForest",
  parameter = list(mtry = c(1, 3),
    nodesize = c(4, 10)))
model <- fit(procedure, x=iris[-5], y=iris$Species)
get_tuning(model)

options(emil_max_indent=4)
ho <- resample("holdout", iris$Species, nfold=5)
result <- evaluate(procedure, iris[-5], iris$Species, resample=ho,
  .save=c(model=TRUE))
get_tuning(result)
```

image.resample *Visualize resampling scheme*

Description

Class specific extension to [image](#).

Usage

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

```
image(resample("holdout", 60, test_fraction=1/3, nfold=20))  
  
y <- gl(2, 30)  
image(resample("crossvalidation", y, nfold=3, nrepeat=8), col=y)
```

importance_glmnet	<i>Feature importance extractor for elastic net models</i>
-------------------	--

Description

Feature importance extractor for elastic net models

Usage

```
importance_glmnet(object, s, ...)
```

```
importance_ridge_regression(object, s, ...)
```

```
importance_lasso(object, s, ...)
```

Arguments

object	Fitted elastic net model, as produced by fit_glmnet .
s	Regularization parameter lambda.
...	Sent to predict.glmnet .

Value

A feature importance data frame.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_glmnet](#), [predict_glmnet](#), [modeling_procedure](#)

importance_pamr	<i>Feature importance of nearest shrunken centroids.</i>
-----------------	--

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

```
importance_pamr(object, threshold, thres_fun = max, ...)
```

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 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 data frame of feature importance scores.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_pamr](#), [predict_pamr](#), [modeling_procedure](#)

importance_randomForest

Feature importance of random forest.

Description

Feature importance of random forest.

Usage

```
importance_randomForest(object, type = 1, ...)
```

Arguments

object	Fitted randomForest classifier
type	Importance can be assessed in two ways: <ol style="list-style-type: none"> 1. Permuted out-of-bag prediction error (default). This can only be used if the classifier was fitted with argument prediction=TRUE which is default. 2. Total decrease in node impurity.
...	Ignored.

Value

An prediction vector with elements corresponding to variables.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_randomForest](#), [predict_randomForest](#), [modeling_procedure](#)

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

```
impute_knn(x, k = 0.05, distance_matrix = "auto")
```

```
impute_median(x)
```

Arguments

x	Dataset.
k	Number of nearest neighbors to use.
distance_matrix	Distance matrix.

Details

For additional information on the parameters see [pre_impute_knn](#) and [pre_impute](#).

Value

An imputed matrix.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [pre_process](#), [pre_impute_knn](#), [pre_impute_median](#)

Examples

```
x <- matrix(rnorm(36), 6, 6)
x[sample(length(x), 5)] <- NA
impute_knn(x)
impute_median(x)
```

indent	<i>Increase indentation</i>
--------	-----------------------------

Description

Increase indentation

Usage

```
indent(base, indent)
```

Arguments

base	Base indentation level of the function printing the message.
indent	Extra indentation of this message.

Value

An integer that can be used to specify the indentation level of messages printed with [log_message](#).

Author(s)

Christofer Bäcklin

index_fit	<i>Convert a fold to row indexes of fitting or test set</i>
-----------	---

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

[emil](#), [resample](#)

is_blank

Wrapper for several methods to test if a variable is empty

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_multi_procedure	<i>Detect if modeling results contains multiple procedures</i>
--------------------	--

Description

Detect if modeling results contains multiple procedures

Usage

```
is_multi_procedure(result)
```

Arguments

result	Modeling results, as returned by evaluate .
--------	---

Value

Logical scalar.

Author(s)

Christofer Bäcklin

learning_curve	<i>Learning curve analysis</i>
----------------	--------------------------------

Description

This function studies the change in performance as the sizes of the training set is varied. In case the studied modeling procedures cannot produce models on the smallest training sets, please use `.return_error=TRUE` (see [evaluate](#)).

Usage

```
learning_curve(procedure, x, y, test_fraction, nfold = 100, ...,
               .verbose = TRUE)
```

Arguments

procedure	modeling_procedure .
x	Dataset descriptors.
y	Response.
test_fraction	Fraction of dataset to hold out, i.e. use as test set. Defaults 20 logarithmically distributed values ranging from all but 5 observations per class in the largest test set to only 5 observations per class in the smallest test set.

nfold How many holdout folds that should be calculated.
 ... Sent to [evaluate](#).
 .verbose Whether to print an activity log. Set to -1 to also suppress output generated from the procedure's functions.

Author(s)

Christofer Bäcklin

References

Richard O Duda, Peter E Hart, and David G Stork. Pattern Classification. Wiley, 2nd edition, 2000. ISBN 978-0-471-05669-0.

Examples

```
options(emil_max_indent=3)
lc <- learning_curve(c(Linear="lda", Quadratic="qda"),
                    iris[-5], iris$Species, test_fraction=7:2/10)
plot(lc)
```

list_method

List all available methods

Description

This function searches all attached packages for methods compatible with the **emil** framework.

Usage

```
list_method(pos = search())
```

Arguments

pos Location to search in, see [ls](#).

Value

A data frame.

Author(s)

Christofer Bäcklin

Examples

```
list_method()
```

log_message	<i>Print a timestamped and indented log message</i>
-------------	---

Description

To suppress messages below a given indentation level set the global [option](#) setting `emil_max_indent`, as in the example below.

Usage

```
log_message(indent = 1, ..., time = TRUE, domain = "R-emil",
  appendLF = TRUE)
```

Arguments

<code>indent</code>	Indentation level. Messages with <code>indent=0</code> are suppressed.
<code>...</code>	Sent to sprintf .
<code>time</code>	Whether or not to print timestamp.
<code>domain</code>	See message .
<code>appendLF</code>	Whether to finish the message with a linebreak or not.

Author(s)

Christofer Bäcklin

Examples

```
equipment <- c("flashlight", "snacks", "pick")
{
  log_message(1, "Begin descent")
  log_message(2, "Oh no, forgot the %s!", sample(equipment, 1))
  log_message(2, "Hello? Can you throw it down to me?", time=FALSE)
  log_message(1, "Aw shucks, I'm coming back up.")
}

for(verbose in c(TRUE, FALSE)){
  cat("It's", verbose, "\n")
  for(i in 0:3)
    log_message(indent(verbose, i), "Down")
}

options(emil_max_indent = 2)
for(i in 1:3)
  log_message(i, "Down")
```

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`. 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 parameters 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 documentaion page [extension](#).

Usage

```
modeling_procedure(method, parameter = list(), error_fun = NULL, fit_fun,
  predict_fun, importance_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.
parameter	<p>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.</p> <p>When tuning more than one parameter, all combinations of parameter values will be tested, if the elements of parameter are named. To manually specify which parameter value combinations to try, leave the the elements unnamed (see example 3 and 4).</p> <p>Parameters that should have vectors or lists as values, e.g. <code>trControl</code> when using <code>fit_caret</code> to train <code>pkgcaret</code> 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).</p>
error_fun	Performance measure used to evaluate procedures and to tune parameters. See error_fun for details.
fit_fun	The function to be used for model fitting.
predict_fun	The function to be used for model prediction.
importance_fun	The function to be used for calculating or extracting feature importances. See get_importance for details.

Value

An object of class `modeling_procedure`.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [evaluate](#), [fit](#), [tune](#), [predict](#), [get_importance](#)

Examples

```
# 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(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",
                                  trControl = list(trainControl(verboseIter = TRUE, classProbs = TRUE))))
```

name_procedure

Get names for modeling procedures

Description

Get names for modeling procedures

Usage

```
name_procedure(procedure)
```

Arguments

procedure List of modeling procedures.

Value

A character vector of suitable non-duplicate names.

Author(s)

Christofer Bäcklin

na_index

Support function for identifying missing values

Description

Support function for identifying missing values

Usage

```
na_index(data)
```

Arguments

data Fitting and testing data sets, as returned by [pre_split](#).

Value

Data frame containing row and column indices of missing values or NULL if the data doesn't contain any.

Author(s)

Christofer Bäcklin

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(truth, prediction, na.rm = FALSE)
```

Arguments

truth	See error_fun .
prediction	See error_fun .
na.rm	Whether to remove missing values or not.

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_axis	<i>Plots an axis the way an axis should be plotted.</i>
-----------	---

Description

Plots an axis the way an axis should be plotted.

Usage

```
nice_axis(..., las = 1, lwd = 0, lwd.ticks = par("lwd"), lend = 2)
```

Arguments

...	Sent to axis .
las	Rotation of axis labels. Always horizontal by default.
lwd	Width of the line drawn along the plot area. Omitted by default since it overlaps with box and causes it to look thicker where the axis is.
lwd.ticks	Width of the tick lines. These are kept by default.
lend	Line endings, see par .

Author(s)

Christofer Bäcklin

nice_box	<i>Plots a box around a plot</i>
----------	----------------------------------

Description

Plots a box around a plot

Usage

```
nice_box(lend = 2, ljoin = 1, ...)
```

Arguments

lend	Line ending style, see par . Defaults to square.
ljoin	Line joint style, see par . Defaults to mitre, i.e. 90 degree corners in this case.
...	Sent to box .

Author(s)

Christofer Bäcklin

nice_require	<i>Load a package and offer to install if missing</i>
--------------	---

Description

If running R in interactive mode, the user is prompted for installing missing packages. If running in batch mode an error is thrown.

Usage

```
nice_require(pkg, reason)
```

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")
```

notify_once	<i>Print a warning message if not printed earlier</i>
-------------	---

Description

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

Usage

```
notify_once(id, ..., fun = log_message)

reset_notification(id, if_top_level = TRUE)
```

Arguments

id	Warning message id. This is used internally to refer to the message.
...	Sent to warning .
fun	Function to display the notification with. Typical choices are message or warning .
if_top_level	If TRUE the notifications will only be reset if <code>reset_notification</code> was called from a top-level function call. This behaviour prevents the notifications from being reset multiple times during nested calls to functions such as fit and evaluate .

Author(s)

Christofer Bäcklin

plot.learning_curve	<i>Plot results from learning curve analysis</i>
---------------------	--

Description

Plot results from learning curve analysis

Usage

```
## S3 method for class 'learning_curve'
plot(x, ..., summaries = list(mean = mean,
  `95-percentile` = function(x) quantile(x, 0.95)))
```


Arguments

x	Results from learning_curve .
...	Ignored, kept for S3 consistency.
summaries	Named list of summary functions that can reduce a vector of performance estimates to a single quantity.

Value

A [ggplot](#) object.

Author(s)

Christofer Bäcklin

plot.Surv	<i>Plot Surv vector</i>
-----------	-------------------------

Description

Plot Surv vector

Usage

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

Arguments

x	Surv 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

predict.model	<i>Predict the response of unknown observations</i>
---------------	---

Description

Predict the response of unknown observations

Usage

```
## S3 method for class 'model'  
predict(object, x, ..., .verbose = FALSE)
```

Arguments

object	Fitted model.
x	Data set with observations whose response is to be predicted.
...	Sent to the procedure's prediction function.
.verbose	Whether to print an activity log.

Value

See the documentation of procedure's method.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [modeling_procedure](#), [evaluate](#), [fit](#), [tune](#), [get_importance](#)

Examples

```
mod <- fit("lda", x=iris[-5], y=iris$Species)  
prediction <- predict(mod, iris[-5])
```

predict_caret	<i>Predict using a caret method</i>
---------------	--

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

```
predict_caret(object, x, ...)
```

Arguments

object	Fitted caret model.
x	New data to predict the response of.
...	Sent to predict that forwards it to the appropriate predict function in the caret package.

Author(s)

Christofer Bäcklin

predict_cforest	<i>Predict with conditional inference forest</i>
-----------------	--

Description

Prediction function for models fitted with [fit_cforest](#).

Usage

```
predict_cforest(object, x, at, ...)
```

Arguments

object	Fitted cforest classifier, as returned by 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](#), [fit_cforest](#), [modeling_procedure](#)

predict_coxph

Predict using Cox proportional hazards model

Description

Predict using Cox proportional hazards model

Usage

```
predict_coxph(object, x, ...)
```

Arguments

object	Fitted model, as returned by fit_coxph .
x	Observations whose response is to be predicted.
...	Sent to predict.coxph .

Author(s)

Christofer Bäcklin

See Also

[fit_coxph](#)

predict_glmnet

Predict using generalized linear model with elastic net regularization

Description

Due to the way [glmnet](#) is implemented, the regularization alpha can not be modified after the model is fitted.

Usage

```
predict_glmnet(object, x, s, ...)
```

```
predict_ridge_regression(object, x, s, ...)
```

```
predict_lasso(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 a subset of the following elements:

prediction	The response of the modeling problem, i.e. a factor for classification, problems, a numeric for regressions, and a relative risk for survival analyses.
probability	Data frame of predicted class probabilities.
link	Link function values.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_glmnet](#), [importance_glmnet](#), [modeling_procedure](#)

predict_lda

Prediction using already trained prediction model

Description

Wrapper for the **MASS** package implementation.

Usage

```
predict_lda(object, x, ...)
```

Arguments

object	Fitted classifier as produced by evaluate .
x	Dataset of observations to be classified.
...	Sent to predict_lda .

Value

A list with elements:

- prediction: Factor of predicted class memberships.
- probability: Data frame of predicted class probabilities.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_lda](#), [modeling_procedure](#)

predict_lm

Prediction using linear model

Description

Prediction using linear model

Usage

```
predict_lm(object, x, ...)
```

Arguments

object	Fitted classifier produced by fit_lm .
x	Dataset to be predicted upon.
...	Sent to predict.lm

Value

A list with elements:

- prediction: Vector of predicted response.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_lm](#), [modeling_procedure](#)

predict_pamr *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

```
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 <code>pamr.predict</code> .

Value

A list with elements:

- `prediction`: Factor of predicted class memberships.
- `probability`: Data frame of predicted class probabilities.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_pamr](#), [importance_pamr](#), [modeling_procedure](#)

predict_qda *Prediction using already trained classifier.*

Description

Wrapper for the **MASS** package implementation.

Usage

```
predict_qda(object, x, ...)
```

Arguments

object	Fitted classifier as produced by evaluate .
x	Dataset of observations to be classified.
...	Sent to predict.qda .

Value

A list with elements:

- prediction: Factor of predicted class memberships.
- probability: Data frame of predicted class probabilities.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_qda](#), [modeling_procedure](#)

predict_randomForest *Prediction using random forest.*

Description

Prediction using random forest.

Usage

```
predict_randomForest(object, x, ...)
```


Arguments

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

Value

When used for classification, a list with elements:

- prediction: Factor of predicted class memberships.
- probability: Data frame of predicted class probabilities.

When used for regression, a list with the element:

- prediction: Vector of predicted response.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [fit_randomForest](#), [importance_randomForest](#), [modeling_procedure](#)

predict_rpart	<i>Predict using a fitted decision tree</i>
---------------	---

Description

Predict using a fitted decision tree

Usage

```
predict_rpart(object, x)
```

Arguments

object	Fitted decision tree.
x	New data whose response is to be predicted.

Value

Predictions. The exact form depends on the type of application (classification or regression)

Author(s)

Christofer Bäcklin

pre_impute	<i>Basic imputation</i>
------------	-------------------------

Description

This solution is optimized for the scenario that the dataset is very large but only contains missing values in a small number of columns.

Usage

```
pre_impute(data, fun, ...)
```

```
pre_impute_median(data)
```

```
pre_impute_mean(data)
```

Arguments

data	Fitting and test datasets, as returned by pre_split or any other standard pre-processing function.
fun	Function for calculating imputation values. Should take a vector and return a scalar.
...	Sent to fun.

Value

A pair of fitting and testing datasets.

Author(s)

Christofer Bäcklin

pre_impute_knn	<i>Nearest neighbors imputation</i>
----------------	-------------------------------------

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

```
pre_impute_knn(data, k = 0.05, distance_matrix)
```

Arguments

data	Fitting and testing data sets, as returned by pre_split .
k	Number of nearest neighbors to calculate mean from. Set to < 1 to specify a fraction.
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.

Details

Features with fewer than k non-missing values will be removed automatically.

Author(s)

Christofer Bäcklin

Examples

```
x <- iris[-5]
x[sample(nrow(x), 30), 3] <- NA
my.dist <- dist(x)
evaluate(modeling_procedure("lda"), x = x, y = iris$Species,
  pre_process = function(...){
    pre_split(...) %>% pre_impute_knn(k = 4, distance_matrix = 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(data)
```

Arguments

data	Fitting and testing data sets, as returned by pre_split .
------	---

Details

pre_pamr must be run last if chained with other pre-processing functions, since it substantially reshapes the data.

Value

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

Author(s)

Christofer Bäcklin

See Also

[emil](#), [pre_process](#)

pre_process

Data preprocessing

Description

These functions are run in [evaluate](#) just prior to model fitting, to extract fitting and test sets from the entire dataset and apply transformations to pre-process the data (for handling missing values, scaling, compression etc.). They can also be used to adapt the form of the data to a specific 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_convert(data, x_fun, y_fun, ...)
```

```
pre_transpose(data)
```

```
pre_remove(data, feature)
```

```
pre_center(data, y = FALSE, na.rm = TRUE)
```

```
pre_scale(data, y = FALSE, na.rm = TRUE, center = TRUE)
```

```
pre_remove_constant(data)
```

```
pre_remove_correlated(data, cutoff)
```

```
pre_pca(data, ncomponent, scale. = TRUE, ...)
```

Arguments

x	Dataset.
y	Response vector.
fold	A logical or numeric vector with TRUE or positive numbers for fitting observations, FALSE or 0 for test observations, and NA for observations not to be included.
data	Fitting and testing data sets, as returned by pre_split .
x_fun	Function to apply to the descriptors of the datasets (e.g. x). This function will be applied independently to the fitting and testing sets.
y_fun	Function to be applied to the response of the training and test sets (independently).
...	Sent to internal methods, see the code of each function.
feature	The features to be removed. Can be integer, logical or character.
na.rm	A logical value indicating whether NA values should be ignored.
center	Whether to center the data before scaling.
cutoff	See findCorrelation .
ncomponent	Number of PCA components to use. Missing all components are used.
scale.	Sent to prcomp .

Details

When supplied to [evaluate](#), pre-processing functions can be chained (i.e. executed sequentially) after an initiating call to [pre_split](#). This can either be done using the [pipe operator](#) defined in the **magrittr** package or by putting all pre-processing functions in a regular list (see the examples).

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 [evaluate](#) by not supplying a fold to [pre_split](#). See the code of [impute_median](#) for an example.

Value

A list with the following components

fit Fitting set.

test Test set.

feature_selection Logical vector indicating which features were kept (TRUE) and discarded (FALSE).

fold The fold that was used to split the data.

Author(s)

Christofer Bäcklin

See Also

[emil](#), [pre_impute_knn](#)

Examples

```
# Setup an example to work on
x <- as.matrix(iris[-5])
x[sample(600, 6)] <- NA
y <- iris$Species
cv <- resample("crossvalidation", y, nrepeat=3, nfold=4)
procedure <- modeling_procedure("lda")

# Simple dataset splitting
sets <- pre_split(x, y, cv[[1]])

# Chaining using the pipe operator
sets <- pre_split(x, y, cv[[1]]) %>%
  pre_impute_median %>%
  pre_scale

# Integration with `evaluate`
result <- evaluate(procedure, x, y, resample=cv,
  pre_process = function(...){
    pre_split(...) %>%
    pre_impute_median %>%
    pre_scale
  }
)

# or analogously with a list
result <- evaluate(procedure, x, y, resample=cv,
  pre_process = list(pre_split, pre_impute_median, pre_scale))

# Imputing without splitting
x.imputed <- impute_knn(x)

# Using a whole chain without splitting
x.processed <- pre_split(x, y=NULL) %>%
  pre_impute_median %>%
  pre_scale %>%
  (function(data) data$fit$x)
```

```
print.preprocessed_data
```

Print method for pre-processed data

Description

Print method for pre-processed data

Usage

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

Arguments

x Pre-processed data, as produced by [pre_split](#).
... Ignored, kept for S3 consistency.

Value

Nothing

Author(s)

Christofer Bäcklin

pvalue *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

```
pvalue(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

[pvalue.crr](#), [pvalue.survdiff](#), [pvalue.cuminc](#)

pvalue.coxph *Extract p-value from a Cox proportional hazards model*

Description

Based on [summary.coxph](#).

Usage

```
## S3 method for class 'coxph'  
pvalue(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

[pvalue](#)

pvalue.crr *Extracts p-value from a competing risk model*

Description

Extracts p-value from a competing risk model

Usage

```
## S3 method for class 'crr'  
pvalue(x, log_p = FALSE, ...)
```


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

[pvalue](#)

Examples

```
library(cmprsk)
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)
pvalue(x)
```

pvalue.cuminc

Extract p-value from a cumulative incidence estimation

Description

This is also known as Gray's test.

Usage

```
## S3 method for class 'cuminc'
pvalue(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.

Value

p-value.

Author(s)

Christofer Bäcklin

See Also

[pvalue](#)

pvalue.survdiff	<i>Extracts p-value from a logrank test</i>
-----------------	---

Description

Extracts p-value from a logrank test

Usage

```
## S3 method for class 'survdiff'  
pvalue(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. 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) pvalue(x)
```

Author(s)

Christofer Bäcklin

See Also

[pvalue](#)

resample	<i>Resampling schemes</i>
----------	---------------------------

Description

Performance evaluation and parameter 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, test_fraction = 0.5, nfold = 5,
  balanced = is.factor(y), subset)

resample_crossvalidation(y, nfold = 5, nrepeat = 5,
  balanced = is.factor(y), subset)

resample_bootstrap(y, nfold = 10, fit_fraction = if (replace) 1 else 0.632,
  replace = TRUE, balanced = is.factor(y), subset)
```

Arguments

method	The resampling method to use, e.g. "holdout" or "crossvalidation".
y	Observations to be divided.
...	Sent to the method specific function, e.g. "resample_holdout".
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.
test_fraction	Fraction of objects to hold out ($0 < \text{test_fraction} < 1$).
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).
nrepeat	Number of fold sets to generate.
fit_fraction	The size of the training set relative to the entire data set.
replace	Whether to sample with replacement.

Details

Note that when setting up analyzes, the user should not call `resample_holdout` or `resample_crossvalidation` 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", ...)`:

```
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.

`parameter` 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

See Also

[emil](#), [subresample](#), [image.resample](#), [index_fit](#)

Examples

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

roc_curve

Calculate ROC curves

Description

Calculate ROC curves

Usage

```
roc_curve(result, y, resample, class = levels(y), statistic = "probability")
```

```
## S3 method for class 'roc_curve'
as.data.table(x, ...)
```

```
## S3 method for class 'roc_curve'
as.data.frame(x, ...)
```

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

Arguments

result	Modeling results, as returned by evaluate .
y	True response vector used to create result.
resample	Resampling scheme used to create result.
class	The class of interest to create ROC-curves for.
statistic	The name of the statistic (as returned by the prediction function of the modeling procedure).
x	Roc curve object, as returned by roc_curve.
...	Sent to as.data.frame or as.data.table .

Value

A data frame of class "roc".

Author(s)

Christofer Bäcklin

Examples

```
# Generate some noisy data
my.data <- iris
my.data[1:4] <- my.data[1:4] + 2*rnorm(150*4)

# Train and evaluate some classifiers
procedure <- list(lda = modeling_procedure("lda"),
                 qda = modeling_procedure("qda"))
cv <- resample("crossvalidation", iris$Species, nrep=1, nfold=3)
result <- evaluate(procedure, my.data[-5], my.data$Species, resample=cv)

# Study the performance
select(result, fold=TRUE, method=TRUE, error="error")
roc <- roc_curve(result, my.data$Species, cv)
plot(roc)
```

select_.list *emil and dplyr integration*

Description

Modeling results can be converted to tabular format and manipulated using **dplyr** and other Hadleyverse packages. This is accomplished by a class specific `select_` function that differs somewhat in syntax from the default `select_`.

Usage

```
## S3 method for class 'list'
select_(.data, ..., .dots)

## S3 method for class 'modeling_result'
select_(.data, ..., .dots)
```

Arguments

<code>.data</code>	Modeling results, as returned by <code>evaluate</code> .
<code>...</code>	Not used, kept for consistency with <code>dplyr</code> .
<code>.dots</code>	Indices to select on each level of <code>.data</code> , i.e. the first index specifies which top level elements of <code>.data</code> to select, the second specifies second-level-elements etc. The last index must select elements that can be converted to a data frame. In case the desired bottom-level element is related to the observations of a modeling task, e.g. the predictions of a test set, you must supply the resampling scheme used to produce <code>.data</code> at the appropriate level (see the examples). The names of the <code>...</code> arguments specifies the names of the resulting data frame. Non-named arguments will be used to traverse the data but not returned. In summary the <code>...</code> indices can be on the following forms: Simple indices Anything that can be used to subset objects, e.g. integers, logicals, or characters. Functions A function that produces a data frame, vector or factor. Resampling schemes The same resampling scheme that was used to produce the modeling results.

Value

A `data.frame` in long format.

Author(s)

Christofer Bäcklin

See Also

`subtree`

Examples

```

# Produce some results
x <- iris[-5]
y <- iris$Species
names(y) <- sprintf("orchid%03i", seq_along(y))
cv <- resample("crossvalidation", y, nfold=3, nrepeat=2)
procedures <- list(nsc = modeling_procedure("pamr"),
                  rf = modeling_procedure("randomForest"))
result <- evaluate(procedures, x, y, resample=cv)

# Get the foldwise error for the NSC method
result %>% select(fold = TRUE, "nsc", error = "error")

# Compare both methods
require(tidyverse)
result %>%
  select(fold = TRUE, method = TRUE, error = "error") %>%
  spread(method, error)
result %>%
  select(fold = TRUE, method = TRUE, error = "error") %>%
  group_by(method) %>% summarize(mean_error = mean(error))

# Investigate the variability in estimated class 2 probability across folds
result %>%
  select(fold = cv, "nsc", "prediction", probability = function(x) x$probability[,2]) %>%
  spread(fold, probability)

```

subresample

Generate resampling subschemes

Description

A subscheme is a resampling scheme that only includes observations in the training set of a fold. This function automatically fetches the type and parameters of the prototype fold 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=1:12, test_fraction=1/4, nfold=3)
inner.cv <- subresample(cv, y=1:12)
```

subtree

Extract a subset of a tree of nested lists

Description

Modeling results produced by [evaluate](#) comes in the form of nested lists. This function can be used to subset or rearrange parts of the results into vectors, matrices or data frames. Also note the [select](#) function that provides an extension to the **dplyr** package for data manipulation.

Usage

```
subtree(x, i, ..., error_value, warn, 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_value	A template for the return value in case it is missing or invalid. Note that NA is a logical by default, causing subtree to also convert existing results to logicals. To get around this, please specify it as <code>as.numeric(NA)</code> , <code>as.character(NA)</code> , or similar (see the example below).
warn	Specifies whether warnings should be displayed (0), ignored (-1), or break execution (1). Works like the options parameter warn.
simplify	Whether to collapse results into vectors or matrices when possible (TRUE) or to preserve the original tree structure as a list (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

[select](#), [get_prediction](#), [get_importance](#), [get_tuning](#).

Examples

```
l <- list(A=list(a=0:2, b=3:4, c=023-22030),
         B=list(a=5:7, b=8:9))
subtree(l, 1:2, "b")
subtree(l, TRUE, mean, "a")

# More practical examples
x <- iris[-5]
y <- iris$Species
cv <- resample("crossvalidation", y, nfold=5, nrep=3)
procedure <- modeling_procedure("pamr")

# To illustrate the error handling capacities of subtree we'll introduce some
# spurious errors in the pre-processing function. By setting .return_error=TRUE
# they wont break the execution, but will instead be return in the results.
pre_error <- function(data, risk=.1){
  if(runif(1) < risk)
    stop("Oh no! Random error!")
  data
}
result <- evaluate(procedure, x, y, resample=cv,
  .save=c(importance=TRUE), .return_error=TRUE,
  pre_process = function(...){
    pre_split(...) %>%
      pre_error(risk=.3) %>%
      pre_pamr
  }
)
message(sum(sapply(result, inherits, "error")),
  " folds did not complete successfully!")

# Extract error rates. Since some folds fail it will be an ugly list with both
# numeric estimates and NULL values (for the failed folds).
subtree(result, TRUE, "error")

# To put it on a more consistent form we can impute the missing error rates
# with NA to allow automatic simplification into a vector (since it requires
# all values to be on the same form, i.e. numeric(1) rather than a mix
# between numeric(1) and NULL as in the previous example).
subtree(result, TRUE, "error", error_value=as.numeric(NA), warn=-1)

# Sum up feature importance for all classes within each fold and extract.
# Note that the lengths (= 4) must match between the folds for the automatic
# simplification to work.
```

```

subtree(result, TRUE, "importance", function(x){
  if(is.null(x)){
    rep(NA, 3)
  } else {
    colMeans(x[2:4])
  }
})

# The equivalent 'select' command would be ...
require(tidyr)
imp <- result %>% select(fold = TRUE, "importance", function(x){
  if(is.null(x)) return(NULL)
  x %>% gather(Species, Importance, -feature)
})
require(ggplot2)
ggplot(imp, aes(x=Species, y=Importance)) +
  geom_abline(yintercept=0, slope=0, color="hotpink") +
  geom_boxplot() + facet_wrap(~feature)

```

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](#) when needed.

Usage

```

tune(procedure, ..., .verbose = getOption("emil_verbose", FALSE))

is_tuned(procedure)

is_tunable(procedure)

detune(procedure)

```

Arguments

procedure	Modeling procedure, or list of modeling procedures, as produced by modeling_procedure .
...	Sent to evaluate .
.verbose	Whether to print an activity log. Set to -1 to suppress all messages.

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](#), [fit](#), [predict](#), [get_importance](#)

Examples

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

vlines

Add vertical or horizontal lines to a plot

Description

Add vertical or horizontal lines to a plot

Usage

```
vlines(x, lend = 1, ...)
```

```
hlines(y, lend = 1, ...)
```

Arguments

x	Coordinates of vertical lines.
lend	Line ending style, see par .
...	Sent to segments .
y	Coordinates of horizontal lines.

Author(s)

Christofer Bäcklin

Examples

```
plot(0:10, 0:10, type="n")
hlines(0:4*2.5, col="#ddddd")
points(0:10, 0:10)
```

weighted_error_rate	<i>Weighted error rate</i>
---------------------	----------------------------

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

Index

`as.data.frame`, 61
`as.data.frame.roc_curve` (`roc_curve`), 60
`as.data.table`, 61
`as.data.table.roc_curve` (`roc_curve`), 60
`as.modeling_procedure`, 3
`axis`, 38

`box`, 38, 39
`brewer.pal`, 22

`cforest`, 6, 14, 15
`cforest_control`, 14, 15
`coxph`, 6, 15, 56
`crr`, 57
`cuminc`, 57
`cv.glmnet`, 16

`data.frame`, 62
`detune` (`tune`), 66
`dichotomize`, 4
`dist`, 51
`dplyr`, 5

`emil`, 5, 8, 9, 12, 13, 15, 17–20, 23, 26–29, 31, 35, 36, 42, 44–49, 52, 54, 60, 64, 67
`error_fun`, 5, 6, 7, 9, 11, 12, 35, 38
`error_rate`, 6
`error_rate` (`error_fun`), 7
`evaluate`, 5, 8, 8, 10, 11, 13, 19, 23, 24, 32, 33, 35, 36, 40, 42, 45, 48, 52, 53, 61, 62, 64, 66, 67
`extension`, 6, 8, 10, 35

`family`, 16
`fill`, 12
`findCorrelation`, 53
`fit`, 5, 10, 13, 35, 36, 40, 42, 66, 67
`fit_caret`, 6, 14, 35
`fit_cforest`, 14, 43, 44
`fit_coxph`, 15, 44
`fit_glmnet`, 16, 16, 27, 45

`fit_lasso` (`fit_glmnet`), 16
`fit_lda`, 17, 46
`fit_lm`, 18, 46
`fit_pamr`, 10, 18, 28, 47
`fit_qda`, 19, 48
`fit_randomForest`, 12, 20, 29, 49
`fit_ridge_regression` (`fit_glmnet`), 16
`fit_rpart`, 21

`get_color`, 21
`get_color.default`, 22
`get_importance`, 5, 13, 22, 35, 36, 42, 65, 67
`get_performance`, 23
`get_prediction`, 5, 24, 65
`get_response`, 24
`get_tuning`, 5, 25, 65
`ggplot`, 41
`glmnet`, 6, 16, 44

`hlines` (`vlines`), 67

`image`, 26, 59
`image.crossvalidation` (`image.resample`), 26
`image.resample`, 26, 60
`importance_glmnet`, 17, 27, 45
`importance_lasso` (`importance_glmnet`), 27
`importance_pamr`, 19, 27, 47
`importance_randomForest`, 20, 28, 49
`importance_ridge_regression` (`importance_glmnet`), 27
`impute`, 6, 29
`impute_knn` (`impute`), 29
`impute_median`, 53
`impute_median` (`impute`), 29
`indent`, 30
`index_fit`, 30, 60
`index_test` (`index_fit`), 30
`is_blank`, 31
`is_multi_procedure`, 32

- is_tunable (tune), 66
- is_tuned (tune), 66
- lapply, 5
- lasso, 6
- lda, 6, 17
- Learning curve analyses, 7
- learning_curve, 5, 32, 41
- legend, 41
- list_method, 6, 33
- lm, 6, 18
- log_message, 10, 30, 34
- logical, 64
- ls, 33
- mCMap, 9
- message, 34, 40
- modeling_procedure, 5, 6, 8, 9, 13, 15, 17–20, 27–29, 32, 35, 42, 44–49, 66, 67
- mse, 6
- mse (error_fun), 7
- na_fill (fill), 12
- na_index, 37
- name_procedure, 36
- neg_auc, 6
- neg_auc (error_fun), 7
- neg_gmpa, 6, 8, 37
- neg_harrell_c, 6, 8
- neg_harrell_c (error_fun), 7
- nice_axis, 38
- nice_box, 39
- nice_require, 39
- notify_once, 40
- option, 34
- options, 64
- pamr, 6
- pamr.cv, 19
- pamr.predict, 28, 47
- pamr.train, 19
- par, 38, 39, 67
- parLapply, 9
- plot, 26, 41
- plot.learning_curve, 40
- plot.roc_curve (roc_curve), 60
- plot.Surv, 41
- prcomp, 53
- pre_center, 6
- pre_center (pre_process), 52
- pre_convert (pre_process), 52
- pre_impute, 29, 50
- pre_impute_knn, 6, 29, 50, 54
- pre_impute_mean (pre_impute), 50
- pre_impute_median, 6, 29
- pre_impute_median (pre_impute), 50
- pre_pamr, 10, 51, 52
- pre_pca (pre_process), 52
- pre_process, 5, 6, 10–12, 29, 52, 52
- pre_remove (pre_process), 52
- pre_remove_constant (pre_process), 52
- pre_remove_correlated (pre_process), 52
- pre_scale, 6
- pre_scale (pre_process), 52
- pre_split, 6, 37, 50, 51, 53, 55
- pre_split (pre_process), 52
- pre_transpose (pre_process), 52
- predict, 5, 13, 36, 43, 67
- predict.coxph, 44
- predict.glmnet, 27, 45
- predict.lda, 45
- predict.lm, 46
- predict.model, 42
- predict.qda, 48
- predict.caret, 43
- predict.cforest, 15, 43
- predict.coxph, 15, 44
- predict.glmnet, 17, 27, 44
- predict.lasso (predict.glmnet), 44
- predict.lda, 17, 45
- predict.lm, 18, 46
- predict.pamr, 19, 28, 47
- predict.qda, 20, 48
- predict.randomForest, 20, 29, 48
- predict.ridge_regression (predict.glmnet), 44
- predict.rpart, 49
- print.preprocessed_data, 54
- pvalue, 55, 56–58
- pvalue.coxph, 56
- pvalue.crr, 55, 56
- pvalue.cuminc, 55, 57
- pvalue.survdiff, 55, 58
- qda, 6, 20

randomForest, [6](#), [20](#)
resample, [5](#), [10–12](#), [19](#), [26](#), [31](#), [59](#), [63](#), [64](#)
resample_bootstrap(resample), [59](#)
resample_crossvalidation, [5](#)
resample_crossvalidation(resample), [59](#)
resample_holdout, [5](#), [19](#)
resample_holdout(resample), [59](#)
Resampling schemes, [7](#)
reset_notification(notify_once), [40](#)
ridge_regression, [6](#)
rmse, [6](#)
rmse(error_fun), [7](#)
ROC-curves, [7](#)
roc_curve, [6](#), [60](#)
rpart, [6](#), [21](#)

sapply, [5](#)
segments, [67](#)
select, [5](#), [64](#), [65](#)
select_, [62](#)
select_.list, [62](#)
select_.modeling_result(select_.list),
[62](#)
sprintf, [34](#)
subresample, [60](#), [63](#)
subtree, [5](#), [64](#)
summary.coxph, [56](#)
Surv, [4](#), [41](#)
survdiff, [58](#)

train, [14](#)
treeresponse, [43](#)
tune, [5](#), [9–11](#), [13](#), [35](#), [36](#), [42](#), [66](#)

vlines, [67](#)

warning, [40](#)
weighted_error_rate, [6](#), [68](#)