Package ‘MachineShop’

August 6, 2020

Type Package

Title Machine Learning Models and Tools

Version 2.5.0

Date 2020-08-05

Author Brian J Smith [aut, cre]

Maintainer Brian J Smith <brian-j-smith@uiowa.edu>

Description Meta-package for statistical and machine learning with a unified interface for model fitting, prediction, performance assessment, and presentation of results. Approaches for model fitting and prediction of numerical, categorical, or censored time-to-event outcomes include traditional regression models, regularization methods, tree-based methods, support vector machines, neural networks, ensembles, data preprocessing, filtering, and model tuning and selection. Performance metrics are provided for model assessment and can be estimated with independent test sets, split sampling, cross-validation, or bootstrap resampling. Resample estimation can be executed in parallel for faster processing and nested in cases of model tuning and selection. Modeling results can be summarized with descriptive statistics; calibration curves; variable importance; partial dependence plots; confusion matrices; and ROC, lift, and other performance curves.

Depends R (>= 3.6.0)

Imports abind, dials (>= 0.0.4), foreach, ggplot2 (>= 3.3.0), kernlab, magrittr, Matrix, methods, nnet, party, polspline, progress, recipes (>= 0.1.4), rlang, rsample, Rsolnp, survival, tibble, utils

Suggests adabag, BART, bartMachine, C50, cluster, doParallel, e1071, earth, elasticnet, gbm, glmnet, gridExtra, Hmisc, kableExtra, kknn, knitr, lars, MASS, mboost, mda, partykit, pls, randomForest, ranger, rmarkdown, rms, rpart, testthat, tree, xgboost

LazyData true

License GPL-3

URL https://brian-j-smith.github.io/MachineShop/

BugReports https://github.com/brian-j-smith/MachineShop/issues

RoxygenNote 7.1.1

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2020-08-05 23:10:43 UTC

R topics documented:

MachineShop-package 4
. . . . 5
AdaBagModel 6
AdaBoostModel 7
as.MLModel 9
BARTMachineModel 9
BARTModel 11
BlackBoostModel 13
C50Model 15
calibration 16
CForestModel 17
combine 19
confusion 20
CoxModel 21
dependence 22
diff 23
DiscreteVariate 24
EarthModel 25
expand_model 26
expand_params 27
expand_steps 28
extract 29
FDAModel 30
fit 31
GAMBoostModel 33
GBMModel 34
GLMBoostModel 35
GLMModel 36
GLMNetModel 38
Grid 39
ICHomes 40
inputs 40
KNNModel 41
LARSModel 42

LDAModel 44
lift 45
LMModel 46
MDAModel 46
metricinfo 48
metrics 49
MLControl 52
MLMetric 54
MLModel 55
ModeledInput 57
ModelFrame 58
modelinfo 59
models 60
NaiveBayesModel 62
NNetModel 63
ParameterGrid 64
performance 65
performance_curve 67
plot 69
PLSModel 71
POLRModel 72
predict 73
print 74
QDAModel 75
RandomForestModel 76
RangerModel 77
recipe_roles 79
resample 80
response 82
RPartModel 82
SelectedInput 84
SelectedModel 86
settings 87
StackedModel 89
step_kmeans 90
step_kmedoids 92
step_lincomp 94
step_sbf 96
step_spca 98
summary 100
SuperModel 102
SurvMatrix 103
SurvRegModel 104
SVMModel 105
t.test 107
TreeModel 108
TunedInput 109
TunedModel 110

varimp 112
XGBModel 112

Index **116**

`MachineShop-package` *MachineShop: Machine Learning Models and Tools*

Description

Meta-package for statistical and machine learning with a unified interface for model fitting, prediction, performance assessment, and presentation of results. Approaches for model fitting and prediction of numerical, categorical, or censored time-to-event outcomes include traditional regression models, regularization methods, tree-based methods, support vector machines, neural networks, ensembles, data preprocessing, filtering, and model tuning and selection. Performance metrics are provided for model assessment and can be estimated with independent test sets, split sampling, cross-validation, or bootstrap resampling. Resample estimation can be executed in parallel for faster processing and nested in cases of model tuning and selection. Modeling results can be summarized with descriptive statistics; calibration curves; variable importance; partial dependence plots; confusion matrices; and ROC, lift, and other performance curves.

Details

The following set of model fitting, prediction, and performance assessment functions are available for **MachineShop** models.

Training:

`fit`	Model fitting
`resample`	Resample estimation of model performance

Response Values:

`response`	Observed
`predict`	Predicted

Performance Assessment:

`calibration`	Model calibration
`confusion`	Confusion matrix
`dependence`	Parital dependence
`diff`	Model performance differences
`lift`	Lift curves
`performance metrics`	Model performance metrics
`performance_curve`	Model performance curves
`varimp`	Variable importance

Methods for resample estimation include

`BootControl`	Simple bootstrap
`BootOptimismControl`	Optimism-corrected bootstrap
`CVControl`	Repeated K-fold cross-validation
`CVOptimismControl`	Optimism-corrected cross-validation
`OOBControl`	Out-of-bootstrap
`SplitControl`	Split training-testing
`TrainControl`	Training resubstitution

Graphical and tabular summaries of modeling results can be obtained with

`plot`
`print`
`summary`

Further information on package features is available with

`metricinfo`	Performance metric information
`modelinfo`	Model information
`settings`	Global settings

Custom metrics and models can be created with the `MLMetric` and `MLModel` constructors.

Author(s)

Maintainer: Brian J Smith <`brian-j-smith@uiowa.edu`>

See Also

Useful links:

- `https://brian-j-smith.github.io/MachineShop/`
- Report bugs at `https://github.com/brian-j-smith/MachineShop/issues`

. *Quote Operator*

Description

Shorthand notation for the `quote` function. The quote operator simply returns its argument unevaluated and can be applied to any R expression. Useful for calling model constructors with quoted parameter values that are defined in terms of `nobs`, `nvars`, or `y`.

Usage

```
.(expr)
```

Arguments

`expr`	any syntactically valid R expression.

Value

The quoted (unevaluated) expression.

See Also

`quote`

Examples

```
## Stepwise variable selection with BIC
glm_fit <- fit(sale_amount ~ ., ICHomes, GLMStepAICModel(k = .(log(nobs))))
varimp(glm_fit)
```

`AdaBagModel` *Bagging with Classification Trees*

Description

Fits the Bagging algorithm proposed by Breiman in 1996 using classification trees as single classifiers.

Usage

```
AdaBagModel(
  mfinal = 100,
  minsplit = 20,
  minbucket = round(minsplit/3),
  cp = 0.01,
  maxcompete = 4,
  maxsurrogate = 5,
  usesurrogate = 2,
  xval = 10,
  surrogatestyle = 0,
  maxdepth = 30
)
```

Arguments

`mfinal`	number of trees to use.
`minsplit`	minimum number of observations that must exist in a node in order for a split to be attempted.
`minbucket`	minimum number of observations in any terminal node.
`cp`	complexity parameter.
`maxcompete`	number of competitor splits retained in the output.
`maxsurrogate`	number of surrogate splits retained in the output.
`usesurrogate`	how to use surrogates in the splitting process.
`xval`	number of cross-validations.
`surrogatestyle`	controls the selection of a best surrogate.
`maxdepth`	maximum depth of any node of the final tree, with the root node counted as depth 0.

Details

Response Types: `factor`

Automatic Tuning of Grid Parameters: `mfinal`, `maxdepth`

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`bagging`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = AdaBagModel(mfinal = 5))
```

`AdaBoostModel` *Boosting with Classification Trees*

Description

Fits the AdaBoost.M1 (Freund and Schapire, 1996) and SAMME (Zhu et al., 2009) algorithms using classification trees as single classifiers.

Usage

```
AdaBoostModel(
  boos = TRUE,
  mfinal = 100,
  coeflearn = c("Breiman", "Freund", "Zhu"),
  minsplit = 20,
  minbucket = round(minsplit/3),
  cp = 0.01,
  maxcompete = 4,
  maxsurrogate = 5,
  usesurrogate = 2,
  xval = 10,
  surrogatestyle = 0,
  maxdepth = 30
)
```

Arguments

`boos`	if `TRUE`, then bootstrap samples are drawn from the training set using the observation weights at each iteration. If `FALSE`, then all observations are used with their weights.
`mfinal`	number of iterations for which boosting is run.
`coeflearn`	learning algorithm.
`minsplit`	minimum number of observations that must exist in a node in order for a split to be attempted.
`minbucket`	minimum number of observations in any terminal node.
`cp`	complexity parameter.
`maxcompete`	number of competitor splits retained in the output.
`maxsurrogate`	number of surrogate splits retained in the output.
`usesurrogate`	how to use surrogates in the splitting process.
`xval`	number of cross-validations.
`surrogatestyle`	controls the selection of a best surrogate.
`maxdepth`	maximum depth of any node of the final tree, with the root node counted as depth 0.

Details

Response Types: `factor`

Automatic Tuning of Grid Parameters: `mfinal`, `maxdepth`, `coeflearn`*

* included only in randomly sampled grid points

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`boosting`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = AdaBoostModel(mfinal = 5))
```

`as.MLModel` *Coerce to an MLModel*

Description

Function to coerce an `MLModelFit` object to an `MLModel`.

Usage

```
as.MLModel(x, ...)

## S3 method for class 'MLModelFit'
as.MLModel(x, ...)
```

Arguments

`x`	model fit result.
`...`	arguments passed to other methods.

Value

`MLModel` class object.

`BARTMachineModel` *Bayesian Additive Regression Trees Model*

Description

Builds a BART model for regression or classification.

Usage

```
BARTMachineModel(
  num_trees = 50,
  num_burn = 250,
  num_iter = 1000,
  alpha = 0.95,
  beta = 2,
  k = 2,
  q = 0.9,
  nu = 3,
  mh_prob_steps = c(2.5, 2.5, 4)/9,
  verbose = FALSE,
  ...
)
```

Arguments

`num_trees`	number of trees to be grown in the sum-of-trees model.
`num_burn`	number of MCMC samples to be discarded as "burn-in".
`num_iter`	number of MCMC samples to draw from the posterior distribution.
`alpha, beta`	base and power hyperparameters in tree prior for whether a node is nonterminal or not.
`k`	regression prior probability that $E(Y \mid X)$ is contained in the interval (y_{min}, y_{max}), based on a normal distribution.
`q`	quantile of the prior on the error variance at which the data-based estimate is placed.
`nu`	regression degrees of freedom for the inverse $sigma^2$ prior.
`mh_prob_steps`	vector of prior probabilities for proposing changes to the tree structures: (GROW, PRUNE, CHANGE).
`verbose`	logical indicating whether to print progress information about the algorithm.
`...`	additional arguments to `bartMachine`.

Details

Response Types: `binary factor`, `numeric`

Automatic Tuning of Grid Parameters: `alpha`, `beta`, `k`, `nu`

Further model details can be found in the source link below.

In calls to `varimp` for `BARTMachineModel`, argument `metric` may be spedified as `"splits"` (default) for the proportion of time each predictor is chosen for a splitting rule or as `"trees"` for the proportion of times each predictor appears in a tree. Argument `num_replicates` is also available to control the number of BART replicates used in estimating the inclusion proportions [default: 5]. Variable importance is automatically scaled to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`. See example below.

Value

`MLModel` class object.

See Also

`bartMachine`, `fit`, `resample`

Examples

```
model_fit <- fit(sale_amount ~ ., data = ICHomes, model = BARTMachineModel)
varimp(model_fit, metric = "splits", num_replicates = 20, scale = FALSE)
```

`BARTModel` *Bayesian Additive Regression Trees Model*

Description

Flexible nonparametric modeling of covariates for continuous, binary, categorical and time-to-event outcomes.

Usage

```
BARTModel(
  K = NULL,
  sparse = FALSE,
  theta = 0,
  omega = 1,
  a = 0.5,
  b = 1,
  rho = NULL,
  augment = FALSE,
  xinfo = NULL,
  usequants = FALSE,
  sigest = NA,
  sigdf = 3,
  sigquant = 0.9,
  lambda = NA,
  k = 2,
  power = 2,
  base = 0.95,
  tau.num = NULL,
  offset = NULL,
  ntree = NULL,
  numcut = 100,
  ndpost = 1000,
  nskip = NULL,
  keepevery = NULL,
  printevery = 1000
)
```

Arguments

`K`	if provided, then coarsen the times of survival responses per the quantiles $1/K, 2/K, ..., K/K$ to reduce computational burdern.
`sparse`	logical indicating whether to perform variable selection based on a sparse Dirichlet prior rather than simply uniform; see Linero 2016.
`theta, omega`	$theta$ and $omega$ parameters; zero means random.
`a, b`	sparse parameters for $Beta(a, b)$ prior: $0.5 <= a <= 1$ where lower values induce more sparsity and typically $b = 1$.
`rho`	sparse parameter: typically $rho = p$ where p is the number of covariates under consideration.
`augment`	whether data augmentation is to be performed in sparse variable selection.
`xinfo`	optional matrix whose rows are the covariates and columns their cutpoints.
`usequants`	whether covariate cutpoints are defined by uniform quantiles or generated uniformly.
`sigest`	normal error variance prior for numeric response variables.
`sigdf`	degrees of freedom for error variance prior.
`sigquant`	quantile at which a rough estimate of the error standard deviation is placed.
`lambda`	scale of the prior error variance.
`k`	number of standard deviations $f(x)$ is away from +/-3 for categorical response variables.
`power, base`	power and base parameters for tree prior.
`tau.num`	numerator in the tau definition, i.e., $tau = tau.num/(k * sqrt(ntree))$.
`offset`	override for the default $offset$ of $F^-1(mean(y))$ in the multivariate response probability $P(y[j] = 1\|x) = F(f(x)[j] + offset[j])$.
`ntree`	number of trees in the sum.
`numcut`	number of possible covariate cutoff values.
`ndpost`	number of posterior draws returned.
`nskip`	number of MCMC iterations to be treated as burn in.
`keepevery`	interval at which to keep posterior draws.
`printevery`	interval at which to print MCMC progress.

Details

Response Types: `factor`, `numeric`, `Surv`

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`gbart`, `mbart`, `surv.bart`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = BARTModel)
```

`BlackBoostModel` *Gradient Boosting with Regression Trees*

Description

Gradient boosting for optimizing arbitrary loss functions where regression trees are utilized as base-learners.

Usage

```
BlackBoostModel(
  family = NULL,
  mstop = 100,
  nu = 0.1,
  risk = c("inbag", "oobag", "none"),
  stopintern = FALSE,
  trace = FALSE,
  teststat = c("quadratic", "maximum"),
  testtype = c("Teststatistic", "Univariate", "Bonferroni", "MonteCarlo"),
  mincriterion = 0,
  minsplit = 10,
  minbucket = 4,
  maxdepth = 2,
  saveinfo = FALSE,
  ...
)
```

Arguments

`family`	optional `Family` object. Set automatically according to the class type of the response variable.
`mstop`	number of initial boosting iterations.
`nu`	step size or shrinkage parameter between 0 and 1.

`risk`	method to use in computing the empirical risk for each boosting iteration.
`stopintern`	logical inidicating whether the boosting algorithm stops internally when the out-of-bag risk increases at a subsequent iteration.
`trace`	logical indicating whether status information is printed during the fitting process.
`teststat`	type of the test statistic to be applied for variable selection.
`testtype`	how to compute the distribution of the test statistic.
`mincriterion`	value of the test statistic or 1 - p-value that must be exceeded in order to implement a split.
`minsplit`	minimum sum of weights in a node in order to be considered for splitting.
`minbucket`	minimum sum of weights in a terminal node.
`maxdepth`	maximum depth of the tree.
`saveinfo`	logical indicating whether to store information about variable selection in `info` slot of each `partynode`.
`...`	additional arguments to `ctree_control`.

Details

Response Types: `binary factor`, `BinomialVariate`, `NegBinomialVariate`, `numeric`, `PoissonVariate`, `Surv`

Automatic Tuning of Grid Parameters: `mstop`, `maxdepth`

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`blackboost`, `Family`, `ctree_control`, `fit`, `resample`

Examples

```
library(MASS)

fit(type ~ ., data = Pima.tr, model = BlackBoostModel)
```

C50Model	*C5.0 Decision Trees and Rule-Based Model*

Description

Fit classification tree models or rule-based models using Quinlan's C5.0 algorithm.

Usage

```
C50Model(
  trials = 1,
  rules = FALSE,
  subset = TRUE,
  bands = 0,
  winnow = FALSE,
  noGlobalPruning = FALSE,
  CF = 0.25,
  minCases = 2,
  fuzzyThreshold = FALSE,
  sample = 0,
  earlyStopping = TRUE
)
```

Arguments

`trials`	integer number of boosting iterations.
`rules`	logical indicating whether to decompose the tree into a rule-based model.
`subset`	logical indicating whether the model should evaluate groups of discrete predictors for splits.
`bands`	integer between 2 and 1000 specifying a number of bands into which to group rules ordered by their affect on the error rate.
`winnow`	logical indicating use of predictor winnowing (i.e. feature selection).
`noGlobalPruning`	logical indicating a final, global pruning step to simplify the tree.
`CF`	number in (0, 1) for the confidence factor.
`minCases`	integer for the smallest number of samples that must be put in at least two of the splits.
`fuzzyThreshold`	logical indicating whether to evaluate possible advanced splits of the data.
`sample`	value between (0, 0.999) that specifies the random proportion of data to use in training the model.
`earlyStopping`	logical indicating whether the internal method for stopping boosting should be used.

Details

Response Types: `factor`

Automatic Tuning of Grid Parameters: `trials`, `rules`, `winnow`

Latter arguments are passed to `C5.0Control`. Further model details can be found in the source link below.

In calls to `varimp` for `C50Model`, argument `metric` may be spedified as `"usage"` (default) for the percentage of training set samples that fall into all terminal nodes after the split of each predictor or as `"splits"` for the percentage of splits associated with each predictor. Variable importance is automatically scaled to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`. See example below.

Value

`MLModel` class object.

See Also

`C5.0`, `fit`, `resample`

Examples

```
model_fit <- fit(Species ~ ., data = iris, model = C50Model)
varimp(model_fit, metric = "splits", scale = FALSE)
```

`calibration` *Model Calibration*

Description

Calculate calibration estimates from observed and predicted responses.

Usage

```
calibration(
  x,
  y = NULL,
  breaks = 10,
  span = 0.75,
  dist = NULL,
  na.rm = TRUE,
  ...
)
```

Arguments

`x`	observed responses or resample result containing observed and predicted responses.
`y`	predicted responses if not contained in `x`.
`breaks`	value defining the response variable bins within which to calculate observed mean values. May be specified as a number of bins, a vector of breakpoints, or `NULL` to fit smooth curves with splines for predicted survival probabilities and with loess for others.
`span`	numeric parameter controlling the degree of loess smoothing.
`dist`	character string specifying a distribution with which to estimate observed survival means. Possible values are `"empirical"` for the Kaplan-Meier estimator, `"exponential"`, `"extreme"`, `"gaussian"`, `"loggaussian"`, `"logistic"`, `"loglogistic"`, `"lognormal"`, `"rayleigh"`, `"t"`, or `"weibull"` (default).
`na.rm`	logical indicating whether to remove observed or predicted responses that are `NA` when calculating metrics.
`...`	arguments passed to other methods.

Value

`Calibration` class object that inherits from `data.frame`.

See Also

`c`, `plot`

Examples

```
library(survival)

res <- resample(Surv(time, status) ~ ., data = veteran, model = GBMModel,
                control = CVControl(times = c(90, 180, 360)))
cal <- calibration(res)
plot(cal)
```

`CForestModel` *Conditional Random Forest Model*

Description

An implementation of the random forest and bagging ensemble algorithms utilizing conditional inference trees as base learners.

Usage

```
CForestModel(
  teststat = c("quad", "max"),
  testtype = c("Univariate", "Teststatistic", "Bonferroni", "MonteCarlo"),
  mincriterion = 0,
  ntree = 500,
  mtry = 5,
  replace = TRUE,
  fraction = 0.632
)
```

Arguments

`teststat`	character specifying the type of the test statistic to be applied.
`testtype`	character specifying how to compute the distribution of the test statistic.
`mincriterion`	value of the test statistic that must be exceeded in order to implement a split.
`ntree`	number of trees to grow in a forest.
`mtry`	number of input variables randomly sampled as candidates at each node for random forest like algorithms.
`replace`	logical indicating whether sampling of observations is done with or without replacement.
`fraction`	fraction of number of observations to draw without replacement (only relevant if `replace = FALSE`).

Details

Response Types: `factor`, `numeric`, `Surv`

Automatic Tuning of Grid Parameters: `mtry`

Supplied arguments are passed to `cforest_control`. Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`cforest`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = CForestModel)
```

combine *Combine MachineShop Objects*

Description

Combine one or more **MachineShop** objects of the same class.

Usage

```
## S3 method for class 'Calibration'
c(...)

## S3 method for class 'ConfusionList'
c(...)

## S3 method for class 'ConfusionMatrix'
c(...)

## S3 method for class 'LiftCurve'
c(...)

## S3 method for class 'ListOf'
c(...)

## S3 method for class 'PerformanceCurve'
c(...)

## S3 method for class 'Resamples'
c(...)

## S4 method for signature 'SurvMatrix,SurvMatrix'
e1 + e2
```

Arguments

`...`	named or unnamed calibration, confusion, lift, performance curve, summary, or resample results. Curves must have been generated with the same performance metrics and resamples with the same resampling control.
`e1, e2`	objects.

Value

Object of the same class as the arguments.

`confusion` *Confusion Matrix*

Description

Calculate confusion matrices of predicted and observed responses.

Usage

```
confusion(
  x,
  y = NULL,
  cutoff = MachineShop::settings("cutoff"),
  na.rm = TRUE,
  ...
)

ConfusionMatrix(data = NA, ordered = FALSE)
```

Arguments

`x`	factor of observed responses or resample result containing observed and predicted responses.
`y`	predicted responses if not contained in `x`.
`cutoff`	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified. If `NULL`, then binary responses are summed directly over predicted class probabilities, whereas a default cutoff of 0.5 is used for survival probabilities. Class probability summations and survival will appear as decimal numbers that can be interpreted as expected counts.
`na.rm`	logical indicating whether to remove observed or predicted responses that are `NA` when calculating metrics.
`...`	arguments passed to other methods.
`data`	square matrix, or object that can be converted to one, of cross-classified predicted and observed values in the rows and columns, respectively.
`ordered`	logical indicating whether the confusion matrix row and columns should be regarded as ordered.

Value

The return value is a `ConfusionMatrix` class object that inherits from `table` if `x` and `y` responses are specified or a `ConfusionList` object that inherits from `list` if `x` is a `Resamples` object.

See Also

`c`, `plot`, `summary`

Examples

```
res <- resample(Species ~ ., data = iris, model = GBMModel)
(conf <- confusion(res))
plot(conf)
```

`CoxModel` *Proportional Hazards Regression Model*

Description

Fits a Cox proportional hazards regression model. Time dependent variables, time dependent strata, multiple events per subject, and other extensions are incorporated using the counting process formulation of Andersen and Gill.

Usage

```
CoxModel(ties = c("efron", "breslow", "exact"), ...)

CoxStepAICModel(
  ties = c("efron", "breslow", "exact"),
  ...,
  direction = c("both", "backward", "forward"),
  scope = NULL,
  k = 2,
  trace = FALSE,
  steps = 1000
)
```

Arguments

`ties`	character string specifying the method for tie handling.
`...`	arguments passed to `coxph.control`.
`direction`	mode of stepwise search, can be one of `"both"` (default), `"backward"`, or `"forward"`.
`scope`	defines the range of models examined in the stepwise search. This should be a list containing components `upper` and `lower`, both formulae.
`k`	multiple of the number of degrees of freedom used for the penalty. Only `k = 2` gives the genuine AIC; `k = .(log(nobs))` is sometimes referred to as BIC or SBC.
`trace`	if positive, information is printed during the running of `stepAIC`. Larger values may give more information on the fitting process.
`steps`	maximum number of steps to be considered.

Details

Response Types: `Surv`

Default values for the `NULL` arguments and further model details can be found in the source link below.

In calls to `varimp` for `CoxModel` and `CoxStepAICModel`, numeric argument `base` may be specified for the (negative) logarithmic transformation of p-values [defaul: `exp(1)`]. Transformed p-values are automatically scaled in the calculation of variable importance to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`.

#' @return `MLModel` class object.

See Also

`coxph`, `coxph.control`, `stepAIC`, `fit`, `resample`

Examples

```
library(survival)

fit(Surv(time, status) ~ ., data = veteran, model = CoxModel)
```

`dependence` *Partial Dependence*

Description

Calculate partial dependence of a response on select predictor variables.

Usage

```
dependence(
  object,
  data = NULL,
  select = NULL,
  interaction = FALSE,
  n = 10,
  intervals = c("uniform", "quantile"),
  stats = MachineShop::settings("stats.PartialDependence")
)
```

Arguments

`object`	model fit result.
`data`	data frame containing all predictor variables. If not specified, the training data will be used by default.

`select`	expression indicating predictor variables for which to compute partial dependence (see `subset` for syntax) [default: all].
`interaction`	logical indicating whether to calculate dependence on the interacted predictors.
`n`	number of predictor values at which to perform calculations.
`intervals`	character string specifying whether the `n` values are spaced uniformly (`"uniform"`) or according to variable quantiles (`"quantile"`).
`stats`	function, function name, or vector of these with which to compute response variable summary statistics over non-selected predictor variables.

Value

`PartialDependence` class object that inherits from `data.frame`.

See Also

`plot`

Examples

```
gbm_fit <- fit(Species ~ ., data = iris, model = GBMModel)
(pd <- dependence(gbm_fit, select = c(Petal.Length, Petal.Width)))
plot(pd)
```

`diff` *Model Performance Differences*

Description

Pairwise model differences in resampled performance metrics.

Usage

```
## S3 method for class 'MLModel'
diff(x, ...)

## S3 method for class 'Performance'
diff(x, ...)

## S3 method for class 'Resamples'
diff(x, ...)
```

Arguments

`x`	model performance or resample result.
`...`	arguments passed to other methods.

Value

`PerformanceDiff` class object that inherits from `Performance`.

See Also

`t.test`, `plot`, `summary`

Examples

```
## Survival response example
library(survival)

fo <- Surv(time, status) ~ .
control <- CVControl()

gbm_res1 <- resample(fo, data = veteran, GBMModel(n.trees = 25), control)
gbm_res2 <- resample(fo, data = veteran, GBMModel(n.trees = 50), control)
gbm_res3 <- resample(fo, data = veteran, GBMModel(n.trees = 100), control)

res <- c(GBM1 = gbm_res1, GBM2 = gbm_res2, GBM3 = gbm_res3)
res_diff <- diff(res)
summary(res_diff)
plot(res_diff)
```

`DiscreteVariate` *Discrete Variate Constructors*

Description

Create a variate of binomial counts, discrete numbers, negative binomial counts, or Poisson counts.

Usage

```
BinomialVariate(x = integer(), size = integer())

DiscreteVariate(x = integer(), min = -Inf, max = Inf)

NegBinomialVariate(x = integer())

PoissonVariate(x = integer())
```

Arguments

`x`	numeric vector.
`size`	number or numeric vector of binomial trials.
`min, max`	minimum and maximum bounds for discrete numbers.

Value

`BinomialVariate` object class, `DiscreteVariate` that inherits from `numeric`, or `NegBinomialVariate` or `PoissonVariate` that inherit from `DiscreteVariate`.

See Also

`role_binom`

Examples

```
BinomialVariate(rbinom(25, 10, 0.5), size = 10)
PoissonVariate(rpois(25, 10))
```

`EarthModel` *Multivariate Adaptive Regression Splines Model*

Description

Build a regression model using the techniques in Friedman's papers "Multivariate Adaptive Regression Splines" and "Fast MARS".

Usage

```
EarthModel(
  pmethod = c("backward", "none", "exhaustive", "forward", "seqrep", "cv"),
  trace = 0,
  degree = 1,
  nprune = NULL,
  nfold = 0,
  ncross = 1,
  stratify = TRUE
)
```

Arguments

`pmethod`	pruning method.
`trace`	level of execution information to display.
`degree`	maximum degree of interaction.
`nprune`	maximum number of terms (including intercept) in the pruned model.
`nfold`	number of cross-validation folds.
`ncross`	number of cross-validations if `nfold` > 1.
`stratify`	logical indicating whether to stratify cross-validation samples by the response levels.

Details

Response Types: factor, numeric

Automatic Tuning of Grid Parameters: nprune, degree*

* included only in randomly sampled grid points

Default values for the NULL arguments and further model details can be found in the source link below.

In calls to varimp for EarthModel, argument metric may be specified as "gcv" (default) for the generalized cross-validation decrease over all subsets that include each predictor, as "rss" for the residual sums of squares decrease, or as "nsubsets" for the number of model subsets that include each predictor. Variable importance is automatically scaled to range from 0 to 100. To obtain unscaled importance values, set scale = FALSE. See example below.

Value

MLModel class object.

See Also

earth, fit, resample

Examples

```
model_fit <- fit(Species ~ ., data = iris, model = EarthModel)
varimp(model_fit, metric = "nsubsets", scale = FALSE)
```

---

expand_model *Model Expansion Over Tuning Parameters*

Description

Expand a model over all combinations of a grid of tuning parameters.

Usage

```
expand_model(x, ..., random = FALSE)
```

Arguments

x	model function, function name, or call.
...	named vectors or factors or a list of these containing the parameter values over which to expand x.
random	number of points to be randomly sampled from the parameter grid or FALSE if all points are to be returned.

Value

`list` of expanded models.

See Also

SelectedModel

Examples

```
library(MASS)

models <- expand_model(GBMModel, n.trees = c(50, 100),
                                 interaction.depth = 1:2)

fit(medv ~ ., data = Boston, model = SelectedModel(models))
```

---

expand_params *Model Parameters Expansion*

Description

Create a grid of parameter values from all combinations of supplied inputs.

Usage

```
expand_params(..., random = FALSE)
```

Arguments

`...`	named vectors or factors or a list of these containing the parameter values over which to create the grid.
`random`	number of points to be randomly sampled from the parameter grid or `FALSE` if all points are to be returned.

Value

A data frame containing one row for each combination of the supplied inputs.

See Also

TunedModel

Examples

```
library(MASS)

grid <- expand_params(
  n.trees = c(50, 100),
  interaction.depth = 1:2
)

fit(medv ~ ., data = Boston, model = TunedModel(GBMModel, grid = grid))
```

---

expand_steps *Recipe Step Parameters Expansion*

Description

Create a grid of parameter values from all combinations of lists supplied for steps of a preprocessing recipe.

Usage

```
expand_steps(..., random = FALSE)
```

Arguments

`...`	one or more lists containing parameter values over which to create the grid. For each list an argument name should be given as the `id` of the recipe step to which it corresponds.
`random`	number of points to be randomly sampled from the parameter grid or `FALSE` if all points are to be returned.

Value

`RecipeGrid` class object that inherits from `data.frame`.

See Also

`TunedInput`

Examples

```
library(recipes)
library(MASS)

rec <- recipe(medv ~ ., data = Boston) %>%
  step_corr(all_numeric(), -all_outcomes(), id = "corr") %>%
  step_pca(all_numeric(), -all_outcomes(), id = "pca")
```

```
expand_steps(
  corr = list(threshold = c(0.8, 0.9),
              method = c("pearson", "spearman")),
  pca = list(num_comp = 1:3)
)
```

`extract` *Extract Elements of an Object*

Description

Operators acting on data structures to extract elements.

Usage

```
## S3 method for class 'BinomialVariate'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'DiscreteVariate,ANY,missing,missing'
x[i]

## S3 method for class 'ModelFrame'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'ModelFrame,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'ModelFrame,ANY,missing,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'ModelFrame,missing,missing,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'RecipeGrid,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'Resamples,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'Resamples,ANY,missing,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'Resamples,missing,missing,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'SurvMatrix,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	object from which to extract elements.
`i, j, ...`	indices specifying elements to extract.
`drop`	logical indicating that the result be returned as an object coerced to the lowest dimension possible if `TRUE` or with the original dimensions and class otherwise.

FDAModel *Flexible and Penalized Discriminant Analysis Models*

Description

Performs flexible discriminant analysis.

Usage

```
FDAModel(
  theta = NULL,
  dimension = NULL,
  eps = .Machine$double.eps,
  method = .(mda::polyreg),
  ...
)

PDAModel(lambda = 1, df = NULL, ...)
```

Arguments

`theta`	optional matrix of class scores, typically with number of columns less than one minus the number of classes.
`dimension`	dimension of the discriminant subspace, less than the number of classes, to use for prediction.
`eps`	numeric threshold for small singular values for excluding discriminant variables.
`method`	regression function used in optimal scaling. The default of linear regression is provided by `polyreg` from the **mda** package. For penalized discriminant analysis, `gen.ridge` is appropriate. Other possibilities are `mars` for multivariate adaptive regression splines and `bruto` for adaptive backfitting of additive splines. Use the `.` operator to quote specified functions.
`...`	additional arguments to `method` for `FDAModel` and to `FDAModel` for `PDAModel`.
`lambda`	shrinkage penalty coefficient.
`df`	alternative specification of `lambda` in terms of equivalent degrees of freedom.

Details

Response Types: `factor`

Automatic Tuning of Grid Parameters
- FDAModel: `nprune`, `degree`*
- PDAModel: `lambda`

* included only in randomly sampled grid points

The `predict` function for this model additionally accepts the following argument.

`prior` prior class membership probabilities for prediction data if different from the training set.

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`fda`, `predict.fda`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = FDAModel)

fit(Species ~ ., data = iris, model = PDAModel)
```

`fit` *Model Fitting*

Description

Fit a model to estimate its parameters from a data set.

Usage

```
fit(x, ...)

## S3 method for class 'formula'
fit(x, data, model, ...)

## S3 method for class 'matrix'
fit(x, y, model, ...)

## S3 method for class 'ModelFrame'
fit(x, model, ...)
```

```
## S3 method for class 'recipe'
fit(x, model, ...)

## S3 method for class 'MLModel'
fit(x, ...)

## S3 method for class 'MLModelFunction'
fit(x, ...)
```

Arguments

`x`	input specifying a relationship between model predictor and response variables. Alternatively, a model function or call may be given first followed by the input specification.
`...`	arguments passed to other methods.
`data`	data frame containing observed predictors and outcomes.
`model`	model function, function name, or call; ignored and can be omitted when fitting modeled inputs.
`y`	response variable.

Details

User-specified case weights may be specified for `ModelFrames` upon creation with the `weights` argument in its constructor.

Variables in `recipe` specifications may be designated as case weights with the `role_case` function.

Value

`MLModelFit` class object.

See Also

`as.MLModel`, `response`, `predict`, `varimp`

Examples

```
## Survival response example
library(survival)

gbm_fit <- fit(Surv(time, status) ~ ., data = veteran, model = GBMModel)
varimp(gbm_fit)
```

GAMBoostModel *Gradient Boosting with Additive Models*

Description

Gradient boosting for optimizing arbitrary loss functions, where component-wise arbitrary base-learners, e.g., smoothing procedures, are utilized as additive base-learners.

Usage

```
GAMBoostModel(
  family = NULL,
  baselearner = c("bbs", "bols", "btree", "bss", "bns"),
  dfbase = 4,
  mstop = 100,
  nu = 0.1,
  risk = c("inbag", "oobag", "none"),
  stopintern = FALSE,
  trace = FALSE
)
```

Arguments

`family`	optional `Family` object. Set automatically according to the class type of the response variable.
`baselearner`	character specifying the component-wise `base learner` to be used.
`dfbase`	gobal degrees of freedom for P-spline base learners (`"bbs"`).
`mstop`	number of initial boosting iterations.
`nu`	step size or shrinkage parameter between 0 and 1.
`risk`	method to use in computing the empirical risk for each boosting iteration.
`stopintern`	logical inidicating whether the boosting algorithm stops internally when the out-of-bag risk increases at a subsequent iteration.
`trace`	logical indicating whether status information is printed during the fitting process.

Details

Response Types: `binary factor`, `BinomialVariate`, `NegBinomialVariate`, `numeric`, `PoissonVariate`, `Surv`

Automatic Tuning of Grid Parameters: `mstop`

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`gamboost`, `Family`, `baselearners`, `fit`, `resample`

Examples

```
library(MASS)

fit(type ~ ., data = Pima.tr, model = GAMBoostModel)
```

`GBMModel` *Generalized Boosted Regression Model*

Description

Fits generalized boosted regression models.

Usage

```
GBMModel(
  distribution = NULL,
  n.trees = 100,
  interaction.depth = 1,
  n.minobsinnode = 10,
  shrinkage = 0.1,
  bag.fraction = 0.5
)
```

Arguments

`distribution`	optional character string specifying the name of the distribution to use or list with a component `name` specifying the distribution and any additional parameters needed. Set automatically according to the class type of the response variable.
`n.trees`	total number of trees to fit.
`interaction.depth`	maximum depth of variable interactions.
`n.minobsinnode`	minimum number of observations in the trees terminal nodes.
`shrinkage`	shrinkage parameter applied to each tree in the expansion.
`bag.fraction`	fraction of the training set observations randomly selected to propose the next tree in the expansion.

Details

Response Types: `factor`, `numeric`, `PoissonVariate`, `Surv`

Automatic Tuning of Grid Parameters: `n.trees`, `interaction.depth`, `shrinkage`*, `n.minobsinnode`*

* included only in randomly sampled grid points

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`gbm`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = GBMModel)
```

`GLMBoostModel` *Gradient Boosting with Linear Models*

Description

Gradient boosting for optimizing arbitrary loss functions where component-wise linear models are utilized as base-learners.

Usage

```
GLMBoostModel(
  family = NULL,
  mstop = 100,
  nu = 0.1,
  risk = c("inbag", "oobag", "none"),
  stopintern = FALSE,
  trace = FALSE
)
```

Arguments

`family`	optional `Family` object. Set automatically according to the class type of the response variable.
`mstop`	number of initial boosting iterations.
`nu`	step size or shrinkage parameter between 0 and 1.
`risk`	method to use in computing the empirical risk for each boosting iteration.

`stopintern`	logical inidicating whether the boosting algorithm stops internally when the out-of-bag risk increases at a subsequent iteration.
`trace`	logical indicating whether status information is printed during the fitting process.

Details

Response Types: `binary factor`, `BinomialVariate`, `NegBinomialVariate`, `numeric`, `PoissonVariate`, `Surv`

Automatic Tuning of Grid Parameters: `mstop`

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`glmboost`, `Family`, `fit`, `resample`

Examples

```
library(MASS)

fit(type ~ ., data = Pima.tr, model = GLMBoostModel)
```

`GLMModel` *Generalized Linear Model*

Description

Fits generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
GLMModel(family = NULL, quasi = FALSE, ...)

GLMStepAICModel(
  family = NULL,
  quasi = FALSE,
  ...,
  direction = c("both", "backward", "forward"),
  scope = NULL,
  k = 2,
  trace = FALSE,
  steps = 1000
)
```

Arguments

family	optional error distribution and link function to be used in the model. Set automatically according to the class type of the response variable.
quasi	logical indicator for over-dispersion of binomial and Poisson families; i.e., dispersion parameters not fixed at one.
...	arguments passed to `glm.control`.
direction	mode of stepwise search, can be one of `"both"` (default), `"backward"`, or `"forward"`.
scope	defines the range of models examined in the stepwise search. This should be a list containing components `upper` and `lower`, both formulae.
k	multiple of the number of degrees of freedom used for the penalty. Only `k = 2` gives the genuine AIC; `k = .(log(nobs))` is sometimes referred to as BIC or SBC.
trace	if positive, information is printed during the running of `stepAIC`. Larger values may give more information on the fitting process.
steps	maximum number of steps to be considered.

Details

`GLMModel` **Response Types:** `BinomialVariate`, `factor`, `matrix`, `NegBinomialVariate`, `numeric`, `PoissonVariate`

`GLMStepAICModel` **Response Types:** `binary factor`, `BinomialVariate`, `NegBinomialVariate`, `numeric`, `PoissonVariate`

Default values for the `NULL` arguments and further model details can be found in the source link below.

In calls to `varimp` for `GLMModel` and `GLMStepAICModel`, numeric argument `base` may be specified for the (negative) logarithmic transformation of p-values [defaul: `exp(1)`]. Transformed p-values are automatically scaled in the calculation of variable importance to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`.

Value

`MLModel` class object.

See Also

`glm`, `glm.control`, `stepAIC`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = GLMModel)
```

GLMNetModel *GLM Lasso or Elasticnet Model*

Description

Fit a generalized linear model via penalized maximum likelihood.

Usage

```
GLMNetModel(
  family = NULL,
  alpha = 1,
  lambda = 0,
  standardize = TRUE,
  intercept = NULL,
  penalty.factor = .(rep(1, nvars)),
  standardize.response = FALSE,
  thresh = 1e-07,
  maxit = 1e+05,
  type.gaussian = .(ifelse(nvars < 500, "covariance", "naive")),
  type.logistic = c("Newton", "modified.Newton"),
  type.multinomial = c("ungrouped", "grouped")
)
```

Arguments

`family`	optional response type. Set automatically according to the class type of the response variable.
`alpha`	elasticnet mixing parameter.
`lambda`	regularization parameter. The default value `lambda = 0` performs no regularization and should be increased to avoid model fitting issues if the number of predictor variables is greater than the number of observations.
`standardize`	logical flag for predictor variable standardization, prior to model fitting.
`intercept`	logical indicating whether to fit intercepts.
`penalty.factor`	vector of penalty factors to be applied to each coefficient.
`standardize.response`	logical indicating whether to standardize `"mgaussian"` response variables.
`thresh`	convergence threshold for coordinate descent.
`maxit`	maximum number of passes over the data for all lambda values.
`type.gaussian`	algorithm type for guassian models.
`type.logistic`	algorithm type for logistic models.
`type.multinomial`	algorithm type for multinomial models.

Details

Response Types: `BinomialVariate`, `factor`, `matrix`, `numeric`, `PoissonVariate`, `Surv`

Automatic Tuning of Grid Parameters: `lambda`, `alpha`

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`glmnet`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = GLMNetModel(lambda = 0.01))
```

`Grid` *Tuning Grid Control*

Description

Defines control parameters for a tuning grid.

Usage

```
Grid(length = 3, random = FALSE)
```

Arguments

`length`	number of values to be generated for each model parameter in the tuning grid.
`random`	number of unique grid points to sample at random, `Inf` for all random points, or `FALSE` for all fixed points.

Value

`Grid` class object.

See Also

`TunedModel`

Examples

```
TunedModel(GBMModel, grid = Grid(10, random = 5))
```

`ICHomes` *Iowa City Home Sales Dataset*

Description

Characteristics of homes sold in Iowa City, IA from 2005 to 2008 as reported by the county assessor's office.

Usage

```
ICHomes
```

Format

A data frame with 753 observations of 17 variables:

sale_amount sale amount in dollars.

sale_year sale year.

sale_month sale month.

built year in which the home was built.

style home stlye (Home/Condo)

construction home construction type.

base_size base foundation size in sq ft.

add_size size of additions made to the base foundation in sq ft.

garage1_size attached garage size in sq ft.

garage2_size detached garage size in sq ft.

lot_size total lot size in sq ft.

bedrooms number of bedrooms.

basement presence of a basement (No/Yes).

ac presence of central air conditioning (No/Yes).

attic presence of a finished attic (No/Yes).

lon,lat home longitude/latitude coordinates.

`inputs` *Model Inputs*

Description

Model inputs are the predictor and response variables whose relationship is determined by a model fit. Input specifications supported by **MachineShop** are summarized in the table below.

formula	Traditional model formula
matrix	Design matrix of predictors
ModelFrame	Model frame
recipe	Preprocessing recipe roles and steps

Response variable types in the input specifications are defined by the user with the functions and recipe roles:

Response Functions	BinomialVariate
	DiscreteVariate
	factor
	matrix
	NegBinomialVariate
	numeric
	ordered
	PoissonVariate
	Surv
Recipe Roles	role_binom
	role_surv

Inputs may be combined, selected, or tuned with the following meta-input functions.

ModeledInput	Input with a prespecified model
SelectedInput	Input selection from a candidate set
TunedInput	Input tuning over a parameter grid

See Also

fit, resample

KNNModel *Weighted k-Nearest Neighbor Model*

Description

Fit a k-nearest neighbor model for which the k nearest training set vectors (according to Minkowski distance) are found for each row of the test set, and prediction is done via the maximum of summed kernel densities.

Usage

```
KNNModel(
  k = 7,
  distance = 2,
  scale = TRUE,
 kernel = c("optimal", "biweight", "cos", "epanechnikov", "gaussian", "inv", "rank",
   "rectangular", "triangular", "triweight")
)
```

Arguments

`k`	numer of neigbors considered.
`distance`	Minkowski distance parameter.
`scale`	logical indicating whether to scale predictors to have equal standard deviations.
`kernel`	kernel to use.

Details

Response Types: `factor`, `numeric`, `ordinal`

Automatic Tuning of Grid Parameters: `k`, `distance`*, `kernel`*

* included only in randomly sampled grid points

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`kknn`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = KNNModel)
```

`LARSModel` *Least Angle Regression, Lasso and Infinitesimal Forward Stagewise Models*

Description

Fit variants of Lasso, and provide the entire sequence of coefficients and fits, starting from zero to the least squares fit.

Usage

```
LARSModel(
  type = c("lasso", "lar", "forward.stagewise", "stepwise"),
  trace = FALSE,
  normalize = TRUE,
  intercept = TRUE,
  step = NULL,
  use.Gram = TRUE
)
```

Arguments

`type`	model type.
`trace`	logical indicating whether status information is printed during the fitting process.
`normalize`	whether to standardize each variable to have unit L2 norm.
`intercept`	whether to include an intercept in the model.
`step`	algorithm step number to use for prediction. May be a decimal number indicating a fractional distance between steps. If specified, the maximum number of algorithm steps will be `ceiling(step)`; otherwise, `step` will be set equal to the source package default maximum [default: `max.steps`].
`use.Gram`	whether to precompute the Gram matrix.

Details

Response Types: `numeric`

Automatic Tuning of Grid Parameters: `step`

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`lars`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = LARSModel)
```

LDAModel *Linear Discriminant Analysis Model*

Description

Performs linear discriminant analysis.

Usage

```
LDAModel(
  prior = NULL,
  tol = 1e-04,
  method = c("moment", "mle", "mve", "t"),
  nu = 5,
  dimen = NULL,
  use = c("plug-in", "debiased", "predictive")
)
```

Arguments

`prior`	prior probabilities of class membership if specified or the class proportions in the training set otherwise.
`tol`	tolerance for the determination of singular matrices.
`method`	type of mean and variance estimator.
`nu`	degrees of freedom for `method = "t"`.
`dimen`	dimension of the space to use for prediction.
`use`	type of parameter estimation to use for prediction.

Details

Response Types: `factor`

Automatic Tuning of Grid Parameters: `dimen`

The `predict` function for this model additionally accepts the following argument.

`prior` prior class membership probabilities for prediction data if different from the training set.

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`lda`, `predict.lda`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = LDAModel)
```

---

lift *Model Lift Curves*

Description

Calculate lift curves from observed and predicted responses.

Usage

```
lift(x, y = NULL, na.rm = TRUE, ...)
```

Arguments

`x`	observed responses or resample result containing observed and predicted responses.
`y`	predicted responses if not contained in `x`.
`na.rm`	logical indicating whether to remove observed or predicted responses that are `NA` when calculating metrics.
`...`	arguments passed to other methods.

Value

`LiftCurve` class object that inherits from `PerformanceCurve`.

See Also

`c`, `plot`, `summary`

Examples

```
library(MASS)

res <- resample(type ~ ., data = Pima.tr, model = GBMModel)
lf <- lift(res)
plot(lf)
```

LMModel *Linear Models*

Description

Fits linear models.

Usage

```
LMModel()
```

Details

Response Types: `factor`, `matrix`, `numeric`

Further model details can be found in the source link below.

In calls to `varimp` for `LModel`, numeric argument `base` may be specified for the (negative) logarithmic transformation of p-values [defaul: `exp(1)`]. Transformed p-values are automatically scaled in the calculation of variable importance to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`.

Value

`MLModel` class object.

See Also

`lm`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = LMModel)
```

MDAModel *Mixture Discriminant Analysis Model*

Description

Performs mixture discriminant analysis.

Usage

```
MDAModel(
  subclasses = 3,
  sub.df = NULL,
  tot.df = NULL,
  dimension = sum(subclasses) - 1,
  eps = .Machine$double.eps,
  iter = 5,
  method = .(mda::polyreg),
  trace = FALSE,
  ...
)
```

Arguments

`subclasses`	numeric value or vector of subclasses per class.
`sub.df`	effective degrees of freedom of the centroids per class if subclass centroid shrinkage is performed.
`tot.df`	specification of the total degrees of freedom as an alternative to `sub.df`.
`dimension`	dimension of the discriminant subspace to use for prediction.
`eps`	numeric threshold for automatically truncating the dimension.
`iter`	limit on the total number of iterations.
`method`	regression function used in optimal scaling. The default of linear regression is provided by `polyreg` from the **mda** package. For penalized mixture discriminant models, `gen.ridge` is appropriate. Other possibilities are `mars` for multivariate adaptive regression splines and `bruto` for adaptive backfitting of additive splines. Use the `.` operator to quote specified functions.
`trace`	logical indicating whether iteration information is printed.
`...`	additional arguments to `mda.start` and `method`.

Details

Response Types: `factor`

Automatic Tuning of Grid Parameters: `subclasses`

The `predict` function for this model additionally accepts the following argument.

`prior` prior class membership probabilities for prediction data if different from the training set.

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

mda, predict.mda, fit, resample

Examples

```
fit(Species ~ ., data = iris, model = MDAModel)
```

---

metricinfo — *Display Performance Metric Information*

Description

Display information about metrics provided by the **MachineShop** package.

Usage

```
metricinfo(...)
```

Arguments

`...`	metric functions or function names; observed responses; observed and predicted responses; confusion or resample results for which to display information. If none are specified, information is returned on all available metrics by default.

Value

List of named metric elements each containing the following components:

label character descriptor for the metric.

maximize logical indicating whether higher values of the metric correspond to better predictive performance.

arguments closure with the argument names and corresponding default values of the metric function.

response_types data frame of the observed and predicted response variable types supported by the metric.

Examples

```
## All metrics
metricinfo()

## Metrics by observed and predicted response types
names(metricinfo(factor(0)))
names(metricinfo(factor(0), factor(0)))
names(metricinfo(factor(0), matrix(0)))
names(metricinfo(factor(0), numeric(0)))
```

```
## Metric-specific information
metricinfo(auc)
```

`metrics` *Performance Metrics*

Description

Compute measures of agreement between observed and predicted responses.

Usage

```
accuracy(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

auc(
  observed,
  predicted = NULL,
  metrics = c(MachineShop::tpr, MachineShop::fpr),
  stat = MachineShop::settings("stat.Curve"),
  ...
)

brier(observed, predicted = NULL, ...)

cindex(observed, predicted = NULL, ...)

cross_entropy(observed, predicted = NULL, ...)

f_score(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  beta = 1,
  ...
)

fnr(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

fpr(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

kappa2(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

npv(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

ppv(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

pr_auc(observed, predicted = NULL, ...)

precision(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

recall(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

roc_auc(observed, predicted = NULL, ...)

roc_index(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  f = function(sensitivity, specificity) (sensitivity + specificity)/2,
  ...
)

rpp(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

sensitivity(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

specificity(
  observed,
  predicted = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

tnr(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

tpr(observed, predicted = NULL, cutoff = MachineShop::settings("cutoff"), ...)

weighted_kappa2(observed, predicted = NULL, power = 1, ...)

gini(observed, predicted = NULL, ...)

mae(observed, predicted = NULL, ...)

mse(observed, predicted = NULL, ...)

msle(observed, predicted = NULL, ...)

r2(observed, predicted = NULL, dist = NULL, ...)

rmse(observed, predicted = NULL, ...)

rmsle(observed, predicted = NULL, ...)
```

Arguments

`observed`	observed responses; or confusion, performance curve, or resample result containing observed and predicted responses.
`predicted`	predicted responses if not contained in `observed`.
`cutoff`	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified.
`...`	arguments passed to or from other methods.
`metrics`	list of two performance metrics for the calculation [default: ROC metrics].
`stat`	function or character string naming a function to compute a summary statistic at each cutoff value of resampled metrics in performance curves, or `NULL` for resample-specific metrics.
`beta`	relative importance of recall to precision in the calculation of `f_score` [default: F1 score].
`f`	function to calculate a desired sensitivity-specificity tradeoff.
`power`	power to which positional distances of off-diagonals from the main diagonal in confusion matrices are raised to calculate `weighted_kappa2`.
`dist`	character string specifying a distribution with which to estimate the survival mean in the total sum of square component of `r2`. Possible values are `"empirical"` for the Kaplan-Meier estimator, `"exponential"`, `"extreme"`, `"gaussian"`, `"loggaussian"`, `"logistic"`, `"loglogistic"`, `"lognormal"`, `"rayleigh"`, `"t"`, or `"weibull"` (default).

See Also

metricinfo, performance

MLControl *Resampling Controls*

Description

Structures to define and control sampling methods for estimating predictive performance of models in the **MachineShop** package.

Usage

```
BootControl(samples = 25, ...)

BootOptimismControl(samples = 25, ...)

CVControl(folds = 10, repeats = 1, ...)

CVOptimismControl(folds = 10, repeats = 1, ...)

OOBControl(samples = 25, ...)

SplitControl(prop = 2/3, ...)

TrainControl(...)

MLControl(
  times = NULL,
  dist = NULL,
  method = NULL,
  seed = sample(.Machine$integer.max, 1),
  ...
)
```

Arguments

`samples`	number of bootstrap samples.
`...`	arguments passed to `MLControl`.
`folds`	number of cross-validation folds (K).
`repeats`	number of repeats of the K-fold partitioning.
`prop`	proportion of cases to include in the training set (`0` < `prop` < `1`).
`times, dist, method`	arguments passed to `predict`.
`seed`	integer to set the seed at the start of resampling.

Details

`BootControl` constructs an `MLControl` object for simple bootstrap resampling in which models are fit with bootstrap resampled training sets and used to predict the full data set (Efron and Tibshirani 1993).

`BootOptimismControl` constructs an `MLControl` object for optimism-corrected bootstrap resampling (Efron and Gong 1983, Harrell et al. 1996).

`CVControl` constructs an `MLControl` object for repeated K-fold cross-validation (Kohavi 1995). In this procedure, the full data set is repeatedly partitioned into K-folds. Within a partitioning, prediction is performed on each of the K folds with models fit on all remaining folds.

`CVOptimismControl` constructs an `MLControl` object for optimism-corrected cross-validation resampling (Davison and Hinkley 1997, eq. 6.48).

`OOBControl` constructs an `MLControl` object for out-of-bootstrap resampling in which models are fit with bootstrap resampled training sets and used to predict the unsampled cases.

`SplitControl` constructs an `MLControl` object for splitting data into a seperate trianing and test set (Hastie et al. 2009).

`TrainControl` constructs an `MLControl` object for training and performance evaluation to be performed on the same training set (Efron 1986).

The base `MLControl` constructor initializes a set of control parameters that are common to all resampling methods.

Value

`MLControl` class object.

References

Efron B and Tibshirani RJ (1993). An Introduction to the Bootstrap. Monographs on Statistics and Applied Probability 57. Boca Raton, Florida, USA: Chapman & Hall/CRC.

Efron B and Gong G (1983). A leisurely look at the bootstrap, the jackknife, and cross-validation. The American Statistician, 37 (1): 36-48.

Harrell FE, Lee KL, and Mark DB (1996). Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. Statistics in Medicine, 15 (4): 361-387.

Kohavi R (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, 1137-43. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Davison AC and Hinkley DV (1997). Bootstrap Methods and Their Application. New York, NY, USA: Cambridge University Press.

Hastie T, Tibshirani R, and Friedman J (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics. New York, NY, USA: Springer.

Efron B (1986). How biased is the apparent error rate of a prediction rule? Journal of the American Statistical Association, 81 (394): 461-70.

See Also

resample, SelectedInput, SelectedModel, TunedInput, TunedModel

Examples

```
## Bootstrapping with 100 samples
BootControl(samples = 100)

## Optimism-corrected bootstrapping with 100 samples
BootOptimismControl(samples = 100)

## Cross-validation with 5 repeats of 10 folds
CVControl(folds = 10, repeats = 5)

## Optimism-corrected cross-validation with 5 repeats of 10 folds
CVOptimismControl(folds = 10, repeats = 5)

## Out-of-bootstrap validation with 100 samples
OOBControl(samples = 100)

## Split sample validation with 2/3 training and 1/3 testing
SplitControl(prop = 2/3)

## Training set evaluation
TrainControl()
```

MLMetric *MLMetric Class Constructor*

Description

Create a performance metric for use with the **MachineShop** package.

Usage

```
MLMetric(object, name = "MLMetric", label = name, maximize = TRUE)

MLMetric(object) <- value
```

Arguments

object	function to compute the metric, defined to accept observed and predicted as the first two arguments and with an ellipsis (...) to accommodate others.
name	character name of the object to which the metric is assigned.
label	optional character descriptor for the model.
maximize	logical indicating whether higher values of the metric correspond to better predictive performance.
value	list of arguments to pass to the MLMetric constructor.

Value

`MLMetric` class object.

See Also

`metrics`

Examples

```
f2_score <- function(observed, predicted, ...) {
  f_score(observed, predicted, beta = 2, ...)
}

MLMetric(f2_score) <- list(name = "f2_score",
                           label = "F Score (beta = 2)",
                           maximize = TRUE)
```

`MLModel` *MLModel Class Constructor*

Description

Create a model for use with the **MachineShop** package.

Usage

```
MLModel(
  name = "MLModel",
  label = name,
  packages = character(),
  response_types = character(),
  predictor_encoding = c(NA, "model.matrix", "terms"),
  params = list(),
  grid = function(x, length, random, ...) NULL,
  fit = function(formula, data, weights, ...) stop("no fit function"),
  predict = function(object, newdata, times, ...) stop("no predict function"),
  varimp = function(object, ...) NULL,
  ...
)
```

Arguments

`name`	character name of the object to which the model is assigned.
`label`	optional character descriptor for the model.
`packages`	character vector of packages required to use the model.

`response_types`	character vector of response variable types to which the model can be fit. Supported types are `"binary"`, = `"BinomialVariate"`, `"DiscreteVariate"`, `"factor"`, `"matrix"`, `"NegBinomialVariate"`, `"numeric"`, `"ordered"`, `"PoissonVariate"`, and `"Surv"`.
`predictor_encoding`	character string indicating whether the model is fit with predictor variables encoded as a `"model.matrix"`, a data.frame containing the originally specified model `"terms"`, or unspecified (default).
`params`	list of user-specified model parameters to be passed to the `fit` function.
`grid`	tuning grid function whose first agument `x` is a `ModelFrame` of the model fit data and formula, followed by a `length` to use in generating sequences of parameter values, a number of grid points to sample at `random`, and an ellipsis (`...`).
`fit`	model fitting function whose arguments are a `formula`, a `ModelFrame` named `data`, case `weights`, and an ellipsis.
`predict`	model prediction function whose arguments are the `object` returned by `fit`, a `ModelFrame` named `newdata` of predictor variables, optional vector of `times` at which to predict survival, and an ellipsis.
`varimp`	variable importance function whose arguments are the `object` returned by `fit`, optional arguments passed from calls to `varimp`, and an ellipsis.
`...`	arguments passed from other methods.

Details

If supplied, the `grid` function should return a list whose elements are named after and contain values of parameters to include in a tuning grid to be constructed automatically by the package.

Argument `data` in the `fit` function may be converted to a data frame with the `as.data.frame` function as needed. The function should return the object resulting from the model fit.

Values returned by the `predict` functions should be formatted according to the response variable types below.

factor vector or column matrix of probabilities for the second level of binary factors or a matrix whose columns contain the probabilities for factors with more than two levels.

matrix matrix of predicted responses.

numeric vector or column matrix of predicted responses.

Surv matrix whose columns contain survival probabilities at `times` if supplied or a vector of predicted survival means otherwise.

The `varimp` function should return a vector of importance values named after the predictor variables or a matrix or data frame whose rows are named after the predictors.

Value

`MLModel` class object.

See Also

`models`, `fit`, `resample`

Examples

```
## Logistic regression model
LogisticModel <- MLModel(
  name = "LogisticModel",
  response_types = "binary",
  fit = function(formula, data, weights, ...) {
    glm(formula, data = data, weights = weights, family = binomial, ...)
  },
  predict = function(object, newdata, ...) {
    predict(object, newdata = newdata, type = "response")
  },
  varimp = function(object, ...) {
    pchisq(coef(object)^2 / diag(vcov(object)), 1)
  }
)

library(MASS)
res <- resample(type ~ ., data = Pima.tr, model = LogisticModel)
summary(res)
```

`ModeledInput` *ModeledInput Classes*

Description

Class for storing a model input and specification pair for **MachineShop** model fitting.

Usage

```
ModeledInput(x, ...)

## S3 method for class 'formula'
ModeledInput(x, data, model, ...)

## S3 method for class 'matrix'
ModeledInput(x, y, model, ...)

## S3 method for class 'ModelFrame'
ModeledInput(x, model, ...)

## S3 method for class 'recipe'
ModeledInput(x, model, ...)

## S3 method for class 'MLModel'
ModeledInput(x, ...)

## S3 method for class 'MLModelFunction'
ModeledInput(x, ...)
```

Arguments

x	input specifying a relationship between model predictor and response variables. Alternatively, a model function or call may be given first followed by the input specification.
...	arguments passed to other methods.
data	data frame or an object that can be converted to one.
model	model function, function name, or call.
y	response variable.

Value

ModeledFrame or ModeledRecipe class object that inherits from ModelFrame or recipe.

See Also

fit, resample, SelectedInput

Examples

```
## Modeled model frame
mod_mf <- ModeledInput(sale_amount ~ ., data = ICHomes, model = GLMModel)
fit(mod_mf)

## Modeled recipe
library(recipes)

rec <- recipe(sale_amount ~ ., data = ICHomes)
mod_rec <- ModeledInput(rec, model = GLMModel)
fit(mod_rec)
```

ModelFrame *ModelFrame Class*

Description

Class for storing data, formulas, and other attributes for **MachineShop** model fitting.

Usage

```
ModelFrame(x, ...)

## S3 method for class 'formula'
ModelFrame(x, data, na.rm = TRUE, weights = NULL, strata = NULL, ...)

## S3 method for class 'matrix'
ModelFrame(
  x,
  y = NULL,
  na.rm = TRUE,
  offsets = NULL,
  weights = NULL,
  strata = NULL,
  ...
)
```

Arguments

`x`	model `formula` or `matrix` of predictor variables.
`...`	arguments passed to other methods.
`data`	data frame or an object that can be converted to one.
`na.rm`	logical indicating whether to remove cases with `NA` values for any of the model variables.
`weights`	vector of case weights [default: equal].
`strata`	vector of resampling stratification levels [default: none].
`y`	response variable.
`offsets`	numeric vector, matrix, or data frame of values to be added with a fixed coefficient of 1 to linear predictors in compatible regression models.

Value

`ModelFrame` class object that inherits from `data.frame`.

See Also

`fit`, `resample`, `response`, `SelectedInput`

Examples

```
mf <- ModelFrame(ncases / (ncases + ncontrols) ~ agegp + tobgp + alcgp,
                 data = esoph, weights = with(esoph, ncases + ncontrols))
gbm_fit <- fit(mf, model = GBMModel)
varimp(gbm_fit)
```

`modelinfo` *Display Model Information*

Description

Display information about models supplied by the **MachineShop** package.

Usage

```
modelinfo(...)
```

Arguments

`...`	model functions, function names, or calls; observed responses for which to display information. If none are specified, information is returned on all available models by default.

Value

List of named model elements each containing the following components:

label character descriptor for the model.

packages character vector of source packages required to use the model. These need only be installed with the `install.packages` function or by equivalent means; but need not be loaded with, for example, the `library` function.

response_types character vector of response variable types supported by the model.

arguments closure with the argument names and corresponding default values of the model function.

grid logical indicating whether automatic generation of tuning parameter grids is implemented for the model.

varimp logical indicating whether variable importance is defined for the model.

Examples

```
## All models
modelinfo()

## Models by response types
names(modelinfo(factor(0)))
names(modelinfo(factor(0), numeric(0)))

## Model-specific information
modelinfo(GBMModel)
```

`models` *Models*

Description

Model constructor functions supplied by **MachineShop** are summarized in the table below according to the types of response variables with which each can be used.

Function	Categorical	Continuous	Survival
AdaBagModel	f		
AdaBoostModel	f		
BARTModel	f	n	S
BARTMachineModel	b	n	
BlackBoostModel	b	n	S
C50Model	f		
CForestModel	f	n	S
CoxModel			S
CoxStepAICModel			S
EarthModel	f	n	
FDAModel	f		
GAMBoostModel	b	n	S
GBMModel	f	n	S
GLMBoostModel	b	n	S
GLMModel	f	m,n	
GLMStepAICModel	b	n	
GLMNetModel	f	m,n	S
KNNModel	f,o	n	
LARSModel		n	
LDAModel	f		
LMModel	f	m,n	
MDAModel	f		
NaiveBayesModel	f		
NNetModel	f	n	
PDAModel	f		
PLSModel	f	n	
POLRModel	o		
QDAModel	f		
RandomForestModel	f	n	
RangerModel	f	n	S
RPartModel	f	n	S
SurvRegModel			S
SurvRegStepAICModel			S
SVMModel	f	n	
SVMANOVAModel	f	n	
SVMBesselModel	f	n	
SVMLaplaceModel	f	n	
SVMLinearModel	f	n	
SVMPolyModel	f	n	
SVMRadialModel	f	n	
SVMSplineModel	f	n	
SVMTanhModel	f	n	
TreeModel	f	n	

XGBModel	f	n	S
XGBDARTModel	f	n	S
XGBLinearModel	f	n	S
XGBTreeModel	f	n	S

Categorical: b = binary, f = factor, o = ordered
Continuous: m = matrix, n = numeric
Survival: S = Surv

Models may be combined, tuned, or selected with the following meta-model functions.

StackedModel	Stacked regression
SuperModel	Super learner
SelectedModel	Model selection from a candidate set
TunedModel	Model tuning over a parameter grid

See Also

modelinfo, fit, resample

`NaiveBayesModel` *Naive Bayes Classifier Model*

Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using Bayes rule.

Usage

```
NaiveBayesModel(laplace = 0)
```

Arguments

laplace	positive numeric controlling Laplace smoothing.

Details

Response Types: `factor`

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

naiveBayes, fit, resample

Examples

```
fit(Species ~ ., data = iris, model = NaiveBayesModel)
```

NNetModel *Neural Network Model*

Description

Fit single-hidden-layer neural network, possibly with skip-layer connections.

Usage

```
NNetModel(
  size = 1,
  linout = NULL,
  entropy = NULL,
  softmax = NULL,
  censored = FALSE,
  skip = FALSE,
  rang = 0.7,
  decay = 0,
  maxit = 100,
  trace = FALSE,
  MaxNWts = 1000,
  abstol = 1e-04,
  reltol = 1e-08
)
```

Arguments

`size`	number of units in the hidden layer.
`linout`	switch for linear output units. Set automatically according to the class type of the response variable [numeric: `TRUE`, other: `FALSE`].
`entropy`	switch for entropy (= maximum conditional likelihood) fitting.
`softmax`	switch for softmax (log-linear model) and maximum conditional likelihood fitting.
`censored`	a variant on softmax, in which non-zero targets mean possible classes.
`skip`	switch to add skip-layer connections from input to output.
`rang`	Initial random weights on [`-rang`, `rang`].
`decay`	parameter for weight decay.

`maxit`	maximum number of iterations.
`trace`	switch for tracing optimization.
`MaxNWts`	maximum allowable number of weights.
`abstol`	stop if the fit criterion falls below `abstol`, indicating an essentially perfect fit.
`reltol`	stop if the optimizer is unable to reduce the fit criterion by a factor of at least `1 -reltol`.

Details

Response Types: `factor`, `numeric`

Automatic Tuning of Grid Parameters: `size`, `decay`

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`nnet`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = NNetModel)
```

`ParameterGrid` *Tuning Parameters Grid*

Description

Defines a tuning grid from a set of parameters.

Usage

```
ParameterGrid(...)

## S3 method for class 'param'
ParameterGrid(..., length = 3, random = FALSE)

## S3 method for class 'list'
ParameterGrid(x, length = 3, random = FALSE, ...)

## S3 method for class 'parameters'
ParameterGrid(x, length = 3, random = FALSE, ...)
```

Arguments

`...`	named `param` objects as defined in the **dials** package.
`length`	single number or vector of numbers of parameter values to use in constructing a regular grid if `random = FALSE`; ignored otherwise.
`random`	number of unique grid points to sample at random or `FALSE` for all points from a regular grid defined by `length`.
`x`	list of named `param` objects or a `parameters` object.

Value

`ParameterGrid` class object that inherits from `parameters` and `Grid`.

See Also

`TunedModel`

Examples

```
## GBMModel tuning parameters
library(dials)

grid <- ParameterGrid(
  n.trees = trees(),
  interaction.depth = tree_depth(),
  random = 5
)
TunedModel(GBMModel, grid = grid)
```

`performance` *Model Performance Metrics*

Description

Compute measures of model performance.

Usage

```
performance(x, ...)

## S3 method for class 'BinomialVariate'
performance(
  x,
  y,
  metrics = MachineShop::settings("metrics.numeric"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'factor'
performance(
  x,
  y,
  metrics = MachineShop::settings("metrics.factor"),
  cutoff = MachineShop::settings("cutoff"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'matrix'
performance(
  x,
  y,
  metrics = MachineShop::settings("metrics.matrix"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'numeric'
performance(
  x,
  y,
  metrics = MachineShop::settings("metrics.numeric"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'Surv'
performance(
  x,
  y,
  metrics = MachineShop::settings("metrics.Surv"),
  cutoff = MachineShop::settings("cutoff"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'ConfusionList'
performance(x, ...)

## S3 method for class 'ConfusionMatrix'
performance(x, metrics = MachineShop::settings("metrics.ConfusionMatrix"), ...)

## S3 method for class 'Resamples'
performance(x, ...)
```

Arguments

x	observed responses; or confusion or resample result containing observed and predicted responses.
...	arguments passed from the Resamples method to the response type-specific methods or from the method for ConfusionList to ConfusionMatrix.
y	predicted responses if not contained in x.
metrics	metric function, function name, or vector of these with which to calculate performance.
na.rm	logical indicating whether to remove observed or predicted responses that are NA when calculating metrics.
cutoff	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified.

See Also

plot, summary

Examples

```
res <- resample(Species ~ ., data = iris, model = GBMModel)
(perf <- performance(res))
summary(perf)
plot(perf)

## Survival response example
library(survival)

gbm_fit <- fit(Surv(time, status) ~ ., data = veteran, model = GBMModel)

obs <- response(gbm_fit, newdata = veteran)
pred <- predict(gbm_fit, newdata = veteran, type = "prob")
performance(obs, pred)
```

---

performance_curve *Model Performance Curves*

Description

Calculate curves for the analysis of tradeoffs between metrics for assessing performance in classifying binary outcomes over the range of possible cutoff probabilities. Available curves include receiver operating characteristic (ROC) and precision recall.

Usage

```
performance_curve(x, ...)

## Default S3 method:
performance_curve(
  x,
  y,
  metrics = c(MachineShop::tpr, MachineShop::fpr),
  na.rm = TRUE,
  ...
)

## S3 method for class 'Resamples'
performance_curve(
  x,
  metrics = c(MachineShop::tpr, MachineShop::fpr),
  na.rm = TRUE,
  ...
)
```

Arguments

`x`	observed responses or resample result containing observed and predicted responses.
`...`	arguments passed to other methods.
`y`	predicted responses if not contained in `x`.
`metrics`	list of two performance metrics for the analysis [default: ROC metrics]. Precision recall curves can be obtained with `c(precision,recall)`.
`na.rm`	logical indicating whether to remove observed or predicted responses that are `NA` when calculating metrics.

Value

`PerformanceCurve` class object that inherits from `data.frame`.

See Also

`auc`, `c`, `plot`, `summary`

Examples

```
library(MASS)

res <- resample(type ~ ., data = Pima.tr, model = GBMModel)

## ROC curve
roc <- performance_curve(res)
plot(roc)
```

```
auc(roc)
```

---

plot *Model Performance Plots*

Description

Plot measures of model performance and predictor variable importance.

Usage

```
## S3 method for class 'Calibration'
plot(x, type = c("line", "point"), se = FALSE, ...)

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

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

## S3 method for class 'LiftCurve'
plot(
  x,
  find = NULL,
  diagonal = TRUE,
  stat = MachineShop::settings("stat.Curve"),
  ...
)

## S3 method for class 'MLModel'
plot(
  x,
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  type = c("boxplot", "density", "errorbar", "line", "violin"),
  ...
)

## S3 method for class 'PartialDependence'
plot(x, stats = NULL, ...)

## S3 method for class 'Performance'
plot(
  x,
  metrics = NULL,
  stat = MachineShop::settings("stat.Resamples"),
  type = c("boxplot", "density", "errorbar", "violin"),
  ...
)

## S3 method for class 'PerformanceCurve'
plot(
  x,
  type = c("tradeoffs", "cutoffs"),
  diagonal = FALSE,
  stat = MachineShop::settings("stat.Curve"),
  ...
)

## S3 method for class 'Resamples'
plot(
  x,
  metrics = NULL,
  stat = MachineShop::settings("stat.Resamples"),
  type = c("boxplot", "density", "errorbar", "violin"),
  ...
)

## S3 method for class 'VarImp'
plot(x, n = NULL, ...)
```

Arguments

`x`	calibration, confusion, lift, trained model fit, partial dependence, performance, performance curve, resample, or variable importance result.
`type`	type of plot to construct.
`se`	logical indicating whether to include standard error bars.
`...`	arguments passed to other methods.
`find`	numeric true positive rate at which to display reference lines identifying the corresponding rates of positive predictions.
`diagonal`	logical indicating whether to include a diagonal reference line.
`stat`	function or character string naming a function to compute a summary statistic on resampled metrics for trained `MLModel` line plots and `Resamples` model ordering. For `LiftCurve` and `PerformanceCurve` classes, plots are of resampled metrics aggregated by the statistic if given or of resample-specific metrics if `NULL`.
`metrics`	vector of numeric indexes or character names of performance metrics to plot.
`stats`	vector of numeric indexes or character names of partial dependence summary statistics to plot.
`n`	number of most important variables to include in the plot [default: all].

Examples

```
## Factor response example

fo <- Species ~ .
control <- CVControl()

gbm_fit <- fit(fo, data = iris, model = GBMModel, control = control)
plot(varimp(gbm_fit))

gbm_res1 <- resample(fo, iris, GBMModel(n.trees = 25), control)
gbm_res2 <- resample(fo, iris, GBMModel(n.trees = 50), control)
gbm_res3 <- resample(fo, iris, GBMModel(n.trees = 100), control)
plot(gbm_res3)

res <- c(GBM1 = gbm_res1, GBM2 = gbm_res2, GBM3 = gbm_res3)
plot(res)
```

`PLSModel` *Partial Least Squares Model*

Description

Function to perform partial least squares regression.

Usage

```
PLSModel(ncomp = 1, scale = FALSE)
```

Arguments

`ncomp`	number of components to include in the model.
`scale`	logical indicating whether to scale the predictors by the sample standard deviation.

Details

Response Types: `factor`, `numeric`

Automatic Tuning of Grid Parameters: `ncomp`

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`mvr`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = PLSModel)
```

POLRModel *Ordered Logistic or Probit Regression Model*

Description

Fit a logistic or probit regression model to an ordered factor response.

Usage

```
POLRModel(method = c("logistic", "probit", "loglog", "cloglog", "cauchit"))
```

Arguments

`method`	logistic or probit or (complementary) log-log or cauchit (corresponding to a Cauchy latent variable).

Details

Response Types: `ordered`

Further model details can be found in the source link below.

In calls to `varimp` for `POLRModel`, numeric argument `base` may be specified for the (negative) logarithmic transformation of p-values [defaul: `exp(1)`]. Transformed p-values are automatically scaled in the calculation of variable importance to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`.

Value

`MLModel` class object.

See Also

`polr`, `fit`, `resample`

Examples

```
library(MASS)

df <- within(Boston,
             medv <- cut(medv,
                         breaks = c(0, 10, 15, 20, 25, 50),
                         ordered = TRUE))
fit(medv ~ ., data = df, model = POLRModel)
```

predict *Model Prediction*

Description

Predict outcomes with a fitted model.

Usage

```
## S3 method for class 'MLModelFit'
predict(
  object,
  newdata = NULL,
  times = NULL,
  type = c("response", "prob"),
  cutoff = MachineShop::settings("cutoff"),
  dist = NULL,
  method = NULL,
  ...
)
```

Arguments

`object`	model fit result.
`newdata`	optional data frame with which to obtain predictions. If not specified, the training data will be used by default.
`times`	numeric vector of follow-up times at which to predict survival events/probabilities or `NULL` for predicted survival means.
`type`	specifies prediction on the original outcome scale (`"response"`) or on a probability distribution scale (`"prob"`).
`cutoff`	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified.
`dist`	character string specifying distributional approximations to estimated survival curves. Possible values are `"empirical"`, `"exponential"`, `"rayleigh"`, or `"weibull"`; with defaults of `"empirical"` for predicted survival events/probabilities and `"weibull"` for predicted survival means.
`method`	character string specifying the empirical method of estimating baseline survival curves for Cox proportional hazards-based models. Choices are `"breslow"`, `"efron"` (default), or `"fleming-harrington"`.
`...`	arguments passed to model-specific prediction functions.

See Also

`confusion`, `performance`, `metrics`

Examples

```
## Survival response example
library(survival)

gbm_fit <- fit(Surv(time, status) ~ ., data = veteran, model = GBMModel)
predict(gbm_fit, newdata = veteran, times = c(90, 180, 360), type = "prob")
```

---

print *Print MachineShop Objects*

Description

Print methods for objects defined in the **MachineShop** package.

Usage

```
## S3 method for class 'BinomialVariate'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'Calibration'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'ListOf'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'MLModel'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'ModelFrame'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'ModeledInput'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'Performance'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'PerformanceCurve'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'RecipeGrid'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'Resamples'
print(x, n = MachineShop::settings("max.print"), ...)
```

```
## S3 method for class 'SelectedInput'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'SurvMatrix'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'TrainBit'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'TunedInput'
print(x, n = MachineShop::settings("max.print"), ...)

## S3 method for class 'VarImp'
print(x, n = MachineShop::settings("max.print"), ...)
```

Arguments

`x`	object to print.
`n`	integer number of models or data frame rows to show.
`...`	arguments passed to other methods.

`QDAModel` *Quadratic Discriminant Analysis Model*

Description

Performs quadratic discriminant analysis.

Usage

```
QDAModel(
  prior = NULL,
  method = c("moment", "mle", "mve", "t"),
  nu = 5,
  use = c("plug-in", "predictive", "debiased", "looCV")
)
```

Arguments

`prior`	prior probabilities of class membership if specified or the class proportions in the training set otherwise.
`method`	type of mean and variance estimator.
`nu`	degrees of freedom for `method = "t"`.
`use`	type of parameter estimation to use for prediction.

Details

Response Types: `factor`

The `predict` function for this model additionally accepts the following argument.

`prior` prior class membership probabilities for prediction data if different from the training set.

Default values for the `NULL` arguments and further model details can be found in the source links below.

Value

`MLModel` class object.

See Also

`qda`, `predict.qda`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = QDAModel)
```

`RandomForestModel` *Random Forest Model*

Description

Implementation of Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression.

Usage

```
RandomForestModel(
  ntree = 500,
  mtry = .(if (is.factor(y)) floor(sqrt(nvars)) else max(floor(nvars/3), 1)),
  replace = TRUE,
  nodesize = .(if (is.factor(y)) 1 else 5),
  maxnodes = NULL
)
```

Arguments

`ntree`	number of trees to grow.
`mtry`	number of variables randomly sampled as candidates at each split.
`replace`	should sampling of cases be done with or without replacement?
`nodesize`	minimum size of terminal nodes.
`maxnodes`	maximum number of terminal nodes trees in the forest can have.

Details

Response Types: `factor`, `numeric`

Automatic Tuning of Grid Parameters: `mtry`, `nodesize`*

* included only in randomly sampled grid points

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`randomForest`, `fit`, `resample`

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = RandomForestModel)
```

`RangerModel` *Fast Random Forest Model*

Description

Fast implementation of random forests or recursive partitioning.

Usage

```
RangerModel(
  num.trees = 500,
  mtry = NULL,
  importance = c("impurity", "impurity_corrected", "permutation"),
  min.node.size = NULL,
  replace = TRUE,
  sample.fraction = ifelse(replace, 1, 0.632),
  splitrule = NULL,
  num.random.splits = 1,
  alpha = 0.5,
  minprop = 0.1,
  split.select.weights = NULL,
  always.split.variables = NULL,
  respect.unordered.factors = NULL,
  scale.permutation.importance = FALSE,
  verbose = FALSE
)
```

Arguments

`num.trees`	number of trees.
`mtry`	number of variables to possibly split at in each node.
`importance`	variable importance mode.
`min.node.size`	minimum node size.
`replace`	logical indicating whether to sample with replacement.
`sample.fraction`	fraction of observations to sample.
`splitrule`	splitting rule.
`num.random.splits`	number of random splits to consider for each candidate splitting variable in the `"extratrees"` rule.
`alpha`	significance threshold to allow splitting in the `"maxstat"` rule.
`minprop`	lower quantile of covariate distribution to be considered for splitting in the `"maxstat"` rule.
`split.select.weights`	numeric vector with weights between 0 and 1, representing the probability to select variables for splitting.
`always.split.variables`	character vector with variable names to be always selected in addition to the `mtry` variables tried for splitting.
`respect.unordered.factors`	handling of unordered factor covariates.
`scale.permutation.importance`	scale permutation importance by standard error.
`verbose`	show computation status and estimated runtime.

Details

Response Types: `factor`, `numeric`, `Surv`

Automatic Tuning of Grid Parameters: `mtry`, `min.node.size`*, `splitrule`*

* included only in randomly sampled grid points

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`ranger`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = RangerModel)
```

`recipe_roles` *Set Recipe Roles*

Description

Add to or replace the roles of variables in a preprocessing recipe.

Usage

```
role_binom(recipe, x, size)

role_case(recipe, stratum, weight, replace = FALSE)

role_pred(recipe, offset, replace = FALSE)

role_surv(recipe, time, event)
```

Arguments

`recipe`	existing recipe object.
`x, size`	number of counts and trials for the specification of a `BinomialVariate` outcome.
`stratum`	variable for stratified resampling of cases.
`weight`	numeric variable of case weights for model fitting.
`replace`	logical indicating whether to replace existing roles.
`offset`	numeric variable to be added to a linear predictor, such as in a generalized linear model, with known coefficient 1 rather than an estimated coefficient.
`time, event`	numeric follow up time and 0-1 numeric or logical event indicator for specification of a `Surv` outcome. If the event indicator is omitted, all cases are assumed to have events.

Value

An updated recipe object.

See Also

`recipe`

Examples

```
library(survival)
library(recipes)

rec <- recipe(time + status ~ ., data = veteran) %>%
  role_surv(time = time, event = status) %>%
  role_case(stratum = status)

(res <- resample(rec, model = CoxModel))
summary(res)
```

`resample` *Resample Estimation of Model Performance*

Description

Estimation of the predictive performance of a model estimated and evaluated on training and test samples generated from an observed data set.

Usage

```
resample(x, ...)

## S3 method for class 'formula'
resample(x, data, model, control = MachineShop::settings("control"), ...)

## S3 method for class 'matrix'
resample(x, y, model, control = MachineShop::settings("control"), ...)

## S3 method for class 'ModelFrame'
resample(x, model, control = MachineShop::settings("control"), ...)

## S3 method for class 'recipe'
resample(x, model, control = MachineShop::settings("control"), ...)

## S3 method for class 'MLModel'
resample(x, ...)

## S3 method for class 'MLModelFunction'
resample(x, ...)
```

Arguments

x	input specifying a relationship between model predictor and response variables. Alternatively, a model function or call may be given first followed by the input specification and `control` value.

`...`	arguments passed to other methods.
`data`	data frame containing observed predictors and outcomes.
`model`	model function, function name, or call; ignored and can be omitted when resampling modeled inputs.
`control`	control function, function name, or call defining the resampling method to be employed.
`y`	response variable.

Details

Stratified resampling is performed for the `formula` method according to values of the response variable; i.e. categorical levels for `factor`, continuous for `numeric`, and event status `Surv`.

User-specified stratification variables may be specified for `ModelFrames` upon creation with the `strata` argument in its constructor. Resampling of this class is unstratified by default.

Variables in `recipe` specifications may be designated as case strata with the `role_case` function. Resampling will be unstratified otherwise.

Value

`Resamples` class object.

See Also

`c`, `metrics`, `performance`, `plot`, `summary`

Examples

```
## Factor response example

fo <- Species ~ .
control <- CVControl()

gbm_res1 <- resample(fo, iris, GBMModel(n.trees = 25), control)
gbm_res2 <- resample(fo, iris, GBMModel(n.trees = 50), control)
gbm_res3 <- resample(fo, iris, GBMModel(n.trees = 100), control)

summary(gbm_res1)
plot(gbm_res1)

res <- c(GBM1 = gbm_res1, GBM2 = gbm_res2, GBM3 = gbm_res3)
summary(res)
plot(res)
```

`response` *Extract Response Variable*

Description

Extract the response variable from an object.

Usage

```
response(object, ...)

## S3 method for class 'MLModelFit'
response(object, newdata = NULL, ...)

## S3 method for class 'ModelFrame'
response(object, newdata = NULL, ...)

## S3 method for class 'recipe'
response(object, newdata = NULL, ...)
```

Arguments

`object`	model fit result, `ModelFrame`, or `recipe`.
`...`	arguments passed to other methods.
`newdata`	data frame from which to extract the response variable values if given; otherwise, `object` is used.

Examples

```
## Survival response example
library(survival)

mf <- ModelFrame(Surv(time, status) ~ ., data = veteran)
response(mf)
```

`RPartModel` *Recursive Partitioning and Regression Tree Models*

Description

Fit an `rpart` model.

Usage

```
RPartModel(
  minsplit = 20,
  minbucket = round(minsplit/3),
  cp = 0.01,
  maxcompete = 4,
  maxsurrogate = 5,
  usesurrogate = 2,
  xval = 10,
  surrogatestyle = 0,
  maxdepth = 30
)
```

Arguments

`minsplit`	minimum number of observations that must exist in a node in order for a split to be attempted.
`minbucket`	minimum number of observations in any terminal node.
`cp`	complexity parameter.
`maxcompete`	number of competitor splits retained in the output.
`maxsurrogate`	number of surrogate splits retained in the output.
`usesurrogate`	how to use surrogates in the splitting process.
`xval`	number of cross-validations.
`surrogatestyle`	controls the selection of a best surrogate.
`maxdepth`	maximum depth of any node of the final tree, with the root node counted as depth 0.

Details

Response Types: `factor`, `numeric`, `Surv`

Automatic Tuning of Grid Parameters: `cp`

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`rpart`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = RPartModel)
```

SelectedInput *Selected Model Inputs*

Description

Formula, design matrix, model frame, or recipe selection from a candidate set.

Usage

```
SelectedInput(...)

## S3 method for class 'formula'
SelectedInput(
  ...,
  data,
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff")
)

## S3 method for class 'matrix'
SelectedInput(
  ...,
  y,
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff")
)

## S3 method for class 'ModelFrame'
SelectedInput(
  ...,
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff")
)

## S3 method for class 'recipe'
SelectedInput(
  ...,
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff")
)

## S3 method for class 'list'
SelectedInput(x, ...)
```

Arguments

`...`	inputs specifying relationships between model predictor and response variables. Supplied inputs must all be of the same type and may be named or unnamed.
`data`	data frame or an object that can be converted to one.
`control`	control function, function name, or call defining the resampling method to be employed.
`metrics`	metric function, function name, or vector of these with which to calculate performance. If not specified, default metrics defined in the performance functions are used. Recipe selection is based on the first calculated metric.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for recipe selection.
`cutoff`	argument passed to the `metrics` functions.
`y`	response variable.
`x`	list of inputs followed by arguments passed to their method function.

Value

`SelectedModelFrame` or `SelectedModelRecipe` class object that inherits from `SelectedInput` and `ModelFrame` or `recipe`.

See Also

`fit`, `resample`

Examples

```
## Selected model frame
sel_mf <- SelectedInput(
  sale_amount ~ sale_year + built + style + construction,
  sale_amount ~ sale_year + base_size + bedrooms + basement,
  data = ICHomes
)

fit(sel_mf, model = GLMModel)

## Selected recipe
library(recipes)
library(MASS)

rec1 <- recipe(medv ~ crim + zn + indus + chas + nox + rm, data = Boston)
rec2 <- recipe(medv ~ chas + nox + rm + age + dis + rad + tax, data = Boston)
sel_rec <- SelectedInput(rec1, rec2)
```

```
fit(sel_rec, model = GLMModel)
```

SelectedModel *Selected Model*

Description

Model selection from a candidate set.

Usage

```
SelectedModel(
  ...,
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff")
)
```

Arguments

`...`	model functions, function names, calls, or vectors of these to serve as the candidate set from which to select, such as that returned by `expand_model`.
`control`	control function, function name, or call defining the resampling method to be employed.
`metrics`	metric function, function name, or vector of these with which to calculate performance. If not specified, default metrics defined in the performance functions are used. Model selection is based on the first calculated metric.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for model selection.
`cutoff`	argument passed to the `metrics` functions.

Details

Response Types: `factor`, `numeric`, `ordered`, `Surv`

Value

`SelectedModel` class object that inherits from `MLModel`.

See Also

`fit`, `resample`

Examples

```
model_fit <- fit(sale_amount ~ ., data = ICHomes,
                 model = SelectedModel(GBMModel, GLMNetModel, SVMRadialModel))
(selected_model <- as.MLModel(model_fit))
summary(selected_model)
```

`settings` *MachineShop Settings*

Description

Allow the user to view or change global settings which affect default behaviors of functions in the **MachineShop** package.

Usage

```
settings(...)
```

Arguments

`...`	character names of settings to view, `name = value` pairs giving the values of settings to change, a vector of these, `"reset"` to restore all package defaults, or no arguments to view all settings. Partial matching of setting names is supported.

Value

The setting value if only one is specified to view. Otherwise, a list of the values of specified settings as they existed prior to any requested changes. Such a list can be passed as an argument to `settings` to restore their values.

Settings

`control` function, function name, or call defining a default resampling method [default: `"CVControl"`].

`cutoff` numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified [default: 0.5].

`dist.Surv` character string specifying distributional approximations to estimated survival curves for predicting survival means. Choices are `"empirical"` for the Kaplan-Meier estimator, `"exponential"`, or `"weibull"` (default).

`dist.SurvProbs` character string specifying distributional approximations to estimated survival curves for predicting survival events/probabilities. Choices are `"empirical"` (default) for the Kaplan-Meier estimator, `"exponential"`, or `"weibull"`.

`grid` number of parameter-specific values to generate automatically for tuning of models that have pre-defined grids or a `Grid` function, function name, or call [default: 3].

`max.print` number of models or data rows to show with print methods or `Inf` to show all [default: 10].

`method.EmpiricalSurv` character string specifying the empirical method of estimating baseline survival curves for Cox proportional hazards-based models. Choices are `"breslow"`, `"efron"` (default), or `"fleming-harrington"`.

`metrics.ConfusionMatrix` function, function name, or vector of these with which to calculate performance metrics for confusion matrices [default: `c(Accuracy = "accuracy",Kappa = "kappa2",`Weighted Kappa` = "weighted_kappa2",Sensitivity = "sensitivity",Specificity = "specificity")`].

`metrics.factor` function, function name, or vector of these with which to calculate performance metrics for factor responses [default: `c(Brier = "brier",Accuracy = "accuracy",Kappa = "kappa2",`Weighted Kappa` = "weighted_kappa2",`ROC AUC` = "roc_auc",Sensitivity = "sensitivity",Specificity = "specificity")`].

`metrics.matrix` function, function name, or vector of these with which to calculate performance metrics for matrix responses [default: `c(RMSE = "rmse",R2 = "r2",MAE = "mae")`].

`metrics.numeric` function, function name, or vector of these with which to calculate performance metrics for numeric responses [default: `c(RMSE = "rmse",R2 = "r2",MAE = "mae")`].

`metrics.Surv` function, function name, or vector of these with which to calculate performance metrics for survival responses [default: `c(`C-Index` = "cindex",Brier = "brier",`ROC AUC` = "roc_auc",Accuracy = "accuracy")`].

`progress.resample` logical indicating whether to display a progress bar during resampling [default: `TRUE`]. Displayed only if a computing cluster is not registered or is registered with the **doSNOW** package.

`require` names of installed packages to load during parallel execution of resampling algorithms [default: `c("MachineShop","survival","recipes")`].

`reset` character names of settings to reset to their default values.

`RHS.formula` non-modifiable character vector of operators and functions allowed in traditional formula specifications.

`stat.Curve` function or character string naming a function to compute one summary statistic at each cutoff value of resampled metrics in performance curves, or `NULL` for resample-specific metrics [default: `"base::mean"`].

`stat.Resamples` function or character string naming a function to compute one summary statistic to control the ordering of models in plots [default: `"base::mean"`].

`stat.train` function or character string naming a function to compute one summary statistic on resampled performance metrics for input selection or tuning or for model selection or tuning [default: `"base::mean"`].

`stats.PartialDependence` function, function name, or vector of these with which to compute partial dependence summary statistics [default: `c(Mean = "base::mean")`].

`stats.Resamples` function, function name, or vector of these with which to compute summary statistics on resampled performance metrics [default: `c(Mean = "base::mean",Median = "stats::median",SD = "stats::sd",Min = "base::min",Max = "base::max")`].

`verbose.resample` logical indicating whether to enable verbose messages when resampling [default: `FALSE`].

Examples

```
## View all current settings
settings()

## Change settings
presets <- settings(control = "BootControl", grid = 10)

## View one setting
settings("control")

## View multiple settings
settings("control", "grid")

## Restore the previous settings
settings(presets)
```

`StackedModel` *Stacked Regression Model*

Description

Fit a stacked regression model from multiple base learners.

Usage

```
StackedModel(..., control = MachineShop::settings("control"), weights = NULL)
```

Arguments

`...`	model functions, function names, calls, or vector of these to serve as base learners.
`control`	control function, function name, or call defining the resampling method to be employed for the estimation of base learner weights.
`weights`	optional fixed base learner weights.

Details

Response Types: `factor`, `numeric`, `ordered`, `Surv`

Value

`StackedModel` class object that inherits from `MLModel`.

References

Breiman, L. (1996) *Stacked Regression.* Machine Learning, 24, 49–64.

See Also

`fit`, `resample`

Examples

```
model <- StackedModel(GBMModel, SVMRadialModel, GLMNetModel(lambda = 0.01))
model_fit <- fit(sale_amount ~ ., data = ICHomes, model = model)
predict(model_fit, newdata = ICHomes)
```

`step_kmeans` *K-Means Clustering Variable Reduction*

Description

Creates a *specification* of a recipe step that will convert numeric variables into one or more by averaging within k-means clusters.

Usage

```
step_kmeans(
  recipe,
  ...,
  k = 5,
  center = TRUE,
  scale = TRUE,
  algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"),
  max_iter = 10,
  num_start = 1,
  replace = TRUE,
  prefix = "KMeans",
  role = "predictor",
  skip = FALSE,
  id = recipes::rand_id("kmeans")
)

## S3 method for class 'step_kmeans'
tidy(x, ...)

tunable.step_kmeans(x, ...)
```

Arguments

`recipe`	recipe object to which the step will be added.
`...`	one or more selector functions to choose which variables will be used to compute the components. See `selections` for more details. These are not currently used by the `tidy` method.

`k`	number of k-means clusterings of the variables. The value of `k` is constrained to be between 1 and one less than the number of original variables.
`center, scale`	logicals indicating whether to mean center and standard deviation scale the original variables prior to deriving components, or functions or names of functions for the centering and scaling.
`algorithm`	character string specifying the clustering algorithm to use.
`max_iter`	maximum number of algorithm iterations allowed.
`num_start`	number of random cluster centers generated for starting the Hartigan-Wong algorithm.
`replace`	logical indicating whether to replace the original variables.
`prefix`	character string prefix added to a sequence of zero-padded integers to generate names for the resulting new variables.
`role`	analysis role that added step variables should be assigned. By default, they are designated as model predictors.
`skip`	logical indicating whether to skip the step when the recipe is baked. While all operations are baked when `prep` is run, some operations may not be applicable to new data (e.g. processing outcome variables). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	unique character string to identify the step.
`x`	`step_kmeans` object.

Details

K-means clustering partitions variables into k groups such that the sum of squares between the variables and their assigned cluster means is minimized. Variables within each cluster are then averaged to derive a new set of k variables.

Value

Function `step_kmeans` creates a new step whose class is of the same name and inherits from `step_lincomp`, adds it to the sequence of existing steps (if any) in the recipe, and returns the updated recipe. For the `tidy` method, a tibble with columns `terms` (selectors or variables selected), `cluster` assignments, `sqdist` (squared distance from cluster centers), and `name` of the new variable names.

References

Forgy EW (1965). Cluster analysis of multivariate data: efficiency vs interpretability of classifications. Biometrics 21, 768–769.

Hartigan JA and Wong MA (1979). A K-means clustering algorithm. Applied Statistics 28, 100–108.

Lloyd SP (1957, 1982). Least squares quantization in PCM. Technical Note, Bell Laboratories. Published in 1982 in IEEE Transactions on Information Theory 28, 128–137.

MacQueen J (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, eds L. M. Le Cam & J. Neyman, 1, 281–297. Berkeley, CA: University of California Press.

See Also

kmeans, recipe, prep, bake

Examples

```
library(recipes)

rec <- recipe(rating ~ ., data = attitude)
kmeans_rec <- rec %>%
  step_kmeans(all_predictors(), k = 3)
kmeans_prep <- prep(kmeans_rec, training = attitude)
kmeans_data <- bake(kmeans_prep, attitude)

pairs(kmeans_data, lower.panel = NULL)

tidy(kmeans_rec, number = 1)
tidy(kmeans_prep, number = 1)
```

---

step_kmedoids *K-Medoids Clustering Variable Selection*

Description

Creates a *specification* of a recipe step that will partition numeric variables according to k-medoids clustering and select the cluster medoids.

Usage

```
step_kmedoids(
  recipe,
  ...,
  k = 5,
  center = TRUE,
  scale = TRUE,
  method = c("pam", "clara"),
  metric = "euclidean",
  optimize = FALSE,
  num_samp = 50,
  samp_size = 40 + 2 * k,
  replace = TRUE,
  prefix = "KMedoids",
  role = "predictor",
  skip = FALSE,
  id = recipes::rand_id("kmedoids")
)

tunable.step_kmedoids(x, ...)
```

Arguments

`recipe`	recipe object to which the step will be added.
`...`	one or more selector functions to choose which variables will be used to compute the components. See `selections` for more details. These are not currently used by the `tidy` method.
`k`	number of k-medoids clusterings of the variables. The value of `k` is constrained to be between 1 and one less than the number of original variables.
`center, scale`	logicals indicating whether to mean center and median absolute deviation scale the original variables prior to cluster partitioning, or functions or names of functions for the centering and scaling; not applied to selected variables.
`method`	character string specifying one of the clustering methods provided by the **cluster** package. The `clara` (clustering large applications) method is an extension of `pam` (partitioning around medoids) designed to handle large datasets.
`metric`	character string specifying the distance metric for calculating dissimilarities between observations as `"euclidean"`, `"manhattan"`, or `"jaccard"` (`clara` only).
`optimize`	logical indicator or 0:5 integer level specifying optimization for the `pam` clustering method.
`num_samp`	number of sub-datasets to sample for the `clara` clustering method.
`samp_size`	number of cases to include in each sub-dataset.
`replace`	logical indicating whether to replace the original variables.
`prefix`	if the original variables are not replaced, the selected variables are added to the dataset with the character string prefix added to their names; otherwise, the original variable names are retained.
`role`	analysis role that added step variables should be assigned. By default, they are designated as model predictors.
`skip`	logical indicating whether to skip the step when the recipe is baked. While all operations are baked when `prep` is run, some operations may not be applicable to new data (e.g. processing outcome variables). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	unique character string to identify the step.
`x`	`step_kmedoids` object.

Details

K-medoids clustering partitions variables into k groups such that the dissimilarity between the variables and their assigned cluster medoids is minimized. Cluster medoids are then returned as a set of k variables.

Value

Function `step_kmedoids` creates a new step whose class is of the same name and inherits from `step_sbf`, adds it to the sequence of existing steps (if any) in the recipe, and returns the updated recipe. For the `tidy` method, a tibble with columns `terms` (selectors or variables selected), `cluster` assignments, `selected` (logical indicator of selected cluster medoids), `silhouette` (silhouette values), and `name` of the selected variable names.

References

Kaufman L and Rousseeuw PJ (1990). Finding Groups in Data: An Introduction to Cluster Analysis. Wiley: New York.

Reynolds A, Richards G, de la Iglesia B and Rayward-Smith V (1992). Clustering rules: a comparison of partitioning and hierarchical clustering algorithms. Journal of Mathematical Modelling and Algorithms 5, 475–504.

See Also

`pam`, `clara`, `recipe`, `prep`, `bake`

Examples

```
library(recipes)

rec <- recipe(rating ~ ., data = attitude)
kmedoids_rec <- rec %>%
  step_kmedoids(all_predictors(), k = 3)
kmedoids_prep <- prep(kmedoids_rec, training = attitude)
kmedoids_data <- bake(kmedoids_prep, attitude)

pairs(kmedoids_data, lower.panel = NULL)

tidy(kmedoids_rec, number = 1)
tidy(kmedoids_prep, number = 1)
```

---

step_lincomp *Linear Components Variable Reduction*

Description

Creates a *specification* of a recipe step that will compute one or more linear combinations of a set of numeric variables according to a user-specified transformation matrix.

Usage

```
step_lincomp(
  recipe,
  ...,
  transform,
  num_comp = 5,
  options = list(),
  center = TRUE,
  scale = TRUE,
  replace = TRUE,
  prefix = "LinComp",
  role = "predictor",
  skip = FALSE,
  id = recipes::rand_id("lincomp")
)

## S3 method for class 'step_lincomp'
tidy(x, ...)

tunable.step_lincomp(x, ...)
```

Arguments

recipe	recipe object to which the step will be added.
...	one or more selector functions to choose which variables will be used to compute the components. See `selections` for more details. These are not currently used by the `tidy` method.
transform	function whose first argument `x` is a matrix of variables with which to compute linear combinations and second argument `step` is the current step. The function should return a transformation `matrix` or `Matrix` of variable weights in its columns, or return a list with element `` `weights` `` containing the transformation matrix and possibly with other elements to be included as attributes in output from the `tidy` method.
num_comp	number of components to derive. The value of `num_comp` will be constrained to a minimum of 1 and maximum of the number of original variables when `prep` is run.
options	list of elements to be added to the step object for use in the `transform` function.
center, scale	logicals indicating whether to mean center and standard deviation scale the original variables prior to deriving components, or functions or names of functions for the centering and scaling.
replace	logical indicating whether to replace the original variables.
prefix	character string prefix added to a sequence of zero-padded integers to generate names for the resulting new variables.
role	analysis role that added step variables should be assigned. By default, they are designated as model predictors.
skip	logical indicating whether to skip the step when the recipe is baked. While all operations are baked when `prep` is run, some operations may not be applicable to new data (e.g. processing outcome variables). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
id	unique character string to identify the step.
x	`step_lincomp` object.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (selectors or variables selected), `weight` of each variable in the linear transformations, and `name` of the new variable names.

See Also

recipe, prep, bake

Examples

```
library(recipes)

pca_mat <- function(x, step) {
  prcomp(x)$rotation[, 1:step$num_comp, drop = FALSE]
}

rec <- recipe(rating ~ ., data = attitude)
lincomp_rec <- rec %>%
  step_lincomp(all_numeric(), -all_outcomes(),
               transform = pca_mat, num_comp = 3, prefix = "PCA")

lincomp_prep <- prep(lincomp_rec, training = attitude)
lincomp_data <- bake(lincomp_prep, attitude)

pairs(lincomp_data, lower.panel = NULL)

tidy(lincomp_rec, number = 1)
tidy(lincomp_prep, number = 1)
```

---

step_sbf *Variable Selection by Filtering*

Description

Creates a *specification* of a recipe step that will select variables from a candidate set according to a user-specified filtering function.

Usage

```
step_sbf(
  recipe,
  ...,
  filter,
  multivariate = FALSE,
  options = list(),
  replace = TRUE,
  prefix = "SBF",
  role = "predictor",
  skip = FALSE,
  id = recipes::rand_id("sbf")
)
```

```
## S3 method for class 'step_sbf'
tidy(x, ...)
```

Arguments

`recipe`	recipe object to which the step will be added.
`...`	one or more selector functions to choose which variables will be used to compute the components. See `selections` for more details. These are not currently used by the `tidy` method.
`filter`	function whose first argument `x` is a univariate vector or a `multivariate` data frame of candidate variables from which to select, second argument `y` is the response variable as defined in preceding recipe steps, and third argument `step` is the current step. The function should return a logical value or vector of length equal the number of variables in `x` indicating whether to select the corresponding variable, or return a list or data frame with element `` `selected` `` containing the logical(s) and possibly with other elements of the same length to be included in output from the `tidy` method.
`multivariate`	logical indicating that candidate variables be passed to the `x` argument of the `filter` function separately as univariate vectors if `FALSE`, or altogether in one multivariate data frame if `TRUE`.
`options`	list of elements to be added to the step object for use in the `filter` function.
`replace`	logical indicating whether to replace the original variables.
`prefix`	if the original variables are not replaced, the selected variables are added to the dataset with the character string prefix added to their names; otherwise, the original variable names are retained.
`role`	analysis role that added step variables should be assigned. By default, they are designated as model predictors.
`skip`	logical indicating whether to skip the step when the recipe is baked. While all operations are baked when `prep` is run, some operations may not be applicable to new data (e.g. processing outcome variables). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	unique character string to identify the step.
`x`	`step_sbf` object.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (selectors or variables selected), `selected` (logical indicator of selected variables), and `name` of the selected variable names.

See Also

`recipe`, `prep`, `bake`

Examples

```
library(recipes)

glm_filter <- function(x, y, step) {
  model_fit <- glm(y ~ ., data = data.frame(y, x))
  p_value <- drop1(model_fit, test = "F")[-1, "Pr(>F)"]
  p_value < step$threshold
}

rec <- recipe(rating ~ ., data = attitude)
sbf_rec <- rec %>%
  step_sbf(all_numeric(), -all_outcomes(),
           filter = glm_filter, options = list(threshold = 0.05))

sbf_prep <- prep(sbf_rec, training = attitude)
sbf_data <- bake(sbf_prep, attitude)

pairs(sbf_data, lower.panel = NULL)

tidy(sbf_rec, number = 1)
tidy(sbf_prep, number = 1)
```

---

step_spca *Sparse Principal Components Analysis Variable Reduction*

Description

Creates a *specification* of a recipe step that will derive sparse principal components from one or more numeric variables.

Usage

```
step_spca(
  recipe,
  ...,
  num_comp = 5,
  sparsity = 0,
  num_var = NULL,
  shrinkage = 1e-06,
  center = TRUE,
  scale = TRUE,
  max_iter = 200,
  tol = 0.001,
  replace = TRUE,
  prefix = "SPCA",
  role = "predictor",
  skip = FALSE,
  id = recipes::rand_id("spca")
)

tunable.step_spca(x, ...)
```

Arguments

`recipe`	recipe object to which the step will be added.
`...`	one or more selector functions to choose which variables will be used to compute the components. See `selections` for more details. These are not currently used by the `tidy` method.
`num_comp`	number of components to derive. The value of `num_comp` will be constrained to a minimum of 1 and maximum of the number of original variables when `prep` is run.
`sparsity, num_var`	sparsity (L1 norm) penalty for each component or number of variables with non-zero component loadings. Larger sparsity values produce more zero loadings. Argument `sparsity` is ignored if `num_var` is given. The argument value may be a single number applied to all components or a vector of component-specific numbers.
`shrinkage`	numeric shrinkage (quadratic) penalty for the components to improve conditioning; larger values produce more shrinkage of component loadings toward zero.
`center, scale`	logicals indicating whether to mean center and standard deviation scale the original variables prior to deriving components, or functions or names of functions for the centering and scaling.
`max_iter`	maximum number of algorithm iterations allowed.
`tol`	numeric tolerance for the convergence criterion.
`replace`	logical indicating whether to replace the original variables.
`prefix`	character string prefix added to a sequence of zero-padded integers to generate names for the resulting new variables.
`role`	analysis role that added step variables should be assigned. By default, they are designated as model predictors.
`skip`	logical indicating whether to skip the step when the recipe is baked. While all operations are baked when `prep` is run, some operations may not be applicable to new data (e.g. processing outcome variables). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	unique character string to identify the step.
`x`	`step_spca` object.

Details

Sparse principal components analysis (SPCA) is a variant of PCA in which the original variables may have zero loadings in the linear combinations that form the components.

Value

Function `step_spca` creates a new step whose class is of the same name and inherits from `step_lincomp`, adds it to the sequence of existing steps (if any) in the recipe, and returns the updated recipe. For the `tidy` method, a tibble with columns `terms` (selectors or variables selected), `weight` of each variable loading in the components, and `name` of the new variable names; and with attribute `pev` containing the proportions of explained variation.

References

Zou H, Hastie T and Tibshirani R (2006). Sparse principal component analysis. Journal of Computational and Graphical Statistics, 15(2):265–286.

See Also

`spca`, `recipe`, `prep`, `bake`

Examples

```
library(recipes)

rec <- recipe(rating ~ ., data = attitude)
spca_rec <- rec %>%
  step_spca(all_predictors(), num_comp = 5, sparsity = 1)
spca_prep <- prep(spca_rec, training = attitude)
spca_data <- bake(spca_prep, attitude)

pairs(spca_data, lower.panel = NULL)

tidy(spca_rec, number = 1)
tidy(spca_prep, number = 1)
```

`summary` *Model Performance Summaries*

Description

Summary statistics for resampled model performance metrics.

Usage

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

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

## S3 method for class 'MLModel'
```

```
summary(
  object,
  stats = MachineShop::settings("stats.Resamples"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'Performance'
summary(
  object,
  stats = MachineShop::settings("stats.Resamples"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'PerformanceCurve'
summary(object, stat = MachineShop::settings("stat.Curve"), ...)

## S3 method for class 'Resamples'
summary(
  object,
  stats = MachineShop::settings("stats.Resamples"),
  na.rm = TRUE,
  ...
)
```

Arguments

`object`	confusion, lift, trained model fit, performance, performance curve, or resample result.
`...`	arguments passed to other methods.
`stats`	function, function name, or vector of these with which to compute summary statistics.
`na.rm`	logical indicating whether to exclude missing values.
`stat`	function or character string naming a function to compute a summary statistic at each cutoff value of resampled metrics in `PerformanceCurve`, or `NULL` for resample-specific metrics.

Value

An object of summmary statistics.

Examples

```
## Factor response example

fo <- Species ~ .
control <- CVControl()
```

```
gbm_res1 <- resample(fo, iris, GBMModel(n.trees = 25), control)
gbm_res2 <- resample(fo, iris, GBMModel(n.trees = 50), control)
gbm_res3 <- resample(fo, iris, GBMModel(n.trees = 100), control)
summary(gbm_res3)

res <- c(GBM1 = gbm_res1, GBM2 = gbm_res2, GBM3 = gbm_res3)
summary(res)
```

SuperModel *Super Learner Model*

Description

Fit a super learner model to predictions from multiple base learners.

Usage

```
SuperModel(
  ...,
  model = GBMModel,
  control = MachineShop::settings("control"),
  all_vars = FALSE
)
```

Arguments

`...`	model functions, function names, calls, or vector of these to serve as base learners.
`model`	model function, function name, or call defining the super model.
`control`	control function, function name, or call defining the resampling method to be employed for the estimation of base learner weights.
`all_vars`	logical indicating whether to include the original predictor variables in the super model.

Details

Response Types: `factor`, `numeric`, `ordered`, `Surv`

Value

`SuperModel` class object that inherits from `MLModel`.

References

van der Lann, M.J., Hubbard A.E. (2007) *Super Learner.* Statistical Applications in Genetics and Molecular Biology, 6(1).

See Also

`fit`, `resample`

Examples

```
model <- SuperModel(GBMModel, SVMRadialModel, GLMNetModel(lambda = 0.01))
model_fit <- fit(sale_amount ~ ., data = ICHomes, model = model)
predict(model_fit, newdata = ICHomes)
```

`SurvMatrix` *SurvMatrix Class Constructors*

Description

Create a matrix of survival events or probabilites.

Usage

```
SurvEvents(data = NA, times = NULL)

SurvProbs(data = NA, times = NULL)
```

Arguments

`data`	matrix, or object that can be coerced to one, with survival events or probabilities at points in time in the columns and cases in the rows.
`times`	numeric vector of survival times for the columns.

Value

Object that is of the same class as the constructor name and inherits from `SurvMatrix`. Examples of these are predicted survival events and probabilities returned by the predict function.

See Also

`performance`, `metrics`

`SurvRegModel` *Parametric Survival Model*

Description

Fits the accelerated failure time family of parametric survival models.

Usage

```
SurvRegModel(
  dist = c("weibull", "exponential", "gaussian", "logistic", "lognormal",
    "logloglogistic"),
  scale = NULL,
  parms = NULL,
  ...
)

SurvRegStepAICModel(
  dist = c("weibull", "exponential", "gaussian", "logistic", "lognormal",
    "logloglogistic"),
  scale = NULL,
  parms = NULL,
  ...,
  direction = c("both", "backward", "forward"),
  scope = NULL,
  k = 2,
  trace = FALSE,
  steps = 1000
)
```

Arguments

`dist`	assumed distribution for y variable.
`scale`	optional fixed value for the scale.
`parms`	list of fixed parameters.
`...`	arguments passed to `survreg.control`.
`direction`	mode of stepwise search, can be one of `"both"` (default), `"backward"`, or `"forward"`.
`scope`	defines the range of models examined in the stepwise search. This should be a list containing components `upper` and `lower`, both formulae.
`k`	multiple of the number of degrees of freedom used for the penalty. Only `k = 2` gives the genuine AIC; `k = .(log(nobs))` is sometimes referred to as BIC or SBC.
`trace`	if positive, information is printed during the running of `stepAIC`. Larger values may give more information on the fitting process.
`steps`	maximum number of steps to be considered.

Details

Response Types: `Surv`

Default values for the `NULL` arguments and further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`psm`, `survreg`, `survreg.control`, `stepAIC`, `fit`, `resample`

`stepAIC`, `fit`, `resample`

Examples

```
library(survival)

fit(Surv(time, status) ~ ., data = veteran, model = SurvRegModel)
```

`SVMModel` *Support Vector Machine Models*

Description

Fits the well known C-svc, nu-svc, (classification) one-class-svc (novelty) eps-svr, nu-svr (regression) formulations along with native multi-class classification formulations and the bound-constraint SVM formulations.

Usage

```
SVMModel(
  scaled = TRUE,
  type = NULL,
 kernel = c("rbfdot", "polydot", "vanilladot", "tanhdot", "laplacedot", "besseldot",
    "anovadot", "splinedot"),
  kpar = "automatic",
  C = 1,
  nu = 0.2,
  epsilon = 0.1,
  cache = 40,
  tol = 0.001,
  shrinking = TRUE
)

SVMANOVAModel(sigma = 1, degree = 1, ...)
```

```
SVMBesselModel(sigma = 1, order = 1, degree = 1, ...)

SVMLaplaceModel(sigma = NULL, ...)

SVMLinearModel(...)

SVMPolyModel(degree = 1, scale = 1, offset = 1, ...)

SVMRadialModel(sigma = NULL, ...)

SVMSplineModel(...)

SVMTanhModel(scale = 1, offset = 1, ...)
```

Arguments

`scaled`	logical vector indicating the variables to be scaled.
`type`	type of support vector machine.
`kernel`	kernel function used in training and predicting.
`kpar`	list of hyper-parameters (kernel parameters).
`C`	cost of constraints violation defined as the regularization term in the Lagrange formulation.
`nu`	parameter needed for nu-svc, one-svc, and nu-svr.
`epsilon`	parameter in the insensitive-loss function used for eps-svr, nu-svr and eps-bsvm.
`cache`	cache memory in MB.
`tol`	tolerance of termination criterion.
`shrinking`	whether to use the shrinking-heuristics.
`sigma`	inverse kernel width used by the ANOVA, Bessel, and Laplacian kernels.
`degree`	degree of the ANOVA, Bessel, and polynomial kernel functions.
`...`	arguments passed to `SVMModel`.
`order`	order of the Bessel function to be used as a kernel.
`scale`	scaling parameter of the polynomial and hyperbolic tangent kernels as a convenient way of normalizing patterns without the need to modify the data itself.
`offset`	offset used in polynomial and hyperbolic tangent kernels.

Details

Response Types: `factor`, `numeric`

Automatic Tuning of Grid Parameters
- SVMANOVAModel: `C`, `degree`
- SVMBesselModel: `C`, `order`, `degree`
- SVMLaplaceModel: `C`, `sigma`
- SVMLinearModel: `C`

- SVMPolyModel: C, degree, scale
- SVMRadialModel: C, sigma

Arguments kernel and kpar are automatically set by the kernel-specific constructor functions. Default values for the NULL arguments and further model details can be found in the source link below.

Value

MLModel class object.

See Also

ksvm, fit, resample

Examples

```
fit(sale_amount ~ ., data = ICHomes, model = SVMRadialModel)
```

t.test *Paired t-Tests for Model Comparisons*

Description

Paired t-test comparisons of resampled performance metrics from different models.

Usage

```
## S3 method for class 'PerformanceDiff'
t.test(x, adjust = "holm", ...)
```

Arguments

x	performance difference result.
adjust	p-value adjustment for multiple statistical comparisons as implemented by p.adjust.
...	arguments passed to other methods.

Value

PerformanceDiffTest class object that inherits from array. p-values and mean differences are contained in the lower and upper triangular portions, respectively, of the first two dimensions. Model pairs are contined in the third dimension.

Examples

```
## Numeric response example
fo <- sale_amount ~ .
control <- CVControl()

gbm_res1 <- resample(fo, ICHomes, GBMModel(n.trees = 25), control)
gbm_res2 <- resample(fo, ICHomes, GBMModel(n.trees = 50), control)
gbm_res3 <- resample(fo, ICHomes, GBMModel(n.trees = 100), control)

res <- c(GBM1 = gbm_res1, GBM2 = gbm_res2, GBM3 = gbm_res3)
res_diff <- diff(res)
t.test(res_diff)
```

TreeModel *Classification and Regression Tree Models*

Description

A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side.

Usage

```
TreeModel(
  mincut = 5,
  minsize = 10,
  mindev = 0.01,
  split = c("deviance", "gini"),
  k = NULL,
  best = NULL,
  method = c("deviance", "misclass")
)
```

Arguments

`mincut`	minimum number of observations to include in either child node.
`minsize`	smallest allowed node size: a weighted quantity.
`mindev`	within-node deviance must be at least this times that of the root node for the node to be split.
`split`	splitting criterion to use.
`k`	scalar cost-complexity parameter defining a subtree to return.
`best`	integer alternative to `k` requesting the number of terminal nodes of a subtree in the cost-complexity sequence to return.
`method`	character string denoting the measure of node heterogeneity used to guide cost-complexity pruning.

Details

Response Types: `factor`, `numeric`

Further model details can be found in the source link below.

Value

`MLModel` class object.

See Also

`tree`, `prune.tree`, `fit`, `resample`

Examples

```
fit(Species ~ ., data = iris, model = TreeModel)
```

`TunedInput` *Tuned Model Inputs*

Description

Recipe tuning over a grid of parameter values.

Usage

```
TunedInput(x, ...)

## S3 method for class 'recipe'
TunedInput(
  x,
  grid = expand_steps(),
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff"),
  ...
)
```

Arguments

`x`	untrained `recipe`.
`...`	arguments passed to other methods.
`grid`	`RecipeGrid` containing parameter values at which to evaluate a recipe, such as those returned by `expand_steps`.

`control`	control function, function name, or call defining the resampling method to be employed.
`metrics`	metric function, function name, or vector of these with which to calculate performance. If not specified, default metrics defined in the performance functions are used. Recipe selection is based on the first calculated metric.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for recipe tuning.
`cutoff`	argument passed to the `metrics` functions.

Value

`TunedModelRecipe` class object that inherits from `TunedInput` and `recipe`.

See Also

`fit`, `resample`

Examples

```
library(recipes)
library(MASS)

rec <- recipe(medv ~ ., data = Boston) %>%
  step_pca(all_numeric(), -all_outcomes(), id = "pca")

grid <- expand_steps(
  pca = list(num_comp = 1:2)
)

fit(TunedInput(rec, grid = grid), model = GLMModel)
```

`TunedModel` *Tuned Model*

Description

Model tuning over a grid of parameter values.

Usage

```
TunedModel(
  model,
  grid = MachineShop::settings("grid"),
  fixed = NULL,
  control = MachineShop::settings("control"),
  metrics = NULL,
  stat = MachineShop::settings("stat.train"),
  cutoff = MachineShop::settings("cutoff")
)
```

Arguments

`model`	model function, function name, or call defining the model to be tuned.
`grid`	data frame containing parameter values at which to evaluate a single model supplied to `models`, such as that returned by `expand_params`; the number of parameter-specific values to generate automatically if the model has a pre-defined grid; or a call to `Grid` or `ParameterGrid`.
`fixed`	list of fixed parameter values to combine with those in `grid`.
`control`	control function, function name, or call defining the resampling method to be employed.
`metrics`	metric function, function name, or vector of these with which to calculate performance. If not specified, default metrics defined in the performance functions are used. Model selection is based on the first calculated metric.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for model tuning.
`cutoff`	argument passed to the `metrics` functions.

Details

Response Types: `factor`, `numeric`, `ordered`, `Surv`

Value

`TunedModel` class object that inherits from `MLModel`.

See Also

`fit`, `resample`

Examples

```
# Automatically generated grid
model_fit <- fit(sale_amount ~ ., data = ICHomes,
                 model = TunedModel(GBMModel))
varimp(model_fit)
(tuned_model <- as.MLModel(model_fit))
summary(tuned_model)
plot(tuned_model, type = "l")

# Randomly sampled grid points
fit(sale_amount ~ ., data = ICHomes,
    model = TunedModel(GBMModel, grid = Grid(length = 1000, random = 5)))

# User-specified grid
fit(sale_amount ~ ., data = ICHomes,
    model = TunedModel(GBMModel,
                       grid = expand_params(n.trees = c(50, 100),
                                            interaction.depth = 1:2,
                                            n.minobsinnode = c(5, 10))))
```

varimp *Variable Importance*

Description

Calculate measures of the relative importance of predictors in a model.

Usage

```
varimp(object, scale = TRUE, ...)
```

Arguments

object	model fit result.
scale	logical indicating whether importance measures should be scaled to range from 0 to 100.
...	arguments passed to model-specific variable importance functions.

Value

VarImp class object.

See Also

plot

Examples

```
## Survival response example
library(survival)

gbm_fit <- fit(Surv(time, status) ~ ., data = veteran, model = GBMModel)
(vi <- varimp(gbm_fit))
plot(vi)
```

XGBModel *Extreme Gradient Boosting Models*

Description

Fits models within an efficient implementation of the gradient boosting framework from Chen & Guestrin.

Usage

```
XGBModel(params = list(), nrounds = 1, verbose = 0, print_every_n = 1)

XGBDARTModel(
  objective = NULL,
  aft_loss_distribution = "normal",
  aft_loss_distribution_scale = 1,
  base_score = 0.5,
  eta = 0.3,
  gamma = 0,
  max_depth = 6,
  min_child_weight = 1,
  max_delta_step = .(0.7 * is(y, "PoissonVariate")),
  subsample = 1,
  colsample_bytree = 1,
  colsample_bylevel = 1,
  colsample_bynode = 1,
  lambda = 1,
  alpha = 0,
  tree_method = "auto",
  sketch_eps = 0.03,
  scale_pos_weight = 1,
  refresh_leaf = 1,
  process_type = "default",
  grow_policy = "depthwise",
  max_leaves = 0,
  max_bin = 256,
  num_parallel_tree = 1,
  sample_type = "uniform",
  normalize_type = "tree",
  rate_drop = 0,
  one_drop = 0,
  skip_drop = 0,
  ...
)

XGBLinearModel(
  objective = NULL,
  aft_loss_distribution = "normal",
  aft_loss_distribution_scale = 1,
  base_score = 0.5,
  lambda = 0,
  alpha = 0,
  updater = "shotgun",
  feature_selector = "cyclic",
  top_k = 0,
  ...
)
```

```
XGBTreeModel(
  objective = NULL,
  aft_loss_distribution = "normal",
  aft_loss_distribution_scale = 1,
  base_score = 0.5,
  eta = 0.3,
  gamma = 0,
  max_depth = 6,
  min_child_weight = 1,
  max_delta_step = .(0.7 * is(y, "PoissonVariate")),
  subsample = 1,
  colsample_bytree = 1,
  colsample_bylevel = 1,
  colsample_bynode = 1,
  lambda = 1,
  alpha = 0,
  tree_method = "auto",
  sketch_eps = 0.03,
  scale_pos_weight = 1,
  refresh_leaf = 1,
  process_type = "default",
  grow_policy = "depthwise",
  max_leaves = 0,
  max_bin = 256,
  num_parallel_tree = 1,
  ...
)
```

Arguments

`params`	list of model parameters as described in the XGBoost documentation.
`nrounds`	maximum number of boosting iterations.
`verbose`	numeric value controlling the amount of output printed during model fitting, such that 0 = none, 1 = performance information, and 2 = additional information.
`print_every_n`	numeric value designating the fitting iterations at at which to print output when `verbose` > `0`.

`objective`	character string specifying the learning task and objective. Possible values for supported response variable types are as follows.
	`factor`: `"multi:softprob"`, `"binary:logistic"` (2 levels only)
	`numeric`: `"reg:squarederror"`, `"reg:logistic"`, `"reg:gamma"`, `"reg:tweedie"`, `"rank:pairwise"`, `"rank:ndcg"`, `"rank:map"`
	`PoissonVariate`: `"count:poisson"`
	`Surv`: `"survival:cox"`, `"survival:aft"`
	The first values listed are the defaults for the corresponding response types.

`aft_loss_distribution`	character string specifying the distribution for the accelerated failure time objective (`"survival:aft"`) as `"normal"`, `"logistic"`, or `"extreme"`.
`aft_loss_distribution_scale`	numeric scaling parameter for the accelerated failure time distribution.
`base_score`	initial numeric prediction score of all instances, global bias.
`eta, gamma, max_depth, min_child_weight, max_delta_step, subsample, colsample_bytree, colsample_bylevel`	see `params` reference.
`...`	arguments passed to `XGBModel`.

Details

Response Types: `factor`, `numeric`, `PoissonVariate`, `Surv`

Automatic Tuning of Grid Parameters
- XGBDARTModel: `nrounds`, `max_depth`, `eta`, `gamma*`, `min_child_weight*`, `subsample`, `colsample_bytree`, `rate_drop`, `skip_drop`
- XGBLinearModel: `nrounds`, `lambda`, `alpha`
- XGBTreeModel: `nrounds`, `max_depth`, `eta`, `gamma*`, `min_child_weight*`, `subsample`, `colsample_bytree`

* included only in randomly sampled grid points

Default values for the `NULL` arguments and further model details can be found in the source link below.

In calls to `varimp` for `XGBTreeModel`, argument `metric` may be spedified as `"Gain"` (default) for the fractional contribution of each predictor to the total gain of its splits, as `"Cover"` for the number of observations related to each predictor, or as `"Frequency"` for the percentage of times each predictor is used in the trees. Variable importance is automatically scaled to range from 0 to 100. To obtain unscaled importance values, set `scale = FALSE`. See example below.

Value

`MLModel` class object.

See Also

`xgboost`, `fit`, `resample`

Examples

```
model_fit <- fit(Species ~ ., data = iris, model = XGBTreeModel)
varimp(model_fit, metric = "Frequency", scale = FALSE)
```

Index

∗ **datasets**
ICHomes, 40
+,SurvMatrix,SurvMatrix-method (combine), 19
., 5, 30, 47
[,DiscreteVariate,ANY,missing,missing-method (extract), 29
[,ModelFrame,ANY,ANY,ANY-method (extract), 29
[,ModelFrame,ANY,missing,ANY-method (extract), 29
[,ModelFrame,missing,missing,ANY-method (extract), 29
[,RecipeGrid,ANY,ANY,ANY-method (extract), 29
[,Resamples,ANY,ANY,ANY-method (extract), 29
[,Resamples,ANY,missing,ANY-method (extract), 29
[,Resamples,missing,missing,ANY-method (extract), 29
[,SurvMatrix,ANY,ANY,ANY-method (extract), 29
[.BinomialVariate (extract), 29
[.ModelFrame (extract), 29

accuracy (metrics), 49
AdaBagModel, 6, 61
AdaBoostModel, 7, 61
as.MLModel, 9, 32
auc, 68
auc (metrics), 49
Automatic Tuning, 7, 8, 10, 14, 16, 18, 26, 31, 33, 35, 36, 39, 42–44, 47, 64, 71, 77, 78, 83, 106, 115

bagging, 7
bake, 92, 94, 96, 97, 100
bartMachine, 10, 11
BARTMachineModel, 9, 61
BARTModel, 11, 61
base learner, 33
baselearners, 34
BinomialVariate, 41, 79
BinomialVariate (DiscreteVariate), 24
blackboost, 14
BlackBoostModel, 13, 61
boosting, 9
BootControl, 5
BootControl (MLControl), 52
BootOptimismControl, 5
BootOptimismControl (MLControl), 52
brier (metrics), 49
bruto, 30, 47

c, 17, 20, 45, 68, 81
c.Calibration (combine), 19
c.ConfusionList (combine), 19
c.ConfusionMatrix (combine), 19
c.LiftCurve (combine), 19
c.ListOf (combine), 19
c.PerformanceCurve (combine), 19
c.Resamples (combine), 19
C5.0, 16
C5.0Control, 16
C50Model, 15, 61
calibration, 4, 16, 19, 70
cforest, 18
cforest_control, 18
CForestModel, 17, 61
cindex (metrics), 49
clara, 93, 94
combine, 19
confusion, 4, 19, 20, 48, 51, 67, 70, 73, 101
ConfusionMatrix (confusion), 20
control, 19, 81, 85–87, 89, 102, 110, 111
controls (MLControl), 52
CoxModel, 21, 61
coxph, 22
coxph.control, 21, 22

CoxStepAICModel, *61*
CoxStepAICModel (CoxModel), 21
cross_entropy (metrics), 49
ctree_control, *14*
curves (performance_curve), 67
CVControl, *5*
CVControl (MLControl), 52
CVOptimismControl, *5*
CVOptimismControl (MLControl), 52

data frame, *22*, *32*, *58*, *59*, *73*, *81*, *82*, *85*, *111*
dependence, *4*, 22, *70*
diff, *4*, 23
difference, *107*
DiscreteVariate, 24, *41*

earth, *26*
EarthModel, 25, *61*
expand_model, 26, *86*
expand_params, 27, *111*
expand_steps, 28, *109*
extract, 29

f_score (metrics), 49
factor, *41*
Family, *13*, *14*, *33–36*
fda, *31*
FDAModel, 30, *61*
fit, *4*, *7*, *9*, *11*, *13*, *14*, *16*, *18*, *22*, *26*, *31*, 31, *34–37*, *39*, *41–44*, *46*, *48*, *56*, *58*, *59*, *62–64*, *70–73*, *76–78*, *82*, *83*, *85*, *86*, *90*, *101*, *103*, *105*, *107*, *109–112*, *115*
fitting, *79*
fnr (metrics), 49
formula, *41*, *59*
fpr (metrics), 49

gamboost, *34*
GAMBoostModel, 33, *61*
gbart, *13*
gbm, *35*
GBMModel, 34, *61*
gen.ridge, *30*, *47*
gini (metrics), 49
glm, *37*
glm.control, *37*
glmboost, *36*
GLMBoostModel, 35, *61*
GLMModel, 36, *61*
glmnet, *39*
GLMNetModel, 38, *61*
GLMStepAICModel, *61*
GLMStepAICModel (GLMModel), 36
Grid, 39, *87*, *111*

ICHomes, 40
input, *32*, *58*, *80*
inputs, 40, *85*
install.packages, *60*

kappa2 (metrics), 49
kknn, *42*
kmeans, *92*
KNNModel, 41, *61*
ksvm, *107*

lars, *43*
LARSModel, 42, *61*
lda, *44*
LDAModel, 44, *61*
library, *60*
lift, *4*, *19*, 45, *70*, *101*
lm, *46*
LMModel, 46, *61*
loess, *17*

MachineShop (MachineShop-package), 4
MachineShop-package, 4
mae (metrics), 49
mars, *30*, *47*
Matrix, *95*
matrix, *41*, *59*, *95*
mbart, *13*
mda, *48*
MDAModel, 46, *61*
metric, *48*, *67*, *85*, *86*, *110*, *111*
metricinfo, *5*, 48, *52*
metrics, *4*, *19*, 49, *55*, *68*, *73*, *81*, *88*, *103*
MLControl, 52
MLMetric, *5*, 54
MLMetric<- (MLMetric), 54
MLModel, *5*, 55
MLModelFunction (models), 60
model, *26*, *32*, *58*, *60*, *80*, *81*, *86*, *89*, *102*, *111*
model.matrix, *56*
modeled inputs, *32*, *81*
ModeledFrame (ModeledInput), 57
ModeledInput, *41*, 57

ModeledRecipe (ModeledInput), 57
ModelFrame, 41, 56, 58, 82
modelinfo, 5, 59, 62
models, 4, 56, 60
mse (metrics), 49
msle (metrics), 49
mvr, 71

naiveBayes, 63
NaiveBayesModel, 61, 62
NegBinomialVariate, 41
NegBinomialVariate (DiscreteVariate), 24
nnet, 64
NNetModel, 61, 63
npv (metrics), 49
numeric, 41

observed, 48
observed responses, 17, 20, 45, 48, 51, 60, 67, 68
OOBControl, 5
OOBControl (MLControl), 52
ordered, 41

p.adjust, 107
pam, 93, 94
ParameterGrid, 64, 111
parameters, 65
partial dependence, 88
PDAModel, 61
PDAModel (FDAModel), 30
performance, 4, 23, 52, 65, 70, 73, 81, 85, 86, 88, 101, 103, 110, 111
performance curve, 19, 51, 70, 101
performance_curve, 4, 67
plot, 5, 17, 20, 23, 24, 45, 67, 68, 69, 81, 112
plots, 88
PLSModel, 61, 71
PoissonVariate, 41
PoissonVariate (DiscreteVariate), 24
polr, 72
POLRModel, 61, 72
polyreg, 30, 47
ppv (metrics), 49
pr_auc (metrics), 49
precision (metrics), 49
predict, 4, 31, 32, 44, 47, 52, 73, 76, 103
predict.fda, 31
predict.lda, 44
predict.mda, 48
predict.qda, 76
predicted, 48
predicted responses, 17, 20, 45, 51, 67, 68
prep, 91–97, 99, 100
print, 5, 74
prune.tree, 109
psm, 105

qda, 76
QDAModel, 61, 75
quote, 5, 6

r2 (metrics), 49
randomForest, 77
RandomForestModel, 61, 76
ranger, 78
RangerModel, 61, 77
recall (metrics), 49
recipe, 28, 41, 79, 82, 90, 92–97, 99, 100, 109
recipe_roles, 79
resample, 4, 7, 9, 11, 13, 14, 16–20, 22, 23, 26, 31, 34–37, 39, 41–46, 48, 51, 54, 56, 58, 59, 62–64, 67, 68, 70–72, 76–78, 80, 83, 85, 86, 90, 101, 103, 105, 107, 109–111, 115
resampling, 79
response, 4, 32, 59, 82
rmse (metrics), 49
rmsle (metrics), 49
roc_auc (metrics), 49
roc_index (metrics), 49
role_binom, 25, 41
role_binom (recipe_roles), 79
role_case, 32, 81
role_case (recipe_roles), 79
role_pred (recipe_roles), 79
role_surv, 41
role_surv (recipe_roles), 79
rpart, 83
RPartModel, 61, 82
rpp (metrics), 49

SelectedInput, 41, 54, 58, 59, 84
SelectedModel, 27, 54, 62, 86
SelectedModelFrame (SelectedInput), 84
SelectedModelRecipe (SelectedInput), 84
selection, 88
selections, 90, 93, 95, 97, 99

sensitivity *(metrics)*, 49
settings, *5*, 87
spca, *100*
specificity *(metrics)*, 49
SplitControl, *5*
SplitControl *(MLControl)*, 52
StackedModel, *62*, 89
step_kmeans, 90
step_kmedoids, 92
step_lincomp, *91*, 94, *100*
step_sbf, *93*, 96
step_spca, 98
stepAIC, *22*, *37*, *105*
strata, *81*
subset, *23*
summary, *5*, *19*, *20*, *24*, *45*, *67*, *68*, *81*, *88*, 100
SuperModel, *62*, 102
Surv, *41*, *79*
surv.bart, *13*
SurvEvents *(SurvMatrix)*, 103
SurvMatrix, 103
SurvProbs *(SurvMatrix)*, 103
survreg, *105*
survreg.control, *104*, *105*
SurvRegModel, *61*, 104
SurvRegStepAICModel, *61*
SurvRegStepAICModel *(SurvRegModel)*, 104
SVMANOVAModel, *61*
SVMANOVAModel *(SVMModel)*, 105
SVMBesselModel, *61*
SVMBesselModel *(SVMModel)*, 105
SVMLaplaceModel, *61*
SVMLaplaceModel *(SVMModel)*, 105
SVMLinearModel, *61*
SVMLinearModel *(SVMModel)*, 105
SVMModel, *61*, 105
SVMPolyModel, *61*
SVMPolyModel *(SVMModel)*, 105
SVMRadialModel, *61*
SVMRadialModel *(SVMModel)*, 105
SVMSplineModel, *61*
SVMSplineModel *(SVMModel)*, 105
SVMTanhModel, *61*
SVMTanhModel *(SVMModel)*, 105

t.test, *24*, 107
tidy.step_kmeans *(step_kmeans)*, 90
tidy.step_lincomp *(step_lincomp)*, 94
tidy.step_sbf *(step_sbf)*, 96
tnr *(metrics)*, 49
tpr *(metrics)*, 49
TrainControl, *5*
TrainControl *(MLControl)*, 52
tree, *109*
TreeModel, *61*, 108
tunable.step_kmeans *(step_kmeans)*, 90
tunable.step_kmedoids *(step_kmedoids)*, 92
tunable.step_lincomp *(step_lincomp)*, 94
tunable.step_spca *(step_spca)*, 98
TunedInput, *28*, *41*, *54*, 109
TunedModel, *27*, *39*, *54*, *62*, *65*, 110
TunedModelRecipe *(TunedInput)*, 109
tuning, *87*, *88*

variable importance, *70*
varimp, *4*, *10*, *16*, *22*, *26*, *32*, *37*, *46*, *56*, *72*, 112, *115*

weighted_kappa2 *(metrics)*, 49
weights, *32*

XGBDARTModel, *62*
XGBDARTModel *(XGBModel)*, 112
XGBLinearModel, *62*
XGBLinearModel *(XGBModel)*, 112
XGBModel, *62*, 112
xgboost, *115*
XGBTreeModel, *62*
XGBTreeModel *(XGBModel)*, 112