Package 'MachineShop'

January 30, 2026

Type Package

Title Machine Learning Models and Tools

Version 3.9.2

Date 2026-01-30

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 (>= 4.1.0)

Imports abind, cli (>= 3.1.0), dials (>= 0.0.4), foreach, ggplot2 (>= 3.4.0), kernlab, magrittr, Matrix (>= 1.5-0), methods, nnet, party, polspline, progress, recipes (>= 1.0.0), rlang, rsample (>= 1.1.0), Rsolnp, survival, tibble, utils

Suggests adabag, BART, bartMachine, C50, censored, cluster, doParallel, e1071, earth, elasticnet, generics, gbm, glmnet, gridExtra, Hmisc, kableExtra, kknn, knitr, lars, MASS, mboost, mda, ParBayesianOptimization, parsnip (>= 1.1.0), partykit, pls, pso, randomForest, randomForestSRC, ranger, rBayesianOptimization, rmarkdown, rms, rpart, testthat, tree, xgboost (>= 3.1.2)

Additional_repositories https://brian-j-smith.github.io/drat

LazyData true

License GPL-3

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

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

RoxygenNote 7.3.3

VignetteBuilder knitr

Encoding UTF-8

Collate 'classes.R' 'conditions.R' 'MachineShop-package.R'
'MLControl.R' 'MLInput.R' 'MLMetric.R' 'MLModel.R'
'MLOptimization.R' 'ML_AdaBagModel.R' 'ML_AdaBoostModel.R'
'ML_BARTMachineModel.R' 'ML_BARTModel.R' 'ML_BlackBoostModel.R'
'ML_C50Model.R' 'ML_CForestModel.R' 'ML_CoxModel.R'
'ML_EarthModel.R' 'ML_FDAModel.R' 'ML_GAMBoostModel.R'
'ML_GBMModel.R' 'ML_GLMBoostModel.R' 'ML_GLMModel.R'
'ML_GLMNetModel.R' 'ML_KNNModel.R' 'ML_LARSModel.R'
'ML_LDAModel.R' 'ML_LMModel.R' 'ML_MDAModel.R' 'ML_NNetModel.R'
'ML_NaiveBayesModel.R' 'ML_ParsnipModel.R' 'ML_PLSModel.R'
'ML_POLRModel.R' 'ML_QDAModel.R' 'ML_RFSRCModel.R'
'ML_RPartModel.R' 'ML_RandomForestModel.R' 'ML_RangerModel.R'
'ML_SVMModel.R' 'ML_StackedModel.R' 'ML_SuperModel.R'
'ML_SurvRegModel.R' 'ML_TreeModel.R' 'ML_XGBModel.R'
'ModelFrame.R' 'ModelRecipe.R' 'ModelSpecification.R'
'TrainedInputs.R' 'TrainedModels.R' 'TrainingParams.R'
'append.R' 'calibration.R' 'case_comps.R' 'coerce.R'
'combine.R' 'confusion.R' 'convert.R' 'data.R' 'dependence.R'
'diff.R' 'expand.R' 'extract.R' 'fit.R' 'grid.R' 'metricinfo.R'
'metrics.R' 'metrics_factor.R' 'metrics_numeric.R'
'modelinfo.R' 'models.R' 'performance.R' 'performance_curve.R'
'plot.R' 'predict.R' 'print.R' 'recipe_roles.R' 'reexports.R'
'resample.R' 'response.R' 'rfe.R' 'settings.R' 'step_kmeans.R'
'step_kmedoids.R' 'step_lincomp.R' 'step_sbf.R' 'step_spca.R'
'summary.R' 'survival.R' 'utils.R' 'varimp.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-01-30 15:40:02 UTC

Contents

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

Tuning Grids:

expand_model	Model expansion over tuning parameters
expand_modelgrid	Model tuning grid expansion
expand_params	Model parameters expansion
expand_steps	Recipe step parameters expansion

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
rfe	Recursive feature elimination
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

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

```
## Requires prior installation of suggested package adabag to run

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*

* excluded from grids by default

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

Value

MLModel class object.

See Also

boosting, fit, resample

Examples

```
## Requires prior installation of suggested package adabag to run

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

---

as.data.frame *Coerce to a Data Frame*

Description

Functions to coerce objects to data frames.

Usage

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

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

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

Arguments

x	ModelFrame, resample results, resampled performance estimates, model performance differences, or t-test comparisons of the differences.
...	arguments passed to other methods.

Value

data.frame class object.

---

as.MLInput *Coerce to an MLInput*

Description

Function to coerce an object to MLInput.

Usage

```
as.MLInput(x, ...)

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

## S3 method for class 'ModelSpecification'
as.MLInput(x, ...)
```

Arguments

x	model fit result or **MachineShop** model specification.
...	arguments passed to other methods.

Value

MLInput class object.

as.MLModel *Coerce to an MLModel*

Description

Function to coerce an object to MLModel.

Usage

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

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

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

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

Arguments

x	model fit result, **MachineShop** model specification, or **parsnip** model specification.
...	arguments passed to other methods.

Value

MLModel class object.

See Also

ParsnipModel

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 $E(Y\|X)$ 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 type may be specified 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

```
## Not run:
## Requires prior installation of suggested package bartMachine, java, and
## setting of java options as shown below to run

if (packageVersion("bartMachine") >= "1.4") {
  options(
    java.parameters = c(
      "-Xmx20g", "--add-modules=jdk.incubator.vector", "-XX:+UseZGC"
    )
  )
} else {
  options(java.parameters = "-Xmx20g")
}

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

## End(Not run)
```

BARTModel *Bayesian Additive Regression Trees Model*

Description

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

Usage

```
BARTModel(
  K = integer(),
  sparse = FALSE,
  theta = 0,
  omega = 1,
  a = 0.5,
  b = 1,
  rho = numeric(),
```

```
  augment = FALSE,
  xinfo = matrix(NA, 0, 0),
  usequants = FALSE,
  sigest = NA,
  sigdf = 3,
  sigquant = 0.9,
  lambda = NA,
  k = 2,
  power = 2,
  base = 0.95,
  tau.num = numeric(),
  offset = numeric(),
  ntree = integer(),
  numcut = 100,
  ndpost = 1000,
  nskip = integer(),
  keepevery = integer(),
  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 argument values and further model details can be found in the source See Also links below.

Value

MLModel class object.

See Also

gbart, mbart, surv.bart, fit, resample

Examples

```
## Requires prior installation of suggested package BART to run

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 argument values and further model details can be found in the source See Also links below.

Value

MLModel class object.

See Also

blackboost, Family, ctree_control, fit, resample

Examples

```
## Requires prior installation of suggested packages mboost and partykit to run

data(Pima.tr, package = "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 type may be specified 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

```
## Requires prior installation of suggested package C50 to run

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

---

calibration *Model Calibration*

Description

Calculate calibration estimates from observed and predicted responses.

Usage

```
calibration(
  x,
  y = NULL,
  weights = NULL,
  breaks = 10,
  span = 0.75,
  distr = character(),
  pool = FALSE,
  na.rm = TRUE,
  ...
)
```

Arguments

x	observed responses or resample result containing observed and predicted responses.
y	predicted responses if not contained in x.
weights	numeric vector of non-negative case weights for the observed x responses [default: equal weights].
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.
distr	character string specifying a distribution with which to estimate the observed survival mean. Possible values are "empirical" for the Kaplan-Meier estimator, "exponential", "extreme", "gaussian", "loggaussian", "logistic", "loglogistic", "lognormal", "rayleigh", "t", or "weibull". Defaults to the distribution that was used in predicting mean survival times.
pool	logical indicating whether to compute a single calibration curve on predictions pooled over all resampling iterations or to compute them for each iteration individually and return the mean calibration curve. Pooling can result in large memory allocation errors when fitting smooth curves with breaks = NULL. The current default is changed from versions <= 3.8.0 of the package which only implemented pool = TRUE.
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

```
## Requires prior installation of suggested package gbm to run

library(survival)

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

`case_weights` *Extract Case Weights*

Description

Extract the case weights from an object.

Usage

```
case_weights(object, newdata = NULL)
```

Arguments

`object`	model fit result, `ModelFrame`, or `recipe`.
`newdata`	dataset from which to extract the weights if given; otherwise, `object` is used. The dataset should be given as a `ModelFrame` or as a data frame if `object` contains a `ModelFrame` or a `recipe`, respectively.

Examples

```
## Training and test sets
inds <- sample(nrow(ICHomes), nrow(ICHomes) * 2 / 3)
trainset <- ICHomes[inds, ]
testset <- ICHomes[-inds, ]

## ModelFrame case weights
trainmf <- ModelFrame(sale_amount ~ . - built, data = trainset, weights = built)
testmf <- ModelFrame(formula(trainmf), data = testset, weights = built)
mf_fit <- fit(trainmf, model = GLMModel)
rmse(response(mf_fit, testmf), predict(mf_fit, testmf),
     case_weights(mf_fit, testmf))

## Recipe case weights
library(recipes)
rec <- recipe(sale_amount ~ ., data = trainset) %>%
  role_case(weight = built, replace = TRUE)
```

```
rec_fit <- fit(rec, model = GLMModel)
rmse(response(rec_fit, testset), predict(rec_fit, testset),
     case_weights(rec_fit, testset))
```

`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 parameter: `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 'Resample'
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,
  weights = 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.
weights	numeric vector of non-negative case weights for the observed x responses [default: equal weights].
cutoff	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified. If NULL, then factor 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 Resample object.

See Also

c, plot, summary

Examples

```
## Requires prior installation of suggested package gbm to run

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 = list(),
  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 argument values and further model details can be found in the source See Also links 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.

Value

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"),
  distr = character(),
  method = character(),
  stats = MachineShop::settings("stats.PartialDependence"),
  na.rm = TRUE
)
```

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"`).
`distr, method`	arguments passed to `predict`.
`stats`	function, function name, or vector of these with which to compute response variable summary statistics over non-selected predictor variables.
`na.rm`	logical indicating whether to exclude missing predicted response values from the calculation of summary statistics.

Value

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

See Also

`plot`

Examples

```
## Requires prior installation of suggested package gbm to run

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 'Resample'
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

```
## Requires prior installation of suggested package gbm to run

## 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 = integer(),
  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`*

* excluded from grids by default

Default argument values and further model details can be found in the source See Also link below.

In calls to `varimp` for `EarthModel`, argument `type` may be specified as `"nsubsets"` (default) for the number of model subsets that include each predictor, as `"gcv"` for the generalized cross-validation decrease over all subsets that include each predictor, or as `"rss"` for the residual sums of squares decrease. 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

```
## Requires prior installation of suggested package earth to run

model_fit <- fit(Species ~ ., data = iris, model = EarthModel)
varimp(model_fit, method = "model", type = "gcv", 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(object, ..., random = FALSE)
```

Arguments

object	model function, function name, or object; or another object that can be coerced to a model.
...	named vectors or factors or a list of these containing the parameter values over which to expand object.
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

```
## Requires prior installation of suggested package gbm to run

data(Boston, package = "MASS")

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

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

---

expand_modelgrid *Model Tuning Grid Expansion*

Description

Expand a model grid of tuning parameter values.

Usage

```
expand_modelgrid(...)

## S3 method for class 'formula'
expand_modelgrid(formula, data, model, info = FALSE, ...)

## S3 method for class 'matrix'
expand_modelgrid(x, y, model, info = FALSE, ...)

## S3 method for class 'ModelFrame'
expand_modelgrid(input, model, info = FALSE, ...)

## S3 method for class 'recipe'
expand_modelgrid(input, model, info = FALSE, ...)

## S3 method for class 'ModelSpecification'
expand_modelgrid(object, ...)

## S3 method for class 'MLModel'
expand_modelgrid(model, ...)

## S3 method for class 'MLModelFunction'
expand_modelgrid(model, ...)
```

Arguments

...	arguments passed from the generic function to its methods and from the MLModel and MLModelFunction methods to others. The first argument of each expand_modelgrid method is positional and, as such, must be given first in calls to them.
formula, data	formula defining the model predictor and response variables and a data frame containing them.
model	model function, function name, or object; or another object that can be coerced to a model. A model can be given first followed by any of the variable specifications.
info	logical indicating whether to return model-defined grid construction information rather than the grid values.
x, y	matrix and object containing predictor and response variables.
input	input object defining and containing the model predictor and response variables.
object	model specification.

Details

The expand_modelgrid function enables manual extraction and viewing of grids created automatically when a TunedModel is fit.

Value

A data frame of parameter values or NULL if data are required for construction of the grid but not supplied.

See Also

TunedModel

Examples

```
expand_modelgrid(TunedModel(GBMModel, grid = 5))


## Requires prior installation of suggested package glmnet to run
expand_modelgrid(TunedModel(GLMNetModel, grid = c(alpha = 5, lambda = 10)),
                 sale_amount ~ ., data = ICHomes)


gbm_grid <- ParameterGrid(
  n.trees = dials::trees(),
  interaction.depth = dials::tree_depth(),
  size = 5
)
expand_modelgrid(TunedModel(GBMModel, grid = gbm_grid))

rf_grid <- ParameterGrid(
  mtry = dials::mtry(),
```

```
  nodesize = dials::max_nodes(),
  size = c(3, 5)
)
expand_modelgrid(TunedModel(RandomForestModel, grid = rf_grid),
                 sale_amount ~ ., data = ICHomes)
```

---

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 data frames or vectors 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

```
## Requires prior installation of suggested package gbm to run

data(Boston, package = "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)
data(Boston, package = "MASS")

rec <- recipe(medv ~ ., data = Boston) %>%
  step_corr(all_numeric_predictors(), id = "corr") %>%
  step_pca(all_numeric_predictors(), 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]

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

## 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,ANY,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 'Resample,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

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

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

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

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

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 = matrix(NA, 0, 0),
  dimension = integer(),
  eps = .Machine$double.eps,
  method = .(mda::polyreg),
  ...
)

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

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`

* excluded from grids by default

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 argument values and further model details can be found in the source See Also links below.

Value

`MLModel` class object.

See Also

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

Examples

```
## Requires prior installation of suggested package mda to run

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


## Requires prior installation of suggested package mda to run

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

`fit` *Model Fitting*

Description

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

Usage

```
fit(...)

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

## S3 method for class 'matrix'
```

```
fit(x, y, model, ...)

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

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

## S3 method for class 'ModelSpecification'
fit(object, verbose = FALSE, ...)

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

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

Arguments

`...`	arguments passed from the generic function to its methods, from the `MLModel` and `MLModelFunction` methods to first arguments of others, and from others to the `ModelSpecification` method. The first argument of each `fit` method is positional and, as such, must be given first in calls to them.
`formula, data`	formula defining the model predictor and response variables and a data frame containing them.
`model`	model function, function name, or object; or another object that can be coerced to a model. A model can be given first followed by any of the variable specifications.
`x, y`	matrix and object containing predictor and response variables.
`input`	input object defining and containing the model predictor and response variables.
`object`	model specification.
`verbose`	logical indicating whether to display printed output generated by some model-specific fit functions to aid in monitoring progress and diagnosing errors.

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

```
## Requires prior installation of suggested package gbm to run

## 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 parameter: `mstop`

Default argument values and further model details can be found in the source See Also links below.

Value

`MLModel` class object.

See Also

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

Examples

```
## Requires prior installation of suggested package mboost to run

data(Pima.tr, package = "MASS")

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

GBMModel *Generalized Boosted Regression Model*

Description

Fits generalized boosted regression models.

Usage

```
GBMModel(
  distribution = character(),
  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*

* excluded from grids by default

Default argument values and further model details can be found in the source See Also link below.

Value

MLModel class object.

See Also

gbm, fit, resample

Examples

```
## Requires prior installation of suggested package gbm to run

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 parameter: mstop

Default argument values and further model details can be found in the source See Also links below.

Value

MLModel class object.

See Also

glmboost, Family, fit, resample

Examples

```
## Requires prior installation of suggested package mboost to run

data(Pima.tr, package = "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 = list(),
  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 argument values and further model details can be found in the source See Also links 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 = logical(),
  penalty.factor = .(rep(1, nvars)),
  standardize.response = FALSE,
  thresh = 1e-07,
  maxit = 1e+05,
  type.gaussian = .(if (nvars < 500) "covariance" else "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 argument values and further model details can be found in the source See Also link below.

Value

MLModel class object.

See Also

glmnet, fit, resample

Examples

```
## Requires prior installation of suggested package glmnet to run

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

`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
`ModelSpecification`	Model specification
`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.

`ModelSpecification`	Model specification
`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*

* excluded from grids by default

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

Value

MLModel class object.

See Also

kknn, fit, resample

Examples

```
## Requires prior installation of suggested package kknn to run

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 = numeric(),
  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 parameter: `step`

Default argument values and further model details can be found in the source See Also link below.

Value

`MLModel` class object.

See Also

`lars`, `fit`, `resample`

Examples

```
## Requires prior installation of suggested package lars to run

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

`LDAModel` *Linear Discriminant Analysis Model*

Description

Performs linear discriminant analysis.

Usage

```
LDAModel(
  prior = numeric(),
  tol = 1e-04,
  method = c("moment", "mle", "mve", "t"),
  nu = 5,
  dimen = integer(),
  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 parameter: `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 argument values and further model details can be found in the source See Also 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, weights = NULL, na.rm = TRUE, ...)
```

Arguments

`x`	observed responses or resample result containing observed and predicted responses.
`y`	predicted responses if not contained in `x`.
`weights`	numeric vector of non-negative case weights for the observed `x` responses [default: equal weights].
`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

```
## Requires prior installation of suggested package gbm to run

data(Pima.tr, package = "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 = numeric(),
  tot.df = numeric(),
  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 parameter: 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 argument values and further model details can be found in the source See Also links below.

Value

`MLModel` class object.

See Also

`mda`, `predict.mda`, `fit`, `resample`

Examples

```
## Requires prior installation of suggested package mda to run

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,
  weights = NULL,
  cutoff = MachineShop::settings("cutoff"),
  ...
)

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

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

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

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

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

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

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

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

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

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

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

pr_auc(
  observed,
  predicted = NULL,
  weights = NULL,
  multiclass = c("pairs", "all"),
  ...
)

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

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

roc_auc(
  observed,
  predicted = NULL,
  weights = NULL,
  multiclass = c("pairs", "all"),
  ...
)

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

```
)

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

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

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

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

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

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

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

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

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

r2(
  observed,
  predicted = NULL,
  weights = NULL,
```

```
  method = c("mse", "pearson", "spearman"),
  distr = character(),
  ...
)

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

rmsle(observed, predicted = NULL, weights = 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.
weights	numeric vector of non-negative case weights for the observed responses [default: equal weights].
cutoff	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified. If NULL, then confusion matrix-based metrics are computed on predicted class probabilities if given.
...	arguments passed to or from other methods.
multiclass	character string specifying the method for computing generalized area under the performance curve for multiclass factor responses. Options are to average over areas for each pair of classes ("pairs") or for each class versus all others ("all").
metrics	vector of two metric functions or function names that define a curve under which to calculate area [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].
fun	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.
method	character string specifying whether to compute r2 as the coefficient of determination ("mse") or as the square of "pearson" or "spearman" correlation.
distr	character string specifying a distribution with which to estimate the observed 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". Defaults to the distribution that was used in predicting mean survival times.

References

Hand, D. J., & Till, R. J. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, *45*, 171-186.

See Also

metricinfo, performance

MLControl *Resampling Controls*

Description

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

Usage

```
BootControl(
  samples = 25,
  weights = TRUE,
  seed = sample(.Machine$integer.max, 1)
)

BootOptimismControl(
  samples = 25,
  weights = TRUE,
  seed = sample(.Machine$integer.max, 1)
)

CVControl(
  folds = 10,
  repeats = 1,
  weights = TRUE,
  seed = sample(.Machine$integer.max, 1)
)

CVOptimismControl(
  folds = 10,
  repeats = 1,
  weights = TRUE,
  seed = sample(.Machine$integer.max, 1)
)

OOBControl(
  samples = 25,
  weights = TRUE,
```

```
  seed = sample(.Machine$integer.max, 1)
)

SplitControl(
  prop = 2/3,
  weights = TRUE,
  seed = sample(.Machine$integer.max, 1)
)

TrainControl(weights = TRUE, seed = sample(.Machine$integer.max, 1))
```

Arguments

samples	number of bootstrap samples.
weights	logical indicating whether to return case weights in resampled output for the calculation of performance metrics.
seed	integer to set the seed at the start of resampling.
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).

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 separate training 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).

Value

Object that inherits from the MLControl class.

References

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Chapman & Hall/CRC.

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

Harrell, F. E., Lee, K. L., & Mark, D. B. (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 *IJCAI'95: Proceedings of the 14th International Joint Conference on Artificial Intelligence* (vol. 2, pp. 1137-1143). Morgan Kaufmann Publishers Inc.

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap methods and their application*. Cambridge University Press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction* (2nd ed.). 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

`set_monitor`, `set_predict`, `set_strata`, `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 <- MLMetric(
  function(observed, predicted, ...) {
    f_score(observed, predicted, beta = 2, ...)
  },
  name = "f2_score",
  label = "F Score (beta = 2)",
  maximize = TRUE
)
```

MLModel *MLModel and MLModelFunction Class Constructors*

Description

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

Usage

```
MLModel(
  name = "MLModel",
  label = name,
  packages = character(),
  response_types = character(),
  weights = FALSE,
  predictor_encoding = c(NA, "model.frame", "model.matrix"),
  na.rm = FALSE,
  params = list(),
  gridinfo = tibble::tibble(param = character(), get_values = list(), default =
    logical()),
  fit = function(formula, data, weights, ...) stop("No fit function."),
  predict = function(object, newdata, times, ...) stop("No predict function."),
  varimp = function(object, ...) NULL,
  ...
)

MLModelFunction(object, ...)
```

Arguments

name	character name of the object to which the model is assigned.
label	optional character descriptor for the model.
packages	character vector of package names upon which the model depends. Each name may be optionally followed by a comment in parentheses specifying a version requirement. The comment should contain a comparison operator, whitespace and a valid version number, e.g. "xgboost (>= 1.3.0)".
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".
weights	logical value or vector of the same length as response_types indicating whether case weights are supported for the responses.
predictor_encoding	character string indicating whether the model is fit with predictor variables encoded as a "model.frame", a "model.matrix", or unspecified (default).

na.rm	character string or logical specifying removal of "all" (TRUE) cases with missing values from model fitting and prediction, "none" (FALSE), or only those whose missing values are in the "response" variable.
params	list of user-specified model parameters to be passed to the fit function.
gridinfo	tibble of information for construction of tuning grids consisting of a character column param with the names of parameters in the grid, a list column get_values with functions to generate grid points for the corresponding parameters, and an optional logical column default indicating which parameters to include by default in regular grids. Values functions may optionally include arguments n and data for the number of grid points to generate and a ModelFrame of the model fit data and formula, respectively; and must include 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 to other methods.
object	function that returns an MLModel object when called without any supplied argument values.

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.

Arguments data and newdata in the fit and predict functions may be converted to data frames with as.data.frame() if needed for their operation. The fit 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 matrix whose columns contain the probabilities for multi-level factors or vector of probabilities for the second level of binary factors.

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.

The predict and varimp functions are additionally passed a list named .MachineShop containing the input and model from fit. This argument may be included in the function definitions as needed for their implementations. Otherwise, it will be captured by the ellipsis.

Value

An MLModel or MLModelFunction class object.

See Also

models, fit, resample

Examples

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

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

ModelFrame *ModelFrame Class*

Description

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

Usage

```
ModelFrame(...)

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

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

Arguments

...	arguments passed from the generic function to its methods. The first argument of each ModelFrame method is positional and, as such, must be given first in calls to them.
formula, data	formula defining the model predictor and response variables and a data frame containing them. In the associated method, arguments groups, strata, and weights will be evaluated as expressions, whose objects are searched for first in the accompanying data environment and, if not found there, next in the calling environment.
groups	vector of values defining groupings of case observations, such as repeated measurements, to keep together during resampling [default: none].
strata	vector of values to use in conducting stratified resample estimation of model performance [default: none].
weights	numeric vector of non-negative case weights for the y response variable [default: equal weights].
na.rm	character string or logical specifying removal of "all" (TRUE) cases with missing values, "none" (FALSE), or only those whose missing values are in the "response" variable.
x, y	matrix and object containing predictor and response variables.
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

```
## Requires prior installation of suggested package gbm to run
```

```
mf <- ModelFrame(ncases / (ncases + ncontrols) ~ agegp + tobgp + alcgp,
                 data = esoph, weights = 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 objects; 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.

weights logical value or vector of the same length as `response_types` indicating whether case weights are supported for the responses.

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 model-specific variable importance is defined.

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	
ParsnipModel	f	m,n	S
PDAModel	f		

PLSModel	f	n	
POLRModel	o		
QDAModel	f		
RandomForestModel	f	n	
RangerModel	f	n	S
RFSRCModel	f	m,n	S
RFSRCFastModel	f	m,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.

ModelSpecification	Model specification
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

ModelSpecification *Model Specification*

Description

Specification of a relationship between response and predictor variables and a model to define a relationship between them.

Usage

```
ModelSpecification(...)

## Default S3 method:
ModelSpecification(
  input,
  model,
  control = MachineShop::settings("control"),
  metrics = NULL,
  cutoff = MachineShop::settings("cutoff"),
  stat = MachineShop::settings("stat.TrainingParams"),
  ...
)

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

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

## S3 method for class 'ModelFrame'
ModelSpecification(input, model, ...)

## S3 method for class 'recipe'
ModelSpecification(input, model, ...)
```

Arguments

`...`	arguments passed from the generic function to its methods. The first argument of each `ModelSpecification` method is positional and, as such, must be given first in calls to them.
`input`	input object defining and containing the model predictor and response variables.
`model`	model function, function name, or object; or another object that can be coerced to a model.
`control`	control function, function name, or object defining the resampling method to be employed. If `NULL` or if the model specification contains any `SelectedInput` or `SelectedModel` objects, then object-specific control structures and training parameters are used for selection and tuning, as usual, and objects are trained sequentially with nested resampling. Otherwise, • tuning of input and model objects is performed simultaneously over a global grid of their parameter values, and

- the specified `control` method and training parameters below override those of any included `TunedInput` or `TunedModel`.

`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.
`cutoff`	argument passed to the `metrics` functions.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for model tuning.
`formula, data`	formula defining the model predictor and response variables and a data frame containing them.
`x, y`	matrix and object containing predictor and response variables.

Value

`ModelSpecification` class object.

See Also

`fit`, `resample`, `set_monitor`, `set_optim`

Examples

```
## Requires prior installation of suggested package gbm to run

modelspec <- ModelSpecification(
  sale_amount ~ ., data = ICHomes, model = GBMModel
)
fit(modelspec)
```

`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

```
## Requires prior installation of suggested package e1071 to run

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 = logical(),
  entropy = logical(),
  softmax = logical(),
  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 argument values and further model details can be found in the source See Also 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(..., size = 3, random = FALSE)

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

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

Arguments

`...`	named `param` objects as defined in the **dials** package.
`size`	single integer or vector of integers whose positions or names match the given parameters and which specify the number of values used to construct the grid.
`random`	number of unique points to sample at random from the grid defined by `size`, or `FALSE` for all points.
`object`	list of named `param` objects or a `parameters` object. This is a positional argument that must be given first in calls to its methods.

Value

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

See Also

`TunedModel`

Examples

```
## GBMModel tuning parameters
grid <- ParameterGrid(
  n.trees = dials::trees(),
  interaction.depth = dials::tree_depth(),
  random = 5
)
TunedModel(GBMModel, grid = grid)
```

ParsnipModel *Parsnip Model*

Description

Convert a model specification from the **parsnip** package to one that can be used with the **MachineShop** package.

Usage

```
ParsnipModel(object, ...)
```

Arguments

object	model specification from the **parsnip** package.
...	tuning parameters with which to update object.

Value

ParsnipModel class object that inherits from MLModel.

See Also

as.MLModel, fit, resample

Examples

```
## Requires prior installation of suggested package parsnip to run

prsp_model <- parsnip::linear_reg(engine = "glmnet")

model <- ParsnipModel(prsp_model, penalty = 1, mixture = 1)
model

model_fit <- fit(sale_amount ~ ., data = ICHomes, model = model)
predict(model_fit)
```

performance *Model Performance Metrics*

Description

Compute measures of model performance.

Usage

```
performance(x, ...)

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

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

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

## S3 method for class 'numeric'
performance(
  x,
  y,
  weights = NULL,
```

```
  metrics = MachineShop::settings("metrics.numeric"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'Surv'
performance(
  x,
  y,
  weights = NULL,
  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 'MLModel'
performance(x, ...)

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

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

Arguments

x	observed responses; or confusion, trained model fit, resample, or rfe result.
...	arguments passed from the Resample method to the response type-specific methods or from the method for ConfusionList to ConfusionMatrix. Elliptical arguments in the response type-specific methods are passed to metrics supplied as a single MLMetric function and are ignored otherwise.
y	predicted responses if not contained in x.
weights	numeric vector of non-negative case weights for the observed x responses [default: equal weights].
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

```
## Requires prior installation of suggested package gbm to run

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)
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,
  weights = NULL,
  metrics = c(MachineShop::tpr, MachineShop::fpr),
  na.rm = TRUE,
  ...
)

## S3 method for class 'Resample'
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.
weights	numeric vector of non-negative case weights for the observed x responses [default: equal weights].
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

```
## Requires prior installation of suggested package gbm to run

data(Pima.tr, package = "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 = numeric(),
  diagonal = TRUE,
  stat = MachineShop::settings("stat.Curve"),
  ...
)

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

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

## S3 method for class 'VariableImportance'
plot(x, n = Inf, ...)
```

Arguments

x	calibration, confusion, lift, trained model fit, partial dependence, performance, performance curve, resample, rfe, 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 Resample model ordering. The original ordering is preserved if a value of NULL is given. 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.

Examples

```
## Requires prior installation of suggested package gbm to run

## 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

```
## Requires prior installation of suggested package pls to run

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

```
data(Boston, package = "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 = numeric(),
  type = c("response", "raw", "numeric", "prob", "default"),
  cutoff = MachineShop::settings("cutoff"),
  distr = character(),
  method = character(),
  verbose = FALSE,
  ...
)

## S4 method for signature 'MLModelFit'
predict(object, ...)
```

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 (`"response"`), numeric (`"numeric"`), or probability (`"prob"`) scale; or the `"raw"` predictions returned by the model. Option `"default"` is deprecated and will be removed in the future; use `"raw"` instead.

cutoff	numeric (0, 1) threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified.
distr	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" or "efron" (default).
verbose	logical indicating whether to display printed output generated by some model-specific predict functions to aid in monitoring progress and diagnosing errors.
...	arguments passed from the S4 to the S3 method.

See Also

confusion, performance, metrics

Examples

```
## Requires prior installation of suggested package gbm to run

## 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("print_max"), ...)

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

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

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

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

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

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

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

## S3 method for class 'ModelFrame'
print(x, n = MachineShop::settings("print_max"), id = FALSE, data = TRUE, ...)

## S3 method for class 'ModelRecipe'
print(x, n = MachineShop::settings("print_max"), id = FALSE, data = TRUE, ...)

## S3 method for class 'ModelSpecification'
print(x, n = MachineShop::settings("print_max"), id = FALSE, ...)

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

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

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

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

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

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

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

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

Arguments

x	object to print.
n	integer number of models or data frame rows to show.
...	arguments passed to other methods, including the one described below.
	level = 0 current nesting level of the corresponding object in recursive calls to print. The amount of information displayed decreases and increases with positive and negative levels, respectively.
id	logical indicating whether to show object identifiers.
data	logical indicating whether to show model data.

QDAModel *Quadratic Discriminant Analysis Model*

Description

Performs quadratic discriminant analysis.

Usage

```
QDAModel(
  prior = numeric(),
  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 argument values and further model details can be found in the source See Also links below.

Value

MLModel class object.

See Also

qda, predict.qda, fit, resample

Examples

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

---

quote *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.

Usage

```
.(expr)
```

Arguments

expr	any syntactically valid R expression.

Details

Useful for calling model functions with quoted parameter values defined in terms of one or more of the following variables.

nobs number of observations in data to be fit.

nvars number of predictor variables.

y the response variable.

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)
```

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 = integer()
)
```

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*

* excluded from grids by default

Default argument values and further model details can be found in the source See Also link below.

Value

MLModel class object.

See Also

randomForest, fit, resample

Examples

```
## Requires prior installation of suggested package randomForest to run

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 = integer(),
  importance = c("impurity", "impurity_corrected", "permutation"),
  min.node.size = integer(),
  replace = TRUE,
  sample.fraction = if (replace) 1 else 0.632,
  splitrule = character(),
  num.random.splits = 1,
  alpha = 0.5,
  minprop = 0.1,
  split.select.weights = numeric(),
  always.split.variables = character(),
  respect.unordered.factors = character(),
  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`*

* excluded from grids by default

Default argument values and further model details can be found in the source See Also link below.

Value

`MLModel` class object.

See Also

`ranger`, `fit`, `resample`

Examples

```
## Requires prior installation of suggested package ranger to run

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, group, 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.
group	variable defining groupings of case observations, such as repeated measurements, to keep together during resampling [default: none].
stratum	variable to use in conducting stratified resample estimation of model performance.
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)

df <- within(veteran, {
  y <- Surv(time, status)
  remove(time, status)
})
rec <- recipe(y ~ ., data = df) %>%
  role_case(stratum = y)

(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(...)

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

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

## S3 method for class 'ModelFrame'
resample(input, model, ...)

## S3 method for class 'recipe'
resample(input, model, ...)

## S3 method for class 'ModelSpecification'
resample(object, control = MachineShop::settings("control"), ...)

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

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

Arguments

`...`	arguments passed from the generic function to its methods, from the `MLModel` and `MLModelFunction` methods to first arguments of others, and from others to the `ModelSpecification` method. The first argument of each `fit` method is positional and, as such, must be given first in calls to them.
`formula, data`	formula defining the model predictor and response variables and a data frame containing them.
`model`	model function, function name, or object; or another object that can be coerced to a model. A model can be given first followed by any of the variable specifications.
`x, y`	matrix and object containing predictor and response variables.
`input`	input object defining and containing the model predictor and response variables.

object	model input or specification.
control	control function, function name, or object defining the resampling method to be employed.

Details

Stratified resampling is performed automatically for the formula and matrix methods according to the type of response variable. In general, strata are constructed from numeric proportions for BinomialVariate; original values for character, factor, logical, and ordered; first columns of values for matrix; original values for numeric; and numeric times within event statuses for Surv. Numeric values are stratified into quantile bins and categorical values into factor levels defined by MLControl.

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

Stratification variables may be designated in recipe specifications with the role_case function. Resampling will be unstratified otherwise.

Value

Resample class object.

See Also

c, metrics, performance, plot, summary

Examples

```
## Requires prior installation of suggested package gbm to run

## 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 'ModelSpecification'
response(object, newdata = NULL, ...)

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

Arguments

`object`	model fit, input, or specification containing predictor and response variables.
`...`	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)
```

rfe *Recursive Feature Elimination*

Description

A wrapper method of backward feature selection in which a given model is fit to nested subsets of most important predictor variables in order to select the subset whose resampled predictive performance is optimal.

Usage

```
rfe(...)

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

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

## S3 method for class 'ModelFrame'
rfe(input, model, ...)

## S3 method for class 'recipe'
rfe(input, model, ...)

## S3 method for class 'ModelSpecification'
rfe(
  object,
  select = NULL,
  control = MachineShop::settings("control"),
  props = 4,
  sizes = integer(),
  random = FALSE,
  recompute = TRUE,
  optimize = c("global", "local"),
  samples = c(rfe = 1, varimp = 1),
  metrics = NULL,
  stat = c(resample = MachineShop::settings("stat.Resample"), permute =
    MachineShop::settings("stat.TrainingParams")),
  progress = FALSE,
  ...
)

## S3 method for class 'MLModel'
rfe(model, ...)

## S3 method for class 'MLModelFunction'
```

```
rfe(model, ...)
```

Arguments

...	arguments passed from the generic function to its methods, from the MLModel and MLModelFunction methods to first arguments of others, and from others to the ModelSpecification method. The first argument of each fit method is positional and, as such, must be given first in calls to them.
formula, data	formula defining the model predictor and response variables and a data frame containing them.
model	model function, function name, or object; or another object that can be coerced to a model. A model can be given first followed by any of the variable specifications.
x, y	matrix and object containing predictor and response variables.
input	input object defining and containing the model predictor and response variables.
object	model input or specification.
select	expression indicating predictor variables that can be eliminated (see subset for syntax) [default: all].
control	control function, function name, or object defining the resampling method to be employed.
props	numeric vector of the proportions of most important predictor variables to retain in fitted models or an integer number of equal spaced proportions to generate automatically; ignored if sizes are given.
sizes	integer vector of the set sizes of most important predictor variables to retain.
random	logical indicating whether to eliminate variables at random with probabilities proportional to their importance.
recompute	logical indicating whether to recompute variable importance after eliminating each set of variables.
optimize	character string specifying a search through all props to identify the globally optimal model ("global") or a search that stops after identifying the first locally optimal model ("local").
samples	numeric vector or list giving the number of permutation samples for each of the rfe and varimp algorithms. One or both of the values may be specified as named arguments or in the order in which their defaults appear. Larger numbers of samples decrease variability in estimated model performances and variable importances at the expense of increased computation time. Samples are more expensive computationally for rfe than for varimp.
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.
stat	functions or character strings naming functions to compute summary statistics on resampled metric values and permuted samples. One or both of the values may be specified as named arguments or in the order in which their defaults appear.
progress	logical indicating whether to display iterative progress during elimination.

Value

TrainingStep class object containing a summary of the numbers of predictor variables retained (size), their names (terms), logical indicators for the optimal model selected (selected), and associated performance metrics (metrics).

See Also

performance, plot, summary, varimp

Examples

```
## Requires prior installation of suggested package gbm to run

(res <- rfe(sale_amount ~ ., data = ICHomes, model = GBMModel))
summary(res)
summary(performance(res))
plot(res, type = "line")
```

RFSRCModel *Fast Random Forest (SRC) Model*

Description

Fast OpenMP computing of Breiman's random forest for a variety of data settings including right-censored survival, regression, and classification.

Usage

```
RFSRCModel(
  ntree = 1000,
  mtry = integer(),
  nodesize = integer(),
  nodedepth = integer(),
  splitrule = character(),
  nsplit = 10,
  block.size = integer(),
  samptype = c("swor", "swr"),
  membership = FALSE,
  sampsize = if (samptype == "swor") function(x) 0.632 * x else function(x) x,
  nimpute = 1,
  ntime = integer(),
  proximity = c(FALSE, TRUE, "inbag", "oob", "all"),
  distance = c(FALSE, TRUE, "inbag", "oob", "all"),
  forest.wt = c(FALSE, TRUE, "inbag", "oob", "all"),
  xvar.wt = numeric(),
```

```
  split.wt = numeric(),
  var.used = c(FALSE, "all.trees", "by.tree"),
  split.depth = c(FALSE, "all.trees", "by.tree"),
  do.trace = FALSE,
  statistics = FALSE
)

RFSRCFastModel(
  ntree = 500,
  sampsize = function(x) min(0.632 * x, max(x^0.75, 150)),
  ntime = 50,
  terminal.qualts = FALSE,
  ...
)
```

Arguments

ntree	number of trees.
mtry	number of variables randomly selected as candidates for splitting a node.
nodesize	minumum size of terminal nodes.
nodedepth	maximum depth to which a tree should be grown.
splitrule	splitting rule (see rfsrc).
nsplit	non-negative integer value for number of random splits to consider for each candidate splitting variable.
block.size	interval number of trees at which to compute the cumulative error rate.
samptype	whether bootstrap sampling is with or without replacement.
membership	logical indicating whether to return terminal node membership.
sampsize	function specifying the bootstrap size.
nimpute	number of iterations of the missing data imputation algorithm.
ntime	integer number of time points to constrain ensemble calculations for survival outcomes.
proximity	whether and how to return proximity of cases as measured by the frequency of sharing the same terminal nodes.
distance	whether and how to return distance between cases as measured by the ratio of the sum of edges from each case to the root node.
forest.wt	whether and how to return the forest weight matrix.
xvar.wt	vector of non-negative weights representing the probability of selecting a variable for splitting.
split.wt	vector of non-negative weights used for multiplying the split statistic for a variable.
var.used	whether and how to return variables used for splitting.
split.depth	whether and how to return minimal depth for each variable.

do.trace	number of seconds between updates to the user on approximate time to completion.
statistics	logical indicating whether to return split statistics.
terminal.qualts	logical indicating whether to return terminal node membership information.
...	arguments passed to RFSRCModel.

Details

Response types: factor, matrix, numeric, Surv

Automatic tuning of grid parameters: mtry, nodesize

Default argument values and further model details can be found in the source See Also links below.

In calls to varimp for RFSRCModel, argument type may be specified as "anti" (default) for cases assigned to the split opposite of the random assignments, as "permute" for permutation of OOB cases, or as "random" for permutation replaced with random assignment. 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

rfsrc, rfsrc.fast, fit, resample

Examples

```
## Requires prior installation of suggested package randomForestSRC to run

model_fit <- fit(sale_amount ~ ., data = ICHomes, model = RFSRCModel)
varimp(model_fit, method = "model", type = "random", scale = TRUE)
```

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 parameter: cp

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

Value

MLModel class object.

See Also

rpart, fit, resample

Examples

```
## Requires prior installation of suggested packages rpart and partykit to run

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,
  cutoff = MachineShop::settings("cutoff"),
  stat = MachineShop::settings("stat.TrainingParams")
)

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

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

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

```
)

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

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

Arguments

`...`	inputs defining 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 containing predictor and response variables.
`control`	control function, function name, or object 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.
`cutoff`	argument passed to the `metrics` functions.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for recipe selection.
`y`	response variable.
`x`	list of inputs followed by arguments passed to their method function.

Value

`SelectedModelFrame`, `SelectedModelRecipe`, or `SelectedModelSpecification` class object that inherits from `SelectedInput` and `ModelFrame`, `recipe`, or `ModelSpecification`, respectively.

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)
data(Boston, package = "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(...)

## Default S3 method:
SelectedModel(
  ...,
  control = MachineShop::settings("control"),
  metrics = NULL,
  cutoff = MachineShop::settings("cutoff"),
  stat = MachineShop::settings("stat.TrainingParams")
)

## S3 method for class 'ModelSpecification'
SelectedModel(
  ...,
  control = MachineShop::settings("control"),
  metrics = NULL,
  cutoff = MachineShop::settings("cutoff"),
  stat = MachineShop::settings("stat.TrainingParams")
)

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

Arguments

...	model functions, function names, objects; other objects that can be coerced to models; vectors of these to serve as the candidate set from which to select, such as that returned by `expand_model`; or model specifications.
control	control function, function name, or object 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.
cutoff	argument passed to the `metrics` functions.
stat	function or character string naming a function to compute a summary statistic on resampled metric values for model selection.
x	list of models followed by arguments passed to their method function.

Details

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

Value

`SelectedModel` or `SelectedModelSpecification` class object that inherits from `MLModel` or `ModelSpecification`, respectively.

See Also

`fit`, `resample`

Examples

```
## Requires prior installation of suggested package gbm and glmnet to run

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 object 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].

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

`distr.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"`, `"rayleigh"`, or `"weibull"`.

`grid size` argument to `TuningGrid` indicating the number of parameter-specific values to generate automatically for tuning of models that have pre-defined grids or a `TuningGrid` function, function name, or object [default: 3].

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

`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")`].

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

`require` names of installed packages to load during parallel execution of resampling algorithms [default: `"MachineShop"`].

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

`stat.TrainingParams` 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.Resample` 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")`].

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)
```

`set_monitor` *Training Parameters Monitoring Control*

Description

Set parameters that control the monitoring of resample estimation of model performance and of tuning parameter optimization.

Usage

```
set_monitor(object, ...)

## S3 method for class 'MLControl'
set_monitor(object, progress = TRUE, verbose = FALSE, ...)

## S3 method for class 'MLOptimization'
set_monitor(object, progress = FALSE, verbose = FALSE, ...)

## S3 method for class 'ModelSpecification'
set_monitor(object, which = c("all", "control", "optim"), ...)
```

Arguments

`object`	resampling control, tuning parameter optimization, or model specification object.
`...`	arguments passed from the `ModelSpecification` method to the others.
`progress`	logical indicating whether to display iterative progress during resampling or optimization. In the case of resampling, a progress bar will be displayed if a computing cluster is not registered or is registered with the **doSNOW** package.
`verbose`	numeric or logical value specifying the level of progress detail to print, with 0 (`FALSE`) indicating none and 1 (`TRUE`) or higher indicating increasing amounts of detail.
`which`	character string specifying the monitoring parameters to set as `"all"`, `"control"`, or optimization (`"optim"`).

Value

Argument `object` updated with the supplied parameters.

See Also

`resample`, `set_optim`, `set_predict`, `set_strata`

Examples

```
CVControl() %>% set_monitor(verbose = TRUE)
```

`set_optim` *Tuning Parameter Optimization*

Description

Set the optimization method and control parameters for tuning of model parameters.

Usage

```
set_optim_bayes(object, ...)

## S3 method for class 'ModelSpecification'
set_optim_bayes(
  object,
  num_init = 5,
  times = 10,
  each = 1,
  acquisition = c("ucb", "ei", "eips", "poi"),
  kappa = stats::qnorm(conf),
  conf = 0.995,
  epsilon = 0,
  control = list(),
  packages = c("ParBayesianOptimization", "rBayesianOptimization"),
  random = FALSE,
  progress = verbose,
  verbose = 0,
  ...
)

set_optim_bfgs(object, ...)

## S3 method for class 'ModelSpecification'
set_optim_bfgs(
  object,
  times = 10,
  control = list(),
  random = FALSE,
  progress = FALSE,
  verbose = 0,
  ...
)

set_optim_grid(object, ...)
```

```
## S3 method for class 'TrainingParams'
set_optim_grid(object, random = FALSE, progress = FALSE, ...)

## S3 method for class 'ModelSpecification'
set_optim_grid(object, ...)

## S3 method for class 'TunedInput'
set_optim_grid(object, ...)

## S3 method for class 'TunedModel'
set_optim_grid(object, ...)

set_optim_pso(object, ...)

## S3 method for class 'ModelSpecification'
set_optim_pso(
  object,
  times = 10,
  each = NULL,
  control = list(),
  random = FALSE,
  progress = FALSE,
  verbose = 0,
  ...
)

set_optim_sann(object, ...)

## S3 method for class 'ModelSpecification'
set_optim_sann(
  object,
  times = 10,
  control = list(),
  random = FALSE,
  progress = FALSE,
  verbose = 0,
  ...
)

set_optim_method(object, ...)

## S3 method for class 'ModelSpecification'
set_optim_method(
  object,
  fun,
  label = "Optimization Function",
  packages = character(),
```

```
  params = list(),
  random = FALSE,
  progress = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

object	input or model object.
...	arguments passed to the TrainingParams method of set_optim_grid from its other methods.
num_init	number of grid points to sample for the initialization of Bayesian optimization.
times	maximum number of times to repeat the optimization step. Multiple sets of model parameters are evaluated automatically at each step of the BFGS algorithm to compute a finite-difference approximation to the gradient.
each	number of times to sample and evaluate model parameters at each optimization step. This is the swarm size in particle swarm optimization, which defaults to floor(10 + 2 * sqrt(length(bounds))).
acquisition	character string specifying the acquisition function as "ucb" (upper confidence bound), "ei" (expected improvement), "eips" (expected improvement per second), or "poi" (probability of improvement).
kappa, conf	upper confidence bound ("ucb") quantile or its probability to balance exploitation against exploration. Argument kappa takes precedence if both are given and multiplies the predictive standard deviation added to the predictive mean in the acquisition function. Larger values encourage exploration of the model parameter space.
epsilon	improvement methods ("ei", "eips", and "poi") parameter to balance exploitation against exploration. Values should be between -0.1 and 0.1 with larger ones encouraging exploration.
control	list of control parameters passed to bayesOpt by set_optim_bayes with package "ParBayesianOptimization", to BayesianOptimization by set_optim_bayes with package "rBayesianOptimization", to optim by set_optim_bfgs and set_optim_sann, and to psoptim by set_optim_pso.
packages	R package or packages to use for the optimization method, or an empty vector if none are needed. The first package in set_optim_bayes is used unless otherwise specified by the user.
random	number of points to sample for a random grid search, or FALSE for an exhaustive grid search. Used when a grid search is specified or as the fallback method for non-numeric model parameters present during other optimization methods.
progress	logical indicating whether to display iterative progress during optimization.
verbose	numeric or logical value specifying the level of progress detail to print, with 0 (FALSE) indicating none and 1 (TRUE) or higher indicating increasing amounts of detail.

`fun` user-defined optimization function to which the arguments below are passed in order. An ellipsis can be included in the function definition when using only a subset of the arguments and ignoring others. A tibble returned by the function with the same number of rows as model evaluations will be included in a `TrainingStep` summary of optimization results; other types of return values will be ignored.

optim function that takes a numeric vector or list of named model parameters as the first argument, optionally accepts the maximum number of iterations as argument `max_iter`, and returns a scalar measure of performance to be maximized. Parameter names are available from the `grid` and `bounds` arguments described below. If the function cannot be evaluated at a given set of parameter values, then `-Inf` is returned.

grid data frame containing a tuning grid of all model parameters.

bounds named list of lower and upper bounds for each finite numeric model parameter in `grid`. The types (integer or double) of the original parameter values are preserved in the bounds.

params list of optimization parameters as supplied to `set_optim_method`.

monitor list of the `progress` and `verbose` values.

`label` character descriptor for the optimization method.

`params` list of user-specified model parameters to be passed to `fun`.

Details

The optimization functions implement the following methods.

`set_optim_bayes` Bayesian optimization with a Gaussian process model (Snoek et al. 2012).

`set_optim_bfgs` limited-memory modification of quasi-Newton BFGS optimization (Byrd et al. 1995).

`set_optim_grid` exhaustive or random grid search.

`set_optim_pso` particle swarm optimization (Bratton and Kennedy 2007, Zambrano-Bigiarini et al. 2013).

`set_optim_sann` simulated annealing (Belisle 1992). This method depends critically on the control parameter settings. It is not a general-purpose method but can be very useful in getting to good parameter values on a very rough optimization surface.

`set_optim_method` user-defined optimization function.

The package-defined optimization functions evaluate and return values of the tuning parameters that are of same type (e.g. integer, double, character) as given in the `object` grid. Sequential optimization of numeric tuning parameters is performed over a hypercube defined by their minimum and maximum grid values. Non-numeric parameters are optimized with grid searches.

Value

Argument `object` updated with the specified optimization method and control parameters.

References

Belisle, C. J. P. (1992). Convergence theorems for a class of simulated annealing algorithms on Rd. *Journal of Applied Probability*, *29*, 885–895.

Bratton, D. & Kennedy, J. (2007), Defining a standard for particle swarm optimization. In *IEEE Swarm Intelligence Symposium, 2007* (pp. 120-127).

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, *16*, 1190–1208.

Snoek, J., Larochelle, H., & Adams, R.P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. arXiv:1206.2944 [stat.ML].

Zambrano-Bigiarini, M., Clerc, M., & Rojas, R. (2013). Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. In *IEEE Congress on Evolutionary Computation, 2013* (pp. 2337-2344).

See Also

BayesianOptimization, bayesOpt, optim, psoptim, set_monitor, set_predict, set_strata

Examples

```
ModelSpecification(
  sale_amount ~ ., data = ICHomes,
  model = TunedModel(GBMModel)
) %>% set_optim_bayes(package = "rBayesianOptimization")
```

---

set_predict *Resampling Prediction Control*

Description

Set parameters that control prediction during resample estimation of model performance.

Usage

```
set_predict(
  object,
  times = numeric(),
  distr = character(),
  method = character(),
  ...
)
```

Arguments

object	control object.
times, distr, method	arguments passed to predict.
...	arguments passed to other methods.

Value

Argument `object` updated with the supplied parameters.

See Also

`resample`, `set_monitor`, `set_optim`, `set_strata`

Examples

```
CVControl() %>% set_predict(times = 1:3)
```

`set_strata` *Resampling Stratification Control*

Description

Set parameters that control the construction of strata during resample estimation of model performance.

Usage

```
set_strata(object, breaks = 4, nunique = 5, prop = 0.1, size = 20, ...)
```

Arguments

`object`	control object.
`breaks`	number of quantile bins desired for stratification of numeric data during resampling.
`nunique`	number of unique values at or below which numeric data are stratified as categorical.
`prop`	minimum proportion of data in each strata.
`size`	minimum number of values in each strata.
`...`	arguments passed to other methods.

Details

The arguments control resampling strata which are constructed from numeric proportions for `BinomialVariate`; original values for `character`, `factor`, `logical`, `numeric`, and `ordered`; first columns of values for `matrix`; and numeric times within event statuses for `Surv`. Stratification of survival data by event status only can be achieved by setting `breaks = 1`. Numeric values are stratified into quantile bins and categorical values into factor levels. The number of bins will be the largest integer less than or equal to `breaks` satisfying the `prop` and `size` control argument thresholds. Categorical levels below the thresholds will be pooled iteratively by reassigning values in the smallest nominal level to the remaining ones at random and by combining the smallest adjacent ordinal levels. Missing values are replaced with non-missing values sampled at random with replacement.

Value

Argument `object` updated with the supplied parameters.

See Also

`resample`, `set_monitor`, `set_optim`, `set_predict`

Examples

```
CVControl() %>% set_strata(breaks = 3)
```

`StackedModel` *Stacked Regression Model*

Description

Fit a stacked regression model from multiple base learners.

Usage

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

Arguments

`...`	model functions, function names, objects; other objects that can be coerced to models; or vector of these to serve as base learners.
`control`	control function, function name, or object 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

```
## Requires prior installation of suggested packages gbm and glmnet to run

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, ...)

## S3 method for class 'step_kmeans'
tunable(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, E. W. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, *21*, 768-769.

Hartigan, J. A., & Wong, M. A. (1979). A K-means clustering algorithm. *Applied Statistics*, *28*, 100-108.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129-137.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (vol. 1, pp. 281-297). 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")
)

## S3 method for class 'step_kmedoids'
tunable(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., & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis*. Wiley.

Reynolds, A., Richards, G., de la Iglesia, B., & 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

```
## Requires prior installation of suggested package cluster to run

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, ...)

## S3 method for class 'step_lincomp'
tunable(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_predictors(),
               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_predictors(),
           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 = integer(),
  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")
)

## S3 method for class 'step_spca'
tunable(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., & Tibshirani, R. (2006). Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, *15*(2), 265-286.

See Also

spca, recipe, prep, bake

Examples

```
## Requires prior installation of suggested package elasticnet to run

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.Resample"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'MLModelFit'
summary(object, .type = c("default", "glance", "tidy"), ...)

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

## S3 method for class 'PerformanceCurve'
```

```
summary(object, stat = MachineShop::settings("stat.Curve"), ...)

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

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

Arguments

object	confusion, lift, trained model fit, performance, performance curve, resample, or rfe 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.
.type	character string specifying that `unMLModelFit(object)` be passed to `summary` ("default"), `glance`, or `tidy`.
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

```
## Requires prior installation of suggested package gbm to run

## 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, objects; other objects that can be coerced to models; or vector of these to serve as base learners.
`model`	model function, function name, or object defining the super model; or another object that can be coerced to the model.
`control`	control function, function name, or object 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 Laan, M. J., Polley, E. C., & Hubbard, A. E. (2007). Super learner. *Statistical Applications in Genetics and Molecular Biology*, *6*(1).

See Also

`fit`, `resample`

Examples

```
## Requires prior installation of suggested packages gbm and glmnet to run

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 = numeric(), distr = character())

SurvProbs(data = NA, times = numeric(), distr = character())
```

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.
distr	character string specifying the survival distribution from which the matrix values were derived.

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 = 0,
  parms = list(),
  ...
)

SurvRegStepAICModel(
  dist = c("weibull", "exponential", "gaussian", "logistic", "lognormal",
    "logloglogistic"),
  scale = 0,
  parms = list(),
  ...,
  direction = c("both", "backward", "forward"),
  scope = list(),
  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 argument values and further model details can be found in the source See Also links below.

Value

`MLModel` class object.

See Also

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

`stepAIC`, `fit`, `resample`

Examples

```
## Requires prior installation of suggested packages rms and Hmisc to run

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 = character(),
 kernel = c("rbfdot", "polydot", "vanilladot", "tanhdot", "laplacedot", "besseldot",
    "anovadot", "splinedot"),
  kpar = "automatic",
  C = 1,
  nu = 0.2,
  epsilon = 0.1,
  prob.model = FALSE,
  cache = 40,
  tol = 0.001,
  shrinking = TRUE
```

```
)

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

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

SVMLaplaceModel(sigma = numeric(), ...)

SVMLinearModel(...)

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

SVMRadialModel(sigma = numeric(), ...)

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.
prob.model	logical indicating whether to calculate the scaling parameter of the Laplacian distribution fitted on the residuals of numeric response variables. Ignored in the case of a factor response variable.
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 from the other constructors.
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:

- SVMModel: `NULL`
- SVMANOVAModel: `C`, `degree`
- SVMBesselModel: `C`, `order`, `degree`
- SVMLaplaceModel: `C`, `sigma`
- SVMLinearModel: `C`
- SVMPolyModel: `C`, `degree`, `scale`
- SVMRadialModel: `C`, `sigma`

The kernel-specific constructor functions `SVMANOVAModel`, `SVMBesselModel`, `SVMLaplaceModel`, `SVMLinearModel`, `SVMPolyModel`, `SVMRadialModel`, `SVMSplineModel`, and `SVMTanhModel` are special cases of `SVMModel` which automatically set its `kernel` and `kpar` arguments. These are called directly in typical usage unless `SVMModel` is needed to specify a more general model.

Default argument values and further model details can be found in the source See Also 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`	method of p-value adjustment for multiple statistical comparisons as implemented by `p.adjust`.
`...`	arguments passed to other methods.

Details

The t-test statistic for pairwise model differences of R resampled performance metric values is calculated as

$$t = \frac{\bar{x}_R}{\sqrt{F s_R^2 / R}},$$

where $\bar{x}_R$ and s_R^2 are the sample mean and variance. Statistical testing for a mean difference is then performed by comparing t to a t_{R-1} null distribution. The sample variance in the t statistic is known to underestimate the true variances of cross-validation mean estimators. Underestimation of these variances will lead to increased probabilities of false-positive statistical conclusions. Thus, an additional factor F is included in the t statistic to allow for variance corrections. A correction of $F = 1 + K/(K-1)$ was found by Nadeau and Bengio (2003) to be a good choice for cross-validation with K folds and is thus used for that resampling method. The extension of this correction by Bouchaert and Frank (2004) to $F = 1 + TK/(K-1)$ is used for cross-validation with K folds repeated T times. For other resampling methods $F = 1$.

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 contained in the third dimension.

References

Nadeau, C., & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, *52*, 239–81.

Bouckaert, R. R., & Frank, E. (2004). Evaluating the replicability of significance tests for comparing learning algorithms. In H. Dai, R. Srikant, & C. Zhang (Eds.), *Advances in knowledge discovery and data mining* (pp. 3–12). Springer.

Examples

```
## Requires prior installation of suggested package gbm to run

## 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 = numeric(),
  best = integer(),
  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

```
## Requires prior installation of suggested package tree to run

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

`TunedInput` *Tuned Model Inputs*

Description

Recipe tuning over a grid of parameter values.

Usage

```
TunedInput(object, ...)

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

Arguments

`object`	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 object 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.
`cutoff`	argument passed to the `metrics` functions.
`stat`	function or character string naming a function to compute a summary statistic on resampled metric values for recipe tuning.

Value

TunedModelRecipe class object that inherits from TunedInput and recipe.

See Also

fit, resample, set_optim

Examples

```
library(recipes)
data(Boston, package = "MASS")

rec <- recipe(medv ~ ., data = Boston) %>%
  step_pca(all_numeric_predictors(), 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(
  object,
  grid = MachineShop::settings("grid"),
  control = MachineShop::settings("control"),
  metrics = NULL,
  cutoff = MachineShop::settings("cutoff"),
  stat = MachineShop::settings("stat.TrainingParams")
)
```

Arguments

object	model function, function name, or object defining the model to be tuned.
grid	single integer or vector of integers whose positions or names match the parameters in the model's pre-defined tuning grid if one exists and which specify the number of values used to construct the grid; TuningGrid function, function name, or object; ParameterGrid object; or data frame containing parameter values at which to evaluate the model, such as that returned by expand_params.

control	control function, function name, or object 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.
cutoff	argument passed to the metrics functions.
stat	function or character string naming a function to compute a summary statistic on resampled metric values for model tuning.

Details

The expand_modelgrid function enables manual extraction and viewing of grids created automatically when a TunedModel is fit.

Response types: factor, numeric, ordered, Surv

Value

TunedModel class object that inherits from MLModel.

See Also

fit, resample, set_optim

Examples

```
## Requires prior installation of suggested package gbm to run
## May require a long runtime

# 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 = TuningGrid(size = 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)
  )
))
```

TuningGrid *Tuning Grid Control*

Description

Defines control parameters for a tuning grid.

Usage

```
TuningGrid(size = 3, random = FALSE)
```

Arguments

size	single integer or vector of integers whose positions or names match the parameters in a model's tuning grid and which specify the number of values used to construct the grid.
random	number of unique points to sample at random from the grid defined by size. If size is a single unnamed integer, then random = Inf will include all values of all grid parameters in the constructed grid, whereas random = FALSE will include all values of default grid parameters.

Details

Returned TuningGrid objects may be supplied to TunedModel for automated construction of model tuning grids. These grids can be extracted manually and viewed with the expand_modelgrid function.

Value

TuningGrid class object.

See Also

TunedModel, expand_modelgrid

Examples

```
TunedModel(XGBTreeModel, grid = TuningGrid(10, random = 5))
```

unMLModelFit *Revert an MLModelFit Object*

Description

Function to revert an `MLModelFit` object to its original class.

Usage

```
unMLModelFit(object)
```

Arguments

`object`	model fit result.

Value

The supplied object with its `MLModelFit` classes and fields removed.

varimp *Variable Importance*

Description

Calculate measures of relative importance for model predictor variables.

Usage

```
varimp(
  object,
  method = c("permute", "model"),
  scale = TRUE,
  sort = c("decreasing", "increasing", "asis"),
  ...
)
```

Arguments

`object`	model fit result.
`method`	character string specifying the calculation of variable importance as permutation-base (`"permute"`) or model-specific (`"model"`). If model-specific importance is specified but not defined, the permutation-based method will be used instead with its default values (below). Permutation-based variable importance is defined as the relative change in model predictive performances between datasets with and without permuted values for the associated variable (Fisher et al. 2019).

`scale` logical value or vector indicating whether importance values are scaled to a maximum of 100.

`sort` character string specifying the sort order of importance values to be `"decreasing"`, `"increasing"`, or as predictors appear in the model formula (`"asis"`).

`...` arguments passed to model-specific or permutation-based variable importance functions. These include the following arguments and default values for `method = "permute"`.

`select = NULL` expression indicating predictor variables for which to compute variable importance (see `subset` for syntax) [default: all].

`samples = 1` number of times to permute the values of each variable. Larger numbers of samples decrease variability in the estimates at the expense of increased computation time.

`prop = numeric()` proportion of observations to sample without replacement at each round of variable permutations [default: all]. Subsampling of observations can decrease computation time.

`size = integer()` number of observations to sample at each round of permutations [default: all].

`times = numeric()` numeric vector of follow-up times at which to predict survival probabilities or `NULL` for predicted survival means.

`metric = NULL` metric function or function name with which to calculate performance. If not specified, the first applicable default metric from the performance functions is used.

`compare = c("-", "/")` character specifying the relative change to compute in comparing model predictive performances between datasets with and without permuted values. The choices are difference (`"-"`) and ratio (`"/"`).

`stats = MachineShop::settings("stat.TrainingParams")` function, function name, or vector of these with which to compute summary statistics on the set of variable importance values from the permuted datasets.

`na.rm = TRUE` logical indicating whether to exclude missing variable importance values from the calculation of summary statistics.

`progress = TRUE` logical indicating whether to display iterative progress during computation.

Details

The `varimp` function supports calculation of variable importance with the permutation-based method of Fisher et al. (2019) or with model-based methods where defined. Permutation-based importance is the default and has the advantages of being available for any model, any performance metric defined for the associated response variable type, and any predictor variable in the original training dataset. Conversely, model-specific importance is not defined for some models and will fall back to the permutation method in such cases; is generally limited to metrics implemented in the source packages of models; and may be computed on derived, rather than original, predictor variables. These disadvantages can make comparisons of model-specific importance across different classes of models infeasible. A downside of the permutation-based approach is increased computation time. To counter this, the permutation algorithm can be run in parallel simply by loading a parallel backend for the **foreach** package `%dopar%` function, such as **doParallel** or **doSNOW**.

Permutation variable importance is interpreted as the contribution of a predictor variable to the predictive performance of a model as measured by the performance metric used in the calculation. Importance of a predictor is conditional on and, with the default scaling, relative to the values of all other predictors in the analysis.

Value

`VariableImportance` class object.

References

Fisher, A., Rudin, C., & Dominici, F. (2019). All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, *20*, 1-81.

See Also

`plot`

Examples

```
## Requires prior installation of suggested package gbm to run

## 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 with an efficient implementation of the gradient boosting framework from Chen & Guestrin.

Usage

```
XGBModel(
  nrounds = 100,
  ...,
  objective = character(),
  aft_loss_distribution = "normal",
  base_score = 0.5,
  verbose = 0,
```

```
  print_every_n = 1
)

XGBDARTModel(
  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,
  alpha = 0,
  lambda = 1,
  tree_method = "auto",
  scale_pos_weight = 1,
  refresh_leaf = 1,
  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(
  alpha = 0,
  lambda = 0,
  updater = "shotgun",
  feature_selector = "cyclic",
  top_k = 0,
  ...
)

XGBTreeModel(
  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,
  alpha = 0,
  lambda = 1,
  tree_method = "auto",
  scale_pos_weight = 1,
  refresh_leaf = 1,
  grow_policy = "depthwise",
  max_leaves = 0,
  max_bin = 256,
  num_parallel_tree = 1,
  ...
)
```

Arguments

nrounds	number of boosting iterations.
...	model parameters as described below and in the XGBoost documentation and arguments passed to XGBModel from the other constructors.
objective	optional character string defining the learning task and objective. Set automatically if not specified according to the following values available for supported response variable types. 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:aft", "survival:cox" The first values listed are the defaults for the corresponding response types.
aft_loss_distribution	character string specifying a distribution for the accelerated failure time objective ("survival:aft") as "extreme", "logistic", or "normal".
base_score	initial prediction score of all observations, global bias.
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.
eta	shrinkage of variable weights at each iteration to prevent overfitting.
gamma	minimum loss reduction required to split a tree node.
max_depth	maximum tree depth.
min_child_weight	minimum sum of observation weights required of nodes.
max_delta_step, tree_method, scale_pos_weight, updater, refresh_leaf, grow_policy, max_leaves, max_bin, num_parallel_tree	other tree booster parameters.
subsample	subsample ratio of the training observations.

colsample_bytree, colsample_bylevel, colsample_bynode
: subsample ratio of variables for each tree, level, or split.

alpha, lambda
: L1 and L2 regularization terms for variable weights.

sample_type, normalize_type
: type of sampling and normalization algorithms.

rate_drop
: rate at which to drop trees during the dropout procedure.

one_drop
: integer indicating whether to drop at least one tree during the dropout procedure.

skip_drop
: probability of skipping the dropout procedure during a boosting iteration.

feature_selector, top_k
: character string specifying the feature selection and ordering method, and number of top variables to select in the "greedy" and "thrifty" feature selectors.

Details

Response types: factor, numeric, PoissonVariate, Surv

Automatic tuning of grid parameters:

- XGBModel: NULL
- XGBDARTModel: nrounds, eta*, gamma*, max_depth, min_child_weight*, subsample*, colsample_bytree*, rate_drop*, skip_drop*
- XGBLinearModel: nrounds, alpha, lambda
- XGBTreeModel: nrounds, eta*, gamma*, max_depth, min_child_weight*, subsample*, colsample_bytree*

* excluded from grids by default

The booster-specific constructor functions XGBDARTModel, XGBLinearModel, and XGBTreeModel are special cases of XGBModel which automatically set the XGBoost booster parameter. These are called directly in typical usage unless XGBModel is needed to specify a more general model.

Default argument values and further model details can be found in the source See Also link below.

In calls to varimp for XGBTreeModel, argument type may be specified 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

```
## Requires prior installation of suggested package xgboost to run

model_fit <- fit(
  Species ~ ., data = iris, model = XGBTreeModel(nthread = 1)
```

```
)
varimp(model_fit, method = "model", type = "Frequency", scale = FALSE)
```

Index