

Package ‘MachineShop’

February 15, 2019

Type Package

Title Machine Learning Models and Tools

Version 1.2.0

Date 2019-02-15

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 common interface for model fitting, prediction, performance assessment, and presentation of results. Supports predictive modeling of numerical, categorical, and censored time-to-event outcomes and resample (bootstrap and cross-validation) estimation of model performance.

Imports abind, foreach, ggplot2, Hmisc, kernlab, magrittr, methods, party, polyspline, recipes (>= 0.1.4), rsample, Rsolnp, survival, utils

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

License GPL-3

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

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

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2019-02-15 16:20:03 UTC

R topics documented:

MachineShop-package	3
.	5

AdaBagModel	6
AdaBoostModel	7
BARTMachineModel	8
BARTModel	10
BlackBoostModel	11
C50Model	13
calibration	14
CForestModel	15
confusion	16
CoxModel	17
dependence	18
diff	19
EarthModel	20
expand.model	21
FDAModel	22
fit	23
GAMBoostModel	24
GBMModel	25
GLMBoostModel	26
GLMModel	28
GLMNetModel	29
Grid	30
KNNModel	31
LARSModel	32
LDAModel	33
lift	34
LMMModel	35
MDAModel	35
metricinfo	37
metrics	37
MLControl	39
MLMetric	41
MLModel	42
ModelFrame	44
modelinfo	45
NaiveBayesModel	46
NNetModel	46
performance	48
performance_curve	49
plot	50
PLSModel	52
POLRModel	53
predict	54
QDAModel	55
RandomForestModel	56
RangerModel	57
resample	58
response	60

RPartModel	60
StackedModel	62
summary	63
SuperModel	64
SurvMatrix	65
SurvRegModel	66
SVMMModel	67
t.test	69
TreeModel	70
tune	71
varimp	73
XGBModel	74

Index	76
--------------	-----------

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

Description

Meta-package for statistical and machine learning with a common interface for model fitting, prediction, performance assessment, and presentation of results. Supports predictive modeling of numerical, categorical, and censored time-to-event outcomes and resample (bootstrap and cross-validation) estimation of model performance.

Details

MachineShop provides a unified interface to machine learning and statistical models provided by other packages. Supported models are summarized in the table below according to the types of response variables with which each can be used. Additional model information can be obtained with the `modelinfo` function.

Model Objects	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
EarthModel	f	n	
FDAModel	f		
GAMBoostModel	b	n	S
GBMModel	f	n	S
GLMBoostModel	b	n	S
GLMModel	b	n	
GLMNetModel	f	m,n	S

KNNModel	f,o	n	
LARSModel		n	
LDAModel	f		
LMMModel	f	m,n	
MDAModel	f		
NaiveBayesModel	f		
NNetModel	f	n	
PDAModel	f		
PLSModel	f	n	
POLRModel	o		
QDAModel	f		
RandomForestModel	f	n	
RangerModel	f	n	S
RPartModel	f	n	S
StackedModel	f,o	m,n	S
SuperModel	f,o	m,n	S
SurvRegModel			S
SVMModel	f	n	
TreeModel	f	n	
XGBModel	f	n	

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

The following set of standard model training, prediction, performance assessment, and tuning functions are available for the model objects.

Training:

<code>fit</code>	Model Fitting
<code>resample</code>	Resample Estimation of Model Performance
<code>tune</code>	Model Tuning and Selection

Prediction:

<code>predict</code>	Model Prediction
----------------------	------------------

Performance Assessment:

<code>calibration</code>	Model Calibration
<code>confusion</code>	Confusion Matrix
<code>dependence</code>	Parital Dependence
<code>diff</code>	Model Performance Differences
<code>lift</code>	Lift Curves
<code>performance</code>	Model Performance Metrics
<code>performance_curve</code>	Model Performance Curves

varimp Variable Importance

Methods for resample estimation include

<code>BootControl</code>	Simple Bootstrap
<code>CVControl</code>	Repeated K-Fold Cross-Validation
<code>OOBControl</code>	Out-of-Bootstrap
<code>SplitControl</code>	Split Training-Testing
<code>TrainControl</code>	Training Resubstitution

Tabular and graphical summaries of modeling results can be obtained with

`summary`
`plot`

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

Author(s)

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

See Also

Useful links:

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

Quote Operator

Description

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

Usage

```
.(expr)
```

Arguments

`expr` any syntactically valid R expression.

Value

The quoted (unevaluated) expression.

See Also

[quote](#)

Examples

```
## Stepwise variable selection with BIC
library(MASS)

glmfit <- fit(medv ~ ., Boston, GLMStepAICModel(k = .(log(nobs))))
varimp(glmfit)
```

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

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

Details**Response Types:** factor**Automatic Tuning Grid Parameters:** mfinal, maxdepth

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

Value

MLModel class object.

See Also[bagging](#), [fit](#), [resample](#), [tune](#)**Examples**

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

 AdaBoostModel

Boosting with Classification Trees

Description

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

Usage

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

Arguments

boos	if TRUE, then bootstrap samples are drawn from the training set using the observation weights at each iteration. If FALSE, then all observations are used with their weights.
mfinal	number of iterations for which boosting is run.
coeflearn	learning algorithm.
minsplit	minimum number of observations that must exist in a node in order for a split to be attempted.
minbucket	minimum number of observations in any terminal node.
cp	complexity parameter.
maxcompete	number of competitor splits retained in the output.

maxsurrogate	number of surrogate splits retained in the output.
usesurrogate	how to use surrogates in the splitting process.
xval	number of cross-validations.
surrogatestyle	controls the selection of a best surrogate.
maxdepth	maximum depth of any node of the final tree, with the root node counted as depth 0.

Details

Response Types: factor

Automatic Tuning Grid Parameters: mfinal, maxdepth, coeflearn*

* included only in randomly sampled grid points

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

Value

MLModel class object.

See Also

[boosting](#), [fit](#), [resample](#), [tune](#)

Examples

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

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 X^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, numeric

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

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

In calls to [varimp](#) for BARTMachineModel, argument `metric` may be 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](#), [tune](#)

Examples

```
library(MASS)

modelfit <- fit(medv ~ ., data = Boston, model = BARTMachineModel())
varimp(modelfit, metric = "splits", num_replicates = 20, scale = FALSE)
```

BARTModel

*Bayesian Additive Regression Trees Model***Description**

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

Usage

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

Arguments

K	if provided, then coarsen the times of survival responses per the quantiles $1/K, 2/K, \dots, K/K$ 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	<i>theta</i> and <i>omega</i> parameters; zero means random.
a, b	sparse parameters for $Beta(a, b)$ prior: $0.5 \leq a \leq 1$ 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 <i>tau</i> definition, i.e., $\tau = \tau.num / (k * \sqrt{ntree})$.
offset	override for the default <i>offset</i> of $F^{-1}(\text{mean}(y))$ in the multivariate response probability $P(y[j] = 1 x) = F(f(x)[j] + \text{offset}[j])$.

nree	number of trees in the sum.
numcut	number of possible covariate cutoff values.
ndpost	number of posterior draws returned.
nskip	number of MCMC iterations to be treated as burn in.
keepevery	interval at which to keep posterior draws.
printevery	interval at which to print MCMC progress.

Details

Response Types: factor, numeric, Surv

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

Value

MLModel class object.

See Also

[gbart](#), [mbart](#), [surv.bart](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

modelfit <- fit(medv ~ ., data = Boston, 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	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 indicating 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, numeric, Surv

Automatic Tuning Grid Parameters: mstop, maxdepth

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

Value

MLModel class object.

See Also

[blackboost](#), [Family](#), [ctree_control](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

C50Model

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

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

Usage

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

Arguments

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

Details

Response Types: factor

Automatic Tuning Grid Parameters: trials, rules, winnow

Latter arguments are passed to [C5.0Control](#). Further model details can be found in the source link below.

In calls to `varimp` for C50Model, argument `metric` may be 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](#), [tune](#)

Examples

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

calibration

Model Calibration

Description

Calculate calibration estimates from observed and predicted responses.

Usage

```
Calibration(...)
```

```
calibration(x, y = NULL, breaks = 10, ...)
```

Arguments

...	named or unnamed calibration output to combine together with the Calibration constructor.
x	observed responses or Resamples object of observed and predicted responses.
y	predicted responses.
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 survival responses and loess for others.

Value

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

See Also

[response](#), [predict](#), [resample](#), [plot](#)

Examples

```

library(survival)
library(MASS)

res <- resample(Surv(time, status != 2) ~ sex + age + year + thickness + ulcer,
               data = Melanoma, model = GBMModel,
               control = CVControl(times = 365 * c(2, 5, 10)))
cal <- calibration(res)
plot(cal)

```

CForestModel

Conditional Random Forest Model

Description

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

Usage

```

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

```

Arguments

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

Details

Response Types: factor, numeric, Surv

Automatic Tuning Grid Parameters: mtry

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

Value

MLModel class object.

See Also

[cforest](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

confusion

Confusion Matrix

Description

Calculate confusion matrices of predicted and observed responses.

Usage

```
Confusion(...)

confusion(x, y = NULL, cutoff = 0.5, ...)
```

Arguments

...	named or unnamed confusion output to combine together with the Confusion constructor.
x	factor of observed responses or Resamples object of observed and predicted responses.
y	predicted responses.
cutoff	threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified. If NULL, then binary responses are summed directly over predicted class probabilities, whereas a default cutoff of 0.5 is used for survival probabilities. Class probability summations and survival will appear as decimal numbers that can be interpreted as expected counts.

Value

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

See Also

[response](#), [predict](#), [resample](#), [plot](#), [summary](#)

Examples

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

CoxModel

*Proportional Hazards Regression Model***Description**

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

Usage

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

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

Arguments

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

Details

Response Types: Surv

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

Value

MLModel class object.

See Also

[cph](#), [coxph](#), [coxph.control](#), [stepAIC](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(survival)
library(MASS)

fit(Surv(time, status != 2) ~ sex + age + year + thickness + ulcer,
    data = Melanoma, model = CoxModel())
```

dependence

Partial Dependence

Description

Calculate partial dependence of a response on select predictor variables.

Usage

```
dependence(object, data = NULL, select = NULL, interaction = FALSE,
           n = 10, intervals = c("uniform", "quantile"), stats = c(Mean =
           base::mean))
```

Arguments

object	MLModelFit object.
data	data.frame containing all predictor variables. If not specified, the training data will be used by default.
select	expression indicating predictor variables for which to compute partial dependence (see subset for syntax) [default: all].
interaction	logical indicating whether to calculate dependence on the interacted predictors.
n	number of predictor values at which to perform calculations.
intervals	character string specifying whether the n values are spaced uniformly ("uniform") or according to variable quantiles ("quantile").
stats	function, one or more function names, or list of named functions with which to aggregate the response variable over the non-selected predictor variables.

Value

PartialDependence class object that inherits from data.frame.

See Also[fit](#), [plot](#)**Examples**

```
gbmfit <- fit(Species ~ ., data = iris, model = GBMModel)
(pd <- dependence(gbmfit, 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 'Performance'
diff(x, ...)

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

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

Arguments

x object containing resampled metrics.
... arguments to be passed to other methods.

Value

PerformanceDiff class object that inherits from Performance.

See Also[performance](#), [resample](#), [tune](#), [plot](#), [summary](#), [t.test](#)

Examples

```

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

fo <- Surv(time, status != 2) ~ sex + age + year + thickness + ulcer
control <- CVControl()

gbmres1 <- resample(fo, Melanoma, GBMModel(n.trees = 25), control)
gbmres2 <- resample(fo, Melanoma, GBMModel(n.trees = 50), control)
gbmres3 <- resample(fo, Melanoma, GBMModel(n.trees = 100), control)

res <- Resamples(GBM1 = gbmres1, GBM2 = gbmres2, GBM3 = gbmres3)
perfdiff <- diff(res)
summary(perfdiff)
plot(perfdiff)

```

EarthModel

Multivariate Adaptive Regression Splines Model

Description

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

Usage

```

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

```

Arguments

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

Details

Response Types: factor, numeric

Automatic Tuning Grid Parameters: nprune, degree*

* included only in randomly sampled grid points

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

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

Value

MModel class object.

See Also

[earth](#), [fit](#), [resample](#), [tune](#)

Examples

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

expand.model

Model Expansion Over a Grid of Tuning Parameters

Description

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

Usage

```
expand.model(x, ...)
```

Arguments

`x` MModel function, function name, or object.
`...` vectors, factors, or a list containing the parameter values.

Value

A list of MModel objects created from the parameter combinations.

See Also

[modelinfo](#), [tune](#)

Examples

```
expand.model(GBMModel, n.trees = c(25, 50, 100),
             interaction.depth = 1:3,
             n.minobsinnode = c(5, 10))
```

 FDAModel

Flexible and Penalized Discriminant Analysis Models

Description

Performs flexible discriminant analysis.

Usage

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

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

Arguments

theta	optional matrix of class scores, typically with number of columns less than one minus the number of classes.
dimension	dimension of the discriminant subspace, less than the number of classes, to use for prediction.
eps	numeric threshold for small singular values for excluding discriminant variables.
method	regression function used in optimal scaling. The default of linear regression is provided by polyreg from the mda package. For penalized discriminant analysis, gen.ridge is appropriate. Other possibilities are mars for multivariate adaptive regression splines and bruto for adaptive backfitting of additive splines. Use the <code>.</code> 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 Grid Parameters • FDAModel: nprune, degree*

- PDAModel: lambda

* included only in randomly sampled grid points

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

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

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

Value

MLModel class object.

See Also

[fda](#), [predict.fda](#), [fit](#), [resample](#), [tune](#)

Examples

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

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

fit

Model Fitting

Description

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

Usage

```
fit(x, ...)
```

```
## S3 method for class 'formula'
```

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

```
## S3 method for class 'ModelFrame'
```

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

```
## S3 method for class 'recipe'
```

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

Arguments

x	defined relationship between model predictors and an outcome. May be a <code>ModelFrame</code> containing a formula, data, and optionally case weights; a formula; or a recipe.
...	arguments passed to other methods.
data	<code>data.frame</code> containing observed predictors and outcomes.
model	<code>MLModel</code> object, constructor function, or character string naming a constructor function that returns an <code>MLModel</code> object.

Details

User-specified case weights may be specified for [ModelFrames](#) upon creation with the `weights` argument in its constructor.

Variables in a recipe may be used as case weights by defining a "case_weight" [role](#) for them.

Value

`MLModelFit` class object.

See Also

[ModelFrame](#), [recipe](#), [modelinfo](#), [tune](#), [predict](#), [varimp](#)

Examples

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

gbmfit <- fit(Surv(time, status != 2) ~ sex + age + year + thickness + ulcer,
             data = Melanoma, model = GBMModel)
varimp(gbmfit)
```

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	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	global 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 indicating 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, numeric, Surv

Automatic Tuning Grid Parameters: mstop

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

Value

MLModel class object.

See Also

[gamboost](#), [Family](#), [baselearners](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

 GBMModel

Generalized Boosted Regression Model

Description

Fits generalized boosted regression models.

Usage

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

Arguments

distribution	either a character string specifying the name of the distribution to use or a 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, Surv

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

* included only in randomly sampled grid points

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

Value

MLModel class object.

See Also

[gbm](#), [fit](#), [resample](#), [tune](#)

Examples

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

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	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 indicating 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, numeric, Surv

Automatic Tuning Grid Parameters: mstop

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

Value

MLModel class object.

See Also

[glmboost](#), [Family](#), [fit](#), [resample](#), [tune](#)

Examples

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

 GLMModel

Generalized Linear Model

Description

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

Usage

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

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

Arguments

family	description of the error distribution and link function to be used in the model. Set automatically according to the class type of the response variable.
...	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: $k = \log(\text{nobs})$ 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: binary factor, numeric

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

Value

MLModel class object.

See Also

[glm](#), [glm.control](#), [stepAIC](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

GLMNetModel

*GLM Lasso or Elasticnet Model***Description**

Fit a generalized linear model via penalized maximum likelihood.

Usage

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

Arguments

family	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 gaussian models.
type.logistic	algorithm type for logistic models.
type.multinomial	algorithm type for multinomial models.

Details

Response Types: factor, matrix, numeric, Surv

Automatic Tuning Grid Parameters: lambda, alpha

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

Value

MLModel class object.

See Also

[glmnet](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

fit(medv ~ ., data = Boston, model = GLMNetModel(lambda = 0.01))
```

Grid

Tuning Grid Control

Description

Defines the control parameters for a tuning grid.

Usage

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

Arguments

length	number of values to be generated for each model parameter in the tuning grid.
random	number of points to be randomly sampled from the tuning grid or FALSE if all points are to be used.

See Also

[tune](#)

`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

<code>k</code>	numer of neighbors considered.
<code>distance</code>	Minkowski distance parameter.
<code>scale</code>	logical indicating whether to scale predictors to have equal standard deviations.
<code>kernel</code>	kernel to use.

Details

Response Types: factor, numeric, ordinal

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

* included only in randomly sampled grid points

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

Value

MLModel class object.

See Also

[kkn](#), [fit](#), [resample](#), [tune](#)

Examples

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

LARSMoDel	<i>Least Angle Regression, Lasso and Infinitesimal Forward Stagewise Models</i>
-----------	---

Description

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

Usage

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

Arguments

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

Details

Response Types: numeric

Automatic Tuning Grid Parameters: step

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

Value

MLModel class object.

See Also

[lars](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

LDAModel

*Linear Discriminant Analysis Model***Description**

Performs linear discriminant analysis.

Usage

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

Arguments

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

Details

Response Types: factor

Automatic Tuning Grid Parameters: dimen

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

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

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

Value

MLModel class object.

See Also

[lda](#), [predict.lda](#), [fit](#), [resample](#), [tune](#)

Examples

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

lift	<i>Model Lift</i>
------	-------------------

Description

Calculate lift estimates from observed and predicted responses.

Usage

```
Lift(...)
```

```
lift(x, y = NULL, ...)
```

Arguments

...	named or unnamed lift output to combine together with the Lift constructor.
x	observed responses or Resamples object of observed and predicted responses.
y	predicted responses.

Value

Lift class object that inherits from Curves.

See Also

[response](#), [predict](#), [resample](#), [plot](#)

Examples

```
library(MASS)

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

`LMModel`*Linear Models*

Description

Fits linear models.

Usage

```
LMModel()
```

Details

Response Types: factor, matrix, numeric

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

Value

LMModel class object.

See Also

[lm](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

`MDAModel`*Mixture Discriminant Analysis Model*

Description

Performs mixture discriminant analysis.

Usage

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

Arguments

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

Details

Response Types: factor

Automatic Tuning Grid Parameters: subclasses

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

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

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

Value

MLModel class object.

See Also

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

Examples

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

`metricinfo`*Display Performance Metric Information*

Description

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

Usage

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

Arguments

... one or more metric functions, function names, observed response, observed and predicted responses, or a Resamples object. If none are specified, information is returned on all available metrics by default.

Value

List of named metrics containing a descriptive "label", whether to "maximize" the metric for better performance, the function "arguments", and supported response variable "types" for each.

See Also

[metrics](#), [resample](#)

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)))
```

`metrics`*Performance Metrics*

Description

Compute measures of agreement between observed and predicted responses.

Usage

```
accuracy(observed, predicted = NULL, cutoff = 0.5, ...)  
auc(observed, predicted = NULL, metrics = c(MachineShop::tpr,  
      MachineShop::fpr), stat = base::mean, ...)  
brier(observed, predicted = NULL, ...)  
cindex(observed, predicted = NULL, ...)  
cross_entropy(observed, predicted = NULL, ...)  
f_score(observed, predicted = NULL, cutoff = 0.5, beta = 1, ...)  
fnr(observed, predicted = NULL, cutoff = 0.5, ...)  
fpr(observed, predicted = NULL, cutoff = 0.5, ...)  
kappa2(observed, predicted = NULL, cutoff = 0.5, ...)  
npv(observed, predicted = NULL, cutoff = 0.5, ...)  
ppv(observed, predicted = NULL, cutoff = 0.5, ...)  
pr_auc(observed, predicted = NULL, ...)  
precision(observed, predicted = NULL, cutoff = 0.5, ...)  
recall(observed, predicted = NULL, cutoff = 0.5, ...)  
roc_auc(observed, predicted = NULL, ...)  
roc_index(observed, predicted = NULL, cutoff = 0.5,  
  f = function(sensitivity, specificity) (sensitivity + specificity)/2,  
  ...)  
rpp(observed, predicted = NULL, cutoff = 0.5, ...)  
sensitivity(observed, predicted = NULL, cutoff = 0.5, ...)  
specificity(observed, predicted = NULL, cutoff = 0.5, ...)  
tnr(observed, predicted = NULL, cutoff = 0.5, ...)  
tpr(observed, predicted = NULL, cutoff = 0.5, ...)  
weighted_kappa2(observed, predicted = NULL, power = 1, ...)
```

```

gini(observed, predicted = NULL, ...)
mae(observed, predicted = NULL, ...)
mse(observed, predicted = NULL, ...)
msle(observed, predicted = NULL, ...)
r2(observed, predicted = NULL, ...)
rmse(observed, predicted = NULL, ...)
rmsle(observed, predicted = NULL, ...)

```

Arguments

observed	observed responses, Curves object, or ConfusionMatrix of observed and predicted responses.
predicted	predicted responses.
cutoff	threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified.
...	arguments passed to or from other methods.
metrics	list of two performance metrics for the calculation [default: ROC metrics].
stat	function to compute a summary statistic at each cutoff value of resampled metrics in Curves , or NULL for resample-specific metrics.
beta	relative importance of recall to precision in the calculation of <code>f_score</code> [default: F1 score].
f	function to calculate a desired sensitivity-specificity tradeoff.
power	power to which positional distances of off-diagonals from the main diagonal in confusion matrices are raised to calculate <code>weighted_kappa2</code> .

See Also

[metricinfo](#), [confusion](#), [performance](#), [performance_curve](#)

MLControl

Resampling Controls

Description

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

`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.

CVControl constructs an MLControl object for repeated K-fold cross-validation. 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.

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.

TrainControl constructs an MLControl object for training and performance evaluation to be performed on the same training set.

Usage

```
MLControl(times = numeric(), seed = NULL, ...)
```

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

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

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

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

```
TrainControl(...)
```

Arguments

times	numeric vector of follow-up times at which to predict survival probabilities.
seed	integer to set the seed at the start of resampling. This is set to a random integer by default (NULL).
...	arguments to be passed to MLControl.
samples	number of bootstrap samples.
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 ($0 < \text{prop} < 1$).

Value

MLControl class object.

See Also

[resample](#)

Examples

```
## 100 bootstrap samples
BootControl(samples = 100)

## 5 repeats of 10-fold cross-validation
CVControl(folds = 10, repeats = 5)

## 100 out-of-bootstrap samples
OOBControl(samples = 100)

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

## Same training and test set
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. Must be defined to accept observed and predicted as the first two arguments and with an ellipsis (. . .) to accommodate others.
name	character string name for the instantiated MLMetric object; same as the metric function name.
label	descriptive label for the metric.
maximize	logical indicating whether to maximize the metric for better performance.
value	list of arguments to pass to the MLMetric constructor.

Value

MLMetric class object.

See Also

[metrics](#), [metricinfo](#)

Examples

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

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

MLModel*MLModel Class Constructor*

Description

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

Usage

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

Arguments

name	character string name for the instantiated MLModel object; same name as the object to which the model is assigned.
label	descriptive label for the model.
packages	character vector of packages whose namespaces are required by the model.
types	character vector of response variable types to which the model can be fit. Supported types are "binary", "factor", "matrix", "numeric", "ordered", and "Surv".
params	list of user-specified model parameters to be passed to the fit function.
grid	tuning grid function whose first argument x is a ModelFrame of the model fit data and formula, followed by a length to use in generating sequences of parameter values, a number of grid points to sample at random, and an ellipsis (...).
design	character string indicating whether the type of design matrix used to fit the model is a "model.matrix", a data.frame of the original predictor variable "terms", or unknown (default).
fit	model fitting function whose arguments are a formula, a data frame, case weights, and an ellipsis.

predict	model prediction function whose arguments are the object returned by fit, a newdata frame of predictor variables, optional vector of times at which to predict survival, and an ellipsis.
varimp	variable importance function whose arguments are the object returned by fit, optional arguments passed from calls to varimp , and an ellipsis.
...	arguments passed from other methods.

Details

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

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

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

matrix a matrix of predicted responses.

numeric a vector or column matrix of predicted responses.

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

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

Value

MLModel class object.

See Also

[modelinfo](#), [fit](#), [resample](#), [tune](#)

Examples

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

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

ModelFrame

ModelFrame Class

Description

Class for storing a data frame, formula, and optionally weights for fitting MLModels.

Usage

```
ModelFrame(x, ...)

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

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

Arguments

<code>x</code>	model formula or matrix of predictor variables.
<code>...</code>	arguments passed to other methods.
<code>data</code>	<code>data.frame</code> or an object that can be converted to one.
<code>weights</code>	vector of case weights.
<code>strata</code>	vector of stratification levels.
<code>na.action</code>	action to take if cases contain missing values. The default is first any <code>na.action</code> attribute of <code>data</code> , second a <code>na.action</code> setting of options , and third <code>na.fail</code> if unset.
<code>y</code>	response variable.

Value

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

See Also

[formula](#), [na.fail](#), [na.omit](#), [na.pass](#)

Examples

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

modelinfo

Display Model Information

Description

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

Usage

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

Arguments

... MLModel objects, constructor functions, constructor function names, or supported responses for which to display information. If none are specified, information is returned on all available models by default.

Value

List of named models containing a descriptive "label", source "packages", supported response variable "types", the constructor "arguments", whether there is a pre-defined "grid" of tuning parameters, and whether a "varimp" function is implemented for each.

See Also

[fit](#), [resample](#), [tune](#)

Examples

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

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](#), [tune](#)

Examples

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

NNetModel *Neural Network Model*

Description

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

Usage

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

Arguments

size	number of units in the hidden layer.
linout	switch for linear output units.
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 Grid Parameters: size, decay

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

Value

MModel class object.

See Also

[nnet](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

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

performance

*Model Performance Metrics***Description**

Compute measures of model performance.

Usage

```
performance(x, ...)

## S3 method for class 'Resamples'
performance(x, ..., na.rm = TRUE)

## S3 method for class 'factor'
performance(x, y, metrics = c(Brier =
  MachineShop::brier, Accuracy = MachineShop::accuracy, Kappa =
  MachineShop::kappa2, `Weighted Kappa` = MachineShop::weighted_kappa2,
  ROCAUC = MachineShop::roc_auc, Sensitivity = MachineShop::sensitivity,
  Specificity = MachineShop::specificity), cutoff = 0.5, ...)

## S3 method for class 'matrix'
performance(x, y, metrics = c(RMSE = MachineShop::rmse,
  R2 = MachineShop::r2, MAE = MachineShop::mae), ...)

## S3 method for class 'numeric'
performance(x, y, metrics = c(RMSE = MachineShop::rmse,
  R2 = MachineShop::r2, MAE = MachineShop::mae), ...)

## S3 method for class 'Surv'
performance(x, y, metrics = c(CIndex =
  MachineShop::cindex, Brier = MachineShop::brier, ROCAUC =
  MachineShop::roc_auc, Accuracy = MachineShop::accuracy), cutoff = 0.5,
  ...)

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

## S3 method for class 'ConfusionMatrix'
performance(x, metrics = c(Accuracy =
  MachineShop::accuracy, Kappa = MachineShop::kappa2), ...)
```

Arguments

`x` observed responses or class containing observed and predicted responses.
`...` arguments passed from the Resamples method to the response type-specific methods or from the method for Confusion to ConfusionMatrix.

na.rm	logical indicating whether to remove observed or predicted responses that are NA when calculating metrics.
y	predicted responses.
metrics	function, one or more function names, or list of named functions to include in the calculation of performance metrics.
cutoff	threshold above which binary factor probabilities are classified as events and below which survival probabilities are classified.

See Also

[response](#), [predict](#), [resample](#), [confusion](#), [metrics](#), [plot](#), [summary](#)

Examples

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

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

fo <- Surv(time, status != 2) ~ sex + age + year + thickness + ulcer
gbmfit <- fit(fo, data = Melanoma, model = GBMModel)

obs <- response(fo, data = Melanoma)
pred <- predict(gbmfit, newdata = Melanoma, type = "prob")
performance(obs, pred)
```

performance_curve *Performance Curves*

Description

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

```
Curves(...)

performance_curve(x, ...)

## S3 method for class 'Resamples'
performance_curve(x, metrics = c(MachineShop::tpr,
```

```
MachineShop::fpr), na.rm = TRUE, ...)

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

Arguments

...	named or unnamed performance_curve output to combine together with the Curves constructor.
x	observed responses or Resamples object of observed and predicted responses.
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.
y	predicted responses.

Value

Curves class object that inherits from data.frame.

See Also

[response](#), [predict](#), [resample](#), [metrics](#), [auc](#), [plot](#), [summary](#)

Examples

```
library(MASS)

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

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

plot

Model Performance Plots

Description

Plot measures of model performance and predictor variable importance.

Usage

```

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

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

## S3 method for class 'MLModelTune'
plot(x, metrics = NULL, stat = base::mean,
     type = c("boxplot", "density", "errorbar", "line", "violin"), ...)

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

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

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

## S3 method for class 'Curves'
plot(x, type = c("tradeoffs", "cutoffs"),
     diagonal = FALSE, stat = base::mean, ...)

## S3 method for class 'Lift'
plot(x, find = NULL, diagonal = TRUE,
     stat = base::mean, ...)

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

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

```

Arguments

x	object to plot.
metrics	vector of numeric indexes or character names of performance metrics to plot.
stat	function to compute a summary statistic on resampled values for MLModelTune line plots and Resamples model sorting. For Curves and Lift classes, plots are of resampled metrics aggregated by the statistic if given or of resample-specific metrics if NULL. Curves, or NULL for resample-specific metrics.
type	type of plot to construct.
...	arguments passed to other methods.

<code>se</code>	logical indicating whether to include standard error bars.
<code>diagonal</code>	logical indicating whether to include a diagonal reference line.
<code>find</code>	numeric true positive rate at which to display reference lines identifying the corresponding rates of positive predictions.
<code>stats</code>	vector of numeric indexes or character names of partial dependence summary statistics to plot.
<code>n</code>	number of most important variables to include in the plot [default: all].

See Also

[performance](#), [resample](#), [diff](#), [tune](#), [calibration](#), [confusion](#), [lift](#), [dependence](#), [varimp](#)

Examples

```
## Factor response example

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

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

gbmres1 <- resample(fo, iris, GBMModel(n.trees = 25), control)
gbmres2 <- resample(fo, iris, GBMModel(n.trees = 50), control)
gbmres3 <- resample(fo, iris, GBMModel(n.trees = 100), control)
plot(gbmres3)

res <- Resamples(GBM1 = gbmres1, GBM2 = gbmres2, GBM3 = gbmres3)
plot(res)
```

 PLSModel

Partial Least Squares Model

Description

Function to perform partial least squares regression.

Usage

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

Arguments

<code>ncomp</code>	number of components to include in the model.
<code>scale</code>	logical indicating whether to scale the predictors by the sample standard deviation.

Details

Response Types: factor, numeric

Automatic Tuning Grid Parameters: ncomp

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

Value

MLModel class object.

See Also

[mvr](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

fit(medv ~ ., data = Boston, 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.

Value

MLModel class object.

See Also

[polr](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

df <- Boston
df$medv <- cut(Boston$medv, breaks = c(0, 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", "prob"), cutoff = 0.5, ...)
```

Arguments

object	MLModelFit object from a model fit.
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.
type	specifies prediction on the original outcome scale ("response") or on a probability distribution scale ("prob").
cutoff	threshold above which binary factor probabilities are classified as events, below which survival probabilities are classified, and at which expected values are rounded for integer outcomes.
...	arguments passed to model-specific prediction functions.

See Also

[fit](#), [confusion](#), [performance](#)

Examples

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

gbmfit <- fit(Surv(time, status != 2) ~ sex + age + year + thickness + ulcer,
             data = Melanoma, model = GBMModel)
predict(gbmfit, newdata = Melanoma, times = 365 * c(2, 5, 10), type = "prob")
```

QDAModel

Quadratic Discriminant Analysis Model

Description

Performs quadratic discriminant analysis.

Usage

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

Arguments

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

Details

Response Types: factor

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

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

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

Value

MLModel class object.

See Also

[qda](#), [predict.qda](#), [fit](#), [resample](#), [tune](#)

Examples

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

RandomForestModel	<i>Random Forest Model</i>
-------------------	----------------------------

Description

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

Usage

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

Arguments

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

Details

Response Types: factor, numeric

Automatic Tuning Grid Parameters: mtry, nodesize*

* included only in randomly sampled grid points

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

Value

MLModel class object.

See Also

[randomForest](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)
```

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

RangerModel

Fast Random Forest Model

Description

Fast implementation of random forests or recursive partitioning.

Usage

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

Arguments

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

Details

Response Types: factor, numeric, Surv

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

* included only in randomly sampled grid points

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

Value

MLModel class object.

See Also

[ranger](#), [fit](#), [resample](#), [tune](#)

Examples

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

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

```
Resamples(...)
```

```
resample(x, ...)
```

```
## S3 method for class 'formula'  
resample(x, data, model, control = CVControl, ...)
```

```
## S3 method for class 'ModelFrame'  
resample(x, model, control = CVControl, ...)
```

```
## S3 method for class 'recipe'  
resample(x, model, control = CVControl, ...)
```

Arguments

...	named or unnamed resample output to combine together with the Resamples constructor.
x	defined relationship between model predictors and an outcome. May be a <code>ModelFrame</code> containing a formula, data, and optionally case weights; a formula; or a recipe.
data	<code>data.frame</code> containing observed predictors and outcomes.
model	<code>MLModel</code> object, constructor function, or character string naming a constructor function that returns an <code>MLModel</code> object.
control	<code>MLControl</code> object, control function, or character string naming a control function defining the resampling method to be employed.

Details

Output being combined from more than one model with the `Resamples` constructor must have been generated with the same resampling control object.

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

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

Variables in a recipe may be used for stratification by defining a "case_strata" `role` for them. Resampling will be unstratified if no variables have that role.

Value

`Resamples` class object.

See Also

[ModelFrame](#), [recipe](#), [modelinfo](#), [MLControl](#), [performance](#), [metricinfo](#), [plot](#), [summary](#)

Examples

```
## Factor response example

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

gbmres1 <- resample(fo, iris, GBMModel(n.trees = 25), control)
gbmres2 <- resample(fo, iris, GBMModel(n.trees = 50), control)
gbmres3 <- resample(fo, iris, GBMModel(n.trees = 100), control)

summary(gbmres1)
plot(gbmres1)

res <- Resamples(GBM1 = gbmres1, GBM2 = gbmres2, GBM3 = gbmres3)
summary(res)
plot(res)
```

response	<i>Extract Response Variable</i>
----------	----------------------------------

Description

Extract the response variable from an object.

Usage

```
response(object, ...)  
  
## S3 method for class 'formula'  
response(object, data, ...)  
  
## S3 method for class 'recipe'  
response(object, data, ...)
```

Arguments

object	object containing response variable values.
...	arguments passed to other methods.
data	data.frame containing the values of a response variable defined in a formula.

See Also

[recipe](#)

Examples

```
## Survival response example  
library(survival)  
library(MASS)  
  
fo <- Surv(time, status != 2) ~ sex + age + year + thickness + ulcer  
response(fo, data = Melanoma)
```

RPartModel	<i>Recursive Partitioning and Regression Tree Models</i>
------------	--

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 Grid Parameters: cp

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

Value

MModel class object.

See Also

[rpart](#), [fit](#), [resample](#), [tune](#)

Examples

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

`StackedModel`*Stacked Regression Model*

Description

Fit a stacked regression model from multiple base learners.

Usage

```
StackedModel(..., control = CVControl, weights = NULL)
```

Arguments

<code>...</code>	MLModel objects to serve as base learners.
<code>control</code>	MLControl object, control function, or character string naming a control function defining the resampling method to be employed for the estimation of base learner weights.
<code>weights</code>	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](#), [tune](#)

Examples

```
library(MASS)

model <- StackedModel(GBMModel, SVMRadialModel, GLMNetModel(lambda = 0.01))
modelfit <- fit(medv ~ ., data = Boston, model = model)
predict(modelfit, newdata = Boston)
```

summary

*Model Performance Summary***Description**

Summary statistics for resampled model performance metrics.

Usage

```
## S3 method for class 'Performance'
summary(object, stats = c(Mean = base::mean, Median
  = stats::median, SD = stats::sd, Min = base::min, Max = base::max),
  na.rm = TRUE, ...)
```

```
## S3 method for class 'Resamples'
summary(object, stats = c(Mean = base::mean, Median =
  stats::median, SD = stats::sd, Min = base::min, Max = base::max),
  na.rm = TRUE, ...)
```

```
## S3 method for class 'MLModelTune'
summary(object, stats = c(Mean = base::mean, Median
  = stats::median, SD = stats::sd, Min = base::min, Max = base::max),
  na.rm = TRUE, ...)
```

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

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

```
## S3 method for class 'Curves'
summary(object, stat = base::mean, ...)
```

Arguments

object	object to summarize.
stats	function, one or more function names, or list of named functions to include in the calculation of summary statistics.
na.rm	logical indicating whether to exclude missing values.
...	arguments passed to other methods.
stat	function to compute a summary statistic at each cutoff value of resampled metrics in Curves, or NULL for resample-specific metrics.

Value

array with summary statistics in the second dimension, metrics in the first for single models, and models and metrics in the first and third, respectively, for multiple models.

See Also

[performance](#), [resample](#), [diff](#), [tune](#), [confusion](#)

Examples

```
## Factor response example

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

gbmres1 <- resample(fo, iris, GBMModel(n.trees = 25), control)
gbmres2 <- resample(fo, iris, GBMModel(n.trees = 50), control)
gbmres3 <- resample(fo, iris, GBMModel(n.trees = 100), control)
summary(gbmres3)

res <- Resamples(GBM1 = gbmres1, GBM2 = gbmres2, GBM3 = gbmres3)
summary(res)
```

SuperModel

Super Learner Model

Description

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

Usage

```
SuperModel(..., model = GBMModel, control = CVControl,
  all_vars = FALSE)
```

Arguments

...	MLModel objects to serve as base learners.
model	MLModel object, constructor function, or character string naming a constructor function to serve as the super model.
control	MLControl object, control function, or character string naming a control function defining the resampling method to be employed for the estimation of base learner weights.
all_vars	logical indicating whether to include the original predictor variables in the super model.

Details

Response Types: factor, numeric, ordered, Surv

Value

SuperModel class object that inherits from MLModel.

References

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

See Also

[fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

model <- SuperModel(GBMModel, SVMRadialModel, GLMNetModel(lambda = 0.01))
modelfit <- fit(medv ~ ., data = Boston, model = model)
predict(modelfit, newdata = Boston)
```

SurvMatrix

SurvMatrix Class Constructor

Description

Create an object of predicted survival events or probabilities for use with metrics provided by the **MachineShop** package.

Usage

```
SurvEvents(object = numeric(), times = NULL)
```

```
SurvProbs(object = numeric(), times = NULL)
```

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

Arguments

object	matrix, or object that can be converted to one, of predicted survival events or probabilities with columns and rows representing prediction times and cases, respectively.
times	numeric vector of the survival prediction times.
x	object from which to extract elements.
i, j, ...	indices specifying elements to extract.
drop	logical indicating that the result be returned as a numeric coerced to the lowest dimension possible if TRUE or as a 2-dimensional SurvMatrix object otherwise.

Value

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

See Also

[metrics](#), [predict](#)

SurvRegModel

Parametric Survival Model

Description

Fits the accelerated failure time family of parametric survival models.

Usage

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

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

Arguments

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

Details

Response Types: Surv

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

Value

MLModel class object.

See Also

[psm](#), [survreg](#), [survreg.control](#), [stepAIC](#), [fit](#), [resample](#), [tune](#)
[stepAIC](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(survival)
library(MASS)

fit(Surv(time, status != 2) ~ sex + age + year + thickness + ulcer,
    data = Melanoma, model = SurvRegModel())
```

SVMModel

Support Vector Machine Models

Description

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

Usage

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

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

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

```
SVMLaplaceModel(sigma = NULL, ...)
```

```
SVMLinearModel(...)
```

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

```
SVMRadialModel(sigma = NULL, ...)
```

```
SVMSplineModel(...)
```

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

Arguments

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

Details

Response Types: factor, numeric

Automatic Tuning Grid Parameters • SVMANOVAModel: C, degree

- SVMBesselModel: C, order, degree
- SVMLaplaceModel: C, sigma
- SVMLinearModel: C
- SVMPolyModel: C, degree, scale
- SVMRadialModel: C, sigma

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

Value

MLModel class object.

See Also

[ksvm](#), [fit](#), [resample](#), [tune](#)

Examples

```
library(MASS)

fit(medv ~ ., data = Boston, 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	object containing paired differences between resampled metrics.
adjust	p-value adjustment for multiple statistical comparisons as implemented by p.adjust .
...	arguments passed to other methods.

Value

HTestPerformanceDiff 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.

See Also

[diff](#)

Examples

```
## Numeric response example
library(MASS)

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

gbmres1 <- resample(fo, Boston, GBMModel(n.trees = 25), control)
gbmres2 <- resample(fo, Boston, GBMModel(n.trees = 50), control)
gbmres3 <- resample(fo, Boston, GBMModel(n.trees = 100), control)

res <- Resamples(GBM1 = gbmres1, GBM2 = gbmres2, GBM3 = gbmres3)
perfdiff <- diff(res)
t.test(perfdiff)
```

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"))
```

Arguments

<code>mincut</code>	minimum number of observations to include in either child node.
<code>minsize</code>	smallest allowed node size: a weighted quantity.
<code>mindev</code>	within-node deviance must be at least this times that of the root node for the node to be split.
<code>split</code>	splitting criterion to use.

Details

Response Types: factor, numeric

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

Value

MModel class object.

See Also

[tree](#), [fit](#), [resample](#), [tune](#)

Examples

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

Description

Evaluate a model over a grid of tuning parameters or a list of specified models and select the best one according to resample estimation of predictive performance.

Usage

```
tune(x, ...)
```

```
## S3 method for class 'formula'
```

```
tune(x, data, models, grid = 3, fixed = NULL,
```

```
  control = CVControl, metrics = NULL, stat = base::mean,
```

```
  maximize = TRUE, ...)
```

```
## S3 method for class 'ModelFrame'
```

```
tune(x, models, grid = 3, fixed = NULL,
```

```
  control = CVControl, metrics = NULL, stat = base::mean,
```

```
  maximize = TRUE, ...)
```

```
## S3 method for class 'recipe'
```

```
tune(x, models, grid = 3, fixed = NULL,
```

```
  control = CVControl, metrics = NULL, stat = base::mean,
```

```
  maximize = TRUE, ...)
```

Arguments

x	defined relationship between model predictors and an outcome. May be a <code>ModelFrame</code> containing a formula, data, and optionally case weights; a formula; or a recipe.
...	arguments passed to the metrics functions.
data	<code>data.frame</code> containing observed predictors and outcomes.
models	<code>MLModel</code> function, function name, object or list of the aforementioned elements, such as that returned by expand.model .
grid	<code>data.frame</code> containing parameter values at which to evaluate a single model supplied to <code>models</code> , the number of parameter-specific values to generate automatically if the model has a pre-defined grid, or a call to Grid . Ignored in the case of a list of models.
fixed	list of fixed parameter values to combine with those in <code>grid</code> .
control	MLControl object, control function, or character string naming a control function defining the resampling method to be employed.
metrics	function, one or more function names, or list of named functions to include in the calculation of performance metrics. The default performance metrics are used unless otherwise specified. Model selection is based on the first specified metric.

stat	function to compute a summary statistic on resampled values of the metric for model selection.
maximize	logical indicating whether to select the model having the maximum or minimum value of the performance metric. Set automatically if a package metrics function is explicitly specified for the model selection.

Value

MLModelTune class object that inherits from MLModel.

See Also

[ModelFrame](#), [recipe](#), [modelinfo](#), [expand.model](#), [Grid](#), [MLControl](#), [fit](#), [plot](#), [summary](#)

Examples

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

fo <- medv ~ .

# User-specified grid
(gbmtune1 <- tune(fo, data = Boston, model = GBMModel,
  grid = expand.grid(n.trees = c(25, 50, 100),
    interaction.depth = 1:3,
    n.minobsinnode = c(5, 10)),
  control = CVControl(folds = 10, repeats = 5)))

# Automatically generated grid
(gbmtune2 <- tune(fo, data = Boston, model = GBMModel, grid = 3,
  control = CVControl(folds = 10, repeats = 5)))

# Randomly sampled grid points
(gbmtune3 <- tune(fo, data = Boston, model = GBMModel,
  grid = Grid(length = 1000, random = 10),
  control = CVControl(folds = 10, repeats = 5)))

summary(gbmtune3)
plot(gbmtune3, type = "line")

gbmfit <- fit(fo, data = Boston, model = gbmfit3)
varimp(gbmfit)
```

varimp	<i>Variable Importance</i>
--------	----------------------------

Description

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

Usage

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

Arguments

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

Value

VarImp class object.

See Also

[fit](#), [plot](#)

Examples

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

gbmfit <- fit(Surv(time, status != 2) ~ sex + age + year + thickness + ulcer,
             data = Melanoma, model = GBMModel)
(vi <- varimp(gbmfit))
plot(vi)
```

Description

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

Usage

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

```
XGBDARTModel(objective = NULL, base_score = 0.5, eta = 0.3,
              gamma = 0, max_depth = 6, min_child_weight = 1,
              max_delta_step = 0, subsample = 1, colsample_bytree = 1,
              colsample_bylevel = 1, lambda = 1, alpha = 0,
              tree_method = "auto", sketch_eps = 0.03, scale_pos_weight = 1,
              update = "grow_colmaker,prune", refresh_leaf = 1,
              process_type = "default", grow_policy = "depthwise",
              max_leaves = 0, max_bin = 256, sample_type = "uniform",
              normalize_type = "tree", rate_drop = 0, one_drop = 0,
              skip_drop = 0, ...)
```

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

```
XGBTreeModel(objective = NULL, base_score = 0.5, eta = 0.3,
              gamma = 0, max_depth = 6, min_child_weight = 1,
              max_delta_step = 0, subsample = 1, colsample_bytree = 1,
              colsample_bylevel = 1, lambda = 1, alpha = 0,
              tree_method = "auto", sketch_eps = 0.03, scale_pos_weight = 1,
              update = "grow_colmaker,prune", refresh_leaf = 1,
              process_type = "default", grow_policy = "depthwise",
              max_leaves = 0, max_bin = 256, ...)
```

Arguments

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

objective	character string specifying the learning task and objective. Set automatically according to the class type of the response variable.
base_score	initial numeric prediction score of all instances, global bias.
eta, gamma, max_depth, min_child_weight, max_delta_step, subsample, colsample_bytree, colsample_by	see params reference.
...	arguments to be passed to XGBModel.

Details

Response Types: factor, numeric

Automatic Tuning Grid Parameters

- XGBDARTModel: nrounds, max_depth, eta, gamma*, min_child_weight*, subsample, colsample_bytree, rate_drop, skip_drop

- XGBLinearModel: nrounds, lambda, alpha

- XGBTreeModel: nrounds, max_depth, eta, gamma*, min_child_weight*, subsample, colsample_bytree

* included only in randomly sampled grid points

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

In calls to [varimp](#) for XGBTreeModel, argument `metric` may be 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

MModel class object.

See Also

[xgboost](#), [fit](#), [resample](#), [tune](#)

Examples

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

Index

- ., [5](#), [22](#), [36](#)
- [,SurvMatrix,ANY,ANY,ANY-method (SurvMatrix), [65](#)

- accuracy (metrics), [37](#)
- AdaBagModel, [3](#), [6](#)
- AdaBoostModel, [3](#), [7](#)
- auc, [50](#)
- auc (metrics), [37](#)
- Automatic Tuning, [7–9](#), [12](#), [13](#), [15](#), [21](#), [23](#), [25–27](#), [30–33](#), [36](#), [47](#), [53](#), [56](#), [58](#), [61](#), [68](#), [75](#)

- bagging, [7](#)
- bartMachine, [9](#)
- BARTMachineModel, [3](#), [8](#)
- BARTModel, [3](#), [10](#)
- baselearners, [25](#)
- blackboost, [12](#)
- BlackBoostModel, [3](#), [11](#)
- boosting, [8](#)
- BootControl, [5](#)
- BootControl (MLControl), [39](#)
- brier (metrics), [37](#)
- bruto, [22](#), [36](#)

- C5.0, [14](#)
- C5.0Control, [13](#)
- C50Model, [3](#), [13](#)
- Calibration (calibration), [14](#)
- calibration, [4](#), [14](#), [52](#)
- cforest, [16](#)
- cforest_control, [15](#)
- CForestModel, [3](#), [15](#)
- cindex (metrics), [37](#)
- Confusion (confusion), [16](#)
- confusion, [4](#), [16](#), [39](#), [49](#), [52](#), [54](#), [64](#)
- ConfusionMatrix, [39](#)
- CoxModel, [3](#), [17](#)
- coxph, [18](#)

- coxph.control, [17](#), [18](#)
- CoxStepAICModel (CoxModel), [17](#)
- cph, [18](#)
- cross_entropy (metrics), [37](#)
- ctree_control, [12](#)
- Curves, [39](#)
- Curves (performance_curve), [49](#)
- CVControl, [5](#)
- CVControl (MLControl), [39](#)

- dependence, [4](#), [18](#), [52](#)
- diff, [4](#), [19](#), [52](#), [64](#), [69](#)

- earth, [21](#)
- EarthModel, [3](#), [20](#)
- expand.model, [21](#), [71](#), [72](#)

- f_score (metrics), [37](#)
- Family, [12](#), [25](#), [27](#)
- fda, [23](#)
- FDAModel, [3](#), [22](#)
- fit, [4](#), [7–9](#), [11](#), [12](#), [14](#), [16](#), [18](#), [19](#), [21](#), [23](#), [23](#), [25–28](#), [30–33](#), [35](#), [36](#), [43](#), [45–47](#), [53–56](#), [58](#), [61](#), [62](#), [65](#), [67](#), [68](#), [70](#), [72](#), [73](#), [75](#)
- fnr (metrics), [37](#)
- formula, [44](#)
- fpr (metrics), [37](#)

- gamboost, [25](#)
- GAMBoostModel, [3](#), [24](#)
- gbart, [11](#)
- gbm, [26](#)
- GBMModel, [3](#), [25](#)
- gen.ridge, [22](#), [36](#)
- gini (metrics), [37](#)
- glm, [28](#)
- glm.control, [28](#)
- glmboost, [27](#)
- GLMBoostModel, [3](#), [26](#)

- GLMModel, [3](#), [28](#)
- glmnet, [30](#)
- GLMNetModel, [3](#), [29](#)
- GLMStepAICModel (GLMModel), [28](#)
- Grid, [30](#), [71](#), [72](#)

- kappa2 (metrics), [37](#)
- kknn, [31](#)
- KNNModel, [4](#), [31](#)
- ksvm, [68](#)

- lars, [32](#)
- LARSModel, [4](#), [32](#)
- lda, [33](#)
- LDAModel, [4](#), [33](#)
- Lift (lift), [34](#)
- lift, [4](#), [34](#), [52](#)
- lm, [35](#)
- LMMModel, [4](#), [35](#)

- MachineShop (MachineShop-package), [3](#)
- MachineShop-package, [3](#)
- mae (metrics), [37](#)
- mars, [22](#), [36](#)
- mbart, [11](#)
- mda, [36](#)
- MDAModel, [4](#), [35](#)
- metricinfo, [37](#), [39](#), [41](#), [59](#)
- metrics, [37](#), [37](#), [41](#), [49](#), [50](#), [66](#), [72](#)
- MLControl, [39](#), [59](#), [62](#), [64](#), [71](#), [72](#)
- MLMetric, [5](#), [41](#)
- MLMetric<- (MLMetric), [41](#)
- MLModel, [5](#), [42](#)
- model.matrix, [42](#)
- ModelFrame, [24](#), [42](#), [44](#), [59](#), [72](#)
- ModelFrames, [24](#), [59](#)
- modelinfo, [3](#), [22](#), [24](#), [43](#), [45](#), [59](#), [72](#)
- mse (metrics), [37](#)
- msle (metrics), [37](#)
- mvr, [53](#)

- na.fail, [44](#)
- na.omit, [44](#)
- na.pass, [44](#)
- naiveBayes, [46](#)
- NaiveBayesModel, [4](#), [46](#)
- nnet, [47](#)
- NNetModel, [4](#), [46](#)
- npv (metrics), [37](#)

- OOBControl, [5](#)
- OOBControl (MLControl), [39](#)
- options, [44](#)

- p.adjust, [69](#)
- PDAModel, [4](#)
- PDAModel (FDAModel), [22](#)
- performance, [4](#), [19](#), [39](#), [48](#), [52](#), [54](#), [59](#), [64](#), [71](#)
- performance_curve, [4](#), [39](#), [49](#)
- plot, [5](#), [14](#), [17](#), [19](#), [34](#), [49](#), [50](#), [50](#), [59](#), [72](#), [73](#)
- PLSModel, [4](#), [52](#)
- polr, [54](#)
- POLRModel, [4](#), [53](#)
- polyreg, [22](#), [36](#)
- ppv (metrics), [37](#)
- pr_auc (metrics), [37](#)
- precision (metrics), [37](#)
- predict, [4](#), [14](#), [17](#), [23](#), [24](#), [33](#), [34](#), [36](#), [49](#), [50](#), [54](#), [55](#), [66](#)
- predict.fda, [23](#)
- predict.lda, [33](#)
- predict.mda, [36](#)
- predict.qda, [55](#)
- psm, [67](#)

- qda, [55](#)
- QDAModel, [4](#), [55](#)
- quote, [5](#), [6](#)

- r2 (metrics), [37](#)
- randomForest, [56](#)
- RandomForestModel, [4](#), [56](#)
- ranger, [58](#)
- RangerModel, [4](#), [57](#)
- recall (metrics), [37](#)
- recipe, [24](#), [59](#), [60](#), [72](#)
- resample, [4](#), [7–9](#), [11](#), [12](#), [14](#), [16–19](#), [21](#), [23](#), [25–28](#), [30–37](#), [40](#), [43](#), [45–47](#), [49](#), [50](#), [52–56](#), [58](#), [58](#), [61](#), [62](#), [64](#), [65](#), [67](#), [68](#), [70](#), [75](#)
- Resamples (resample), [58](#)
- response, [14](#), [17](#), [34](#), [49](#), [50](#), [60](#)
- rmse (metrics), [37](#)
- rmsle (metrics), [37](#)
- roc_auc (metrics), [37](#)
- roc_index (metrics), [37](#)
- role, [24](#), [59](#)
- rpart, [61](#)
- RPartModel, [4](#), [60](#)

rpp (metrics), 37

sensitivity (metrics), 37

specificity (metrics), 37

SplitControl, 5

SplitControl (MLControl), 39

StackedModel, 4, 62

stepAIC, 18, 28, 67

subset, 18

summary, 5, 17, 19, 49, 50, 59, 63, 72

SuperModel, 4, 64

surv.bart, 11

SurvEvents (SurvMatrix), 65

SurvMatrix, 65

SurvMatrix, ANY, ANY, ANY (SurvMatrix), 65

SurvProbs (SurvMatrix), 65

survreg, 67

survreg.control, 66, 67

SurvRegModel, 4, 66

SurvRegStepAICModel (SurvRegModel), 66

SVMANOVAModel (SVMModel), 67

SVMBesselModel (SVMModel), 67

SVMLaplaceModel (SVMModel), 67

SVMLinearModel (SVMModel), 67

SVMModel, 4, 67

SVMPolyModel (SVMModel), 67

SVMRadialModel (SVMModel), 67

SVMSplineModel (SVMModel), 67

SVMTanhModel (SVMModel), 67

t.test, 19, 69

tnr (metrics), 37

tpr (metrics), 37

TrainControl, 5

TrainControl (MLControl), 39

tree, 70

TreeModel, 4, 70

tune, 4, 7–9, 11, 12, 14, 16, 18, 19, 21–28, 30–33, 35, 36, 43, 45–47, 52–56, 58, 61, 62, 64, 65, 67, 68, 70, 71, 75

varimp, 5, 9, 13, 21, 24, 43, 52, 73, 75

weighted_kappa2 (metrics), 37

XGBDARTModel (XGBModel), 74

XGBLinearModel (XGBModel), 74

XGBModel, 4, 74

xgboost, 75

XGBTreeModel (XGBModel), 74