Package ‘MLInterfaces’

May 15, 2025

Title Uniform interfaces to R machine learning procedures for data in Bioconductor containers

Version 1.88.1

Description This package provides uniform interfaces to machine learning code for data in R and Bioconductor containers.

License LGPL

Depends R (>= 3.5), Rcpp, methods, BiocGenerics (>= 0.13.11), Biobase, annotate, cluster

Imports gdata, pls, sfsmisc, MASS, rpart, genefilter, fpc, ggvis, shiny, gbm, RColorBrewer, hwriter, threejs (>= 0.2.2), mlbench, stats4, tools, grDevices, graphics, stats, magrittr, SummarizedExperiment

Suggests class, e1071, ipred, randomForest, gpls, pamr, nnet, ALL, hgu95av2.db, som, hu6800.db, lattice, caret (>= 5.07), golubEsets, ada, keggorthology, kernlab, mboost, party, klaR, BiocStyle, knitr, testthat

Enhances parallel

LazyLoad yes

biocViews Classification, Clustering

NeedsCompilation no

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/MLInterfaces

git_branch RELEASE_3_21

git_last_commit 4cddc2c

git_last_commit_date 2025-05-08

Repository Bioconductor 3.21

Date/Publication 2025-05-14

Author Vincent Carey [cre, aut] (ORCID: <https://orcid.org/0000-0003-4046-0063>),
Jess Mar [aut],

Jason Vertrees [ctb],
Laurent Gatto [ctb],
Phylis Atieno [ctb] (Translated vignettes from Sweave to Rmarkdown / HTML.)

Maintainer Vincent Carey <stvjc@channing.harvard.edu>

Contents

balKfold.xvspec . . . 2
brennan_2013_tabS7exc . . . 4
classifierOutput-class . . . 5
clusteringOutput-class . . . 7
confuMat-methods . . . 8
confuTab . . . 9
fs.absT . . . 10
fsHistory . . . 11
hclustWidget . . . 12
learnerSchema-class . . . 13
MLearn . . . 14
MLIntInternals . . . 19
performance-analytics . . . 20
planarPlot-methods . . . 21
plspinHcube . . . 22
predict.classifierOutput . . . 23
projectedLearner-class . . . 25
projectLearnerToGrid . . . 26
RAB . . . 28
raboostCont-class . . . 29
varImpStruct-class . . . 29
xvalLoop . . . 31
xvalSpec . . . 32

Index **34**

`balKfold.xvspec` *generate a partition function for cross-validation, where the partitions are approximately balanced with respect to the distribution of a response variable*

Description

generate a partition function for cross-validation, where the partitions are approximately balanced with respect to the distribution of a response variable

Usage

```
balKfold.xvspec(K)
```

Arguments

K	number of partitions to be computed

Details

This function returns a closure. The symbol K is bound in the environment of the returned function.

Value

A closure consisting of a function that can be used as a partitionFunc for passage in xvalSpec.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

Examples

```
## The function is currently defined as
function (K)
function(data, clab, iternum) {
    clabs <- data[[clab]]
    narr <- nrow(data)
    cnames <- unique(clabs)
    ilist <- list()
    for (i in 1:length(cnames)) ilist[[cnames[i]]] <- which(clabs ==
        cnames[i])
    clens <- lapply(ilist, length)
    nrep <- lapply(clens, function(x) ceiling(x/K))
    grpinds <- list()
    for (i in 1:length(nrep)) grpinds[[i]] <- rep(1:K, nrep[[i]])[1:clens[[i]]]
    (1:narr)[-which(unlist(grpinds) == iternum)]
  }
# try it out
library("MASS")
data(crabs)
p1c = balKfold.xvspec(5)
inds = p1c( crabs, "sp", 3 )
table(crabs$sp[inds] )
inds2 = p1c( crabs, "sp", 4 )
table(crabs$sp[inds2] )
allc = 1:200
# are test sets disjoint?
intersect(setdiff(allc,inds), setdiff(allc,inds2))
```

brennan_2013_tabS7exc *Clinical characterization of 158 GBM samples from https://doi.org/10.1016/j.cell.2013.09.034 supp table S7*

Description

The clinical characteristics table of https://doi.org/10.1016/j.cell.2013.09.034 in supplemental table S7 was aligned with the GBM samples in curatedTCGAData (selecting GBM and version 2.0.1 with curatedTCGAData 1.17.0.

Usage

```
data("brennan_2013_tabS7exc")
```

Format

A data frame with 158 observations on the following 16 variables.

`Case_ID` a character vector

`Secondary_or_Recurrent` a character vector

`Age_at_Procedure` a numeric vector

`Gender` a character vector

`Path_Dx` a character vector

`MGMT_Status` a character vector

`Methylation_Class_2012` a character vector

`G_CIMP_methylation` a character vector

`IDH1_status` a character vector

`Expression_Subclass` a character vector

`Therapy_Class` a character vector

`Vital_Status` a character vector

`OS_days` a numeric vector

`Progression_Status` a character vector

`PFS_days` a numeric vector

`V16` a logical vector

Details

Simple intersection on Case_ID in Supp Tab 7 with patientID in the GBM from curatedTCGAData.

Source

https://doi.org/10.1016/j.cell.2013.09.034

References

The Somatic Genomic Landscape of Glioblastoma by Cameron W. Brennan, Roel G.W. Verhaak, Aaron McKenna, and others, Cell Oct 10 2013.

Examples

```
data(brennan_2013_tabS7exc)
head(brennan_2013_tabS7exc)
```

`classifierOutput-class` *Class "classifierOutput"*

Description

This class summarizes the output values from different classifiers.

Objects from the Class

Objects are typically created during the application of a supervised machine learning algorithm to data and are the value returned. It is very unlikely that any user would create such an object by hand.

Slots

`testOutcomes`: Object of class `"factor"` that lists the actual outcomes in the records on the test set

`testPredictions`: Object of class `"factor"` that lists the predictions of outcomes in the test set

`testScores`: Object of class `"ANY"` – this element will include matrices or vectors or arrays that include information that is typically related to the posterior probability of occupancy of the predicted class or of all classes. The actual contents of this slot can be determined by inspecting the converter element of the learnerSchema used to select the model.

`trainOutcomes`: Object of class `"factor"` that lists the actual outcomes in records on the training set

`trainPredictions`: Object of class `"factor"` that lists the predicted outcomes in the training set

`trainScores`: Object of class `"ANY"` see the description of `testScores` above; the same information is returned, but applicable to the training set records.

`trainInd`: Object of class `"numeric"` with of indices of data to be used for training.

`RObject`: Object of class `"ANY"` – when the `trainInd` parameter of the `MLearn` call is numeric, this slot holds the return value of the underlying R function that carried out the predictive modeling. For example, if `rpartI` was used as MLearn `method`, `Robject` holds an instance of the `rpart` S3 class, and `plot` and `text` methods can be applied to this. When the `trainInd` parameter of the `MLearn` call is an instance of `xvalSpec`, this slot holds a `list` of results of cross-validatory iterations. Each element of this list has two elements: `test.idx`, giving the numeric indices of the test cases for the associated cross-validation iteration, and `mlans`, which is the `classifierOutput` for the associated iteration. See the example for an illustration of 'digging out' the predicted probabilities associated with each cross-validation iteration executed through an xvalSpec specification.

`embeddedCV`**:** logical value that is TRUE if the procedure in use performs its own cross-validation

`fsHistory`**:** list of features selected through cross-validation process

`learnerSchema`**:** propagation of the learner schema object used in the call

`call`**:** Object of class `"call"` – records the call used to generate the classifierOutput RObject

Methods

confuMat `signature(obj = "classifierOutput")`: Compute the confusion matrix for test records.

confuMatTrain `signature(obj = "classifierOutput")`: Compute the confusion matrix for training set. Typically yields optimistically biased information on misclassification rate.

RObject `signature(obj = "classifierOutput")`: The R object returned by the underlying classifier. This can then be passed on to specific methods for those objects, when they exist.

trainInd `signature(obj = "classifierOutput")`: Returns the indices of data used for training.

show `signature(object = "classifierOutput")`: A print method that provides a summary of the output of the classifier.

predictions `signature(object = "classifierOutput")`: Print the predicted classes for each sample/individual. The predictions for the training set are the training outcomes.

predictions `signature(object = "classifierOutput", t = "numeric")`: Print the predicted classes for each sample/individual that have a `testScore` greater or equal than `t`. The predictions for the training set are the training outcomes. Non-predicted cases and cases that matche multiple classes are returned as `NA`s.

predScore `signature(object = "classifierOutput")`: Returns the scores for predicted class for each sample/individual. The scores for the training set are set to 1.

predScores `signature(object = "classifierOutput")`: Returns the prediction scores for all classes for each sample/individual. The scores for the training set are set to 1 for the appropriate class, 0 otherwise.

testScores `signature(object = "classifierOutput")`: ...

testPredictions `signature(object = "classifierOutput")`: Print the predicted classes for each sample/individual in the test set.

testPredictions `signature(object = "classifierOutput", t = "numeric")`: Print the predicted classes for each sample/individual in the test set that have a `testScore` greater or equal than `t`. Non-predicted cases and cases that matche multiple classes are returned as `NA`s.

trainScores `signature(object = "classifierOutput")`: ...

trainPredictions `signature(object = "classifierOutput")`: Print the predicted classes for each sample/individual in the train set.

trainPredictions `signature(object = "classifierOutput", t = "numeric")`: Print the predicted classes for each sample/individual in the train set that have a `testScore` greater or equal than `t`. Non-predicted cases and cases that matche multiple classes are returned as `NA`s.

fsHistory `signature(object = "classifierOutput")`: ...

Author(s)

V. Carey

Examples

```
showClass("classifierOutput")
library(golubEsets)
data(Golub_Train) # now cross-validate a neural net
set.seed(1234)
xv5 = xvalSpec("LOG", 5, balKfold.xvspec(5))
m2 = MLearn(ALL.AML~., Golub_Train[1000:1050,], nnetI, xv5,
   size=5, decay=.01, maxit=1900 )
testScores(RObject(m2)[[1]]$mlans)
alls = lapply(RObject(m2), function(x) testScores(x$mlans))
```

---

clusteringOutput-class *container for clustering outputs in uniform structure*

Description

container for clustering outputs in uniform structure

Objects from the Class

Objects can be created by calls of the form `new("clusteringOutput", ...)`.

Slots

`partition`: Object of class `"integer"`, labels for observations as clustered

`silhouette`: Object of class `"silhouette"`, structure from Rousseeuw cluster package measuring cluster membership strength per observation

`prcomp`: Object of class `"prcompObj"` a wrapped instance of stats package prcomp output

`call`: Object of class `"call"` for auditing

`learnerSchema`: Object of class `"learnerSchema"`, a formal object indicating the package, function, and other attributes of the clustering algorithm employed to generate this object

`RObject`: Object of class `"ANY"`, the unaltered output of the function called according to learnerSchema

`converter`: converter propagated from call

`distFun`: distfun propagated from call

Methods

RObject `signature(x = "clusteringOutput")`: extract the unaltered output of the R function or method called according to learnerSchema

plot `signature(x = "clusteringOutput", y = "ANY")`: a 4-panel plot showing features of the clustering, including the scree plot for a principal components transformation and a display of the partition in PC1xPC2 plane. For a clustering method that does not have a native plot procedure, such as kmeans, the parameter y should be bound to a data frame or matrix with feature data for all records; an image plot of robust feature z-scores (z=(x-median(x))/mad(x)) and the cluster indices is produced in the northwest panel.

show `signature(object = "clusteringOutput")`: concise report

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

Examples

```
showClass("clusteringOutput")
```

`confuMat-methods` *Compute the confusion matrix for a classifier.*

Description

This function will compute the confusion matrix for a classifier's output

Methods

obj = "classifOutput", ... Typically, an instance of class "`classifierOutput`" is built on a training subset of the input data. The model is then used to predict the class of samples in the test set. When the true class labels for the test set are available the confusion matrix is the cross-tabulation of the true labels of the test set against the predictions from the classifier. An optional `t` score threshold can also be specified.

obj = "classifierOutput", type="character", ... For instances of classifierOutput, it is possible to specify the `type` of confusion matrix desired. The default is `test`, which tabulates classes from the test set against the associated predictions. If `type` is `train`, the training class vector is tabulated against the predictions on the training set. An optional `t` score threshold can also be specified.

obj = "classifierOutput", type="numeric" For instances of classifierOutput, it is possible to specify the minimum score feature classification threshold. Features with a score less than the threshold are classified as `NA` in the confustion `train` or `test` confusion matrix.

Examples

```
library(golubEsets)
data(Golub_Merge)
smallG <- Golub_Merge[101:150,]
k1 <- MLearn(ALL.AML~., smallG, knnI(k=1), 1:30)
confuMat(k1)
confuMat(k1, "train")
```

---

confuTab *Compute confusion tables for a confusion matrix.*

Description

Given a `n` by `n` confusion matrix, the function returns a `list` of `n` 2 by 2 tables with false positives, false negatives, false positives and true negative for each initial variables.

Usage

```
confuTab(obj, naAs0. = FALSE)
```

Arguments

`obj`	An instance of class `table`. Must be square.
`naAs0.`	A `logical`, defining if `NA`s are to be replaced by `0`s.

Value

A `list` of length `nrow(obj)` and names `rownames(obj)`.

Author(s)

Laurent Gatto <lg390@cam.ac.uk>

See Also

The `tp`, `tn`, `fp`, `fn`, methods to extract the respective classification outcomes from a contingency matrix.

Examples

```
## the confusion matrix
cm <- table(iris$Species, sample(iris$Species))
## the 3 confusion tables
(ct <- confuTab(cm))
```

fs.absT *support for feature selection in cross-validation*

Description

support for feature selection in cross-validation

Usage

```
fs.absT(N)
fs.probT(p)
fs.topVariance(p)
```

Arguments

N	number of features to retain; features are ordered by descending value of abs(two-sample t stat.), and the top N are used.
p	cumulative probability (in (0,1)) in the distribution of absolute t statistics above which we retain features

Details

This function returns a function that will be used as a parameter to `xvalSpec` in applications of `MLearn`.

Value

a function is returned, that will itself return a formula consisting of the selected features for application of `MLearn`.

Note

The functions `fs.absT` and `fs.probT` are two examples of approaches to embedded feature selection that make sense for two-sample prediction problems. For selection based on linear models or other discrimination measures, you will need to create your own selection helper, following the code in these functions as examples.

fs.topVariance performs non-specific feature selection based on the variance. Argument p is the variance percentile beneath which features are discarded.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

See Also

`MLearn`

Examples

```
library("MASS")
data(crabs)
# we will demonstrate this procedure with the crabs data.
# first, create the closure to pick 3 features
demFS = fs.absT(3)
# run it on the entire dataset with features excluding sex
demFS(sp~.-sex, crabs)
# emulate cross-validation by excluding last 50 records
demFS(sp~.-sex, crabs[1:150,])
# emulate cross-validation by excluding first 50 records -- different features retained
demFS(sp~.-sex, crabs[51:200,])
```

`fsHistory` *extract history of feature selection for a cross-validated machine learner*

Description

extract history of feature selection for a cross-validated machine learner

Usage

```
fsHistory(x)
```

Arguments

x	instance of `classifierOutput`

Details

returns a list of names of selected features

Value

a list; the names of variables are made 'syntactic'

Author(s)

Vince Carey <stvjc@channing.harvard.edu>

Examples

```
data(iris)
iris2 = iris[ iris$Species %in% levels(iris$Species)[1:2], ]
iris2$Species = factor(iris2$Species) # drop unused levels
x1 = MLearn(Species~., iris2, ldaI, xvalSpec("LOG", 3,
   balKfold.xvspec(3), fs.absT(3)))
fsHistory(x1)
```

hclustWidget *shiny-oriented GUI for cluster or classifier exploration*

Description

shiny-oriented GUI for cluster or classifier exploration

Usage

```
hclustWidget(mat, featureName = "feature", title =
                 paste0("hclustWidget for ", deparse(substitute(mat))),
                 minfeats = 2, auxdf = NULL)

mlearnWidget(eset, infmla)
```

Arguments

mat	matrix with feature vectors in rows
featureName	name to be used for control that asks for number of features to use
title	widget title
minfeats	lower bound on number of features to use
auxdf	data.frame with number of rows equal to nrow(mat), with metadata to be displayed in hovering tooltip
eset	instance of ExpressionSet-class
infmla	instance of formula, with dependent variable values obtained in pData(eset), and independent variable specified as "." or as properly munged elements of featureNames(eset).

Details

Experimental tool to illustrate impacts of choice of distance, agglomeration method, etc.

Value

a shinyApp result that will display in active browser

Note

mlearnWidget will attempt to nicely produce a variable importance plot using randomForestI. This means that the annotation package for probe identifiers should be loaded or an error will be thrown.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

Examples

```
# should run with example(hclustWidget, ask=FALSE)
if (interactive()) {
 library(shiny)
 library(MASS)
 data(crabs)
 cr = data.matrix(crabs[,-c(1:3)])
 au = crabs[,1:3]
 show(hclustWidget(cr, auxdf=au))
## must use stop widget button to proceed
  library(ALL)
  library(hgu95av2.db)
  data(ALL)
  show(mlearnWidget(ALL[1:500,], mol.biol~.))
 }
```

---

learnerSchema-class   *Class "learnerSchema" – convey information on a machine learning function to the MLearn wrapper*

Description

conveys information about machine learning functions in CRAN packages, for example, to MLearn wrapper

Objects from the Class

Objects can be created by calls of the form `new("learnerSchema", ...)`.

Slots

`packageName`**:** Object of class `"character"` string naming the package in which the function to be used is defined.

`mlFunName`**:** Object of class `"character"` string naming the function to be used

`converter`**:** Object of class `"function"` function with parameters obj, data, trainInd, that will produce a classifierOutput instance

Methods

MLearn `signature(formula = "formula", data = "ExpressionSet", method = "learnerSchema", trainInd = "numeric")`: execute desired learner passing a formula and ExpressionSet

MLearn `signature(formula = "formula", data = "data.frame", method = "learnerSchema", trainInd = "numeric")`: execute desired learner passing a formula

show `signature(object = "learnerSchema")`: concise display

Author(s)

Vince Carey <stvjc@channing.harvard.edu>

Examples

```
showClass("learnerSchema")
```

MLearn *revised MLearn interface for machine learning*

Description

revised MLearn interface for machine learning, emphasizing a schematic description of external learning functions like knn, lda, nnet, etc.

Usage

```
MLearn( formula, data, .method, trainInd, ... )
makeLearnerSchema(packname, mlfunname, converter, predicter)
```

Arguments

`formula`	standard model formula
`data`	data.frame or ExpressionSet instance
`.method`	instance of learnerSchema
`trainInd`	obligatory numeric vector of indices of data to be used for training; all other data are used for testing, or instance of the xvalSpec class
`...`	additional named arguments passed to external learning function
`packname`	character – name of package harboring a learner function
`mlfunname`	character – name of function to use
`converter`	function – with parameters (obj, data, trainInd) that tells how to convert the material in obj [produced by [packname::mlfunname]] into a classifierOutput instance.
`predicter`	function – with parameters (obj, newdata, ...) that tells how to use the material in `obj` to predict `newdata`.

Details

The purpose of the MLearn methods is to provide a uniform calling sequence to diverse machine learning algorithms. In R package, machine learning functions can have parameters `(x, y, ...)` or `(formula, data, ...)` or some other sequence, and these functions can return lists or vectors or other sorts of things. With MLearn, we always have calling sequence `MLearn(formula, data, .method, trainInd, ...)`, and `data` can be a `data.frame` or `ExpressionSet`. `MLearn` will always return an S4 instance of `classifierObject` or `clusteringObject`.

At this time (1.13.x), NA values in predictors trigger an error.

To obtain documentation on the older (pre bioc 2.1) version of the MLearn method, please use help(MLearn-OLD).

randomForestI randomForest. Note, that to obtain the default performance of randomForestB, you need to set mtry and sampsize parameters to sqrt(number of features) and table([training set response factor]) respectively, as these were not taken to be the function's defaults. Note you can use xvalSpec("NOTEST") as trainInd, to use all the samples; the RObject() result will print the misclassification matrix estimate along with OOB error rate estimate.

knnI(k=1,l=0) knn; special support bridge required, defined in MLint

knn.cvI(k=1,l=0) knn.cv; special support bridge required, defined in MLint. This option uses the embedded leave-one-out cross-validation of `knn.cv`, and thereby achieves high performance. You can have more general cross-validation using `knnI` with an `xvalSpec`, but it will be slower. When using this learner schema, you should use the numerical `trainInd` setting with `1:N` where `N` is the number of samples.

dldaI diagDA; special support bridge required, defined in MLint

nnetI nnet

rpartI rpart

ldaI lda

svmI svm

qdaI qda

logisticI(threshold) glm – with binomial family, expecting a dichotomous factor as response variable, not bulletproofed against other responses yet. If response probability estimate exceeds threshold, predict 1, else 0

adaI ada

BgbmI gbm, forcing the Bernoulli loss function.

blackboostI blackboost – you MUST supply a family parameter relevant for mboost package procedures

lvqI lvqtest after building codebook with lvqinit and updating with olvq1. You will need to write your own detailed schema if you want to tweak tuning parameters.

naiveBayesI naiveBayes

baggingI bagging

sldaI slda

ksvmI ksvm

hclustI(distMethod, agglomMethod) hclust – you must explicitly specify distance and agglomeration procedure.

kmeansI(centers, algorithm) kmeans – you must explicitly specify centers and algorithm name.

If the `parallel` package is attached, cross-validation will be distributed to cores using `mclapply`.

Value

Instances of classifierOutput or clusteringOutput

Author(s)

Vince Carey <stvjc@channing.harvard.edu>

See Also

Try `example(hclustWidget, ask=FALSE)` for an interactive approach to cluster analysis tuning.

Examples

```
library("MASS")
data(crabs)
set.seed(1234)
kp = sample(1:200, size=120)
rf1 = MLearn(sp~CW+RW, data=crabs, randomForestI, kp, ntree=600 )
rf1
nn1 = MLearn(sp~CW+RW, data=crabs, nnetI, kp, size=3, decay=.01,
    trace=FALSE )
nn1
RObject(nn1)
knn1 = MLearn(sp~CW+RW, data=crabs, knnI(k=3,l=2), kp)
knn1
names(RObject(knn1))
dlda1 = MLearn(sp~CW+RW, data=crabs, dldaI, kp )
dlda1
names(RObject(dlda1))
lda1 = MLearn(sp~CW+RW, data=crabs, ldaI, kp )
lda1
names(RObject(lda1))
slda1 = MLearn(sp~CW+RW, data=crabs, sldaI, kp )
slda1
names(RObject(slda1))
svm1 = MLearn(sp~CW+RW, data=crabs, svmI, kp )
svm1
names(RObject(svm1))
ldapp1 = MLearn(sp~CW+RW, data=crabs, ldaI.predParms(method="debiased"), kp )
ldapp1
names(RObject(ldapp1))
qda1 = MLearn(sp~CW+RW, data=crabs, qdaI, kp )
qda1
names(RObject(qda1))
logi = MLearn(sp~CW+RW, data=crabs, glmI.logistic(threshold=0.5), kp, family=binomial ) # need family
logi
names(RObject(logi))
rp2 = MLearn(sp~CW+RW, data=crabs, rpartI, kp)
rp2
## recode data for RAB
#nsp = ifelse(crabs$sp=="O", -1, 1)
#nsp = factor(nsp)
#ncrabs = cbind(nsp,crabs)
#rab1 = MLearn(nsp~CW+RW, data=ncrabs, RABI, kp, maxiter=10)
#rab1
#
# new approach to adaboost
#
ada1 = MLearn(sp ~ CW+RW, data = crabs, .method = adaI,
    trainInd = kp, type = "discrete", iter = 200)
```

```
ada1
confuMat(ada1)
#
lvq.1 = MLearn(sp~CW+RW, data=crabs, lvqI, kp )
lvq.1
nb.1 = MLearn(sp~CW+RW, data=crabs, naiveBayesI, kp )
confuMat(nb.1)
bb.1 = MLearn(sp~CW+RW, data=crabs, baggingI, kp )
confuMat(bb.1)
#
# new mboost interface -- you MUST supply family for nonGaussian response
#
require(party)  # trafo ... killing cmd check
blb.1 = MLearn(sp~CW+RW+FL, data=crabs, blackboostI, kp, family=mboost::Binomial() )
confuMat(blb.1)
#
# ExpressionSet illustration
#
data(sample.ExpressionSet)
#  needed to increase training set size to avoid a new randomForest condition
# on empty class
set.seed(1234)
X = MLearn(type~., sample.ExpressionSet[100:250,], randomForestI, 1:19, importance=TRUE )
library(randomForest)
library(hgu95av2.db)
opar = par(no.readonly=TRUE)
par(las=2)
plot(getVarImp(X), n=10, plat="hgu95av2", toktype="SYMBOL")
par(opar)
#
# demonstrate cross validation
#
nn1cv = MLearn(sp~CW+RW, data=crabs[c(1:20,101:120),],
   nnetI, xvalSpec("LOO"), size=3, decay=.01, trace=FALSE )
confuMat(nn1cv)
nn2cv = MLearn(sp~CW+RW, data=crabs[c(1:20,101:120),], nnetI,
   xvalSpec("LOG",5, balKfold.xvspec(5)), size=3, decay=.01,
   trace=FALSE )
confuMat(nn2cv)
nn3cv = MLearn(sp~CW+RW+CL+BD+FL, data=crabs[c(1:20,101:120),], nnetI,
   xvalSpec("LOG",5, balKfold.xvspec(5), fsFun=fs.absT(2)), size=3, decay=.01,
   trace=FALSE )
confuMat(nn3cv)
nn4cv = MLearn(sp~.-index-sex, data=crabs[c(1:20,101:120),], nnetI,
   xvalSpec("LOG",5, balKfold.xvspec(5), fsFun=fs.absT(2)), size=3, decay=.01,
   trace=FALSE )
confuMat(nn4cv)
#
# try with expression data
#
library(golubEsets)
data(Golub_Train)
litg = Golub_Train[ 100:150, ]
```

```
g1 = MLearn(ALL.AML~. , litg, nnetI,
   xvalSpec("LOG",5, balKfold.xvspec(5),
   fsFun=fs.probT(.75)), size=3, decay=.01, trace=FALSE )
confuMat(g1)
#
# computations related to ALL that were used for rda and may be used elsewhere
#
library(ALL)
data(ALL)
#
# restrict to BCR/ABL or NEG
#
bio <- which( ALL$mol.biol %in% c("BCR/ABL", "NEG"))
#
# restrict to B-cell
#
isb <- grep("^B", as.character(ALL$BT))
kp <- intersect(bio,isb)
all2 <- ALL[,kp]
mads = apply(exprs(all2),1,mad)
kp = which(mads>1)  # get around 250 genes
vall2 = all2[kp, ]
vall2$mol.biol = factor(vall2$mol.biol) # drop unused levels

# illustrate clustering support

cl1 = MLearn(~CW+RW+CL+FL+BD, data=crabs, hclustI(distFun=dist, cutParm=list(k=4)))
plot(cl1)

cl1a = MLearn(~CW+RW+CL+FL+BD, data=crabs, hclustI(distFun=dist, cutParm=list(k=4)),
   method="complete")
plot(cl1a)

cl2 = MLearn(~CW+RW+CL+FL+BD, data=crabs, kmeansI, centers=5, algorithm="Hartigan-Wong")
plot(cl2, crabs[,-c(1:3)])

c3 = MLearn(~CL+CW+RW, crabs, pamI(dist), k=5)
c3
plot(c3, data=crabs[,c("CL", "CW", "RW")])


#  new interfaces to PLS thanks to Laurent Gatto

set.seed(1234)
kp = sample(1:200, size=120)

#plsda.1 = MLearn(sp~CW+RW, data=crabs, plsdaI, kp, probMethod="Bayes")
#plsda.1
#confuMat(plsda.1)
#confuMat(plsda.1,t=.65) ## requires at least 0.65 post error prob to assign species
#
#plsda.2 = MLearn(type~., data=sample.ExpressionSet[100:250,], plsdaI, 1:16)
#plsda.2
```

```
#confuMat(plsda.2)
#confuMat(plsda.2,t=.65) ## requires at least 0.65 post error prob to assign outcome

## examples for predict
#clout <- MLearn(type~., sample.ExpressionSet[100:250,], svmI , 1:16)
#predict(clout, sample.ExpressionSet[100:250,17:26])
```

`MLIntInternals` *MLInterfaces infrastructure*

Description

These functions are internal tools for `MLInterfaces`. Users will generally not call these functions directly.

Usage

```
getGrid(x)
```

Arguments

`x`	a vector or matrix or ExpressionSet

Details

Forthcoming.

Value

Functions with 'new' as prefix are constructor helpers.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

performance-analytics *Assessing classifier performance*

Description

Methods to calculate the number of true positives (tp), true negatives (tn), false negatives (fn), false positive (fp), accuracy (acc), precision, recall (same as sensitivity), specificity, F1 and macroF1 scores.

Each method also accepts an naAs0 argument definiting if NAs should be replaced by 0 (default is FALSE).

Methods

Methods tp, tn, fp, fn, F1, acc and specificity:

signature(obj = "table")

Methods recall (sensitivity), precision and macroF1:

signature(obj = "classifierOutput", type = "character")

signature(obj = "classifierOutput", type = "missing")

signature(obj = "classifierOutput", type = "numeric")

signature(obj = "table")

Examples

```
## the confusion matrix
cm <- table(iris$Species, sample(iris$Species))
tp(cm)
tn(cm)
fp(cm)
fn(cm)
acc(cm)
precision(cm)
recall(cm)
F1(cm)
macroF1(cm)
```

planarPlot-methods *Methods for Function planarPlot in Package 'MLInterfaces'*

Description

show the classification boundaries on the plane dictated by two genes in an ExpressionSet

Methods

clo = "classifierOutput", eset = "ExpressionSet", classifLab = "character" uses two genes in the ExpressionSet to exhibit the decision boundaries in the plane

clo = "classifierOutput", eset = "data.frame", classifLab = "character" uses two columns in the data.frame to exhibit the decision boundaries in the plane

Examples

```
library(ALL)
library(hgu95av2.db)
data(ALL)
#
# restrict to BCR/ABL or NEG
#
bio <- which( ALL$mol.biol %in% c("BCR/ABL", "NEG"))
#
# restrict to B-cell
#
isb <- grep("^B", as.character(ALL$BT))
kp <- intersect(bio,isb)
all2 <- ALL[,kp]
#
# sample 2 genes at random
#
set.seed(1234)
ng <- nrow(exprs(all2)) # pick 5 in case any NAs come back
pick <- sample(1:ng, size=5, replace=FALSE)
gg <- all2[pick,]
sym <- unlist(mget(featureNames(gg), hgu95av2SYMBOL))
bad = which(is.na(sym))
if (length(bad)>0) {
  gg = gg[-bad,]
  sym = sym[-bad]
  }
gg = gg[1:2,]
sym = sym[1:2]
featureNames(gg) <- sym
gg$class = factor(ifelse(all2$mol.biol=="NEG", "NEG", "POS"))

cl1 <- which( gg$class == "NEG" )
cl2 <- which( gg$class != "NEG" )
```

```
#
# create balanced training sample
#
trainInds <- c( sample(cl1, size=floor(length(cl1)/2) ),
      sample(cl2, size=floor(length(cl2)/2)) )
#
# run rpart
#
tgg <- MLearn(class~., gg, rpartI, trainInds, minsplit=4 )
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
planarPlot( tgg, gg, "class" )
title("rpart")
points(exprs(gg)[1,trainInds], exprs(gg)[2,trainInds], col=ifelse(gg$class[trainInds]=="NEG", "yellow", "black")
#
# run nnet
#
ngg <- MLearn( class~., gg, nnetI, trainInds, size=8 )
planarPlot( ngg, gg, "class" )
points(exprs(gg)[1,trainInds], exprs(gg)[2,trainInds], col=ifelse(gg$class[trainInds]=="NEG", "yellow", "black")
title("nnet")
#
# run knn
#
kgg <- MLearn( class~.,  gg, knnI(k=3,l=1), trainInds)
planarPlot( kgg, gg, "class" )
points(exprs(gg)[1,trainInds], exprs(gg)[2,trainInds], col=ifelse(gg$class[trainInds]=="NEG", "yellow", "black")
title("3-nn")
#
# run svm
#
sgg <- MLearn( class~., gg, svmI, trainInds )
planarPlot( sgg, gg, "class" )
points(exprs(gg)[1,trainInds], exprs(gg)[2,trainInds], col=ifelse(gg$class[trainInds]=="NEG", "yellow", "black")
title("svm")
par(opar)
```

plspinHcube *shiny app for interactive 3D visualization of mlbench hypercube*

Description

shiny app for interactive 3D visualization of mlbench hypercube

Usage

```
plspinHcube(insbwidth=4)
```

Arguments

`insbwidth`	numeric, sidebar width

Value

Runs shinyApp on ui and server that render gaussian data at hypercube vertices.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

See Also

mlbench.hypercube

Examples

```
if (interactive()) plspinHcube()
```

---

predict.classifierOutput

Predict method for `classifierOutput` *objects*

Description

This function predicts values based on models trained with MLInterfaces' `MLearn` interface to many machine learning algorithms.

Usage

```
## S3 method for class 'classifierOutput'
predict(object, newdata, ...)
```

Arguments

`object`	An instance of class `classifierOutput`.
`newdata`	An object containing the new input data: either a `matrix`, a `data.frame` or an `ExpressionSet`.
`...`	Other arguments to be passed to the algorithm-specific predict methods.

Details

This S3 method will extract the ML model from the `classifierOutput` instance and call either a generic predict method or, if available, a specficly written wrapper to do classes prediction and class probabilities.

Value

Currently, a list with

`testPredictions`	A factor with class predictions.
`testScores`	A `numeric` or `matrix` with class probabilities.

Note

The function output will most likely be updated in a near future to a `classifierOutput` (or similar) object.

Author(s)

Laurent Gatto <lg390@cam.ac.uk>

See Also

`MLearn` and `classifierOutput`.

Examples

```
## Not run:
set.seed(1234)
data(sample.ExpressionSet)
trainInd <- 1:16

clout.svm <- MLearn(type~., sample.ExpressionSet[100:250,], svmI, trainInd)
predict(clout.svm, sample.ExpressionSet[100:250,-trainInd])

clout.ksvm <- MLearn(type~., sample.ExpressionSet[100:250,], ksvmI, trainInd)
predict(clout.ksvm, sample.ExpressionSet[100:250,-trainInd])

clout.nnet <- MLearn(type~., sample.ExpressionSet[100:250,], nnetI, trainInd, size=3, decay=.01 )
predict(clout.nnet, sample.ExpressionSet[100:250,-trainInd])

clout.knn <- MLearn(type~., sample.ExpressionSet[100:250,], knnI(k=3), trainInd)
predict(clout.knn, sample.ExpressionSet[100:250,-trainInd],k=1)
predict(clout.knn, sample.ExpressionSet[100:250,-trainInd],k=3)

#clout.plsda <- MLearn(type~., sample.ExpressionSet[100:250,], plsdaI, trainInd)
#predict(clout.plsda, sample.ExpressionSet[100:250,-trainInd])

clout.nb <- MLearn(type~., sample.ExpressionSet[100:250,], naiveBayesI, trainInd)
predict(clout.nb, sample.ExpressionSet[100:250,-trainInd])

# this can fail if training set does not yield sufficient diversity in response vector;
# setting seed seems to help with this example, but other applications may have problems
#
clout.rf <- MLearn(type~., sample.ExpressionSet[100:250,], randomForestI, trainInd)
predict(clout.rf, sample.ExpressionSet[100:250,-trainInd])

## End(Not run) # end of dontrun
```

projectedLearner-class

Class "projectedLearner"

Description

helps depict prediction hyperregions from high-dimensional models

Objects from the Class

Objects can be created by calls of the form `new("projectedLearner", ...)`.

Slots

`fittedLearner`**:** Object of class "classifierOutput"

`trainingSetPCA`**:** Object of class "prcomp"

`trainingLabels`**:** Object of class "ANY" given labels for features used in training

`testLabels`**:** Object of class "ANY" given labels for features used in testing

`gridFeatsProjectedToTrainingPCs`**:** Object of class "matrix" rotated coordinates of training features

`gridPredictions`**:** Object of class "ANY" predicted labels for all grid points

`trainFeatsProjectedToTrainingPCs`**:** Object of class "matrix" rotated coordinates of training features

`testFeatsProjectedToTrainingPCs`**:** Object of class "matrix" rotated coordinates of test features

`trainPredictions`**:** Object of class "ANY" predicted labels for training features

`testPredictions`**:** Object of class "ANY" predicted labels for test features

`theCall`**:** Object of class "call" call used to generate this wonderful thing

Methods

learnerIn3D `signature(x = "projectedLearner")`: uses `rgl` to give a dynamic 3d-like projection of labels in colored regions. See `projectLearnerToGrid` for an example.

plot `signature(x = "projectedLearner", y = "ANY")`: pairs plot of the tesselated PCA of the training features

plotOne `signature(x = "projectedLearner")`: a 2d plot of tesselation projection for selected axes of the PCA

show `signature(object = "projectedLearner")`: object housing numerical resources for the renderings

Note

`plot` may need to be modified when there are many features/PCs in use

`plotOne` has additional arguments `ind1`, `ind2`, and `type`. ind1 and ind2 specify the PCs to display. type is one of `"showTestPredictions"` (default), `"showTrainPredictions"`, `"showTestLabels"`, `"showTrainLabels"`. These indicate what will be used to locate glyphs with labels in the projected scatterplots.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

References

None.

Examples

```
showClass("projectedLearner")
```

`projectLearnerToGrid` *create learned tesselation of feature space after PC transformation*

Description

create learned tesselation of feature space after PC transformation

Usage

```
projectLearnerToGrid(formula, data, learnerSchema,
   trainInds, ...,
   dropIntercept = TRUE,
   ngpts = 20,
   predExtras = list(),
   predWrapper = force)
```

Arguments

`formula`	standard formula, typically of the form "y~." where y denotes the class label variable to be predicted by all remaining features in the input data frame
`data`	a data.frame instance
`learnerSchema`	an instance of `learnerSchema-class`
`trainInds`	integer vector of rows of `data` to use for training
`...`	additional parameters for use with `learnerSchema`
`dropIntercept`	logical indicating whether to include column of 1s among feature column-vectors

`ngpts`	number of equispaced points along the range of each input feature to use in forming a grid in feature space
`predExtras`	a list with named elements giving binding to extra parameters needed to predict labels for the learner in use. For example, with `ldaI`, set `predExtras=list(type="class")`
`predWrapper`	Sometimes a function call is needed to extract the predicted labels from the RObject applied to the `fittedLearner` slot of the output object; this parameter defines that call.

Value

instance of `projectedLearner-class`

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

References

none.

Examples

```
library(mlbench)
# demostrate with 3 dimensional hypercube problem
kk = mlbench.hypercube()
colnames(kk$x) = c("f1", "f2", "f3")
hcu = data.frame(cl=kk$classes, kk$x)
set.seed(1234)
sam = sample(1:nrow(kk$x), size=nrow(kk$x)/2)
ldap = projectLearnerToGrid(cl~., data=hcu, ldaI,
   sam, predWrapper=function(x)x$class)
plot(ldap)
confuMat(ldap@fittedLearner)
nnetp = projectLearnerToGrid(cl~., data=hcu, nnetI, sam, size=2,
   decay=.01, predExtras=list(type="class"))
plot(nnetp)
confuMat(nnetp@fittedLearner)
#if (requireNamespace("rgl") && interactive()) {
#    learnerIn3D(nnetp)
#    ## customising the rgl plot
#    learnerIn3D(nnetp, size = 10, alpha = 0.1)
#}
```

`RAB` *real adaboost (Friedman et al)*

Description

read adaboost ... a demonstration version

Usage

```
RAB(formula, data, maxiter=200, maxdepth=1)
```

Arguments

`formula`	formula – the response variable must be coded -1, 1
`data`	data
`maxiter`	maxiter
`maxdepth`	maxdepth – passed to rpart

Value

an instance of raboostCont

Author(s)

Vince Carey <stvjc@channing.harvard.edu>

References

Friedman et al Ann Stat 28/2 337

Examples

```
library(MASS)
library(rpart)
data(Pima.tr)
data(Pima.te)
Pima.all = rbind(Pima.tr, Pima.te)
tonp = ifelse(Pima.all$type == "Yes", 1, -1)
tonp = factor(tonp)
Pima.all = data.frame(Pima.all[,1:7], mtype=tonp)
fit1 = RAB(mtype~ped+glu+npreg+bmi+age, data=Pima.all[1:200,], maxiter=10, maxdepth=5)
pfit1 = Predict(fit1, newdata=Pima.tr)
table(Pima.tr$type, pfit1)
```

raboostCont-class *Class "raboostCont" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("raboostCont", ...)`. ~~ describe objects here ~~

Slots

`.Data`: Object of class `"list"` ~~

`formula`: Object of class `"formula"` ~~

`call`: Object of class `"call"` ~~

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2.

Methods

`Predict` is an S4 method that can apply to instances of this class.

Author(s)

VJ Carey <stvjc@channing.harvard.edu>

Examples

```
showClass("raboostCont")
```

varImpStruct-class *Class "varImpStruct" – collect data on variable importance from various machine learning methods*

Description

collects data on variable importance

Objects from the Class

Objects can be created by calls of the form `new("varImpStruct", ...)`. These are matrices of importance measures with separate slots identifying algorithm generating the measures and variable names.

Slots

`.Data`: Object of class `"matrix"` actual importance measures

`method`: Object of class `"character"` tag

`varnames`: Object of class `"character"` conformant vector of names of variables

Extends

Class `"matrix"`, from data part. Class `"structure"`, by class `"matrix"`. Class `"array"`, by class `"matrix"`. Class `"vector"`, by class "matrix", with explicit coerce. Class `"vector"`, by class "matrix", with explicit coerce.

Methods

plot `signature(x = "varImpStruct")`: make a bar plot, you can supply arguments `plat` and `toktype` which will use `lookUp(...,plat,toktype)` from the `annotate` package to translate probe names to, e.g., gene symbols.

show `signature(object = "varImpStruct")`: simple abbreviated display

getVarImp `signature(object = "classifOutput", fixNames="logical")`: extractor of variable importance structure; fixNames parameter is to remove leading X used to make variable names syntactic by randomForest (ca 1/2008). You can set fixNames to false if using hu6800 platform, because all featureNames are syntactic as given.

report `signature(object = "classifOutput", fixNames="logical")`: extractor of variable importance data, with annotation; fixNames parameter is to remove leading X used to make variable names syntactic by randomForest (ca 1/2008). You can set fixNames to false if using hu6800 platform, because all featureNames are syntactic as given.

Examples

```
library(golubEsets)
data(Golub_Merge)
library(hu6800.db)
smallG <- Golub_Merge[1001:1060,]
set.seed(1234)
opar=par(no.readonly=TRUE)
par(las=2, mar=c(10,11,5,5))
rf2 <- MLearn(ALL.AML~., smallG, randomForestI, 1:40, importance=TRUE,
 sampsize=table(smallG$ALL.AML[1:40]), mtry=sqrt(ncol(exprs(smallG))))
plot( getVarImp( rf2, FALSE ), n=10, plat="hu6800", toktype="SYMBOL")
par(opar)
report( getVarImp( rf2, FALSE ), n=10, plat="hu6800", toktype="SYMBOL")
```

`xvalLoop` *Cross-validation in clustered computing environments*

Description

Use cross-validation in a clustered computing environment

Usage

```
xvalLoop( cluster, ... )
```

Arguments

`cluster`	Any S4-class object, used to indicate how to perform clustered computations.
`...`	Additional arguments used to inform the clustered computation.

Details

Cross-validiation usually involves repeated calls to the same function, but with different arguments. This provides an obvious place for using clustered computers to enhance execution. The method xval is structured to exploit this; `xvalLoop` provides an easy mechanism to change how `xval` performs cross-validation.

The idea is to write an `xvalLoop` method that returns a function. The function is then used to execute the cross-validation. For instance, the default method returns the function `lapply`, so the cross-validation is performed by using `lapply`. A different method might return a function that executed lapply-like functions, but sent different parts of the function to different computer nodes.

An accompanying vignette illustrates the technique in greater detail. An effective division of labor is for experienced cluster programmers to write lapply-like methods for their favored clustering environment. The user then only has to add the cluster object to the list of arguments to `xval` to get clustered calculations.

Value

A function taking arguments like those for `lapply`

Examples

```
## Not run:
library(golubEsets)
data(Golub_Merge)
smallG <- Golub_Merge[200:250,]

# Evaluation on one node

lk1 <- xval(smallG, "ALL.AML", knnB, xvalMethod="LOO", group=as.integer(0))
table(lk1,smallG$ALL.AML)
```

```
# Evaluation on several nodes -- a cluster programmer might write the following...

library(snow)
setOldClass("spawnedMPIcluster")

setMethod("xvalLoop", signature( cluster = "spawnedMPIcluster"),
## use the function returned below to evalutae
## the central cross-validation loop in xval
function( cluster, ... ) {
    clusterExportEnv <- function (cl, env = .GlobalEnv)
    {
        unpackEnv <- function(env) {
            for ( name in ls(env) ) assign(name, get(name, env), .GlobalEnv )
            NULL
        }
        clusterCall(cl, unpackEnv, env)
    }
    function(X, FUN, ...) { # this gets returned to xval
        ## send all visible variables from the parent (i.e., xval) frame
        clusterExportEnv( cluster, parent.frame(1) )
        parLapply( cluster, X, FUN, ... )
    }
})

# ... and use the cluster like this...

cl <- makeCluster(2, "MPI")
clusterEvalQ(cl, library(MLInterfaces))

lk1 <- xval(smallG, "ALL.AML", knnB, xvalMethod="LOO", group=as.integer(0), cluster = cl)
table(lk1,smallG$ALL.AML)

## End(Not run)
```

xvalSpec	*container for information specifying a cross-validated machine learning exercise*

Description

container for information specifying a cross-validated machine learning exercise

Usage

```
xvalSpec(type, niter=0,
      partitionFunc = function(data, classLab,iternum) { (seq_len(nrow(data)))[-iternum] },
         fsFun = function(formula, data) formula )
```

Arguments

- `type` — a string, "LOO" indicating leave-one-out cross-validation, or "LOG" indicating leave-out-group, or "NOTEST", indicating the entire dataset is used in a single training run.
- `niter` — numeric specification of the number of cross-validation iterations to use. Ignored if `type` is `"LOO"`.
- `partitionFunc` — function, with parameters data (bound to data.frame), clab (bound to character string), iternum (bound to numeric index into sequence of 1:`niter`). This function's job is to provide the indices of training cases for each cross-validation step. An example is `balKfold.xvspec`, which computes a series of indices that are approximately balanced with respect to frequency of outcome types.
- `fsFun` — function, with parameters formula, data. The function must return a formula suitable for defining a model on the basis of the main input data. A candidate fsFun is given in example for fsHistory function.

Details

If `type == "LOO"`, no other parameters are inspected. If `type == "LOG"` a value for `partitionFunc` must be supplied. We recommend using `balKfold.xvspec(K)`. The values of `niter` and `K` in this usage must be the same. This redundancy will be removed in a future upgrade.

If the `parallel` package is attached and symbol `mc_fork` is loaded, cross-validation will be distributed to cores using `mclapply`.

Value

An instance of `classifierOutput`, with a special structure. The `RObject` return slot is populated with a list of `niter` cross-validation results. Each element of this list is itself a list with two elements: `test.idx` (the indices of the test set for the associated cross-validation iteration, and `mlans`, the `classifierOutput` generated at each iteration. Thus there are `classifierOutput` instances nested within the main `classifierOutput` returned when a `xvalSpec` is used.

Author(s)

Vince Carey <stvjc@channing.harvard.edu>

Examples

```
library("MASS")
data(crabs)
set.seed(1234)
#
# demonstrate cross validation
#
nn1cv = MLearn(sp~CW+RW, data=crabs, nnetI, xvalSpec("LOG",
   5, balKfold.xvspec(5)), size=3, decay=.01 )
nn1cv
confuMat(nn1cv)
names(RObject(nn1cv)[[1]])
RObject(RObject(nn1cv)[[1]]$mlans)
```

Index

∗ **classes**
classifierOutput-class, 5
clusteringOutput-class, 7
learnerSchema-class, 13
projectedLearner-class, 25
raboostCont-class, 29
varImpStruct-class, 29
∗ **classif**
confuMat-methods, 8
MLIntInternals, 19
∗ **datasets**
brennan_2013_tabS7exc, 4
∗ **manip**
balKfold.xvspec, 2
∗ **methods**
confuMat-methods, 8
performance-analytics, 20
planarPlot-methods, 21
xvalLoop, 31
∗ **models**
balKfold.xvspec, 2
fs.absT, 10
fsHistory, 11
hclustWidget, 12
MLearn, 14
plspinHcube, 22
projectLearnerToGrid, 26
RAB, 28
xvalSpec, 32

acc *(performance-analytics)*, 20
acc,table-method *(performance-analytics)*, 20
ada, *15*
adaI *(MLearn)*, 14

bagging, *15*
baggingI *(MLearn)*, 14
balKfold.xvspec, 2, *33*
BgbmI *(MLearn)*, 14
blackboost, *15*
blackboostI *(MLearn)*, 14
brennan_2013_tabS7exc, 4

classifierOutput, *6*, *8*, *11*, *23*, *24*, *33*
classifierOutput-class, 5
classifOutput *(MLIntInternals)*, 19
clusteringOutput-class, 7
clusteringSchema-class *(clusteringOutput-class)*, 7
clustOutput *(MLIntInternals)*, 19
confuMat *(confuMat-methods)*, 8
confuMat,classifierOutput,character-method *(confuMat-methods)*, 8
confuMat,classifierOutput,missing-method *(confuMat-methods)*, 8
confuMat,classifierOutput,numeric-method *(confuMat-methods)*, 8
confuMat,classifierOutput-method *(confuMat-methods)*, 8
confuMat-methods, 8
confuTab, 9
cverrs *(MLIntInternals)*, 19

DAB *(RAB)*, 28
daboostCont-class *(raboostCont-class)*, 29
diagDA, *15*
dlda *(MLearn)*, 14
dlda2 *(MLearn)*, 14
dldaI *(MLearn)*, 14

es2df *(MLIntInternals)*, 19

F1 *(performance-analytics)*, 20
F1,table-method *(performance-analytics)*, 20
fn, *9*
fn *(performance-analytics)*, 20
fn,table-method *(performance-analytics)*, 20

fp, *9*
fp *(*performance-analytics*)*, 20
fp,table-method
*(*performance-analytics*)*, 20
fs.absT, 10
fs.probT *(*fs.absT*)*, 10
fs.topVariance *(*fs.absT*)*, 10
fsHistory, 11
fsHistory,classifierOutput-method
*(*classifierOutput-class*)*, 5

gbm, *15*
gbm2 *(*MLearn*)*, 14
getConverter *(*clusteringOutput-class*)*, 7
getConverter,clusteringSchema-method
*(*clusteringOutput-class*)*, 7
getDist *(*clusteringOutput-class*)*, 7
getDist,clusteringSchema-method
*(*clusteringOutput-class*)*, 7
getGrid *(*MLIntInternals*)*, 19
getGrid,data.frame-method
*(*MLIntInternals*)*, 19
getGrid,ExpressionSet-method
*(*MLIntInternals*)*, 19
getVarImp *(*varImpStruct-class*)*, 29
getVarImp,classifierOutput,logical-method
*(*varImpStruct-class*)*, 29
getVarImp,classifierOutput,missing-method
*(*varImpStruct-class*)*, 29
getVarImp,classifOutput,logical-method
*(*varImpStruct-class*)*, 29
glm, *15*
glmI.logistic *(*MLearn*)*, 14
groupIndex *(*MLIntInternals*)*, 19

hclust, *15*
hclustConverter *(*MLIntInternals*)*, 19
hclustI *(*MLearn*)*, 14
hclustWidget, 12

kmeans, *15*
kmeansConverter *(*MLIntInternals*)*, 19
kmeansI *(*MLearn*)*, 14
knn, *15*
knn.cv, *15*
knn.cv2 *(*MLearn*)*, 14
knn.cvI *(*MLearn*)*, 14
knn2 *(*MLearn*)*, 14
knnI *(*MLearn*)*, 14
ksvm, *15*
ksvm2 *(*MLearn*)*, 14
ksvmI *(*MLearn*)*, 14

lapply, *31*
lda, *15*
ldaI *(*MLearn*)*, 14
learnerIn3D *(*projectedLearner-class*)*, 25
learnerIn3D,projectedLearner-method
*(*projectedLearner-class*)*, 25
learnerSchema-class, 13
list, *29*
lvq *(*MLearn*)*, 14
lvqI *(*MLearn*)*, 14
lvqtest, *15*

macroF1 *(*performance-analytics*)*, 20
macroF1,classifierOutput,character-method
*(*performance-analytics*)*, 20
macroF1,classifierOutput,missing-method
*(*performance-analytics*)*, 20
macroF1,classifierOutput,numeric-method
*(*performance-analytics*)*, 20
macroF1,numeric,numeric-method
*(*performance-analytics*)*, 20
macroF1,table,missing-method
*(*performance-analytics*)*, 20
macroF1-methods
*(*performance-analytics*)*, 20
makeClusteringSchema *(*MLIntInternals*)*,
19
makeConfuMat *(*MLIntInternals*)*, 19
makeLearnerSchema *(*MLearn*)*, 14
mapPSvec *(*MLIntInternals*)*, 19
mclapply, *15*, *33*
membMat *(*MLIntInternals*)*, 19
mkfmla *(*RAB*)*, 28
mlbench.hypercube, *23*
MLearn, *10*, 14, *24*
MLearn,formula,data.frame,clusteringSchema,ANY-method
*(*MLearn*)*, 14
MLearn,formula,data.frame,learnerSchema,numeric-method
*(*MLearn*)*, 14
MLearn,formula,data.frame,learnerSchema,xvalSpec-method
*(*MLearn*)*, 14
MLearn,formula,ExpressionSet,character,numeric-method
*(*MLearn*)*, 14
MLearn,formula,ExpressionSet,learnerSchema,numeric-method
*(*MLearn*)*, 14

MLearn,formula,ExpressionSet,learnerSchema,xvalSpec-method (MLearn), 14
MLearn,formula,SummarizedExperiment,learnerSchema,numeric-method (MLearn), 14
MLearn_new (MLearn), 14
mlearnWidget (hclustWidget), 12
MLIConverter.Bgbm (MLIntInternals), 19
MLIConverter.blackboost (MLIntInternals), 19
MLIConverter.dlda (MLIntInternals), 19
MLIConverter.knn (MLIntInternals), 19
MLIConverter.knncv (MLIntInternals), 19
MLIConverter.ksvm (MLIntInternals), 19
MLIConverter.ldaPredMeth (MLIntInternals), 19
MLIConverter.logistic (MLIntInternals), 19
MLIConverter.naiveBayes (MLIntInternals), 19
MLIConverter.nnet (MLIntInternals), 19
MLIConverter.plsda (MLIntInternals), 19
MLIConverter.RAB (MLIntInternals), 19
MLIConverter.randomForest (MLIntInternals), 19
MLIConverter.rpart (MLIntInternals), 19
MLIConverter.selftesting (MLIntInternals), 19
MLIConverter.slda (MLIntInternals), 19
MLIConverter.svm (MLIntInternals), 19
MLIConverterListEl.class (MLIntInternals), 19
MLIConverterPredType.class (MLIntInternals), 19
MLIntInternals, 19
MLIPredicter.knn (MLIntInternals), 19
MLIPredicter.ksvm (MLIntInternals), 19
MLIPredicter.naiveBayes (MLIntInternals), 19
MLIPredicter.nnet (MLIntInternals), 19
MLIPredicter.plsda (MLIntInternals), 19
MLIPredicter.randomForest (MLIntInternals), 19
MLIPredicter.svm (MLIntInternals), 19
MLLabel (MLIntInternals), 19
MLOutput (MLIntInternals), 19
MLScore (MLIntInternals), 19

naAs0 (MLIntInternals), 19
naiveBayes, *15*
naiveBayesI (MLearn), 14
nnet, *15*
nnetI (MLearn), 14
nonstandardLearnerSchema-class (learnerSchema-class), 13

pamConverter (MLIntInternals), 19
pamI (MLearn), 14
partPlot (MLIntInternals), 19
performance-analytics, 20
planarPlot (planarPlot-methods), 21
planarPlot,classifierOutput,data.frame,character-method (planarPlot-methods), 21
planarPlot,classifierOutput,ExpressionSet,character-method (planarPlot-methods), 21
planarPlot-methods, 21
planarPlot2 (MLIntInternals), 19
plot (varImpStruct-class), 29
plot,clusteringOutput,ANY-method (clusteringOutput-class), 7
plot,projectedLearner,ANY-method (projectedLearner-class), 25
plot,varImpStruct,ANY-method (varImpStruct-class), 29
plot,varImpStruct-method (varImpStruct-class), 29
plotOne (projectedLearner-class), 25
plotOne,projectedLearner-method (projectedLearner-class), 25
plotXvalRDA (MLearn), 14
plsda2 (MLearn), 14
plsdaI (MLearn), 14
plspinDF (MLIntInternals), 19
plspinHcube, 22
prcomp-class (clusteringOutput-class), 7
prcompObj-class (clusteringOutput-class), 7
precision (performance-analytics), 20
precision,classifierOutput,character-method (performance-analytics), 20
precision,classifierOutput,missing-method (performance-analytics), 20
precision,classifierOutput,numeric-method (performance-analytics), 20
precision,table,missing-method (performance-analytics), 20
precision-methods (performance-analytics), 20
Predict (RAB), 28

Predict,daboostCont-method *(RAB)*, 28
Predict,raboostCont-method *(RAB)*, 28
predict.classifierOutput, 23
predict.dlda2 *(MLIntInternals)*, 19
predict.gbm2 *(MLIntInternals)*, 19
predict.knn.cv2 *(MLIntInternals)*, 19
predict.knn2 *(MLIntInternals)*, 19
predict.lvq *(MLIntInternals)*, 19
predict.RAB *(MLIntInternals)*, 19
predictions *(classifierOutput-class)*, 5
predictions,classifierOutput-method *(classifierOutput-class)*, 5
predScore *(classifierOutput-class)*, 5
predScore,classifierOutput-method *(classifierOutput-class)*, 5
predScores *(classifierOutput-class)*, 5
predScores,classifierOutput-method *(classifierOutput-class)*, 5
probArray *(MLIntInternals)*, 19
probMat *(MLIntInternals)*, 19
projectedLearner-class, 25
projectLearnerToGrid, *25*, 26

qda, *15*
qdaI *(MLearn)*, 14
qualScore *(MLIntInternals)*, 19

RAB, 28
rab *(MLearn)*, 14
RAB4es *(RAB)*, 28
RABI *(MLearn)*, 14
raboostCont-class, 29
randomForest, *15*
randomForestI, *12*
randomForestI *(MLearn)*, 14
recall *(performance-analytics)*, 20
recall,classifierOutput,character-method *(performance-analytics)*, 20
recall,classifierOutput,missing-method *(performance-analytics)*, 20
recall,classifierOutput,numeric-method *(performance-analytics)*, 20
recall,table,missing-method *(performance-analytics)*, 20
recall-methods *(performance-analytics)*, 20
report *(varImpStruct-class)*, 29
report,varImpStruct-method *(varImpStruct-class)*, 29
RObject *(classifierOutput-class)*, 5
RObject,classifierOutput-method *(classifierOutput-class)*, 5
RObject,clusteringOutput-method *(clusteringOutput-class)*, 7
rpart, *15*
rpartI *(MLearn)*, 14

se2df *(MLIntInternals)*, 19
sensitivity *(performance-analytics)*, 20
sensitivity,classifierOutput,character-method *(performance-analytics)*, 20
sensitivity,classifierOutput,missing-method *(performance-analytics)*, 20
sensitivity,classifierOutput,numeric-method *(performance-analytics)*, 20
sensitivity,table,missing-method *(performance-analytics)*, 20
sensitivity-methods *(performance-analytics)*, 20
shinyApp, *23*
show,classifierOutput-method *(classifierOutput-class)*, 5
show,clusteringOutput-method *(clusteringOutput-class)*, 7
show,clusteringSchema-method *(clusteringOutput-class)*, 7
show,learnerSchema-method *(learnerSchema-class)*, 13
show,projectedLearner-method *(projectedLearner-class)*, 25
show,raboostCont-method *(raboostCont-class)*, 29
show,varImpStruct-method *(varImpStruct-class)*, 29
silhouette-class *(clusteringOutput-class)*, 7
silhouetteVec *(MLIntInternals)*, 19
slda, *15*
sldaI *(MLearn)*, 14
SOMBout *(MLIntInternals)*, 19
somout *(MLIntInternals)*, 19
specificity *(performance-analytics)*, 20
specificity,table-method *(performance-analytics)*, 20
standardMLIConverter *(MLearn)*, 14
svm, *15*
svm2 *(MLearn)*, 14
svmI *(MLearn)*, 14

testPredictions
(classifierOutput-class), 5
testPredictions,classifierOutput-method
(classifierOutput-class), 5
testScores (classifierOutput-class), 5
testScores,classifierOutput-method
(classifierOutput-class), 5
tn, *9*
tn (performance-analytics), 20
tn,table-method
(performance-analytics), 20
tonp (RAB), 28
tp, *9*
tp (performance-analytics), 20
tp,table-method
(performance-analytics), 20
trainInd (classifierOutput-class), 5
trainInd,classifierOutput-method
(classifierOutput-class), 5
trainPredictions
(classifierOutput-class), 5
trainPredictions,classifierOutput-method
(classifierOutput-class), 5
trainScores (classifierOutput-class), 5
trainScores,classifierOutput-method
(classifierOutput-class), 5

varImpStruct-class, 29
vector, *29*

x (MLIntInternals), 19
xvalLoop, 31
xvalLoop,ANY-method (xvalLoop), 31
xvalSpec, *3*, *5*, *10*, 32
xvalSpec-class (xvalSpec), 32