Package 'exprso'

July 26, 2018

Title Rapid Deployment of Machine Learning Algorithms

Version 0.4.0

URL http://github.com/tpq/exprso

BugReports http://github.com/tpq/exprso/issues

Description Supervised machine learning has an increasingly important role in data analysis. This package introduces a framework for rapidly building and deploying supervised machine learning in a high-throughput manner. This package provides a user-friendly interface that empowers investigators to execute state-of-the-art binary and multi-class classification, as well as regression, with minimal programming experience necessary.

License GPL-2

LazyData TRUE

VignetteBuilder knitr

RoxygenNote 6.0.1

Imports cluster, MASS, e1071, frbs, lattice, methods, nnet, plyr, randomForest, ROCR, rpart, sampling, stats

Depends R (>= 3.2.2), kernlab

Suggests Biobase, edgeR, GEOquery, h2o, knitr, limma, magrittr, mRMRe, pathClass, propr, RankProd, rmarkdown, testthat

NeedsCompilation no

Author Thomas Quinn [aut, cre], Daniel Tylee [ctb], Samuel Lee [ctb]

Maintainer Thomas Quinn <contacttomquinn@gmail.com>

Repository CRAN

Date/Publication 2018-07-25 22:20:03 UTC

R topics documented:

`array` *Sample ExprsBinary Data*

Description

Sample ExprsBinary Data

Usage

```
data(array)
```

Format

An object of class `ExprsBinary` of length 1.

`arrayExprs` *Import Data as ExprsArray*

Description

A convenience function that builds an `ExprsArray` object. This function is no longer supported. Please use `exprso` instead.

Usage

```
arrayExprs(object, colBy, include, colID, begin, ...)
```

Arguments

`object`	What to import as an `ExprsArray` object. See Details.
`colBy`	A numeric or character index. The column that contains group annotations.
`include`	A list of character vectors. Specifies which annotations in `colBy` to include in which groups. Each element of the list specifies a unique group while each element of the character vector specifies which annotations define that group. For binary classification, the first list element defines the negative, or control, group.
`colID`	A numeric or character index. The column used to name subjects. For `data.frame` or file import only.

begin	A numeric scalar. The j-th column at which feature data starts. For data.frame or file import only.
...	Additional arguments passed along to read.delim. For file import only.

Details

Importing a data.frame object:

This function expects that the imported data.frame has the following format: rows indicate subject entries while columns indicate measured variables. The first several columns should contain annotation information (e.g., age, sex, diagnosis). The remaining columns should contain feature data (e.g., expression values). The argument begin defines the j-th column at which the feature data starts. This function automatically removes any features with NA values. Take care to remove any factor columns before importing.

Importing an ExpressionSet object:

The package Biobase maintains a popular class object called ExpressionSet that often gets used to store expression data. This function converts this eSet object into an ExprsArray object. This function automatically removes any features with NA values.

Importing a file:

arrayExprs can also build an ExprsArray object from a tab-delimited data file, passing along the file and ... argument(s) to read.delim. All rules for data.frame import also apply here. By default, arrayExprs forces stringsAsFactors = FASE.

Value

An ExprsArray object.

See Also

ExprsArray-class, GSE2eSet

---

arrayMulti *Sample ExprsMulti Data*

Description

Sample ExprsMulti Data

Usage

```
data(arrayMulti)
```

Format

An object of class ExprsMulti of length 1.

build *Build Models*

Description

The exprso package includes these build modules:

- buildNB
- buildLDA
- buildSVM
- buildLM
- buildGLM
- buildLR
- buildANN
- buildDT
- buildRF
- buildFRB
- buildDNN

Details

In the case of multi-class classification, each build module can harness the doMulti function to perform "1 vs. all" classifier construction. In the setting of four class labels, a single build call will return four classifiers that work in concert to make a single prediction of an unlabelled subject. For building multiple classifiers across a vast parameter space in a high-throughput manner, see pl.

Like fs methods, build methods have a top argument which allows the user to specify which features to feed INTO the model build. This effectively provides the user with one last opportunity to subset the feature space based on prior feature selection or dimension reduction. For all build methods, @preFilter and @reductionModel will get passed along to the resultant ExprsModel object, again ensuring that any test or validation sets will undergo the same feature selection and dimension reduction in the appropriate steps when deploying the model. Set top = 0 to pass all features through a build method.

build. *Workhorse for build Methods*

Description

Used as a back-end wrapper for creating new build methods.

Usage

```
build.(object, top, uniqueFx, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
uniqueFx	A function call unique to the method.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

---

buildANN *Build Artificial Neural Network Model*

Description

buildANN builds a model using the nnet function from the nnet package.

Usage

```
buildANN(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

buildDNN *Build Deep Neural Network Model*

Description

buildDNN builds a model using the h2o.deeplearning function from the h2o package.

Usage

```
buildDNN(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

buildDT *Build Decision Tree Model*

Description

buildDT builds a model using the rpart function from the rpart package.

Usage

```
buildDT(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Details

Provide `cp` as a numeric scalar to trim the `rpart` decision tree. If provided, this argument is passed to the `rpart::prune` function. Set `cp = 0` to skip pruning (default behavior).

Value

Returns an `ExprsModel` object.

`buildEnsemble` *Build Ensemble*

Description

`buildEnsemble` builds an ensemble from `ExprsModel` or `ExprsPipeline` objects. See Details.

Usage

```
buildEnsemble(object, ...)

## S4 method for signature 'ExprsModel'
buildEnsemble(object, ...)

## S4 method for signature 'ExprsPipeline'
buildEnsemble(object, colBy = 0, how = 0,
  gate = 0, top = 0)
```

Arguments

`object`	An `ExprsPipeline-class` object.
`...`	Additional `ExprsModel` objects to use in the ensemble. Argument applies to the `ExprsModel-class` method only.
`colBy`	A character vector or string. Specifies column(s) to use when filtering by model performance. Listing multiple columns will result in a filter based on the product all listed columns.
`how`	A numeric scalar. Arguments between 0 and 1 will impose a threshold or ceiling filter, respectively, based on the raw value of `colBy`. Arguments between 1 and 100 will impose a filter based on the percentile of `colBy`. The user may also provide "midrange", "median", or "mean" as an argument for these filters.
`gate`	A numeric scalar. Arguments between 0 and 1 will impose a threshold or ceiling filter, respectively, based on the raw value of `colBy`. Arguments between 1 and 100 will impose a filter based on the percentile of `colBy`. The user may also provide "midrange", "median", or "mean" as an argument for these filters.
`top`	A numeric scalar. Determines the top N models based on `colBy` to include after the threshold and ceiling filters. In the case that the `@summary` slot contains the column "boot", this selects the top N models for each unique bootstrap.

Details

This function can combine any number of model objects into an ensemble. These models do not necessarily have to derive from the same `build` method. In this way, it works like `conjoin`.

This function can also build an ensemble from pipeline objects. It does this by calling `pipeFilter`, then joining the remaining models into an ensemble. As an adjunct to this method, consider first combining multiple pipeline objects with `conjoin`.

Value

An `ExprsEnsemble-class` object.

Methods (by class)

- `ExprsModel`: Method to build ensemble from `ExprsModel` objects.
- `ExprsPipeline`: Method to build ensemble from `ExprsPipeline` objects.

`buildFRB` *Build Fuzzy Rule Based Model*

Description

`buildFRB` builds a model using the `frbs` function from the `frbs` package.

Usage

```
buildFRB(object, top = 0, ...)
```

Arguments

`object`	An `ExprsArray` object. The training set.
`top`	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set `top = 0` to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
`...`	Arguments passed to the detailed function.

Value

Returns an `ExprsModel` object.

buildGLM *Build Generalized Linear Model*

Description

buildGLM builds a model using the glm function.

Usage

```
buildGLM(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

buildLDA *Build Linear Discriminant Analysis Model*

Description

buildLDA builds a model using the lda function from the MASS package.

Usage

```
buildLDA(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

---

buildLM *Build Linear Model*

Description

buildLM builds a model using the lm function.

Usage

```
buildLM(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

---

buildLR *Build Logistic Regression Model*

Description

buildLR builds a model using the glm function.

Usage

```
buildLR(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

---

buildNB *Build Naive Bayes Model*

Description

buildNB builds a model using the naiveBayes function from the e1071 package.

Usage

```
buildNB(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

buildRF *Build Random Forest Model*

Description

buildRF builds a model using the randomForest function from the randomForest package.

Usage

```
buildRF(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

buildSVM *Build Support Vector Machine Model*

Description

buildSVM builds a model using the svm function from the e1071 package.

Usage

```
buildSVM(object, top = 0, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
...	Arguments passed to the detailed function.

Value

Returns an ExprsModel object.

---

calcMonteCarlo *Calculate* plMonteCarlo *Performance*

Description

calcMonteCarlo calculates a single performance measure for the results of a plMonteCarlo function call.

Usage

```
calcMonteCarlo(pl, colBy = "valid.acc")
```

Arguments

pl	Specifies the ExprsPipeline object returned by plMonteCarlo.
colBy	A character vector or string. Specifies column(s) to use when summarizing model performance. Listing multiple columns will calculate performance as a product of those listed performances.

Details

For each dataset split (i.e., bootstrap), calcMonteCarlo averages the validation set performance for the "best" model (where "best" is defined as the model with the maximum "internal" cross-validation accuracy, max($train.plCV)). The validation set performance ultimately averaged depends on the supplied colBy argument.

Value

A numeric scalar. The cross-validation accuracy.

---

calcNested *Calculate* plNested *Performance*

Description

calcNested calculates a single performance measure for the results of a plNested function call.

Usage

```
calcNested(pl, colBy = "valid.acc")
```

Arguments

pl	Specifies the ExprsPipeline object returned by plNested.
colBy	A character vector or string. Specifies column(s) to use when summarizing model performance. Listing multiple columns will calculate performance as a product of those listed performances.

Details

For each dataset split (i.e., bootstrap), calcNested averages the validation set performance for the "best" model (where "best" is defined as the model with the maximum "internal" cross-validation accuracy, max($train.plCV)). The validation set performance ultimately averaged depends on the supplied colBy argument.

Value

A numeric scalar. The cross-validation accuracy.

---

calcStats *Calculate Model Performance*

Description

calcStats calculates the performance of a deployed model.

Usage

```
calcStats(object, aucSkip = FALSE, plotSkip = FALSE, verbose = TRUE)

## S4 method for signature 'ExprsPredict'
calcStats(object, aucSkip = FALSE,
  plotSkip = FALSE, verbose = TRUE)

## S4 method for signature 'RegrsPredict'
calcStats(object, aucSkip = FALSE,
  plotSkip = FALSE, verbose = TRUE)
```

Arguments

object	An ExprsPredict or RegrsPredict object.
aucSkip	A logical scalar. Toggles whether to calculate area under the receiver operating characteristic curve. See Details.
plotSkip	A logical scalar. Toggles whether to plot the receiver operating characteristic curve. See Details.
verbose	A logical scalar. Toggles whether to print the results of model performance to console.

Details

For classification, if the argument `aucSkip = FALSE` AND the `ExprsArray` object was an `ExprsBinary` object with at least one case and one control AND `ExprsPredict` contains a coherent `@probability` slot, `calcStats` will calculate classifier performance using the area under the receiver operating characteristic (ROC) curve via the `ROCR` package. Otherwise, `calcStats` will calculate classifier performance traditionally using a confusion matrix. Note that accuracies calculated using `ROCR` may differ from those calculated using a confusion matrix because `ROCR` adjusts the discrimination threshold to optimize sensitivity and specificity. This threshold is automatically chosen as the point along the ROC which minimizes the Euclidean distance from (0, 1).

For regression, accuracy is defined as (1 - MSE / (1 + MSE)). This allows MSE to range from 0 to 1 for use with `pl` and `pipe`. Note that the `aucSkip` and `plotSkip` arguments are ignored for regression.

Value

Returns a `data.frame` of performance metrics.

Methods (by class)

- `ExprsPredict`: Method to calculate performance for classification models.
- `RegrsPredict`: Method to calculate performance for continuous outcome models.

`check.ctrlGS` *Check* `ctrlGS` *Arguments*

Description

This function ensures that the list of arguments for `ctrlGS` meets the criteria required by the `plNested` function. This function forces `aucSkip = TRUE` and `plotSkip = TRUE`.

Usage

```
check.ctrlGS(args)
```

Arguments

`args`	A list of arguments to check.

classCheck *Class Check*

Description

Checks whether an object belongs to a specified class. For back-end use only.

Usage

```
classCheck(x, what, msg)
```

Arguments

x	An object.
what	A character vector. The classes any of which x should have.
msg	A string. An error message if x is not what.

compare *Compare* ExprsArray *Objects*

Description

This method compares the values of all ExprsArray annotations across a specified annotation term for up to two ExprsArray objects. Depending on the composition of each annotation, compare will perform either a chi-squared test or an ANOVA test.

Usage

```
compare(object, array.valid = NULL, colBy = "defineCase", cutoff = 0.05)

## S4 method for signature 'ExprsArray'
compare(object, array.valid = NULL,
  colBy = "defineCase", cutoff = 0.05)
```

Arguments

object	The ExprsArray object used when comparing annotations.
array.valid	A second ExprsArray object used when comparing annotations. Optional. Exclude with array.valid = NULL.
colBy	A character string. The annotation column against which to compare all other annotation terms (i.e., to test as the independent variable).
cutoff	A numeric scalar. The p-value cutoff that determines when the annotation test returns a TRUE result

Details

This method performs two kinds of comparisons. First, it tests all annotation variables against the annotation supplied by the `colBy` argument for each provided `ExprsArray` object. In other words, the `colBy` argument determines which annotation to use as the independent variable for "internal" comparisons. Second, it tests all annotation variables between the provided `ExprsArray` objects. Providing `array.valid = NULL` will skip the between comparisons.

This method will test annotations using either a chi-squared test or an ANOVA test depending on the class of the values stored by the tested column. The presence of a "character" or "factor" in the tested column will trigger a chi-squared test. As such, this method requires the user to select a `colBy` annotation that contains categorical data (i.e., to use as the independent variable).

We anticipate that this method will serve as a useful adjunct to `modCluster`. However, it may also help in quickly determining whether the data `split` has yielded comparable training and test sets in terms of the annotations included in `@annot`.

Value

A list of three logical vectors. The first and second elements of the list correspond to "internal" comparisons for the two provided `ExprsArray` objects, respectively. The third element of the list corresponds to comparisons made between the provided objects.

Methods (by class)

- `ExprsArray`: Method to compare `ExprsArray` objects.

`conjoin` *Combine* `exprso` *Objects*

Description

`conjoin` combines two or more `exprso` objects based on their class.

Usage

```
conjoin(object, ...)

## S4 method for signature 'ExprsArray'
conjoin(object, ...)

## S4 method for signature 'ExprsModel'
conjoin(object, ...)

## S4 method for signature 'ExprsPipeline'
conjoin(object, ...)

## S4 method for signature 'ExprsEnsemble'
conjoin(object, ...)
```

Arguments

object	An ExprsArray, ExprsModel, ExprsPipeline, or ExprsEnsemble object.
...	Two or more objects of the same class.

Details

When applied to two or more ExprsArray objects, this function returns one ExprsArray object as output. This only works on ExprsArray objects that have not undergone feature selection. Any missing annotations in @annot will get replaced with NA values. Note that all combined ExprsArray objects must initially have had the same features in the same order.

When applied to two or more ExprsModel objects, this function returns one ExprsEnsemble object as output. In this way, this function works similar to the buildEnsemble method for ExprsModel objects.

When applied to two or more ExprsPipeline objects, this function returns one ExprsPipeline object as output. To keep track of which ExprsPipeline objects contributed initially to the resultant object, the source gets flagged in the summary slot. For each ExprsPipeline object, if the summary lacks a boot column, all summary entries will receive one unique ID. However, if the summary contains a boot column (e.g., as generated by plMonteCarlo), all models belonging to each bootstrap will receive one unique ID. Afterwards, the old boot columns will get renamed to unboot while the newly assigned unique IDs become the new boot column. This complicated indexing system treats all models derived from one unique cut to a training set as if they had belonged to the same "pseudo-bootstrap". These "pseudo-bootstraps" will get handled like true bootstraps downstream by functions built around pipeFilter and buildEnsemble.

When applied to two or more ExprsEnsemble objects, this function returns one ExprsEnsemble object as output. The resultant object contains all models found within each of the supplied ExprsEnsemble objects.

Value

An ExprsArray, ExprsModel, ExprsPipeline, or ExprsEnsemble object.

Methods (by class)

- ExprsArray: Method to join ExprsArray objects.
- ExprsModel: Method to join ExprsModel objects.
- ExprsPipeline: Method to join ExprsPipeline objects.
- ExprsEnsemble: Method to join ExprsEnsemble objects.

---

ctrlFeatureSelect *Manage* fs *Arguments*

Description

This function organizes fs arguments passed to pl functions.

Usage

```
ctrlFeatureSelect(func, top = 0, ...)
```

Arguments

func	A character string. The fs function to call.
top	Argument passed to the fs function.
...	Additional arguments passed to the fs function.

Value

A list of arguments.

---

ctrlGridSearch *Manage* plGrid *Arguments*

Description

This function organizes plGrid arguments passed to pl functions.

Usage

```
ctrlGridSearch(func, top = 0, ...)
```

Arguments

func	A character string. The pl function to call.
top	Argument passed to the pl function. Leave missing when handling plMonteCarlo or plNested arguments.
...	Additional arguments passed to the pl function.

Value

A list of arguments.

ctrlModSet *Manage* mod *Arguments*

Description

This function organizes mod arguments passed to pl functions.

Usage

```
ctrlModSet(func, ...)
```

Arguments

func	A character string. The mod function to call.
...	Additional arguments passed to the mod function.

Value

A list of arguments.

ctrlSplitSet *Manage* split *Arguments*

Description

This function organizes split arguments passed to pl functions.

Usage

```
ctrlSplitSet(func, percent.include = 67, ...)
```

Arguments

func	A character string. The split function to call.
percent.include	Argument passed to the split function.
...	Additional arguments passed to the split function.

Value

A list of arguments.

defaultArg *Set an args List Element to Default Value*

Description

Set an args List Element to Default Value

Usage

```
defaultArg(what, as, args, verbose = TRUE)
```

Arguments

what	The name of the argument.
as	The value to set it as.
args	An args list. The result of getArgs.
verbose	A boolean. Toggles whether to alert the user that an argument is set.

doMulti *Perform Multiple "1 vs. all" Tasks*

Description

A function to execute multiple "1 vs. all" binary tasks.

Usage

```
doMulti(object, top = 0, method, ...)
```

Arguments

object	An ExprsArray object. The training set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
method	A character string. The method to apply.
...	Arguments passed to the detailed function.

Details

doMulti runs once for each factor level in the "defineCase" column. If a training set is missing any one of the factor levels (e.g., owing to random cuts during cross-validation), the ExprsModule component that would refer to that class label gets replaced with an NA placeholder. Note that this NA placeholder will prevent a classifier from possibly predicting the NA class (i.e., a classifier can only make predictions about class labels that it "knows"). However, these "unknown" classes still impact metrics of classifier performance. Otherwise, see exprso-predict.

Value

A list of the results from method.

ExprsArray-class *An S4 class to store feature and annotation data*

Description

An S4 class to store feature and annotation data

Usage

```
## S4 method for signature 'ExprsArray'
show(object)

## S4 method for signature 'ExprsArray,ANY,ANY'
x[i, j]

## S4 method for signature 'ExprsArray'
x$name

## S4 method for signature 'ExprsArray'
subset(x, subset, select)

## S4 method for signature 'ExprsArray,missing'
plot(x, y, a = 1, b = 2, c = 3, ...)

## S4 method for signature 'ExprsArray'
summary(object)

## S4 method for signature 'ExprsArray'
getFeatures(object)
```

Arguments

object, x	An object of class ExprsArray.
i, j	Subsets entire ExprsArray object via object@annot[i, j]. Returns object@annot[, j] if argument i is missing.

name	Returns object@annot[, name].
subset	Subsets entire ExprsArray object via object@annot[subset,]. Can be used to rearrange feature order.
select	Subsets entire ExprsArray object via object@annot[, select]. Can be used to rearrange subject order.
y	Leave missing. Argument exists because of plot generic definition.
a, b, c	A numeric scalar. Indexes the first, second, and third dimensions to plot. Set c = 0 to plot two dimensions.
...	Additional arguments passed toplot or lattice::cloud.

Methods (by generic)

- show: Method to show ExprsArray object.
- [: Method to subset ExprsArray object.
- $: Method to subset ExprsArray object.
- subset: Method to subset ExprsArray object.
- plot: Method to plot two or three dimensions of data.
- summary: Method to plot summary graphs for a sub-sample of feature data.
- getFeatures: Method to return features within an ExprsArray object.

Slots

exprs A matrix. Stores the feature data.

annot A data.frame. Stores the annotation data.

preFilter Typically a list. Stores feature selection history.

reductionModel Typically a list. Stores dimension reduction history.

See Also

ExprsArray-class
ExprsModel-class
ExprsPipeline-class
ExprsEnsemble-class
ExprsPredict-class
RegrsPredict-class

ExprsBinary-class *An S4 class to store feature and annotation data*

Description

An ExprsArray sub-class for data with binary class outcomes.

ExprsEnsemble-class *An S4 class to store multiple models*

Description

An S4 class to store multiple models

Usage

```
## S4 method for signature 'ExprsEnsemble'
show(object)

## S4 method for signature 'ExprsEnsemble'
getFeatures(object, index)
```

Arguments

object	An ExprsArray, ExprsModel, ExprsPipeline, or ExprsEnsemble object.
index	A numeric scalar. The i-th model from which to retrieve features. If missing, getFeatures will tabulate features across all models.

Methods (by generic)

- show: Method to show ExprsEnsemble object.
- getFeatures: Method to return features within an ExprsEnsemble model.

Slots

machs Typically a list. Stores the models.

See Also

ExprsArray-class
ExprsModel-class
ExprsPipeline-class
ExprsEnsemble-class
ExprsPredict-class
RegrsPredict-class

ExprsMachine-class *An S4 class to store the model*

Description

An ExprsModel sub-class for dichotomous models.

ExprsModel-class *An S4 class to store the model*

Description

An S4 class to store the model

Usage

```
## S4 method for signature 'ExprsModel'
show(object)

## S4 method for signature 'ExprsModel'
getFeatures(object)
```

Arguments

object	An object of class ExprsModel.

Methods (by generic)

- show: Method to show ExprsModel object.
- getFeatures: Method to return features within an ExprsModel object.

Slots

preFilter Typically a list. Stores feature selection history.

reductionModel Typically a list. Stores dimension reduction history.

mach Typically an S4 class. Stores the model.

See Also

ExprsArray-class
ExprsModel-class
ExprsPipeline-class
ExprsEnsemble-class
ExprsPredict-class
RegrsPredict-class

ExprsModule-class *An S4 class to store the model*

Description

An ExprsModel sub-class for multi-class models.

ExprsMulti-class *An S4 class to store feature and annotation data*

Description

An ExprsArray sub-class for data with multiple class outcomes.

exprso *The* exprso *Package*

Description

Welcome to the exprso package!

The exprso function imports data into the learning environment.

See mod to process the data.

See split to split off a test set.

See fs to select features.

See build to build models.

See pl to build models high-throughput.

See pipe to process pipelines.

See buildEnsemble to build ensembles.

See exprso-predict to deploy models.

See conjoin to merge objects.

Usage

```
exprso(x, y, label = 1, switch = FALSE)
```

Arguments

x	A matrix of feature data for all samples. Rows should contain samples and columns should contain features.
y	A vector of outcomes for all samples. If class(y) == "character" or class(y) == "factor", exprso prepares data for binary or multi-class classification. Else, exprso prepares data for regression. If y is a matrix, the program uses the column in label.
label	A numeric scalar or character string. The column to use as the label if y is a matrix.
switch	A logical scalar. Toggles which class label is called Control in binary classification.

Value

An ExprsArray object.

Examples

```
## Not run:
library(exprso)
data(iris)
array <- exprso(iris[,1:4], iris[,5])
arrays <- splitSample(array, percent.include = 67)
array.train <- fsANOVA(arrays[[1]], top = 0)
array.train <- fsPrcomp(array.train, top = 3)
mach <- buildSVM(array.train, top = 5, kernel = "linear", cost = 1)
predict(mach, arrays[[2]])

## End(Not run)
```

---

exprso-predict *Deploy Model*

Description

Deploy a model to predict outcomes from the data.

Usage

```
## S4 method for signature 'ExprsMachine'
predict(object, array, verbose = TRUE)

## S4 method for signature 'ExprsModule'
predict(object, array, verbose = TRUE)

## S4 method for signature 'RegrsModel'
predict(object, array, verbose = TRUE)

## S4 method for signature 'ExprsEnsemble'
predict(object, array, how = "probability",
  verbose = TRUE)
```

Arguments

object	An ExprsModel or ExprsEnsemble object.
array	An ExprsArray object. The target data.
verbose	A logical scalar. Argument passed to calcStats.
how	A character string. Select from "probability" or "majority". See Details. Argument applies to binary classifier ensembles only.

Details

Models can only get deployed on an object of the type used to build the model. Binary classification and regression are handled natively by the machine learning algorithm chosen. Multi-class classification is handled by `doMulti`. Note that a validation set should never get modified once separated from the training set. See `buildEnsemble` to learn about ensembles.

For binary classifier ensembles, when `how = "probability"`, outcomes are based on the average class probability (via `@probability`) estimated by each deployed model. When `how = "majority"`, outcomes are based on consensus voting whereby each deployed model casts a single (all-or-nothing) vote (via `@pred`) in a winner takes all approach. In both scenarios, ties get broken randomly (as weighted by class).

For multi-class classifier ensembles, outcomes are based on the `how = "majority"` method from above. For regression ensembles, outcomes are based on the average predicted value.

Value

Returns an `ExprsPredict` or `RegrsPredict` object.

ExprsPipeline-class *An S4 class to store models built during high-throughput learning*

Description

An S4 class to store models built during high-throughput learning

Usage

```
## S4 method for signature 'ExprsPipeline'
show(object)

## S4 method for signature 'ExprsPipeline,ANY,ANY'
x[i, j]

## S4 method for signature 'ExprsPipeline'
x$name

## S4 method for signature 'ExprsPipeline'
subset(x, subset, select)

## S4 method for signature 'ExprsPipeline'
summary(object)

## S4 method for signature 'ExprsPipeline'
getFeatures(object, index)
```

Arguments

object, x	An object of class ExprsPipeline.
i, j	Subsets entire ExprsPipeline object via object@summary[i, j]. Returns object@summary[, j] if argument i is missing.
name	Returns object@summary[, name].
subset	Subsets entire ExprsPipeline object via object@summary[subset,]. Can be used to rearrange summary table.
select	Subsets entire ExprsPipeline object via object@summary[, select]. Can be used to rearrange summary table.
index	A numeric scalar. The i-th model from which to retrieve features. If missing, getFeatures will tabulate features across all models.

Methods (by generic)

- show: Method to show ExprsPipeline object.
- [: Method to subset ExprsPipeline object.
- $: Method to subset ExprsPipeline object.
- subset: Method to subset ExprsPipeline object.
- summary: Method to summarize ExprsPipeline results.
- getFeatures: Method to return features within an ExprsPredict model.

Slots

summary Typically a data.frame. Stores the parameters and performances for the models.

machs Typically a list. Stores the models referenced in summary slot.

See Also

ExprsArray-class
ExprsModel-class
ExprsPipeline-class
ExprsEnsemble-class
ExprsPredict-class
RegrsPredict-class

ExprsPredict-class *An S4 class to store model predictions*

Description

An S4 class to store model predictions

Usage

```
## S4 method for signature 'ExprsPredict'
show(object)
```

Arguments

object	An object of class ExprsPredict.

Methods (by generic)

- show: Method to show ExprsPredict object.

Slots

pred A factor. Stores class predictions as an unambiguous class assignment.

decision.values Typically a matrix. Stores class predictions as a decision value.

probability Typically a matrix. Stores class predictions as a probability.

actual Typically a factor. Stores known class labels. Used by calcStats.

See Also

ExprsArray-class
ExprsModel-class
ExprsPipeline-class
ExprsEnsemble-class
ExprsPredict-class
RegrsPredict-class

---

forceArg *Force an args List Element to Value*

Description

Force an args List Element to Value

Usage

```
forceArg(what, as, args, verbose = TRUE)
```

Arguments

what	The name of the argument.
as	The value to set it as.
args	An args list. The result of getArgs.
verbose	A boolean. Toggles whether to alert the user that an argument is set.

fs *Select Features*

Description

The exprso package includes these feature selection modules:

- fsSample
- fsNULL
- fsANOVA
- fsInclude
- fsStats
- fsCor
- fsPrcomp
- fsEbayes
- fsEdger
- fsMrmre
- fsPathClassRFE
- fsRankProd
- fsPropd

Details

Considering the high-dimensionality of many datasets, it is prudent and often necessary to prioritize which features to include during model construction. This package provides functions for some of the most frequently used feature selection methods. Each function works as a self-contained wrapper that (1) pre-processes the ExprsArray input, (2) performs the feature selection, and (3) returns an ExprsArray output with an updated feature selection history. These histories get passed along at every step of the way until they eventually get used to pre-process an unlabeled dataset during model deployment (i.e., prediction).

The argument top specifies either the names or the number of features to supply TO the feature selection method, not what the user intends to retrieve FROM the feature selection method. When calling the first feature selection method (or the first build method, if skipping feature selection), a numeric top argument will select a "top ranked" feature set according to their default order in the ExprsArray input.

fs. *Workhorse for fs Methods*

Description

Used as a back-end wrapper for creating new fs methods.

Usage

```
fs.(object, top, uniqueFx, keep, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
uniqueFx	A function call unique to the method.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Details

If the uniqueFx returns a character vector, it is assumed that the fs method is for feature selection only. If the uniqueFx returns a list, it is assumed that the fs method is a reduction model method only.

Value

Returns an ExprsArray object.

fsANOVA *Select Features by ANOVA*

Description

fsANOVA selects features using the aov function. Note that ANOVA assumes equal variances.

Usage

```
fsANOVA(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

---

fsCor *Select Features by Correlation*

Description

fsCor selects features using the cor function. Ranks features by absolute value of correlation.

Usage

```
fsCor(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

fsEbayes *Select Features by Moderated t-test*

Description

fsEbayes selects features using the lmFit and eBayes functions from the limma package. Features ranked by the topTableF function.

Usage

```
fsEbayes(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

fsEdger *Selects Features by Exact Test*

Description

fsEdger selects features using the exactTest function from the edgeR package. This function does not normalize the data, but does estimate dispersion using the estimateCommonDisp and estimateTagwiseDisp functions.

Usage

```
fsEdger(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Details

The user can normalize the data before feature selection using the modTMM function. Note that applying edgeR to already normalized counts differs slightly from applying edgeR with normalization.

Value

Returns an ExprsArray object.

---

fsInclude *Select Features by Explicit Reference*

Description

fsInclude selects features passed to the include argument. Ranks features by the provided order.

Usage

```
fsInclude(object, top = 0, keep = 0, include)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
include	A character vector. The names of features to rank above all others. This preserves the feature order otherwise. Argument for fsInclude only.

Value

Returns an ExprsArray object.

---

fsMrmre *Select Features by mRMR*

Description

fsMrmre selects features using the mRMR.classic function from the mRMRe package.

Usage

```
fsMrmre(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Details

Note that fsMrmre crashes when supplied a very large feature_count owing to its mRMRe implementation.

Value

Returns an ExprsArray object.

fsNULL *Null Feature Selection*

Description

fsNULL selects features by passing along the top argument.

Usage

```
fsNULL(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

fsPathClassRFE *Select Features by Recursive Feature Elimination*

Description

fsPathClassRFE selects features using the fit.rfe function from the pathClass package.

Usage

```
fsPathClassRFE(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

---

fsPrcomp *Reduce Dimensions by PCA*

Description

fsPrcomp reduces dimensions using the prcomp function. The reduction model is saved and deployed automatically on any new data during model validation.

Usage

```
fsPrcomp(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

fsPropd *Select Features by Differential Proportionality Analysis*

Description

fsPropd selects features using the propd function from the propr package.

Usage

```
fsPropd(object, top = 0, keep = 0, modRatios = FALSE, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
modRatios	A logical scalar. Toggles whether to compute theta from the feature ratios as provided. Set modRatios = TRUE if data were recasted by a prior modRatios call. If TRUE, the alpha and weighted arguments will not work.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

fsRankProd *Select Features by Rank Product Analysis*

Description

fsRankProd selects features using the RankProducts function from the RankProd package.

Usage

```
fsRankProd(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

---

fsSample *Select Features by Random Sampling*

Description

fsSample selects features using the sample function.

Usage

```
fsSample(object, top = 0, keep = 0, ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

fsStats *Select Features by Statistical Testing*

Description

fsStats selects features using a base R statistics function (toggled by the how argument).

Usage

```
fsStats(object, top = 0, keep = 0, how = c("t.test", "ks.test",
  "wilcox.test", "var.test"), ...)
```

Arguments

object	An ExprsArray object to undergo feature selection.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
keep	A numeric scalar. Specifies the number of top features that should get returned by the feature selection method. Use of keep is generally not recommended, but can speed up analyses of large data.
how	A character string. Toggles between the "t.test", "ks.test", "wilcox.test", and "var.test" methods.
...	Arguments passed to the detailed function.

Value

Returns an ExprsArray object.

getArgs *Build an args List*

Description

Build an args List

Usage

```
getArgs(...)
```

Arguments

...	Arguments passed down from a calling function.

getFeatures *Retrieve Feature Set*

Description

See the respective S4 class for method details.

Usage

```
getFeatures(object, ...)
```

Arguments

object	An ExprsArray, ExprsModel, ExprsPipeline, or ExprsEnsemble object.
...	See ExprsPipeline-class or ExprsEnsemble-class.

GSE2eSet *Convert GSE to eSet*

Description

A convenience function that builds an eSet object from a GSE data source.

Usage

```
GSE2eSet(gse, colBy, colID)
```

Arguments

gse	A GSE data object retrieved using GEOquery.
colBy	A character string. The GSE column name that contains the feature value. If missing, function will prompt user for a column name after previewing options.
colID	A character string. The GSE column name that contains the feature identity. If missing, function will prompt user for a column name after previewing options.

Details

The NCBI GEO hosts files in GSE or GDS format, the latter of which exists as a curated version the former. These GDS data files easily convert to an ExpressionSet (abbreviated eSet) object using the GDS2eSet function available from the GEOquery package. However, not all GSE data files have a corresponding GDS data file available. To convert GSE data files into eSet objects, exprso provides this convenience function.

However, the user should note that GSE data files do not always get stored in an easy to parse format. Although this function has worked successfully with some GSE data files, we cannot make any guarantee that it will work for all GSE data files.

To acquire GSE data files, use the function getGEO from the GEOquery package (e.g., getGEO("GSExxxxx", GSEMatrix = F
For more information, see the GEOquery package.

Value

An ExpressionSet object.

See Also

ExprsArray-class, arrayExprs

---

makeGridFromArgs *Build Argument Grid*

Description

This function builds an argument grid from any number of arguments. Used to prepare a grid-search for the plGrid and plGridMulti functions.

Usage

```
makeGridFromArgs(array.train, top, how, ...)
```

Arguments

array.train	The array.train argument as fed to plGrid.
top	The top argument as fed to plGrid.
how	The how argument as fed to plGrid.
...	Additional arguments as fed to plGrid.

---

mod *Process Data*

Description

The exprso package includes these data process modules:

- modHistory
- modSubset
- modFilter
- modTransform
- modSample
- modInclude
- modNormalize
- modTMM
- modAcomp

- modCLR
- modRatios
- modScale
- modSkew

---

modAcomp *Compositionally Constrain Data*

Description

modAcomp makes it so that all sample vectors have the same total sum.

Usage

```
modAcomp(object)
```

Arguments

object	An ExprsArray object to undergo pre-processing.

Value

A pre-processed ExprsArray object.

---

modCLR *Log-ratio Transform Data*

Description

modCLR applies a centered log-ratio transformation to the data.

Usage

```
modCLR(object)
```

Arguments

object	An ExprsArray object to undergo pre-processing.

Value

A pre-processed ExprsArray object.

modCluster *Cluster Subjects*

Description

This method clusters subjects based on feature data using any one of seven available clustering algorithms. See Arguments below.

Usage

```
modCluster(object, top = 0, how = "hclust", onlyCluster = FALSE, ...)

## S4 method for signature 'ExprsArray'
modCluster(object, top = 0, how = "hclust",
  onlyCluster = FALSE, ...)
```

Arguments

object	An ExprsArray object. The object containing the subject data to cluster.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
how	A character string. The name of the function used to cluster. Select from "hclust", "kmeans", "agnes", "clara", "diana", "fanny", or "pam".
onlyCluster	A logical scalar. Toggles whether to return a processed cluster object or an updated ExprsArray object.
...	Additional arguments to the cluster function and/or other functions used for clustering (e.g., dist and cutree).

Details

Note that this function will expect the argument k to define the returned number of clusters, except when how = "kmeans" in which case this function will expect the argument centers instead.

Value

Typically an ExprsArray object with subject cluster assignments added to the $cluster column of the @anot slot.

Methods (by class)

- ExprsArray: Method to compare ExprsArray objects.

modFilter *Hard Filter Data*

Description

modFilter imposes a hard filter for (gene expression) feature data.

Usage

```
modFilter(object, threshold, maximum, beta1, beta2)
```

Arguments

object	An ExprsArray object to undergo pre-processing.
threshold	A numeric scalar. The value below which to assign this value.
maximum	A numeric scalar. The value above which to assign this value.
beta1	A numeric scalar. The max - min range above which to include the feature. Inclusive with beta2.
beta2	A numeric scalar. The max / min ratio above which to include the feature. Inclusive with beta1.

Details

This method reproduces the hard filter described by Deb and Reddy (2003) for pre-processing the hallmark Golub ALL/AML dataset. This filter first sets all values less than threshold to threshold and all values greater than maximum to maximum.

Next, this method includes only those features with (a) a range greater than beta1, and also (b) a ratio of maximum feature expression to minimum feature expression greater than beta2.

Value

A pre-processed ExprsArray object.

modHistory *Replicate Data Process History*

Description

modHistory replicates the fs history of a reference object. Used by predict to prepare validation set for model deployment.

Usage

```
modHistory(object, reference)
```

Arguments

object	An ExprsArray object. The object that should undergo a replication of some feature selection and dimension reduction history.
reference	An ExprsArray or ExprsModel object. The object containing the history to use as a template.

Value

A pre-processed ExprsArray object.

---

modInclude *Select Features from Data*

Description

modSelect selects specific features from a data set. Unlike fsInclude, this function does not update @preFilter and returns only those features stated by include.

Usage

```
modInclude(object, include = rownames(object@exprs))
```

Arguments

object	An ExprsArray object to undergo pre-processing.
include	A character vector. The names of features to include.

Value

A pre-processed ExprsArray object.

---

modNormalize *Normalize Data*

Description

modNormalize normalizes feature data.

Usage

```
modNormalize(object, MARGIN = c(1, 2))
```

Arguments

object	An ExprsArray object to undergo pre-processing.
MARGIN	A numeric vector. The margin by which to normalize. Provide MARGIN = 1 to normalize the feature vector. Provide MARGIN = 2 to normalize the subject vector. Provide MARGIN = c(1, 2) to normalize by the subject vector and then by the feature vector.

Details

This method normalizes subject and/or feature vectors according to the formula y = (x - mean(x)) / sd(x).

Value

A pre-processed ExprsArray object.

---

modRatios *Recast Data as Feature Ratios*

Description

modRatios recasts a data set with N feature columns as a new data set with N * (N - 1) / 2 feature ratio columns.

Usage

```
modRatios(object)
```

Arguments

object	An ExprsArray object. The object that should undergo a replication of some feature selection and dimension reduction history.

Value

A pre-processed ExprsArray object.

modSample *Sample Features from Data*

Description

modSample samples features from a data set randomly without replacement. When size = 0, this is equivalent to fsSample, top = 0, but much quicker.

Usage

```
modSample(object, size = 0)
```

Arguments

object	An ExprsArray object to undergo pre-processing.
size	A numeric scalar. The number of randomly sampled features to include in the pre-processed ExprsArray object.

Value

A pre-processed ExprsArray object.

modScale *Scale Data by Factor Range*

Description

modScale scales a data set by making all sample vectors have the same total sum, then multiplying each sample vector by a scale factor.

Usage

```
modScale(object, alpha = 0, uniform = TRUE)
```

Arguments

object	An ExprsArray object. The object that should undergo a replication of some feature selection and dimension reduction history.
alpha	An integer. The maximum range of scale factors used for scaling if uniform = TRUE. The standard deviation of the scale factors if uniform = FALSE. See Details.
uniform	A boolean. Toggles whether to draw scale factors from a uniform distribution or a normal distribution.

Details

If `uniform = TRUE`, scale factors are randomly sampled from the uniform distribution `(0, alpha) + 1`. Otherwise, scale factors are randomly sampled from the normal distribution with a mean of 0 and standard deviation of `alpha`. When using the normal distribution, these scale factors are transformed by taking the absolute value then adding one. For this reason, data are always unscaled when `alpha = 0`.

Value

A pre-processed `ExprsArray` object.

---

modSkew *Skew Data by Factor Range*

Description

`modSkew` skews a data set by making all sample vectors have the same total sum, introducing a new feature, and then making all sample vectors again have the same total sum.

Usage

```
modSkew(object, alpha = 0, uniform = TRUE)
```

Arguments

`object`	An `ExprsArray` object. The object that should undergo a replication of some feature selection and dimension reduction history.
`alpha`	An integer. The maximum range of skew factors used for skewing if `uniform = TRUE`. The standard deviation of the skew factors if `uniform = FALSE`. See Details.
`uniform`	A boolean. Toggles whether to draw skew factors from a uniform distribution or a normal distribution.

Details

If `uniform = TRUE`, skew factors are randomly sampled from the uniform distribution `(0, alpha) + 1`. Otherwise, skew factors are randomly sampled from the normal distribution with a mean of 0 and standard deviation of `alpha`. When using the normal distribution, these skew factors are transformed by taking the absolute value then adding one. For this reason, data are always unskewed when `alpha = 0`.

Value

A pre-processed `ExprsArray` object.

modSubset *Tidy Subset Wrapper*

Description

modSubset function provides a tidy wrapper for the ExprsArray subset method. pipeSubset provides a tidy wrapper for the ExprsPipeline subset method.

Usage

```
modSubset(object, colBy, include)

pipeSubset(object, colBy, include)
```

Arguments

object	An ExprsArray or ExprsPipeline object to subset.
colBy	A numeric or character index. The column that contains group annotations.
include	A character vector. Specifies which annotations in colBy to include in the subset.

Value

An ExprsArray or ExprsPipeline object.

Functions

- pipeSubset: A variant of modSubset.

modSwap *Swap Case Subjects*

Description

This experimental function mutates a percentage of case subjects into noisy positives, false positives, or defined out-groups.

Usage

```
modSwap(object, how = "fp", percent = 10, theta = 1)

## S4 method for signature 'ExprsBinary'
modSwap(object, how = "fp", percent = 10,
  theta = 1)
```

Arguments

object	An ExprsBinary object to mutate.
how	A character string. The method used to mutate case subjects. Select from "rp.1", "rp.2", "fp", "ng", or "tg". Alternatively, another ExprsBinary object. See Details.
percent	A numeric scalar. The percentage of subjects to mutate.
theta	A numeric scalar. Applies a weight to the distribution of means when mutating subjects via the "ng" or "tg" method.

Details

This function includes several methods for distorting the features of ExprsBinary subjects. The "rp.1" method randomizes subject vectors to create "subject noise". The "rp.2" method creates a new subject vector by randomly sampling feature values from the respective feature vector. The "fp" method creates a new subject vector by randomly sampling feature values from the respective control feature vector.

The "ng" and "tg" methods create out-groups by defining new means for each feature. These methods yield fixed distributions around new feature means such that the mean of all new feature means remains constant. The argument theta dictates how much the new feature mean might differ from the original feature mean (where larger theta values lead to more similar new feature means). For the "ng" method, the mean of new feature means equals that of the original features for case subjects only. On the other hand, for the "tg" method, the mean of new feature means equals that of the original features for all subjects.

Alternatively, by providing another ExprsBinary object as the how argument, this function will swap a percentage of case subjects from the main dataset with control subjects from the second dataset.

Value

An ExprsBinary object containing mutated subjects with an index appended to the $mutated column of the @annot slot.

Methods (by class)

- ExprsBinary: A method to mutate ExprsBinary objects.

---

modTMM *Normalize Data*

Description

modTMM normalizes feature data.

Usage

```
modTMM(object, method = "TMM")
```

Arguments

object	An ExprsArray object to undergo pre-processing.
method	A character string. The method used by calcNormFactors. Defaults to the "TMM" method.

Details

This method normalizes data using the calcNormFactors function from the edgeR package. It returns the original counts multiplied by the effective library size factors.

Value

A pre-processed ExprsArray object.

---

modTransform *Log Transform Data*

Description

modTransform log transforms feature data.

Usage

```
modTransform(object, base = exp(1))
```

Arguments

object	An ExprsArray object to undergo pre-processing.
base	A numeric scalar. The base of the logarithm.

Value

A pre-processed ExprsArray object.

---

packageCheck *Package Check*

Description

Checks whether the user has the required package installed. For back-end use only.

Usage

```
packageCheck(package)
```

Arguments

package	A character string. An R package.

pipe *Process Pipelines*

Description

The exprso package includes these pipeline process modules:

- pipeSubset
- pipeFilter
- pipeUnboot

pipeFilter *Filter* ExprsPipeline *Object*

Description

pipeFilter subsets an ExprsPipeline object.

Usage

```
pipeFilter(object, colBy = "valid.acc", how = 0, gate = 0, top = 0)
```

Arguments

object	An ExprsPipeline-class object.
colBy	A character vector or string. Specifies column(s) to use when filtering by model performance. Listing multiple columns will result in a filter based on the product all listed columns.
how, gate	A numeric scalar. Arguments between 0 and 1 will impose a threshold or ceiling filter, respectively, based on the raw value of colBy. Arguments between 1 and 100 will impose a filter based on the percentile of colBy. The user may also provide "midrange", "median", or "mean" as an argument for these filters.
top	A numeric scalar. Determines the top N models based on colBy to include after the threshold and ceiling filters. In the case that the @summary slot contains the column "boot", this selects the top N models for each unique bootstrap.

Details

The filter process occurs in three steps. However, the user may skip any one of these steps by setting the respective argument to 0. First, a threshold filter gets imposed. Any model with a performance less than the threshold filter, how, gets excluded. Second, a ceiling filter gets imposed. Any model with a performance greater than the ceiling filter, gate, gets excluded. Third, an arbitrary subset occurs. The top N models in the ExprsPipeline object get selected based on the argument top. However, in the case that the @summary slot contains the column "boot", pipeFilter selects the top N models per bootstrap.

pipeFilter will apply this filter based on the performance metrics listed in the colBy argument. Listing multiple columns will result in a filter based on the product of all listed columns. To weigh one metric over another, list that column more times.

Value

An ExprsPipeline-class object.

---

pipeUnboot *Rename "boot" Column*

Description

pipeUnboot renames the "boot" column summary to "unboot".

Usage

```
pipeUnboot(object)
```

Arguments

object	An ExprsPipeline-class object.

Details

This method provides a convenient adjunct to pipeFilter owing to how exprso handles ExprsPipeline objects with a "boot" column.

Value

An ExprsPipeline-class object.

---

pl *Deploy Pipeline*

Description

The exprso package includes these automated pipeline modules:

- plCV
- plGrid
- plGridMulti
- plMonteCarlo
- plNested

plCV *Perform Simple Cross-Validation*

Description

Calculates v-fold or leave-one-out cross-validation without selecting a new set of features with each fold. See Details.

Usage

```
plCV(array, top, how, fold, ...)
```

Arguments

array	Specifies the ExprsArray object to undergo cross-validation.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. A numeric vector can also be used to indicate specific features by location, similar to a character vector.
how	A character string. Specifies the build method to iterate.
fold	A numeric scalar. Specifies the number of folds for cross-validation. Set fold = 0 to perform leave-one-out cross-validation.
...	Arguments passed to the how method.

Details

plCV performs v-fold or leave-one-out cross-validation. The argument fold specifies the number of v-folds to use during cross-validation. Set fold = 0 to perform leave-one-out cross-validation. Cross-validation accuracy is defined as the average accuracy from calcStats.

This type of cross-validation is most appropriate if the data has not undergone any prior feature selection. However, it can also serve as an unbiased guide to parameter selection when embedded in plGrid. If using cross-validation in lieu of an independent test set in the setting of one or more feature selection methods, consider using a more "sophisticated" form of cross-validation as implemented in plMonteCarlo or plNested.

When calculating model performance with calcStats, this function forces aucSkip = TRUE and plotSkip = TRUE.

Value

A numeric scalar. The cross-validation accuracy.

plGrid *Perform High-Throughput Machine Learning*

Description

Trains and deploys models across a vast parameter search space.

Usage

```
plGrid(array.train, array.valid = NULL, top, how, fold = 10,
  aucSkip = FALSE, verbose = FALSE, ...)
```

Arguments

array.train	Specifies the ExprsArray object to use as training set.
array.valid	Specifies the ExprsArray object to use as validation set.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. Note that providing a numeric vector for the top argument will have plGrid search across multiple top features. However, by providing a list of numeric vectors as the top argument, the user can force the default handling of numeric vectors.
how	A character string. Specifies the build method to iterate.
fold	A numeric scalar. Specifies the number of folds for cross-validation. Set fold = 0 to perform leave-one-out cross-validation. Argument passed to plCV. Set fold = NULL to skip cross-validation altogether.
aucSkip	A logical scalar. Argument passed to calcStats.
verbose	A logical scalar. Argument passed to exprso-predict.
...	Arguments passed to the how method. Unlike the build method, plGrid allows multiple parameters for each argument, supplied as a vector. See Details.

Details

plGrid will build and exprso-predict for each combination of parameters provided as additional arguments (...). When using plGrid, supplying a numeric vector as the top argument will train and deploy a model of each mentioned size for each combination of parameters provided in .... To skip validation set prediction, use array.valid = NULL. Either way, this function returns an ExprsPipeline-class object which contains a summary of the build parameters and the models themselves. The argument fold controls cross-validation via plCV.

Value

An ExprsPipeline-class object.

plGridMulti *Perform High-Throughput Multi-Class Classification*

Description

Trains and deploys multi-class classifiers across a vast parameter search space.

Usage

```
plGridMulti(array.train, array.valid = NULL, ctrlFS, top, how,
  aucSkip = FALSE, verbose = TRUE, ...)
```

Arguments

array.train	Specifies the ExprsMulti object to use as training set.
array.valid	Specifies the ExprsMulti object to use as validation set.
ctrlFS	A list of arguments handled by ctrlFeatureSelect.
top	A numeric scalar or character vector. A numeric scalar indicates the number of top features that should undergo feature selection. A character vector indicates specifically which features by name should undergo feature selection. Set top = 0 to include all features. Note that providing a numeric vector for the top argument will have plGrid search across multiple top features. However, by providing a list of numeric vectors as the top argument, the user can force the default handling of numeric vectors.
how	A character string. Specifies the build method to iterate.
aucSkip	A logical scalar. Argument passed to calcStats.
verbose	A logical scalar. Argument passed to exprso-predict.
...	Arguments passed to the how method. Unlike the build method, plGrid allows multiple parameters for each argument, supplied as a vector. See Details.

Details

Unlike plGrid, the plGridMulti function accepts a ctrlFS argument, allowing for 1-vs-all classification with implicit feature selection. 1-vs-all classification, this function divides the data into 1-vs-all bins, performs a 1-vs-all feature selection for each bin, and then performs a 1-vs-all classification for that same bin. As such, each ExprsMachine within the ExprsModule will have its own unique feature selection history.

Take note, that plGridMulti does not have built-in plCV support. To use plGridMulti with cross-validation, use plNested.

Value

An ExprsPipeline-class object.

plMonteCarlo *Monte Carlo Cross-Validation*

Description

Perform Monte Carlo style cross-validation.

Usage

```
plMonteCarlo(array, B = 10, ctrlSS, ctrlFS = NULL, ctrlGS, ctrlMS = NULL,
  save = FALSE)
```

Arguments

array	Specifies the ExprsArray object to undergo cross-validation.
B	A numeric scalar. The number of times to split the data.
ctrlSS	Arguments handled by ctrlSplitSet.
ctrlFS	A list of arguments handled by ctrlFeatureSelect.
ctrlGS	Arguments handled by ctrlGridSearch.
ctrlMS	Arguments handled by ctrlModSet. Optional.
save	A logical scalar. Toggles whether to save randomly split training and validation sets.

Details

Analogous to how plGrid manages multiple build and predict tasks, one can think of plMonteCarlo as managing multiple pl tasks.

Specifically, plMonteCarlo will call the provided split function (via ctrlSS) some B times, perform all feature selection tasks (listed via ctrlFS) on each split of the data, and execute the pl function (via ctrlGS) using the bootstrapped set.

To perform multiple feature selection tasks, supply a list of multiple ctrlFeatureSelect argument wrappers to ctrlFS. To reduce the results of plMonteCarlo to a single performance metric, you can feed the returned ExprsPipeline object through the helper function calcMonteCarlo.

When embedding another plMonteCarlo or plNested call within this function (i.e., via ctrlGS), outer-fold model performance will force aucSkip = TRUE and plotSkip = TRUE.

Value

An ExprsPipeline-class object.

Examples

```
## Not run:
require(golubEsets)
data(Golub_Merge)
array <- arrayEset(Golub_Merge, colBy = "ALL.AML", include = list("ALL", "AML"))
array <- modFilter(array, 20, 16000, 500, 5) # pre-filter Golub ala Deb 2003
array <- modTransform(array) # lg transform
array <- modNormalize(array, c(1, 2)) # normalize gene and subject vectors
ss <- ctrlSplitSet(func = "splitStratify", percent.include = 67, colBy = NULL)
fs <- list(ctrlFeatureSelect(func = "fsStats", top = 0, how = "t.test"),
           ctrlFeatureSelect(func = "fsPrcomp", top = 50))
gs <- ctrlGridSearch(func = "plGrid", how = "buildSVM", top = c(2, 3, 4), fold = 10,
                     kernel = c("linear", "radial"), cost = 10^(-3:3), gamma = 10^(-3:3))
boot <- plMonteCarlo(array, B = 3, ctrlSS = ss, ctrlFS = fs, ctrlGS = gs)

## End(Not run)
```

---

plNested *Nested Cross-Validation*

Description

Perform nested cross-validation.

Usage

```
plNested(array, fold = 10, ctrlFS = NULL, ctrlGS, save = FALSE)
```

Arguments

`array`	Specifies the `ExprsArray` object to undergo cross-validation.
`fold`	A numeric scalar. Specifies the number of folds for cross-validation. Set `fold = 0` to perform leave-one-out cross-validation.
`ctrlFS`	A list of arguments handled by `ctrlFeatureSelect`.
`ctrlGS`	Arguments handled by `ctrlGridSearch`.
`save`	A logical scalar. Toggles whether to save each fold.

Details

Analogous to how `plGrid` manages multiple `build` and `predict` tasks, one can think of `plNested` as managing multiple `pl` tasks.

Specifically, `plNested` segregates the data into v-folds, treating each fold as a validation set and the subjects not in that fold as a training set. Then, some `fold` times, it performs all feature selection tasks (listed via `ctrlFS`) on each split of the data, and executes the `pl` function (via `ctrlGS`) using the training set.

To perform multiple feature selection tasks, supply a list of multiple ctrlFeatureSelect argument wrappers to ctrlFS. To reduce the results of plNested to a single performance metric, you can feed the returned ExprsPipeline object through the helper function calcNested.

When calculating model performance with calcStats, this function forces aucSkip = TRUE and plotSkip = TRUE. When embedding another plMonteCarlo or plNested call within this function (i.e., via ctrlGS), outer-fold model performance will force aucSkip = TRUE and plotSkip = TRUE.

Value

An ExprsPipeline-class object.

Examples

```
## Not run:
require(golubEsets)
data(Golub_Merge)
array <- arrayEset(Golub_Merge, colBy = "ALL.AML", include = list("ALL", "AML"))
array <- modFilter(array, 20, 16000, 500, 5) # pre-filter Golub ala Deb 2003
array <- modTransform(array) # lg transform
array <- modNormalize(array, c(1, 2)) # normalize gene and subject vectors
fs <- ctrlFeatureSelect(func = "fsEbayes", top = 0)
gs <- ctrlGridSearch(func = "plGrid", how = "buildANN", top = c(10, 20, 30),
                     size = 1:3, decay = c(0, .5, 1), fold = 0)
nest <- plNested(arrays[[1]], fold = 10, ctrlFS = fs, ctrlGS = gs, save = FALSE)

## End(Not run)
```

---

progress *Make Progress Bar*

Description

Make Progress Bar

Usage

```
progress(i, k, numTicks)
```

Arguments

i	The current iteration.
k	Total iterations.
numTicks	The result of progress.

Value

The next numTicks argument.

RegrsArray-class *An S4 class to store feature and annotation data*

Description

An ExprsArray sub-class for data with continuous outcomes.

RegrsModel-class *An S4 class to store the model*

Description

An ExprsModel sub-class for continuous outcome models.

RegrsPredict-class *An S4 class to store model predictions*

Description

An S4 class to store model predictions

Usage

```
## S4 method for signature 'RegrsPredict'
show(object)
```

Arguments

object	An object of class RegrsPredict.

Methods (by generic)

- show: Method to show RegrsPredict object.

Slots

pred Any. Stores predicted outcome.

actual Any. Stores actual outcome. Used by calcStats.

See Also

ExprsArray-class
ExprsModel-class
ExprsPipeline-class
ExprsEnsemble-class
ExprsPredict-class
RegrsPredict-class

reRank *Serialize "1 vs. all" Feature Selection*

Description

This experimental function converts multiple feature rank lists, derived from "1 vs. all" binary feature selection, into a single feature rank list. This function is not in use in this package.

Usage

```
reRank(fss)
```

Arguments

fss	The result of a doMulti function call.

Details

After passing a feature selection method through doMulti, a set of ranked features gets returned for each one of the total number of levels in the $defineCase factor. In order to proceed with model deployment (at least in the setting of a conventional pipeline where feature selection occurs prior to classifier construction), multiple feature rankings would need to get serialized into a single feature rank list. reRank accomplishes this by calculating the rank sum for each feature across all "1 vs. all" feature selection tasks. Features found in one rank list but not in another receive a numeric rank equal to one more than the maximum rank in that feature rank list. The presence of a NA placeholder (see: doMulti) will not impact reRank.

We note here, however, that a better approach would deploy "1 vs. all" feature selection and classifier construction simultaneously, rather than "1 vs. all" feature selection followed by "1 vs. all" classifier construction. This is now implemented as plGridMulti.

Value

A vector of re-ranked features. See Details.

split *Split Data*

Description

The exprso package includes these split modules:

- splitSample
- splitStratify
- splitBalanced
- splitBy

splitBalanced *Split by Balanced Sampling*

Description

splitBalance is a wrapper that calls splitStratify twice. In the first call, splitStratify is used to create a balanced training set from the total data. In the second call, splitStratify is used to create a balanced validation set from the leftover data. This function ensures that there are always an equal number of samples from each class in the split.

Usage

```
splitBalanced(object, percent.include = 67, ...)
```

Arguments

object	An ExprsArray object to split.
percent.include	Specifies the percent of the total number of subjects to include in the training set.
...	Arguments passed to both splitStratify calls.

Value

Returns a list of two ExprsArray objects.

splitBy *Split by User-defined Group*

Description

splitBy builds a training set and validation set by placing all samples that have the include annotation in the specified colBy column in the training set. The remaining samples get placed in the validation set. This split is not random.

Usage

```
splitBy(object, colBy, include)
```

Arguments

object	An ExprsArray object to split.
colBy	A character string. Specifies the column used to split the data.
include	A character vector. Specifies which annotations in colBy to include in the training set.

Value

Returns a list of two ExprsArray objects.

---

splitSample *Split by Random Sampling*

Description

splitSample builds a training and validation set by randomly sampling the subjects found within the ExprsArray object. Note that this method is not truly random. Instead, splitSample iterates through the random sampling process until it settles on a solution such that both the training and validation set contain at least one subject for each class label. If this method finds no solution after 10 iterations, the function will post an error. Set percent.include = 100 to skip random sampling and return a NULL validation set. Additional arguments (e.g., replace = TRUE) passed along to sample.

Usage

```
splitSample(object, percent.include = 67, ...)
```

Arguments

object	An ExprsArray object to split.
percent.include	Specifies the percent of the total number of subjects to include in the training set.
...	For splitSample: additional arguments passed along to sample. For splitStratify: additional arguments passed along to cut.

Value

Returns a list of two ExprsArray objects.

---

splitStratify *Split by Stratified Sampling*

Description

splitStratify builds a training and validation set through a stratified random sampling process. This function utilizes the strata function from the sampling package as well as the cut function from the base package. The latter function provides a means by which to bin continuous data prior to stratified random sampling. We refer the user to the parameter descriptions to learn the specifics of how to apply binning, although the user might find it easier to instead bin annotations beforehand. When applied to an ExprsMulti object, this function stratifies subjects across all classes found in that dataset.

Usage

```
splitStratify(object, percent.include = 67, colBy = NULL, bin = rep(FALSE,
  length(colBy)), breaks = rep(list(NA), length(colBy)), ...)
```

Arguments

object	An ExprsArray object to split.
percent.include	Specifies the percent of the total number of subjects to include in the training set.
colBy	Specifies a vector of column names by which to stratify in addition to class labels annotation. If colBy = NULL, random sampling will occur across the class label annotation only. For splitStratify only.
bin	A logical vector indicating whether to bin the respective colBy column using cut (e.g., bin = c(FALSE, TRUE)). For splitStratify only.
breaks	A list. Each element of the list should correspond to a breaks argument passed to cut for the respective colBy column. Set an element to NA when not binning that colBy. For splitStratify only.
...	For splitSample: additional arguments passed along to sample. For splitStratify: additional arguments passed along to cut.

Value

Returns a list of two ExprsArray objects.

---

trainingSet *Extract Training Set*

Description

This function extracts the training set from the result of a split method call such as splitSample or splitStratify.

Usage

```
trainingSet(splitSets)
```

Arguments

splitSets	A two-item list. The result of a split method call.

Value

An ExprsArray object.

validationSet *Extract Validation Set*

Description

This function extracts the validation set from the result of a `split` method call such as `splitSample` or `splitStratify`.

Usage

```
validationSet(splitSets)

testSet(splitSets)
```

Arguments

`splitSets`	A two-item list. The result of a `split` method call.

Value

An `ExprsArray` object.

Functions

- `testSet`: A variant of `validationSet`.

Index