Package ‘ClassifyR’

October 18, 2022

Type Package

Title A framework for cross-validated classification problems, with
applications to differential variability and differential
distribution testing

Version 3.0.3

Date 2022-05-11

Author Dario Strbenac, Ellis Patrick, John Ormerod, Graham Mann, Jean Yang

Maintainer Dario Strbenac <dario.strbenac@sydney.edu.au>

VignetteBuilder knitr

Encoding UTF-8

biocViews Classification, Survival

Depends R (>= 4.1.0), methods, S4Vectors (>= 0.18.0),
MultiAssayExperiment (>= 1.6.0), BiocParallel, survival

Imports grid, utils, dplyr, tidyr, rlang, randomForest

Suggests limma, genefilter, edgeR, car, Rmixmod, ggplot2 (>= 3.0.0),
gridExtra (>= 2.0.0), cowplot, BiocStyle, pamr, PoiClaClu,
parathyroidSE, knitr, htmltools, gtable, scales, e1071,
rmarkdown, IRanges, robustbase, glmnet, class

Description The software formalises a framework for classification in R.
There are four stages; Data transformation, feature selection, classifier training,
and prediction. The requirements of variable types and names are
fixed, but specialised variables for functions can also be provided.
The classification framework is wrapped in a driver loop, that
reproducibly carries out a number of cross-validation schemes.
Functions for differential expression, differential variability,
and differential distribution are included. Additional functions
may be developed by the user, by creating an interface to the framework.

License GPL-3

RoxygenNote 7.1.2

Collate 'ROCplot.R' 'classes.R' 'calcPerformance.R' 'constants.R'
'crossValidate.R' 'data.R' 'distribution.R'

'edgesToHubNetworks.R' 'featureSetSummary.R'
'getLocationsAndScales.R' 'interactorDifferences.R'
'interfaceClassify.R' 'interfaceCoxPH.R' 'interfaceCoxnet.R'
'interfaceDLDA.R' 'interfaceElasticNetGLM.R'
'interfaceFisherDiscriminant.R' 'interfaceKNN.R'
'interfaceKTSPclassifier.R' 'interfaceMerge.R'
'interfaceMixModels.R' 'interfaceNSC.R'
'interfaceNaiveBayesKernel.R' 'interfacePCA.R'
'interfacePrevalidation.R' 'interfaceRandomForest.R'
'interfaceSVM.R' 'performancePlot.R' 'plotFeatureClasses.R'
'previousSelection.R' 'previousTrained.R' 'rankingBartlett.R'
'rankingCoxPH.R' 'rankingDMD.R' 'rankingDifferentMeans.R'
'rankingEdgeR.R' 'rankingKolmogorovSmirnov.R'
'rankingKullbackLeibler.R' 'rankingLevene.R'
'rankingLikelihoodRatio.R' 'rankingLimma.R'
'rankingPairsDifferences.R' 'rankingPlot.R'
'rankingSelectMulti.R' 'runTest.R' 'runTests.R'
'samplesMetricMap.R' 'selectionPlot.R' 'simpleParams.R'
'subtractFromLocation.R' 'utilities.R'

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

git_branch RELEASE_3_15

git_last_commit 1d0f471

git_last_commit_date 2022-05-11

Date/Publication 2022-10-18

R topics documented:

`asthma` *Asthma RNA Abundance and Patient Classes*

Description

Data set consists of a matrix of abundances of 2000 most variable gene expression measurements for 190 samples and a factor vector of classes for those samples.

Format

`measurements` has a row for each sample and a column for each gene. `classes` is a factor vector with values No and Yes, indicating if a partiular person has asthma or not.

Source

A Nasal Brush-based Classifier of Asthma Identified by Machine Learning Analysis of Nasal RNA Sequence Data, *Scientific Reports*, 2018. Webpage: `http://www.nature.com/articles/s41598-018-27189-4`

`bartlettRanking` *Ranking of Differential Variability with Bartlett Statistic*

Description

Ranks all features from largest Bartlett statistic to smallest.

Usage

```
## S4 method for signature 'matrix'
bartlettRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
bartlettRanking(measurementsTrain, classesTrain, verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
```

```
bartlettRanking(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information table will be used.

Details

The calculation of the test statistic is performed by the `bartlett.test` function from the `stats` package.

Data tables which consist entirely of non-numeric data cannot be ranked.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# Samples in one class with differential variability to other class.
# First 20 genes are DV.
genesRNAmatrix <- sapply(1:20, function(sample) c(rnorm(25, 9, 1), rnorm(25, 9, 5)))
genesRNAmatrix <- cbind(genesRNAmatrix, sapply(1:80, function(sample) rnorm(50, 9, 1)))
rownames(genesRNAmatrix) <- paste("Sample", 1:50)
colnames(genesRNAmatrix) <- paste("Gene", 1:100)
genesSNPmatrix <- matrix(sample(c("None", "Missense"), 250, replace = TRUE), nrow = 50)
rownames(genesSNPmatrix) <- paste("Sample", 1:50)
colnames(genesSNPmatrix) <- paste("Gene", 1:5)
classes <- factor(rep(c("Poor", "Good"), each = 25))
names(classes) <- paste("Sample", 1:50)


genesDataset <- MultiAssayExperiment(list(RNA = t(genesRNAmatrix), SNP = t(genesSNPmatrix)),
                                     colData = DataFrame(class = classes))

bartlettRanking(genesDataset, targets = "RNA", classesTrain = "class")
```

---

calcExternalPerformance

Add Performance Calculations to a ClassifyResult Object or Calculate for a Pair of Factor Vectors

---

Description

If `calcExternalPerformance` is used, such as when having a vector of known classes and a vector of predicted classes determined outside of the ClassifyR package, a single metric value is calculated. If `calcCVperformance` is used, annotates the results of calling `runTests` with one of the user-specified performance measures.

Usage

```
## S4 method for signature 'factor,factor'
calcExternalPerformance(
  actualOutcomes,
  predictedOutcomes,
  performanceType = c("Balanced Error", "Balanced Accuracy", "Error", "Accuracy",
    "Sample Error", "Sample Accuracy", "Micro Precision", "Micro Recall", "Micro F1",
    "Macro Precision", "Macro Recall", "Macro F1", "Matthews Correlation Coefficient")
)

## S4 method for signature 'Surv,numeric'
calcExternalPerformance(
  actualOutcomes,
  predictedOutcomes,
  performanceType = "C-index"
)

## S4 method for signature 'ClassifyResult'
calcCVperformance(
  result,
  performanceType = c("Balanced Error", "Balanced Accuracy", "Error", "Accuracy",
    "Sample Error", "Sample Accuracy", "Micro Precision", "Micro Recall", "Micro F1",
    "Macro Precision", "Macro Recall", "Macro F1", "Matthews Correlation Coefficient",
    "AUC", "C-index")
)
```

Arguments

`actualOutcomes` A factor vector or survival information specifying each sample's known outcome.

`predictedOutcomes` A factor vector or survival information of the same length as `actualOutcomes` specifying each sample's predicted outcome.

`performanceType` A character vector of length 1. Default: `"Balanced Error"`. Must be one of the following options:

- `"Error"`: Ordinary error rate.
- `"Accuracy"`: Ordinary accuracy.
- `"Balanced Error"`: Balanced error rate.
- `"Balanced Accuracy"`: Balanced accuracy.
- `"Sample Error"`: Error rate for each sample in the data set.
- `"Sample Accuracy"`: Accuracy for each sample in the data set.
- `"Micro Precision"`: Sum of the number of correct predictions in each class, divided by the sum of number of samples in each class.
- `"Micro Recall"`: Sum of the number of correct predictions in each class, divided by the sum of number of samples predicted as belonging to each class.
- `"Micro F1"`: F1 score obtained by calculating the harmonic mean of micro precision and micro recall.
- `"Macro Precision"`: Sum of the ratios of the number of correct predictions in each class to the number of samples in each class, divided by the number of classes.
- `"Macro Recall"`: Sum of the ratios of the number of correct predictions in each class to the number of samples predicted to be in each class, divided by the number of classes.
- `"Macro F1"`: F1 score obtained by calculating the harmonic mean of macro precision and macro recall.
- `"Matthews Correlation Coefficient"`: Matthews Correlation Coefficient (MCC). A score between -1 and 1 indicating how concordant the predicted classes are to the actual classes. Only defined if there are two classes.
- `"AUC"`: Area Under the Curve. An area ranging from 0 to 1, under the ROC.
- `"C-index"`: For survival data, the concordance index, for models which produce risk scores. Ranges from 0 to 1.

`result` An object of class `ClassifyResult`.

Details

All metrics except Matthews Correlation Coefficient are suitable for evaluating classification scenarios with more than two classes and are reimplementations of those available from Intel DAAL.

If `runTests` was run in resampling mode, one performance measure is produced for every resampling. If the leave-k-out mode was used, then the predictions are concatenated, and one performance measure is calculated for all classifications.

`"Balanced Error"` calculates the balanced error rate and is better suited to class-imbalanced data sets than the ordinary error rate specified by `"Error"`. `"Sample Error"` calculates the error rate of each sample individually. This may help to identify which samples are contributing the most to the overall error rate and check them for confounding factors. Precision, recall and F1 score have micro and macro summary versions. The macro versions are preferable because the metric will not have a good score if there is substantial class imbalance and the classifier predicts all samples as belonging to the majority class.

Value

If `calcCVperformance` was run, an updated `ClassifyResult` object, with new metric values in the `performance` slot. If `calcExternalPerformance` was run, the performance metric value itself.

Author(s)

Dario Strbenac

Examples

```
predictTable <- data.frame(sample = paste("A", 1:10, sep = ''),
                           class = factor(sample(LETTERS[1:2], 50, replace = TRUE)))
actual <- factor(sample(LETTERS[1:2], 10, replace = TRUE))
result <- ClassifyResult(DataFrame(),
                         paste("A", 1:10, sep = ''), paste("Gene", 1:50, sep = ''),
                     list(1:50, 1:50), list(1:5, 6:15), list(function(oracle){}), NULL,
                         predictTable, actual)
result <- calcCVperformance(result)
performance(result)
```

`characterOrDataFrame-class` *Union of a Character and a DataFrame*

Description

Allows a slot to be either a character or a DataFrame.

Enables different functionality to be executed depending on whether input data dependent variable is survival or categorical classes.

Author(s)

Dario Strbenac

Dario Strbenac

Examples

```
setClass("Selections", representation(features = "characterOrDataFrame"))
selections <- new("Selections", features = c("BRAF", "NRAS"))
featuresTable <- DataFrame(assay = c("RNA-seq", "Mass spectrometry"),
                           feature = c("PD-1", "MITF"))
omicsSelections <- new("Selections", features = featuresTable)
```

```
setClass("Selections", representation(features = "characterOrDataFrame"))
selections <- new("Selections", features = c("BRAF", "NRAS"))
featuresTable <- DataFrame(assay = c("RNA-seq", "Mass spectrometry"),
                           feature = c("PD-1", "MITF"))
omicsSelections <- new("Selections", features = featuresTable)
```

`classifyInterface` *An Interface for PoiClaClu Package's Classify Function*

Description

More details of Poisson LDA are available in the documentation of `Classify`. Data tables which consist entirely of non-integer data cannot be analysed.

Usage

```
## S4 method for signature 'matrix'
classifyInterface(countsTrain, classesTrain, countsTest, ...)

## S4 method for signature 'DataFrame'
classifyInterface(
  countsTrain,
  classesTrain,
  countsTest,
  ...,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
classifyInterface(
  countsTrain,
  countsTest,
  targets = names(countsTrain),
  classesTrain,
  ...
)
```

Arguments

`countsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or parameters that `Classify` can accept.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `countsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `countsTrain` if it is a `DataFrame` or the column name in `colData(countsTrain)` if `countsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`countsTest`	An object of the same class as `countsTrain` with no samples in common with `countsTrain` and the same number of features as it.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, matrix of scores for each class, or both labels and scores in a `data.frame`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `countsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the sample information table will be used.

Value

Either a factor vector of predicted classes, a matrix of scores for each class, or a table of both the class labels and class scores, depending on the setting of `returnType`.

Author(s)

Dario Strbenac

Examples

```
if(require(PoiClaClu))
{
  readCounts <- CountDataSet(n = 100, p = 1000, 2, 5, 0.1)
  # Rows are for features, columns are for samples.
  trainData <- readCounts[['x']]
  trainClasses <- factor(paste("Class", readCounts[['y']]))
  testData <- readCounts[['xte']]
  storage.mode(trainData) <- storage.mode(testData) <- "integer"
  classified <- classifyInterface(trainData, trainClasses, testData)

 setNames(table(paste("Class", readCounts[["yte"]]) == classified), c("Incorrect", "Correct"))
}
```

ClassifyResult *Container for Storing Classification Results*

Description

Contains a list of models, table of actual sample classes and predicted classes, the identifiers of features selected for each fold of each permutation or each hold-out classification, and performance metrics such as error rates. This class is not intended to be created by the user. It is created by `runTest` or `runTests`.

Constructor

```
ClassifyResult(characteristics, originalNames, originalFeatures,
          rankedFeatures, chosenFeatures, models, tunedParameters, predictions, actualOutcomes, import
```

`characteristics` A `DataFrame` describing the characteristics of classification done. First column must be named `"charateristic"` and second column must be named `"value"`. If using wrapper functions for feature selection and classifiers in this package, the function names will automatically be generated and therefore it is not necessary to specify them.

`originalNames` All sample names.

`originalFeatures` All feature names. Character vector or `DataFrame` with one row for each feature if the data set has multiple kinds of measurements on the same set of samples.

`rankedFeatures` All features, from most to least important. Character vector or a data frame if data set has multiple kinds of measurements on the same set of samples.

`chosenFeatures` Features selected at each fold. Character vector or a data frame if data set has multiple kinds of measurements on the same set of samples.

`models` All of the models fitted to the training data.

`tunedParameters` Names of tuning parameters and the value chosen of each parameter.

`predictions` A data frame containing sample IDs, predicted class or risk and information about the cross-validation iteration in which the prediction was made.

`actualOutcomes` The known class or survival data of each sample.

`importance` The changes in model performance for each selected variable when it is excluded.

`modellingParams` Stores the object used for defining the model building to enable future reuse.

`finalModel` A model built using all of the sample for future use. For any tuning parameters, the most popular value of the parameter in cross-validation is used.

Summary

`result` is a `ClassifyResult` object.

`show(result)`: Prints a short summary of what `result` contains.

Accessors

`result` is a `ClassifyResult` object.

`sampleNames(result)` Returns a vector of sample names present in the data set.

`allFeatureNames(result)` Returns a vector of features present in the data set.

`actualOutcomes(result)` Returns the known outcomes of each sample.

`models(result)` A `list` of the models fitted for each training.

`chosenFeatureNames(result)` A `list` of the features selected for each training.

`predictions(result)` Returns a `DataFrame` which has columns with test sample, cross-validation and prediction information.

`performance(result)` Returns a `list` of performance measures. This is empty until `calcCVperformance` has been used.

`tunedParameters(result)` Returns a `list` of tuned parameter values. If cross-validation is used, this list will be large, as it stores chosen values for every iteration.

`totalPredictions(result)` A single number representing the total number. of predictions made during the cross-validation procedure.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  data(asthma)

  LOOCVparams <- CrossValParams("Leave-k-Out", leave = 1)
  modellingParams <- ModellingParams()
  classified <-
  runTests(measurements, classes, LOOCVparams, modellingParams,
           DataFrame(characteristic = c("dataset", "classification"),
                     value = c("Asthma", "Different Means"))
           )
  class(classified)
#}
```

`coxnetTrainInterface` *An Interface for glmnet Package's coxnet Function*

Description

An elastic net GLM classifier uses a penalty which is a combination of a lasso penalty and a ridge penalty, scaled by a lambda value, to fit a sparse linear model to the data.

Usage

```
coxnetTrainInterface(measurementsTrain, ...)

## S4 method for signature 'matrix'
coxnetTrainInterface(measurementsTrain, survivalTrain, ...)

## S4 method for signature 'DataFrame'
coxnetTrainInterface(
  measurementsTrain,
  survivalTrain,
  lambda = NULL,
  ...,
  verbose = 3
)

coxnetPredictInterface(model, measurementsTest, ...)

## S4 method for signature 'coxnet,matrix'
coxnetPredictInterface(model, measurementsTest, ...)

## S4 method for signature 'coxnet,DataFrame'
coxnetPredictInterface(
  model,
  measurementsTest,
  survivalTest = NULL,
  lambda,
  ...,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'coxnet,MultiAssayExperiment'
coxnetPredictInterface(
  model,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method (e.g. `verbose`) or, for the training function, options that are used by the `glmnet` function. For the testing function, this variable simply contains any parameters passed from the classification framework to it which aren't used by glmnet's `predict` fuction.
`survivalTrain`	A tabular data type of survival information of the same number of rows as the number of samples in `measurementsTrain` and 2 to 3 columns if it is a `matrix` or a `DataFrame`, or a character vector of length 2 to 3 containing the column names in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a vector of column names, those columns will be removed before training.
`lambda`	The lambda value passed directly to `glmnet` if the training function is used or passed as `s` to `predict.glmnet` if the prediction function is used.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`model`	A trained coxnet, as created by the `glmnet` function.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurements` and the same number of features as it.
`survivalTest`	A `Surv` object or columns from the `measurementsTest` table which contains the follow-up time and status information.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, matrix of scores for each class, or both labels and scores in a `data.frame`.
`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"clinical"` is also a valid value and specifies that integer variables from the clinical data table will be used.

Details

The value of the `family` parameter is fixed to `"cox"` so that classification with survival is possible. During classifier training, if more than one lambda value is considered by specifying a vector of them as input or leaving the default value of NULL, then the chosen value is determined based on classifier resubstitution error rate.

Value

For `coxnetTrainInterface`, an object of type `glmnet`. For `coxnetPredictInterface`, a matrix of containing the link and risk functions. `returnType`.

Examples

```
if(require(glmnet))
 {
   set.seed(51773)
   proteinMatrix <- matrix(rnorm(20*10), nrow = 20, ncol = 10)
   survivalOutcome <- Surv(time = rpois(20,20), event = rbinom(20, 1, 0.2))

   trained <- coxnetTrainInterface(proteinMatrix,
                                      survivalOutcome)

   # Resubstituting training data
   predicted <- coxnetPredictInterface(trained, proteinMatrix)

 }
```

coxph-class *Trained coxph Object*

Description

Enables S4 method dispatching on it.

coxphRanking *Ranking of Differential Variability with coxph Statistic*

Description

Ranks all features from largest coxph statistic to smallest.

Usage

```
## S4 method for signature 'matrix'
coxphRanking(measurementsTrain, survivalTrain, ...)

## S4 method for signature 'DataFrame'
coxphRanking(measurementsTrain, survivalTrain, verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
coxphRanking(
  measurementsTrain,
  targets = names(measurementsTrain),
  survivalTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`survivalTrain`	A tabular data type of survival information of the same number of rows as the number of samples in `measurementsTrain` and 2 to 3 columns if it is a `matrix` or a `DataFrame`, or a character vector of length 2 to 3 containing the column names in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a vector of column names, those columns will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"clinical"` is also a valid value and specifies that numeric variables from the clinical data table will be used.

Details

The calculation of the test statistic is performed by the coxph function from the survival package.

Data tables which consist entirely of non-numeric data cannot be ranked.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

`coxphTrainInterface` *An Interface for survival Package's coxph Function*

Description

Cox proportional hazards.

Usage

```
## S4 method for signature 'matrix'
coxphTrainInterface(measurementsTrain, survivalTrain, ...)

## S4 method for signature 'DataFrame'
coxphTrainInterface(measurementsTrain, survivalTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
```

```
coxphTrainInterface(
  measurementsTrain,
  targets = names(measurementsTrain),
  survivalTrain,
  ...
)

## S4 method for signature 'coxph,matrix'
coxphPredictInterface(model, measurementsTest, ...)

## S4 method for signature 'coxph,DataFrame'
coxphPredictInterface(model, measurementsTest, ..., verbose = 3)

## S4 method for signature 'coxph,MultiAssayExperiment'
coxphPredictInterface(
  model,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method (e.g. `verbose`) or options which are accepted by the `coxph` or `predict.coxph` functions.
`survivalTrain`	A tabular data type of survival information of the same number of rows as the number of samples in `measurementsTrain` and 2 to 3 columns if it is a `matrix` or a `DataFrame`, or a character vector of length 2 to 3 containing the column names in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a vector of column names, those columns will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the clinical data table will be used.
`model`	A trained coxph classifier, as created by `coxphTrainInterface`, which has the same form as the output of `coxph`.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.

Value

For `coxphTrainInterface`, the trained Cox proportional hazards model. For `coxphPredictInterface`, a risk score prediction (natural log scale) for each sample.

Examples

```
#'
  # if(require(randomForest))
  # {
  #   # Genes 76 to 100 have differential expression.
  #   genesMatrix <- sapply(1:25, function(sample) c(rnorm(100, 9, 2)))
  #   genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample)
  #                                    c(rnorm(75, 9, 2), rnorm(25, 14, 2))))
  #   classes <- factor(rep(c("Poor", "Good"), each = 25))
  #   colnames(genesMatrix) <- paste("Sample", 1:ncol(genesMatrix), sep = ' ')
  #   rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix), sep = '-')
  #   trainingSamples <- c(1:20, 26:45)
  #   testingSamples <- c(21:25, 46:50)
  #
  #   trained <- randomForestTrainInterface(genesMatrix[, trainingSamples],
  #                                        classes[trainingSamples])
  #   predicted <- randomForestPredictInterface(trained, genesMatrix[, testingSamples])
  # }
```

`crossValidate` *Cross-validation to evaluate classification performance.*

Description

This function has been designed to facilitate the comparison of classification methods using cross-validation. A selection of typical comparisons are implemented.

Usage

```
crossValidate(
  measurements,
  classes,
  nFeatures = 20,
  selectionMethod = "t-test",
  selectionOptimisation = "Resubstitution",
  classifier = "randomForest",
  multiViewMethod = "none",
  dataCombinations = NULL,
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL
```

```
)

## S4 method for signature 'DataFrame'
crossValidate(
  measurements,
  classes,
  nFeatures = 20,
  selectionMethod = "t-test",
  selectionOptimisation = "Resubstitution",
  classifier = "randomForest",
  multiViewMethod = "none",
  dataCombinations = NULL,
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL
)

## S4 method for signature 'MultiAssayExperiment'
crossValidate(
  measurements,
  classes,
  nFeatures = 20,
  selectionMethod = "t-test",
  selectionOptimisation = "Resubstitution",
  classifier = "randomForest",
  multiViewMethod = "none",
  dataCombinations = NULL,
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL
)

## S4 method for signature 'data.frame'
crossValidate(
  measurements,
  classes,
  nFeatures = 20,
  selectionMethod = "t-test",
  selectionOptimisation = "Resubstitution",
  classifier = "randomForest",
  multiViewMethod = "none",
  dataCombinations = NULL,
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL
```

```
)

## S4 method for signature 'matrix'
crossValidate(
  measurements,
  classes,
  nFeatures = 20,
  selectionMethod = "t-test",
  selectionOptimisation = "Resubstitution",
  classifier = "randomForest",
  multiViewMethod = "none",
  dataCombinations = NULL,
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL
)

## S4 method for signature 'list'
crossValidate(
  measurements,
  classes,
  nFeatures = 20,
  selectionMethod = "t-test",
  selectionOptimisation = "Resubstitution",
  classifier = "randomForest",
  multiViewMethod = "none",
  dataCombinations = NULL,
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL
)

## S4 method for signature 'ClassifyResult'
predict(object, newData)
```

Arguments

`measurements`	Either a `DataFrame`, `data.frame`, `matrix`, `MultiAssayExperiment` or a list of these objects containing the training data. For a `matrix` and `data.frame`, the rows are samples and the columns are features. For a `data.frame` or `MultiAssayExperiment` assay the rows are features and the columns are samples, as is typical in Bioconductor.
`classes`	A vector of class labels of class `factor` of the same length as the number of samples in `measurements` or a character vector of length 1 containing the column name in `measurements` if it is a `DataFrame` or the column name in `colData(measurements)` if `measurements` is a `MultiAssayExperiment`. If a

	column name, that column will be removed before training.
`nFeatures`	The number of features to be used for classification. If this is a single number, the same number of features will be used for all comparisons or datasets. If a numeric vector these will be optimised over using `selectionOptimisation`. If a named vector with the same names of multiple datasets, a different number of features will be used for each dataset. If a named list of vectors, the respective number of features will be optimised over. Set to NULL or "all" if all features should be used.
`selectionMethod`	A character vector of feature selection methods to compare. If a named character vector with names corresponding to different datasets, and performing multiview classification, the respective classification methods will be used on each dataset.
`selectionOptimisation`	A character of "Resubstitution", "Nested CV" or "none" specifying the approach used to optimise nFeatures.
`classifier`	A character vector of classification methods to compare. If a named character vector with names corresponding to different datasets, and performing multiview classification, the respective classification methods will be used on each dataset.
`multiViewMethod`	A character vector specifying the multiview method or data integration approach to use.
`dataCombinations`	A character vector or list of character vectors proposing the datasets or, in the case of a list, combination of datasets to use with each element being a vector of datasets to combine.
`nFolds`	A numeric specifying the number of folds to use for cross-validation.
`nRepeats`	A numeric specifying the the number of repeats or permutations to use for cross-validation.
`nCores`	A numeric specifying the number of cores used if the user wants to use parallelisation.
`characteristicsLabel`	A character specifying an additional label for the cross-validation run.
`object`	A trained model to predict with.
`newData`	The data to use to make predictions with.

Details

`classifier` can be any of the following implemented approaches - randomForest, elasticNet, logistic, SVM, DLDA, kNN, naiveBayes, mixturesNormals.

`selectionMethod` can be any of the following implemented approaches - none, t-test, limma, edgeR, NSC, Bartlett, Levene, DMD, likelihoodRatio, KS or KL.

`multiViewMethod` can take a few different values. Using `merge` will merge or bind the datasets after feature selection. Using `prevlidation` will build prevalidated vectors on all the datasets except the clinical data. There must be a dataset called clinical. Using `pca` will perform pca on each dataset and then merge the top few components with the clinical data. There must be a dataset called clinical.

Value

An object of class ClassifyResult

Examples

```
data(asthma)

# Compare randomForest and SVM classifiers.
result <- crossValidate(measurements, classes, classifier = c("randomForest", "SVM"))
# performancePlot(result)


# Compare performance of different datasets.
# First make a toy example dataset with multiple data types. We'll randomly assign different features to be clinical
# set.seed(51773)
# measurements <- DataFrame(measurements, check.names = FALSE)
# mcols(measurements)$dataset <- c(rep("clinical",20),sample(c("gene", "protein"), ncol(measurements)-20, replac
# mcols(measurements)$feature <- colnames(measurements)

# We'll use different nFeatures for each dataset. We'll also use repeated cross-validation with 5 repeats for speed
# set.seed(51773)
#result <- crossValidate(measurements, classes, nFeatures = c(clinical = 5, gene = 20, protein = 30), classifier = "
# performancePlot(result)

# Merge different datasets. But we will only do this for two combinations. If dataCombinations is not specified it wo
# set.seed(51773)
# resultMerge <- crossValidate(measurements, classes, dataCombinations = list(c("clinical", "protein"), c("clinic
# performancePlot(resultMerge)


# performancePlot(c(result, resultMerge))
```

CrossValParams *Parameters for Cross-validation Specification*

Description

Collects and checks necessary parameters required for cross-validation by runTests.

Usage

```
CrossValParams(
  samplesSplits = c("Permute k-Fold", "Permute Percentage Split", "Leave-k-Out",
    "k-Fold"),
  permutations = 100,
  percentTest = 25,
  folds = 5,
  leave = 2,
  tuneMode = c("Resubstitution", "Nested CV", "none"),
  parallelParams = bpparam()
)
```

Arguments

samplesSplits	Default: "Permute k-Fold". A character value specifying what kind of sample splitting to do.
permutations	Default: 100. Number of times to permute the data set before it is split into training and test sets. Only relevant if samplesSplits is either "Permute k-Fold" or "Permute Percentage Split".
percentTest	The percentage of the data set to assign to the test set, with the remainder of the samples belonging to the training set. Only relevant if samplesSplits is "Permute Percentage Split".
folds	The number of approximately equal-sized folds to partition the samples into. Only relevant if samplesSplits is "Permute k-Fold" or "k-Fold".
leave	The number of samples to generate all possible combination of and use as the test set. Only relevant if samplesSplits is "Leave-k-Out". If set to 1, it is the traditional leave-one-out cross-validation, sometimes written as LOOCV.
tuneMode	Default: Resubstitution. The scheme to use for selecting any tuning parameters.
parallelParams	An instance of BiocParallelParam specifying the kind of parallelisation to use. Default is to use two cores less than the total number of cores the computer has, if it has four or more cores, otherwise one core, as is the default of bpparam. To make results fully reproducible, please choose a specific back-end depending on your operating system and also set RNGseed to a number.

Author(s)

Dario Strbenac

Examples

```
CrossValParams() # Default is 100 permutations and 5 folds of each.
snow <- SnowParam(workers = 4, RNGseed = 999)
CrossValParams("Leave-k-Out", leave = 2, parallelParams = snow)
# Fully reproducible Leave-2-out cross-validation on 4 cores,
# even if feature selection or classifier use random sampling.
```

DataFrameOrNULL-class *Union of NULL and DataFrame Class*

Description

Allows cross-validation to accept data as either a DataFrame (for a single data set) or DataFrameList (for a list of tables of related measurements, such as different projects measuring the same outcome and the same kind of measurements). No constructor.

Author(s)

Dario Strbenac

---

differentMeansRanking *Ranking of Differentially Abundant Features*

Description

Uses an ordinary t-test if the data set has two classes or one-way ANOVA if the data set has three or more classes to select differentially expressed features.

Usage

```
## S4 method for signature 'matrix'
differentMeansRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
differentMeansRanking(measurementsTrain, classesTrain, verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
differentMeansRanking(measurementsTrain, targets = NULL, classesTrain, ...)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	Names of data tables to be combined into a single table and used in the analysis.

Details

This ranking method looks for changes in means and uses `rowttests` to rank the features if there are two classes or `rowFtests` if there are three or more classes.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# Genes 76 to 100 have differential expression.
genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
classes <- factor(rep(c("Poor", "Good"), each = 25))
rownames(genesMatrix) <- paste("Sample", 1:50)
colnames(genesMatrix) <- paste("Gene", 1:100)

ranked <- differentMeansRanking(genesMatrix, classes)
head(ranked)
```

`distribution` *Get Frequencies of Feature Selection and Sample-wise Classification Errors*

Description

There are two modes. For aggregating feature selection results, the function counts the number of times each feature was selected in all cross-validations. For aggregating classification results, the error rate for each sample is calculated. This is useful in identifying outlier samples that are difficult to classify.

Arguments

`result`	An object of class `ClassifyResult`.
`...`	Further parameters, such as `colour` and `fill`, passed to `geom_histogram` or `stat_density`, depending on the value of `plotType`.
`dataType`	Whether to calculate sample-wise error rate or the number of times a feature was selected.
`plotType`	Whether to draw a probability density curve or a histogram.
`summaryType`	Whether to summarise the feature selections as a percentage or count.
`plot`	Whether to draw a plot of the frequency of selection or error rate.
`xMax`	Maximum data value to show in plot.
`fontSizes`	A vector of length 3. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values.

Value

If `dataType` is "features", a vector as long as the number of features that were chosen at least once containing the number of times the feature was chosen in cross validations or the percentage of times chosen. If `dataType` is "samples", a vector as long as the number of samples, containing the cross-validation error rate of the sample. If `plot` is `TRUE`, then a plot is also made on the current graphics device.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  data(asthma)
  CVparams <- CrossValParams(permutations = 5)
  result <- runTests(measurements, classes, CVparams, ModellingParams())
  featureDistribution <- distribution(result, "features", summaryType = "count",
                                      plotType = "histogram", binwidth = 1)
  print(head(featureDistribution))
#}
```

`DLDA Interface` *An Interface for sparsediscrim Package's dlda Function*

Description

`DLDAtrainInterface` generates a trained diagonal LDA classifier and `DLDApredictInterface` uses it to make predictions on a test data set.

Usage

```
## S4 method for signature 'matrix'
DLDAtrainInterface(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
DLDAtrainInterface(measurementsTrain, classesTrain, verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
DLDAtrainInterface(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

```
## S4 method for signature 'dlda,matrix'
DLDApredictInterface(model, measurementsTest, ...)

## S4 method for signature 'dlda,DataFrame'
DLDApredictInterface(
  model,
  measurementsTest,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'dlda,MultiAssayExperiment'
DLDApredictInterface(
  model,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method (e.g. `verbose`).
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the sample information data table will be used.
`model`	A fitted model as returned by `DLDAtrainInterface`.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it. Also, if a `DataFrame`, the `class` column must be absent.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, matrix of scores for each class, or both labels and scores in a `data.frame`.

Value

For `DLDAtrainInterface`, a trained DLDA classifier. For `DLDApredictInterface`, either a factor vector of predicted classes, a matrix of scores for each class, or a table of both the class labels and class scores, depending on the setting of `returnType`.

Author(s)

Dario Strbenac

Examples

```
# if(require(sparsediscrim)) Package currently removed from CRAN.
#{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
  colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))
  selected <- colnames(genesMatrix)[91:100]
  trainingSamples <- c(1:20, 26:45)
  testingSamples <- c(21:25, 46:50)

  classifier <- DLDAtrainInterface(genesMatrix[trainingSamples, selected],
                                 classes[trainingSamples])
  DLDApredictInterface(classifier, genesMatrix[testingSamples, selected])
#}
```

dlda-class — *Trained dlda Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

`DMDranking` *Ranking by Differential Distributions with Differences in Means or Medians and a Deviation Measure*

Description

Ranks features by largest Differences in Means/Medians and Deviations.

Usage

```
## S4 method for signature 'matrix'
DMDranking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
DMDranking(
  measurementsTrain,
  classesTrain,
  differences = c("both", "location", "scale"),
  ...,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
DMDranking(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or parameters for `getLocationsAndScales`, such as `location`, `scale`.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.

`differences`	Default: `"both"`. Either `"both"`, `"location"`, or `"scale"`. The type of differences to consider. If both are considered then the absolute difference in location and the absolute difference in scale are summed.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Details

DMD is defined as $\sum_{i=1} \sum_{j=i+1} |location_i - location_j| + |scale_i - scale_j|$. The subscripts denote the class for which the parameter is calculated for.

Value

A vector of feature indices, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# First 20 features have bimodal distribution for Poor class.
# Other 80 features have normal distribution for both classes.
genesMatrix <- sapply(1:20, function(feature)
                            {
                              randomMeans <- sample(c(8, 12), 25, replace = TRUE)
                              c(rnorm(25, randomMeans, 1), rnorm(25, 10, 1))
                            }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:80, function(feature) rnorm(50, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

ranked <- DMDranking(genesMatrix, classes)
head(ranked)
```

`edgeRranking` *Feature Ranking Based on Differential Expression for Count Data*

Description

Performs a differential expression analysis between classes and ranks the features based on test statistics. The data may have overdispersion and this is modelled.

Usage

```
## S4 method for signature 'matrix'
edgeRranking(countsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
edgeRranking(
  countsTrain,
  classesTrain,
  normFactorsOptions = NULL,
  dispOptions = NULL,
  fitOptions = NULL,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
edgeRranking(countsTrain, targets = NULL, ...)
```

Arguments

`countsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the unnormalised counts. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features, unlike the convention used in edgeR.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `countsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `countsTrain` if it is a `DataFrame` or the column name in `colData(counts)` if `countsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`normFactorsOptions`	A named `list` of any options to be passed to `calcNormFactors`.
`dispOptions`	A named `list` of any options to be passed to `estimateDisp`.
`fitOptions`	A named `list` of any options to be passed to `glmFit`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `countsTrain` is a `MultiAssayExperiment`, the names of the data tables of counts to be used.

Details

The differential expression analysis follows the standard `edgeR` steps of estimating library size normalisation factors, calculating dispersion, in this case robustly, and then fitting a generalised linear model followed by a likelihood ratio test.

Data tables which consist entirely of non-numeric data cannot be analysed.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

References

edgeR: a Bioconductor package for differential expression analysis of digital gene expression data, Mark D. Robinson, Davis McCarthy, and Gordon Smyth, 2010, *Bioinformatics*, Volume 26 Issue 1, `https://academic.oup.com/bioinformatics/article/26/1/139/182458`.

Examples

```
if(require(parathyroidSE) && require(PoiClaClu))
{
  data(parathyroidGenesSE)
  expression <- assays(parathyroidGenesSE)[[1]] # Genes in rows, samples in columns.
  sampleNames <- paste("Sample", 1:ncol(parathyroidGenesSE))
  colnames(expression) <- sampleNames
  DPN <- which(colData(parathyroidGenesSE)[, "treatment"] == "DPN")
  control <- which(colData(parathyroidGenesSE)[, "treatment"] == "Control")
  expression <- expression[, c(control, DPN)]
  classes <- factor(rep(c("Contol", "DPN"), c(length(control), length(DPN))))
  expression <- expression[rowSums(expression > 1000) > 8, ] # Make small data set.

  # ClassifyR is using the convention of samples in rows and features in columns.
  ranked <- edgeRranking(t(expression), classes)

  head(ranked)
  plotFeatureClasses(t(expression), classes, "ENSG00000044574",
                     dotBinWidth = 500, xAxisLabel = "Unnormalised Counts")
}
```

`edgesToHubNetworks` *Convert a Two-column Matrix or Data Frame into a Hub Node List*

Description

Interactions between pairs of features (typically a protein-protein interaction, commonly abbreviated as PPI, database) are restructured into a named list. The name of the each element of the list is a feature and the element contains all features which have an interaction with it.

Usage

```
edgesToHubNetworks(edges, minCardinality = 5)
```

Arguments

`edges`	A two-column `matrix` or `data.frame` for which each row specifies a known interaction betwen two interactors. If feature X appears in the first column and feature Y appears in the second, there is no need for feature Y to appear in the first column and feature X in the second.
`minCardinality`	An integer specifying the minimum number of features to be associated with a hub feature for it to be present in the result.

Value

An object of type `FeatureSetCollection`.

Author(s)

Dario Strbenac

References

VAN: an R package for identifying biologically perturbed networks via differential variability analysis, Vivek Jayaswal, Sarah-Jane Schramm, Graham J Mann, Marc R Wilkins and Yee Hwa Yang, 2010, *BMC Research Notes*, Volume 6 Article 430, `https://bmcresnotes.biomedcentral.com/articles/10.1186/1756-0500-6-430`.

Examples

```
  interactor <- c("MITF", "MITF", "MITF", "MITF", "MITF", "MITF",
                  "KRAS", "KRAS", "KRAS", "KRAS", "KRAS", "KRAS",
                  "PD-1")
  otherInteractor <- c("HINT1", "LEF1", "PSMD14", "PIAS3", "UBE2I", "PATZ1",
                       "ARAF", "CALM1", "CALM2", "CALM3", "RAF1", "HNRNPC",
                       "PD-L1")
  edges <- data.frame(interactor, otherInteractor, stringsAsFactors = FALSE)

  edgesToHubNetworks(edges, minCardinality = 4)
```

`elasticNetFeatures` *Extract Vectors of Ranked and Selected Features From an Elastic Net GLM Object*

Description

Provides a ranking of features based on the magnitude of fitted GLM coefficients. Also provides the selected features which are those with a non-zero coefficient.

Usage

```
## S4 method for signature 'multnet'
elasticNetFeatures(model)
```

Arguments

`model`	A fitted multinomial GLM which was created by `glmnet`.

Value

An `list` object. The first element is a vector or data frame of ranked features, the second is a vector or data frame of selected features.

Author(s)

Dario Strbenac

Examples

```
if(require(glmnet))
{
  # Genes 76 to 100 have differential expression.
genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                   c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
classes <- factor(rep(c("Poor", "Good"), each = 25))
rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))

  # alpha is a user-specified tuning parameter.
  # lambda is automatically tuned, based on glmnet defaults, if not user-specified.
  CVparams <- CrossValParams("k-Fold")

  trainParams <- TrainParams(elasticNetGLMtrainInterface, nlambda = 500)
  predictParams <- PredictParams(elasticNetGLMpredictInterface)
  modParams <- ModellingParams(selectParams = NULL, trainParams = trainParams,
                               predictParams = predictParams)

  classified <- runTests(genesMatrix, classes, CVparams, modParams)

  elasticNetFeatures(models(classified)[[1]])
}
```

---

elasticNetGLMtrainInterface

An Interface for glmnet Package's glmnet Function

Description

An elastic net GLM classifier uses a penalty which is a combination of a lasso penalty and a ridge penalty, scaled by a lambda value, to fit a sparse linear model to the data.

Usage

```
elasticNetGLMtrainInterface(measurementsTrain, ...)

## S4 method for signature 'matrix'
elasticNetGLMtrainInterface(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
elasticNetGLMtrainInterface(
  measurementsTrain,
  classesTrain,
  lambda = NULL,
  ...,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
elasticNetGLMtrainInterface(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)

elasticNetGLMpredictInterface(model, measurementsTest, ...)

## S4 method for signature 'multnet,matrix'
elasticNetGLMpredictInterface(model, measurementsTest, ...)

## S4 method for signature 'multnet,DataFrame'
elasticNetGLMpredictInterface(
  model,
  measurementsTest,
  lambda,
  ...,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'multnet,MultiAssayExperiment'
elasticNetGLMpredictInterface(
  model,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method (e.g. `verbose`) or, for the training function, options that are used by the `glmnet` function. For the testing function, this variable simply contains any parameters passed from the classification framework to it which aren't used by glmnet's `predict` fuction.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`lambda`	The lambda value passed directly to `glmnet` if the training function is used or passed as `s` to `predict.glmnet` if the prediction function is used.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the sample information data table will be used.
`model`	A trained elastic net GLM, as created by the `glmnet` function.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, matrix of scores for each class, or both labels and scores in a `data.frame`.

Details

The value of the `family` parameter is fixed to `"multinomial"` so that classification with more than 2 classes is possible and `type.multinomial` is fixed to `"grouped"` so that a grouped lasso penalty is used. During classifier training, if more than one lambda value is considered by specifying a vector of them as input or leaving the default value of NULL, then the chosen value is determined based on classifier resubstitution error rate.

Value

For `elasticNetGLMtrainInterface`, an object of type `glmnet`. For `elasticNetGLMpredictInterface`, either a factor vector of predicted classes, a matrix of scores for each class, or a table of both the class labels and class scores, depending on the setting of `returnType`.

Author(s)

Dario Strbenac

See Also

`elasticNetFeatures` for a function used to extract the features with non-zero coefficients from the model.

Examples

```
if(require(glmnet))
{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                   c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
  colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))

  CVparams <- CrossValParams("k-Fold")

  trainParams <- TrainParams(elasticNetGLMtrainInterface, nlambda = 500)
  predictParams <- PredictParams(elasticNetGLMpredictInterface)
  modParams <- ModellingParams(selectParams = NULL, trainParams = trainParams,
                                predictParams = predictParams)
  classified <- runTests(genesMatrix, classes, CVparams, modParams)

  classified <- calcCVperformance(classified, "Balanced Error")
  head(tunedParameters(classified))
  performance(classified)
}
```

FeatureSetCollection *Container for Storing A Collection of Sets*

Description

This container is the required storage format for a collection of sets. Typically, the elements of a set will either be a set of proteins (i.e. character vector) which perform a particular biological process or a set of binary interactions (i.e. Two-column matrix of feature identifiers).

Constructor

```
FeatureSetCollection(sets)
```

`sets` A named list. The names of the list describe the sets and the elements of the list specify the features which comprise the sets.

Summary

featureSets is a FeatureSetCollection object.

- show(featureSets): Prints a short summary of what featureSets contains.
- length(featureSets): Prints how many sets of features there are.

Subsetting

The FeatureSetCollection may be subsetted to a smaller set of elements or a single set may be extracted as a vector. featureSets is a FeatureSetCollection object.

- featureSets[i:j]: Reduces the object to a subset of the feature sets between elements i and j of the collection.
- featureSets[[i]]: Extract the feature set identified by i. i may be a numeric index or the character name of a feature set.

Author(s)

Dario Strbenac

Examples

```
ontology <- list(c("SESN1", "PRDX1", "PRDX2", "PRDX3", "PRDX4", "PRDX5", "PRDX6",
                   "LRRK2", "PARK7"),
                 c("ATP7A", "CCS", "NQO1", "PARK7", "SOD1", "SOD2", "SOD3",
                   "SZT2", "TNF"),
                 c("AARS", "AIMP2", "CARS", "GARS", "KARS", "NARS", "NARS2",
                   "LARS2", "NARS", "NARS2", "RGN", "UBA7"),
                 c("CRY1", "CRY2", "ONP1SW", "OPN4", "RGR"),
                 c("ESRRG", "RARA", "RARB", "RARG", "RXRA", "RXRB", "RXRG"),
                 c("CD36", "CD47", "F2", "SDC4"),
                 c("BUD31", "PARK7", "RWDD1", "TAF1")
                 )
names(ontology) <- c("Peroxiredoxin Activity", "Superoxide Dismutase Activity",
                     "Ligase Activity", "Photoreceptor Activity",
                     "Retinoic Acid Receptor Activity",
                     "Thrombospondin Receptor Activity",
                     "Regulation of Androgen Receptor Activity")

featureSets <- FeatureSetCollection(ontology)
featureSets
featureSets[3:5]
featureSets[["Photoreceptor Activity"]]

subNetworks <- list(MAPK = matrix(c("NRAS", "NRAS", "NRAS", "BRAF", "MEK",
                                    "ARAF", "BRAF", "CRAF", "MEK", "ERK"), ncol = 2),
                    P53 = matrix(c("ATM", "ATR", "ATR", "P53",
                                   "CHK2", "CHK1", "P53", "MDM2"), ncol = 2)
                    )
networkSets <- FeatureSetCollection(subNetworks)
networkSets
```

`FeatureSetCollectionOrNULL-class` *Union of a FeatureSetCollection and NULL*

Description

Allows a slot to be either a FeatureSetCollectionOrNULL object or empty.

Author(s)

Dario Strbenac

Examples

```
TrainParams(DLDAtrainInterface, transform = NULL) # Use the input data as-is.
```

`featureSetSummary` *Transform a Table of Feature Abundances into a Table of Feature Set Abundances.*

Description

Represents a feature set by the mean or median feature measurement of a feature set for all features belonging to a feature set.

Usage

```
## S4 method for signature 'matrix'
featureSetSummary(
  measurements,
  location = c("median", "mean"),
  featureSets,
  minimumOverlapPercent = 80,
  verbose = 3
)

## S4 method for signature 'DataFrame'
featureSetSummary(
  measurements,
  location = c("median", "mean"),
  featureSets,
  minimumOverlapPercent = 80,
  verbose = 3
)
```

```
## S4 method for signature 'MultiAssayExperiment'
featureSetSummary(
  measurements,
  target = NULL,
  location = c("median", "mean"),
  featureSets,
  minimumOverlapPercent = 80,
  verbose = 3
)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`location`	Default: The median. The type of location to summarise a set of features belonging to a feature set by.
`featureSets`	An object of type `FeatureSetCollection` which defines the feature sets.
`minimumOverlapPercent`	The minimum percentage of overlapping features between the data set and a feature set defined in `featureSets` for that feature set to not be discarded from the anaylsis.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`target`	If the input is a `MultiAssayExperiment`, this specifies which data set will be transformed. Can either be an integer index or a character string specifying the name of the table. Must have length 1.

Details

This feature transformation method is unusual because the mean or median feature of a feature set for one sample may be different to another sample, whereas most other feature transformation methods do not result in different features being compared between samples during classification.

Value

The same class of variable as the input variable `measurements` is, with the individual features summarised to feature sets. The number of samples remains unchanged, so only one dimension of `measurements` is altered.

Author(s)

Dario Strbenac

References

Network-based biomarkers enhance classical approaches to prognostic gene expression signatures, Rebecca L Barter, Sarah-Jane Schramm, Graham J Mann and Yee Hwa Yang, 2014, *BMC Systems Biology*, Volume 8 Supplement 4 Article S5, https://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-8-S4-S5.

Examples

```
sets <- list(Adhesion = c("Gene 1", "Gene 2", "Gene 3"),
             `Cell Cycle` = c("Gene 8", "Gene 9", "Gene 10"))
featureSets <- FeatureSetCollection(sets)

# Adhesion genes have a median gene difference between classes.
genesMatrix <- matrix(c(rnorm(5, 9, 0.3), rnorm(5, 7, 0.3), rnorm(5, 8, 0.3),
                      rnorm(5, 6, 0.3), rnorm(10, 7, 0.3), rnorm(70, 5, 0.1)),
                      nrow = 10)
rownames(genesMatrix) <- paste("Patient", 1:10)
colnames(genesMatrix) <- paste("Gene", 1:10)
classes <- factor(rep(c("Poor", "Good"), each = 5)) # But not used for transformation.

featureSetSummary(genesMatrix, featureSets = featureSets)
```

fisherDiscriminant *Classification Using Fisher's LDA*

Description

Finds the decision boundary using the training set, and gives predictions for the test set. Unlike ordinary LDA, Fisher's version does not have assumptions about the normality of the features. Data tables which consist entirely of non-numeric data cannot be analysed.

Usage

```
## S4 method for signature 'matrix'
fisherDiscriminant(measurementsTrain, classesTrain, measurementsTest, ...)

## S4 method for signature 'DataFrame'
fisherDiscriminant(
  measurementsTrain,
  classesTrain,
  measurementsTest,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
fisherDiscriminant(
  measurementsTrain,
  measurementsTest,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.
`returnType`	Default: `"both"`. Either `"class"`, `"score"`, or `"both"`. Sets the return value from the prediction to either a vector of class labels, score for a sample belonging to the second class, as determined by the factor levels, or both labels and scores in a `data.frame`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Value

A vector or `data.frame` of class prediction information, as long as the number of samples in the test data.

Author(s)

Dario Strbenac

Examples

```
trainMatrix <- matrix(rnorm(1000, 8, 2), nrow = 10)
classes <- factor(rep(c("Poor", "Good"), each = 5))
```

```
# Make first 30 genes increased in value for poor samples.
trainMatrix[1:5, 1:30] <- trainMatrix[1:5, 1:30] + 5

testMatrix <- matrix(rnorm(1000, 8, 2), nrow = 10)

# Make first 30 genes increased in value for sixth to tenth samples.
testMatrix[6:10, 1:30] <- testMatrix[6:10, 1:30] + 5

fisherDiscriminant(trainMatrix, classes, testMatrix)
```

`forestFeatures` *Extract Vectors of Ranked and Selected Features From a Random Forest Object*

Description

Provides a ranking of features based on the total decrease in node impurities from splitting on the variable, averaged over all trees. Also provides the selected features which are those that were used in at least one tree of the forest.

Arguments

`forest`	A trained random forest which was created by `randomForest`.

Value

An `list` object. The first element is a vector or data frame of features, ranked from best to worst using the Gini index. The second element is a vector or data frame of features used in at least one tree.

Author(s)

Dario Strbenac

Examples

```
if(require(randomForest))
{
   # Genes 76 to 100 have differential expression.
   genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
   genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
   classes <- factor(rep(c("Poor", "Good"), each = 25))
   rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
   colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))
   trainingSamples <- c(1:20, 26:45)
   testingSamples <- c(21:25, 46:50)
```

```
    trained <- randomForestTrainInterface(genesMatrix[trainingSamples, ],
                                          classes[trainingSamples], ntree = 10)

    forestFeatures(trained)
  }
```

functionOrList-class *Union of Functions and List of Functions*

Description

Allows a slot to be either a function or a list of functions.

Author(s)

Dario Strbenac

Examples

```
SelectParams(limmaRanking)
SelectParams(list(limmaRanking, differentMeansRanking)) # Ensemble selection.
```

functionOrNULL-class *Union of A Function and NULL*

Description

Allows a slot to be either a function or empty.

Author(s)

Dario Strbenac

Examples

```
PredictParams(NULL) # Training function does both tasks.
PredictParams(DLDApredictInterface)
```

generateCrossValParams

A function to generate a CrossValParams object

Description

A function to generate a CrossValParams object

Usage

```
generateCrossValParams(nRepeats, nFolds, nCores, selectionOptimisation)
```

Arguments

`nRepeats`	A numeric specifying the the number of repeats or permutations to use for cross-validation.
`nFolds`	A numeric specifying the number of folds to use for cross-validation.
`nCores`	A numeric specifying the number of cores used if the user wants to use parallelisation.
`selectionOptimisation`	A character of "Resubstitution", "Nested CV" or "none" specifying the approach used to optimise nFeatures.

Value

CrossValParams object

Examples

```
CVparams <- generateCrossValParams(nRepeats = 20, nFolds = 5, nCores = 8, selectionOptimisation = "none")
```

generateModellingParams

A function to generate a ModellingParams object

Description

A function to generate a ModellingParams object

Usage

```
generateModellingParams(
  datasetIDs,
  measurements,
  nFeatures,
  selectionMethod,
  selectionOptimisation,
  classifier,
  multiViewMethod = "none"
)
```

Arguments

`datasetIDs`	A vector of data set identifiers as long at the number of data sets.
`measurements`	Either a `DataFrame`, `data.frame`, `matrix`, `MultiAssayExperiment` or a list of these objects containing the training data. For a `matrix` and `data.frame`, the rows are samples and the columns are features. For a `data.frame` or `MultiAssayExperiment` assay the rows are features and the columns are samples, as is typical in Bioconductor.
`nFeatures`	The number of features to be used for classification. If this is a single number, the same number of features will be used for all comparisons or datasets. If a numeric vector these will be optimised over using `selectionOptimisation`. If a named vector with the same names of multiple datasets, a different number of features will be used for each dataset. If a named list of vectors, the respective number of features will be optimised over. Set to NULL or "all" if all features should be used.
`selectionMethod`	A character vector of feature selection methods to compare. If a named character vector with names corresponding to different datasets, and performing multiview classification, the respective classification methods will be used on each dataset.
`selectionOptimisation`	A character of "Resubstitution", "Nested CV" or "none" specifying the approach used to optimise nFeatures.
`classifier`	A character vector of classification methods to compare. If a named character vector with names corresponding to different datasets, and performing multiview classification, the respective classification methods will be used on each dataset.
`multiViewMethod`	A character vector specifying the multiview method or data integration approach to use.

Value

ModellingParams object

Examples

```
data(asthma)
# First make a toy example dataset with multiple data types. We'll randomly assign different features to be clinical
set.seed(51773)
measurements <- DataFrame(measurements, check.names = FALSE)
mcols(measurements)$dataset <- c(rep("clinical",20),sample(c("gene", "protein"), ncol(measurements)-20, replace
mcols(measurements)$feature <- colnames(measurements)
modellingParams <- generateModellingParams(datasetIDs = c("clinical", "gene", "protein"),
                                          measurements = measurements,
                                  nFeatures = list(clinical = 10, gene = 10, protein = 10),
                             selectionMethod = list(clinical = "t-test", gene = "t-test", protein = "t-test"),
                                          selectionOptimisation = "none",
                                          classifier = "randomForest",
                                          multiViewMethod = "merge")
```

---

getLocationsAndScales *Calculate Location and Scale*

Description

Calculates the location and scale for each feature.

Usage

```
## S4 method for signature 'matrix'
getLocationsAndScales(measurements, ...)

## S4 method for signature 'DataFrame'
getLocationsAndScales(
  measurements,
  location = c("mean", "median"),
  scale = c("SD", "MAD", "Qn")
)

## S4 method for signature 'MultiAssayExperiment'
getLocationsAndScales(measurements, targets = names(measurements), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`location`	The type of location to be calculated.
`scale`	The type of scale to be calculated.

`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Details

`"SD"` is used to represent standard deviation and `"MAD"` is used to represent median absolute deviation.

Value

A `list` of length 2. The first element contains the location for every feature. The second element contains the scale for every feature.

Author(s)

Dario Strbenac

References

Qn: `http://www.tandfonline.com/doi/pdf/10.1080/01621459.1993.10476408`

Examples

```
genesMatrix <- matrix(rnorm(1000, 8, 4), nrow = 10)
distributionInfo <- getLocationsAndScales(genesMatrix, "median", "MAD")

mean(distributionInfo[["median"]]) # Typical median.
mean(distributionInfo[["MAD"]]) # Typical MAD.
```

HuRI *Human Reference Interactome*

Description

A collection of 45783 pairs of protein gene symbols, as determined by the The Human Reference Protein Interactome Mapping Project. Self-interactions have been removed.

Format

`interactors` is a `Pairs` object containing each pair of interacting proteins.

Source

A Reference Map of the Human Binary Protein Interactome, *Nature*, 2020. Webpage: `http://www.interactome-atlas.org/download`

interactorDifferences *Convert Individual Features into Differences Between Binary Interactors Based on Known Sub-networks*

Description

This conversion is useful for creating a meta-feature table for classifier training and prediction based on sub-networks that were selected based on their differential correlation between classes.

Usage

```
## S4 method for signature 'matrix'
interactorDifferences(measurements, ...)

## S4 method for signature 'DataFrame'
interactorDifferences(
  measurements,
  featurePairs = NULL,
  absolute = FALSE,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
interactorDifferences(measurements, target = NULL, classesColumn, ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`featurePairs`	A object of type `Pairs`.
`absolute`	If TRUE, then the absolute values of the differences are returned.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`target`	If `measurements` is a `MultiAssayExperiment`, the name of the data table to be used.
`classesColumn`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the class column in the table extracted by `colData(multiAssayExperiment)` that contains the samples' outcomes to use for prediction.

Details

The pairs of features known to interact with each other are specified by `networkSets`.

Value

An object of class DataFrame with one column for each interactor pair difference and one row for each sample. Additionally, `mcols(resultTable)` prodvides a DataFrame with a column named "original" containing the name of the sub-network each meta-feature belongs to.

Author(s)

Dario Strbenac

References

Dynamic modularity in protein interaction networks predicts breast cancer outcome, Ian W Taylor, Rune Linding, David Warde-Farley, Yongmei Liu, Catia Pesquita, Daniel Faria, Shelley Bull, Tony Pawson, Quaid Morris and Jeffrey L Wrana, 2009, *Nature Biotechnology*, Volume 27 Issue 2, https://www.nature.com/articles/nbt.1522.

Examples

```
pairs <- Pairs(rep(c('A', 'G'), each = 3), c('B', 'C', 'D', 'H', 'I', 'J'))

# Consistent differences for interactors of A.
measurements <- matrix(c(5.7, 10.1, 6.9, 7.7, 8.8, 9.1, 11.2, 6.4, 7.0, 5.5,
                         3.6, 7.6, 4.0, 4.4, 5.8, 6.2, 8.1, 3.7, 4.4, 2.1,
                         8.5, 13.0, 9.9, 10.0, 10.3, 11.9, 13.8, 9.9, 10.7, 8.5,
                         8.1, 10.6, 7.4, 10.7, 10.8, 11.1, 13.3, 9.7, 11.0, 9.1,
                         round(rnorm(60, 8, 0.3), 1)), nrow = 10)

rownames(measurements) <- paste("Patient", 1:10)
colnames(measurements) <- LETTERS[1:10]

interactorDifferences(measurements, pairs)
```

---

kNNinterface   *An Interface for class Package's knn Function*

Description

More details of k Nearest Neighbours are available in the documentation of knn.

Usage

```
## S4 method for signature 'matrix'
kNNinterface(measurementsTrain, classesTrain, measurementsTest, ...)

## S4 method for signature 'DataFrame'
kNNinterface(
  measurementsTrain,
  classesTrain,
  measurementsTest,
  ...,
  classifierName = "k Nearest Neighbours",
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
kNNinterface(
  measurementsTrain,
  measurementsTest,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or parameters that `knn` can accept.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.
`classifierName`	Default: k Nearest Neighbours. Useful for automated plot annotation by plotting functions within this package.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the sample information data table will be used.

Details

Data tables which consist entirely of non-numeric data cannot be analysed. If `measurements` is an object of class `MultiAssayExperiment`, the factor of sample classes must be stored in the DataFrame accessible by the `colData` function with column name `"class"`.

Value

A factor vector, the same as is returned by knn.

Author(s)

Dario Strbenac

Examples

```
if(require(class))
{
 classes <- factor(rep(c("Healthy", "Disease"), each = 5), levels = c("Healthy", "Disease"))
  measurements <- matrix(c(rnorm(50, 10), rnorm(50, 5)), nrow = 10, byrow = TRUE)
  rownames(measurements) <- paste("Sample", 1:10)
  colnames(measurements) <- paste("mRNA", 1:10)

  # Train with 9 samples, test with one.
  kNNinterface(measurements[1:9, ], classes[1:9], measurements[10, , drop = FALSE])
}
```

KolmogorovSmirnovRanking

Ranking of Differential Distributions with Kolmogorov-Smirnov Distance

---

Description

Ranks features from largest Kolmogorov-Smirnov distance to smallest.

Usage

```
## S4 method for signature 'matrix'
KolmogorovSmirnovRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
KolmogorovSmirnovRanking(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
KolmogorovSmirnovRanking(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or options which are accepted by the function `ks.test`.
`classesTrain`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` or if the measurements are of class `DataFrame` a character vector of length 1 containing the column name in `measurement` is also permitted.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# First 20 features have bimodal distribution for Poor class.
# Other 80 features have normal distribution for both classes.
set.seed(1984)
genesMatrix <- sapply(1:20, function(feature)
                            {
                              randomMeans <- sample(c(8, 12), 25, replace = TRUE)
                              c(rnorm(25, randomMeans, 1), rnorm(25, 10, 1))
                            }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:80, function(feature) rnorm(50, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

ranked <- KolmogorovSmirnovRanking(genesMatrix, classes)
head(ranked)
```

kTSPclassifier *Classification Using k Pairs of Features With Relative Differences Between Classes*

Description

Each pair of features votes for a class based on whether the value of one feature is less than the other feature. If the voting is tied, the the class with the most samples in the training set is voted for.

Usage

```
## S4 method for signature 'matrix'
kTSPclassifier(
  measurementsTrain,
  classesTrain,
  measurementsTest,
  featurePairs = NULL,
  ...
)

## S4 method for signature 'DataFrame'
kTSPclassifier(
  measurementsTrain,
  classesTrain,
  measurementsTest,
  featurePairs = NULL,
  difference = c("unweighted", "weighted"),
  minDifference = 0,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
kTSPclassifier(
  measurementsTrain,
  classesTrain,
  target = names(measurementsTrain)[1],
  featurePairs = NULL,
  ...
)
```

Arguments

`measurementsTrain`
: Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.

`...`	Unused variables by the methods for a `matrix` or a `MultiAssayExperiment` passed to the `DataFrame` method which does the classification.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.
`featurePairs`	An object of class as `Pairs` containing the pairs of features to determine whether the inequality of the first feature being less than the second feature holds, indicating evidence for the second level of the `classesTrain` factor.
`difference`	Default: `"unweighted"`. Either `"unweighted"` or `"weighted"`. In weighted mode, the difference in densities is summed over all features. If unweighted mode, each feature's vote is worth the same.
`minDifference`	Default: 0. The minimum difference in densities for a feature to be allowed to vote. Can be a vector of cutoffs. If no features for a particular sample have a difference large enough, the class predicted is simply the largest class.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, score for a sample belonging to the second class, as determined by the factor levels, or both labels and scores in a `data.frame`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`target`	If `measurementsTrain` is a `MultiAssayExperiment`, the name of the data table to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the sample information data table will be used.

Details

Because this method compares different features, they need to have comparable measurements. For example, RNA-seq counts would be unsuitable since these depend on the length of a feature, whereas F.P.K.M. values would be suitable.

The `featurePairs` to use is recommended to be determined in conjunction with `pairsDifferencesRanking`.

Value

A vector or data frame of class prediction information, as long as the number of samples in the test data.

Author(s)

Dario Strbenac

See Also

pairsDifferencesRanking for a function which could be used to do feature ranking before the k-TSP classifier is run.

Examples

```
# Difference in differences for features A and C between classes.
measurements <- matrix(c(9.9, 10.5, 10.1, 10.9, 11.0, 6.6, 7.7, 7.0, 8.1, 6.5,
                         8.5, 10.5, 12.5, 10.5, 9.5, 8.5, 10.5, 12.5, 10.5, 9.5,
                         6.6, 7.7, 7.0, 8.1, 6.5, 11.2, 11.0, 11.1, 11.4, 12.0,
                         8.1, 10.6, 7.4, 7.1, 10.4, 6.1, 7.3, 2.7, 11.0, 9.1,
                         round(rnorm(60, 8, 1), 1)), ncol = 10)
classes <- factor(rep(c("Good", "Poor"), each = 5))

rownames(measurements) <- paste("Patient", 1:10)
colnames(measurements) <- LETTERS[1:10]

trainIndex <- c(1:4, 6:9)
trainMatrix <- measurements[trainIndex, ]
testMatrix <- measurements[c(5, 10), ]

featurePairs <- Pairs('A', 'C') # Could be ranked by pairsDifferencesRanking function and
                                # selected internally by tuning of features selected.
kTSPclassifier(trainMatrix, classes[trainIndex], testMatrix, featurePairs)
```

KullbackLeiblerRanking

Ranking of Differential Distributions with Kullback-Leibler Distance

---

Description

Ranks features from largest Kullback-Leibler distance between classes to smallest.

Usage

```
## S4 method for signature 'matrix'
KullbackLeiblerRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
KullbackLeiblerRanking(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
KullbackLeiblerRanking(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or options which are accepted by the function `getLocationsAndScales`.
`classesTrain`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` or if the measurements are of class `DataFrame` a character vector of length 1 containing the column name in `measurement` is also permitted.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Details

The distance is defined as $\frac{1}{2} \times \left(\frac{(location_1 - location_2)^2}{scale_1^2} + \frac{(location_1 - location_2)^2}{scale_2^2} + \frac{scale_2^2}{scale_1^2} + \frac{scale_1^2}{scale_2^2}\right)$

The subscripts denote the group which the parameter is calculated for.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# First 20 features have bimodal distribution for Poor class.
# Other 80 features have normal distribution for both classes.
genesMatrix <- sapply(1:20, function(feature)
                            {
                              randomMeans <- sample(c(8, 12), 25, replace = TRUE)
                              c(rnorm(25, randomMeans, 1), rnorm(25, 10, 1))
                            }
                      )
genesMatrix <- cbind(genesMatrix, sapply(1:80, function(feature) rnorm(50, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

ranked <- KullbackLeiblerRanking(genesMatrix, classes)
head(ranked)
```

`leveneRanking` *Selection of Differential Variability with Levene Statistic*

Description

Ranks features by largest Levene statistic.

Usage

```
## S4 method for signature 'matrix'
leveneRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
leveneRanking(measurementsTrain, classesTrain, verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
leveneRanking(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` or if the measurements are of class `DataFrame` a character vector of length 1 containing the column name in `measurement` is also permitted.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information table will be used.

Details

Levene's statistic for unequal variance between groups is a robust version of Bartlett's statistic.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# First 20 features have bimodal distribution for Poor class.
# Other 80 features have normal distribution for both classes.
set.seed(1984)
genesMatrix <- sapply(1:20, function(feature)
                            {
                              randomMeans <- sample(c(8, 12), 25, replace = TRUE)
                              c(rnorm(25, randomMeans, 1), rnorm(25, 10, 1))
                            }
                      )
genesMatrix <- cbind(genesMatrix, sapply(1:80, function(feature) rnorm(50, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

ranked <- leveneRanking(genesMatrix, classes)
head(ranked)
```

---

likelihoodRatioRanking

Ranking of Differential Distributions with Likelihood Ratio Statistic

---

Description

Ranks features from largest difference of log likelihoods (null hypothesis - alternate hypothesis) to smallest.

Usage

```
## S4 method for signature 'matrix'
likelihoodRatioRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
likelihoodRatioRanking(
  measurementsTrain,
  classesTrain,
  alternative = c(location = "different", scale = "different"),
  ...,
  verbose = 3
)
```

```
## S4 method for signature 'MultiAssayExperiment'
likelihoodRatioRanking(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or options which are accepted by the function `getLocationsAndScales`.
`classesTrain`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` or if the measurements are of class `DataFrame` a character vector of length 1 containing the column name in `measurement` is also permitted. Not used if `measurementsTrain` is a `MultiAssayExperiment` object.
`alternative`	Default: `c("different", "different")`. A vector of length 2. The first element specifies the location of the alternate hypothesis. The second element specifies the scale of the alternate hypothesis. Valid values in each element are "same" or "different".
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Details

Likelihood ratio test of null hypothesis that the location and scale are the same for both groups, and an alternate hypothesis that is specified by parameters. The location and scale of features is calculated by `getLocationsAndScales`. The distribution fitted to the data is the normal distribution.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
# First 20 features have bimodal distribution for Poor class.
# Other 80 features have normal distribution for both classes.

genesMatrix <- sapply(1:20, function(feature)
                            {
                              randomMeans <- sample(c(8, 12), 25, replace = TRUE)
                              c(rnorm(25, randomMeans, 1), rnorm(25, 10, 1))
                            }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:80, function(feature) rnorm(50, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

ranked <- likelihoodRatioRanking(genesMatrix, classes)
head(ranked)
```

`limmaRanking` *Ranking of Differentially Abundant Features*

Description

Uses a moderated F-test with empirical Bayes shrinkage to rank differentially expressed features based on differences of means. This means it works when there are three or more classes.

Usage

```
## S4 method for signature 'matrix'
limmaRanking(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
limmaRanking(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
limmaRanking(measurementsTrain, targets = NULL, classesTrain, ...)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or optional settings that are passed to `lmFit`.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurements`.

`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	Names of data tables to be combined into a single table and used in the analysis.

Details

This ranking method looks for changes in means and uses a moderated F-test to do so.

Value

A vector or data frame (if `MultiAssayExperiment` input) of features, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

References

Limma: linear models for microarray data, Gordon Smyth, 2005, In: Bioinformatics and Computational Biology Solutions using R and Bioconductor, Springer, New York, pages 397-420.

Examples

```
#if(require(sparsediscrim))
#{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
  colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))

  ranked <- limmaRanking(genesMatrix, classes)
  head(ranked)
#}
```

`listOrNULL-class` *Union of a List and NULL*

Description

Allows a slot to be either a list or a NULL.

Author(s)

Dario Strbenac

Examples

```
setClass("EasyClassifier", representation(model = "listOrNULL"))
classifier <- new("EasyClassifier", model = NULL) # Optimistic classifier.
```

mixModelsTrain *Classification based on Differential Distribution utilising Mixtures of Normals*

Description

Fits mixtures of normals for every feature, separately for each class.

Usage

```
## S4 method for signature 'matrix'
mixModelsTrain(measurementsTrain, ...)

## S4 method for signature 'DataFrame'
mixModelsTrain(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
mixModelsTrain(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)

## S4 method for signature 'MixModelsListsSet,matrix'
mixModelsPredict(models, measurementsTest, ...)

## S4 method for signature 'MixModelsListsSet,DataFrame'
mixModelsPredict(
  models,
  measurementsTest,
  difference = c("unweighted", "weighted"),
  weighting = c("height difference", "crossover distance"),
  densityXvalues = 1024,
  minDifference = 0,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'MixModelsListsSet,MultiAssayExperiment'
mixModelsPredict(
  models,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or extra arguments for training passed to `mixmodCluster`. The argument `nbCluster` is mandatory.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the clinical data table will be used.
`models`	A `MixModelsListsSet` of models generated by the training function and training class information. There is one element for each class. Another element at the end of the list has the class sizes of the classes in the training data.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.
`difference`	Default: `"unweighted"`. Either `"unweighted"` or `"weighted"`. In weighted mode, the difference in densities is summed over all features. If unweighted mode, each feature's vote is worth the same. Both can be calculated simultaneously.
`weighting`	Default: `"height difference"`. Either `"height difference"`, or `"crossover distance"`. The type of weight to calculate. For `"height difference"`, the weight of each prediction is equal to the sum of the vertical distances for all of the mixture components within one class subtracted from the sum of the components of the other class, summed for each value of x. For `"crossover distance"`, the x positions where the mixture density of the class being considered crosses another class' density is firstly calculated. The predicted class is the class with the highest mixture sum at the particular value of x and the weight is the distance of x from the nearest density crossover point.

`densityXvalues`	Default: 1024. Only relevant when `weight` is `"crossover distance"`. The number of equally-spaced locations at which to calculate y values for each mixture density.
`minDifference`	Default: 0. The minimum difference in sums of mixture densities between the class with the highest sum and the class with the second highest sum for a feature to be allowed to vote. If no features for a particular sample have a difference large enough, the class predicted is simply the largest class.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of predicted classes, a matrix of scores with columns corresponding to classes, as determined by the factor levels of `classesTrain`, or both a column of predicted classes and columns of class scores in a `data.frame`.

Details

If `weighted` is `TRUE`, then a sample's predicted class is the class with the largest sum of weights, each scaled for the number of samples in the training data of each class. Otherwise, when `weighted` is `FALSE`, each feature has an equal vote, and votes for the class with the largest weight, scaled for class sizes in the training set.

If `weight` is `"crossover distance"`, the crossover points are computed by considering the distance between y values of the two densities at every x value. x values for which the sign of the difference changes compared to the difference of the closest lower value of x are used as the crossover points.

Value

For `mixModelsTrain`, a list of trained models of class `MixmodCluster`. For `mixModelsPredict`, a vector of class prediction information (i.e. classes and/or scores), as long as the number of samples in the test data.

Author(s)

Dario Strbenac

Examples

```
# First 25 samples and first 5 genes are mixtures of two normals. Last 25 samples are
# one normal.

genesMatrix <- t(sapply(1:25, function(geneColumn) c(rnorm(5, sample(c(5, 15), replace = TRUE, 5)))))
genesMatrix <- rbind(genesMatrix, sapply(1:5, function(geneColumn) c(rnorm(25, 9, 1))))
genesMatrix <- cbind(genesMatrix, sapply(1:5, function(geneColumn) rnorm(50, 9, 1)))
rownames(genesMatrix) <- paste("Sample", 1:50)
colnames(genesMatrix) <- paste("Gene", 1:10)
classes <- factor(rep(c("Poor", "Good"), each = 25), levels = c("Good", "Poor"))

trainSamples <- c(1:15, 26:40)
testSamples <- c(16:25, 41:50)
selected <- 1:5

trained <- mixModelsTrain(genesMatrix[trainSamples, selected], classes[trainSamples],
                            nbCluster = 1:3)
mixModelsPredict(trained, genesMatrix[testSamples, selected])
```

ModellingParams *Parameters for Data Modelling Specification*

Description

Collects and checks necessary parameters required for data modelling. Apart from data transfomation that needs to be done within cross-validation (e.g. subtractFromLocation), feature selection, model training and prediction, this container also stores a setting for class imbalance rebalancing.

Usage

```
ModellingParams(
  balancing = c("downsample", "upsample", "none"),
  transformParams = NULL,
  selectParams = SelectParams(),
  trainParams = TrainParams(),
  predictParams = PredictParams(),
  doImportance = FALSE
)
```

Arguments

balancing	Default: "downsample". A character value specifying what kind of class balancing to do, if any.
transformParams	Parameters used for feature transformation inside of C.V. specified by a TransformParams instance. Optional, can be NULL.
selectParams	Parameters used during feature selection specified by a SelectParams instance. By default, parameters for selection based on differences in means of numeric data. Optional, can be NULL.
trainParams	Parameters for model training specified by a TrainParams instance. By default, uses diagonal LDA.
predictParams	Parameters for model training specified by a PredictParams instance. By default, uses diagonal LDA.
doImportance	Default: FALSE. Whether or not to carry out removal of each feature, one at a time, which was chosen and then retrain and model and predict the test set, to measure the change in performance metric. Can also be set to TRUE, if required. Modelling run time will be noticeably longer.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
   ModellingParams() # Default is differences in means selection and DLDA.
   ModellingParams(selectParams = NULL, # No feature selection before training.
                   trainParams = TrainParams(randomForestTrainInterface),
                   predictParams = PredictParams(randomForestPredictInterface))
#}
```

`ModellingParamsOrNULL-class` *Union of A ModellingParams Object and NULL*

Description

Allows a slot to be either a ModellingParams class object or empty. No constructor.

`multnet-class` *Trained multnet Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

`naiveBayesKernel` *Classification Using A Bayes Classifier with Kernel Density Estimates*

Description

Kernel density estimates are fitted to the training data and a naive Bayes classifier is used to classify samples in the test data.

Usage

```
## S4 method for signature 'matrix'
naiveBayesKernel(measurementsTrain, classesTrain, measurementsTest, ...)

## S4 method for signature 'DataFrame'
naiveBayesKernel(
  measurementsTrain,
  classesTrain,
  measurementsTest,
  densityFunction = density,
  densityParameters = list(bw = "nrd0", n = 1024, from =
    expression(min(featureValues)), to = expression(max(featureValues))),
  difference = c("unweighted", "weighted"),
  weighting = c("height difference", "crossover distance"),
  minDifference = 0,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
naiveBayesKernel(
  measurementsTrain,
  measurementsTest,
  targets = names(measurements),
  classesTrain,
  ...
)
```

Arguments

`measurementsTrain`
: Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.

`...`
: Unused variables by the three top-level methods passed to the internal method which does the classification.

`classesTrain`
: A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.

`measurementsTest`
: An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.

`densityFunction`	Default: `density`. A function which will return a probability density, which is essentially a list with x and y coordinates.
`densityParameters`	A list of options for `densityFunction`. Default: `list(bw = "nrd0", n = 1024, from = expression(min(featureValues)), to = expression(max(featureValues))`.
`difference`	Default: `"unweighted"`. Either `"unweighted"`, `"weighted"`. In weighted mode, the difference in densities is summed over all features. If unweighted mode, each feature's vote is worth the same.
`weighting`	Default: `"height difference"`. Either `"height difference"` or `"height difference"`. The type of weight to calculate. For `"height difference"`, the weight of each prediction is equal to the vertical distance between the highest density and the second-highest, for a particular value of x. For `"crossover distance"`, the x positions where two densities cross is firstly calculated. The predicted class is the class with the highest density at the particular value of x and the weight is the distance of x from the nearest density crossover point.
`minDifference`	Default: 0. The minimum difference in density height between the highest density and second-highest for a feature to be allowed to vote. If no features for a particular sample have a difference large enough, the class predicted is simply the largest class.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of predicted classes, a matrix of scores with columns corresponding to classes, as determined by the factor levels of `classes`, or both a column of predicted classes and columns of class scores in a `data.frame`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Details

If `difference` is `"weighted"`, then a sample's predicted class is the class with the largest sum of weights, each scaled for the number of samples in the training data of each class. Otherwise, when `difference` is `"unweighted"`, each feature has an equal vote, and votes for the class with the largest weight, scaled for class sizes in the training set.

The variable name of each feature's measurements in the iteration over all features is `featureValues`. This is important to know if each feature's measurements need to be referred to in the specification of `densityParameters`, such as for specifying the range of x values of the density function to be computed. For example, see the default value of `densityParameters` above.

If `weight` is `"crossover distance"`, the crossover points are computed by considering the distance between y values of all of the densities at every x value. x values for which a class density crosses any other class' density are used as the crossover points for that class.

Value

A vector or data frame of class prediction information (i.e. classes and/or scores), as long as the number of samples in the test data.

Author(s)

Dario Strbenac, John Ormerod

Examples

```
trainMatrix <- matrix(rnorm(1000, 8, 2), nrow = 10)
classesTrain <- factor(rep(c("Poor", "Good"), each = 5))
rownames(trainMatrix) <- paste("Sample", 1:10)

# Make first 30 genes increased in value for poor samples.
trainMatrix[1:5, 1:30] <- trainMatrix[1:5, 1:30] + 5

testMatrix <- matrix(rnorm(1000, 8, 2), nrow = 10)
rownames(testMatrix) <- paste("Sample", 11:20)

# Make first 30 genes increased in value for sixth to tenth samples.
testMatrix[6:10, 1:30] <- testMatrix[6:10, 1:30] + 5

naiveBayesKernel(trainMatrix, classesTrain, testMatrix)
```

`NSCtrainInterface` *Interface for* `pamr.train` *Function from* `pamr` *CRAN Package*

Description

Restructures variables from ClassifyR framework to be compatible with `pamr.train` definition.

Restructures variables from ClassifyR framework to be compatible with `pamr.predict` definition.

Extracts the threshold for the minimum training error and then extracts the corresponding gene IDs of the genes that were not eliminated by the thresold.

Usage

```
## S4 method for signature 'matrix'
NSCtrainInterface(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
NSCtrainInterface(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
NSCtrainInterface(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)

## S4 method for signature 'pamrtrained,DataFrame'
NSCpredictInterface(
  model,
  measurementsTest,
  classesColumnTest = NULL,
  ...,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'pamrtrained,MultiAssayExperiment'
NSCpredictInterface(
  model,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)

## S4 method for signature 'pamrtrained'
NSCfeatures(model, measurementsTrain, classesTrain)
```

Arguments

`measurementsTrain`	A `DataFrame` containing the training data.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or optional settings that are passed to `pamr.predict`.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTest` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information table will be used.
`model`	The output of `NSCtrainInterface`, which is identical to the output of `pamr.listgenes`.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` used in the training stage and the same number of features as it. Also, if a `DataFrame`, the `classesTrain` column must be absent.
`classesColumnTest`	Either NULL or a character vector of length 1, specifying the column name to remove from the test set.

`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, score for a sample belonging to the second class, as determined by the factor levels, or both labels and scores in a `data.frame`.

Details

This function is an interface between the ClassifyR framework and `pamr.train`.

This function is an interface between the ClassifyR framework and `pamr.predict`. It selects the highest threshold that gives the minimum error rate in the training data.

When used within ClassifyR cross-validation, the trained model, measurements and classes will automatically be passed to this function in each iteration.

Value

A list with elements as described in `pamr.train`.

Either a factor vector of predicted classes, a matrix of scores for each class, or a table of both the class labels and class scores, depending on the setting of `returnType`.

A list with the first element being empty (no feature ranking is provided) and second element being the selected features.

Author(s)

Dario Strbenac

Dario Strbenac

Dario Strbenac

See Also

`pamr.train` for the function that was interfaced to.

`pamr.predict` for the function that was interfaced to.

`pamr.listgenes` for the function that is interfaced to.

Examples

```
if(require(pamr))
{
 # Samples in one class with differential expression to other class for last 25 features.
  genesMatrix <- sapply(1:100, function(sampleColumn) c(rnorm(25, 9, 1)))
 genesMatrix <- rbind(genesMatrix, cbind(sapply(1:50, function(sampleColumn) rnorm(25, 9, 1)),
                              sapply(1:50, function(sampleColumn) rnorm(25, 14, 1))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))

  NSCtrainInterface(genesMatrix, classes)
}


if(require(pamr))
```

```
{
 # Samples in one class with differential expression to other class for last 25 features.
  genesMatrix <- sapply(1:100, function(sampleColumn) c(rnorm(25, 9, 1)))
 genesMatrix <- rbind(genesMatrix, cbind(sapply(1:50, function(sampleColumn) rnorm(25, 9, 1)),
                              sapply(1:50, function(sampleColumn) rnorm(25, 14, 1))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))

  fit <- NSCtrainInterface(genesMatrix[c(1:20, 26:45), ], classes[c(1:20, 26:45)])
  NSCpredictInterface(fit, genesMatrix[c(21:25, 46:50), ])
}


if(require(pamr))
{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
  colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))

  model <- NSCtrainInterface(genesMatrix, classes)
  selected <- NSCfeatures(model, genesMatrix, classes)
  selected[[2]]
}
```

`numericOrNULL-class` *Union of A Numeric Value and NULL*

Description

Allows a slot to be either a numeric value or empty. No constructor.

Author(s)

Dario Strbenac

`pairsDifferencesRanking` *Ranking of Pairs of Features that are Different Between Classes*

Description

Ranks pre-specified pairs of features by the largest difference of the sum of measurement differences over all samples within a class.

Usage

```
## S4 method for signature 'matrix'
pairsDifferencesRanking(
  measurementsTrain,
  classesTrain,
  featurePairs = NULL,
  ...
)

## S4 method for signature 'DataFrame'
pairsDifferencesRanking(
  measurementsTrain,
  classesTrain,
  featurePairs = NULL,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
pairsDifferencesRanking(
  measurementsTrain,
  target = names(measurementsTrain)[1],
  classesTrain,
  featurePairs = NULL,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` or if the measurements are of class `DataFrame` a character vector of length 1 containing the column name in `measurement` is also permitted.
`featurePairs`	An S4 object of type `Pairs` containing feature identifiers to calculate the sum of differences within each class for.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`target`	If `measurementsTrain` is a `MultiAssayExperiment`, the name of the data table to be used.

Details

Instead of considering whether one feature in a pair of features is consistently lower or higher than the other in the pair, this method takes the sum of differences across all samples within a class, to prevent ties in the ranking of pairs of features.

Value

A `Pairs` object, from the most promising feature pair in the first position to the least promising feature pair in the last position.

Author(s)

Dario Strbenac

References

Simple decision rules for classifying human cancers from gene expression profiles, Aik C Tan, Daniel Q Naiman, Lei Xu, Raimond L. Winslow and Donald Geman, 2005, *Bioinformatics*, Volume 21 Issue 20, `https://academic.oup.com/bioinformatics/article/21/20/3896/203010`.

See Also

`kTSPclassifier` for a classifier which makes use of the pairs of selected features in classification.

Examples

```
featurePairs <- Pairs(c('A', 'C'), c('B', 'C'))

# Difference in differences for features A and C between classes.
measurements <- matrix(c(9.9, 10.5, 10.1, 10.9, 11.0, 6.6, 7.7, 7.0, 8.1, 6.5,
                         8.5, 10.5, 12.5, 10.5, 9.5, 8.5, 10.5, 12.5, 10.5, 9.5,
                         6.6, 7.7, 7.0, 8.1, 6.5, 11.2, 11.0, 11.1, 11.4, 12.0,
                         8.1, 10.6, 7.4, 7.1, 10.4, 6.1, 7.3, 2.7, 11.0, 9.1,
                         round(rnorm(60, 8, 1), 1)), nrow = 10)
classes <- factor(rep(c("Good", "Poor"), each = 5))

rownames(measurements) <- paste("Patient", 1:10)
colnames(measurements) <- LETTERS[1:10]

pairsDifferencesRanking(measurements, classes, featurePairs = featurePairs)
```

pamrtrained-class *Trained pamr Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

---

performancePlot *Plot Performance Measures for Various Classifications*

Description

Draws a graphical summary of a particular performance measure for a list of classifications

Usage

```
## S4 method for signature 'list'
performancePlot(
  results,
  performanceName = "Balanced Accuracy",
  characteristicsList = list(x = "Classifier Name"),
  aggregate = character(),
  coloursList = list(),
  orderingList = list(),
  densityStyle = c("box", "violin"),
  yLimits = NULL,
  fontSizes = c(24, 16, 12, 12),
  title = NULL,
  margin = grid::unit(c(1, 1, 1, 1), "lines"),
  rotate90 = FALSE,
  showLegend = TRUE,
  plot = TRUE
)
```

Arguments

`results`	A list of `ClassifyResult` objects.
`performanceName`	Default: "Balanced Accuracy". The name of the performance measure to make comparisons of. This is one of the names printed in the Performance Measures field when a `ClassifyResult` object is printed, or if none are stored, the performance metric will be calculated.
`characteristicsList`	A named list of characteristics. Each element's name must be one of `"x"`, `"row"`, `"column"`, `fillColour`, or `fillLine`. The value of each element must be a characteristic name, as stored in the `"characteristic"` column of the results' characteristics table. Only `"x"` is mandatory.
`aggregate`	A character vector of the levels of `characteristicsList['x']` to aggregate to a single number by taking the mean. This is particularly meaningful when the cross-validation is leave-k-out, when k is small.

`coloursList`	A named list of plot aspects and colours for the aspects. No elements are mandatory. If specified, each list element's name must be either `"fillColours"` or `"lineColours"`. If a characteristic is associated to fill or line by `characteristicsList` but this list is empty, a palette of colours will be automaticaly chosen.
`orderingList`	An optional named list. Any of the variables specified to `characteristicsList` can be the name of an element of this list and the value of the element is the order in which the factors should be presented in, in case alphabetical sorting is undesirable.
`densityStyle`	Default: "box". Either `"violin"` for violin plot or `"box"` for box plot.
`yLimits`	The minimum and maximum value of the performance metric to plot.
`fontSizes`	A vector of length 4. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the font size of the titles of grouped plots, if any are produced. In other words, when `rowVariable` or `columnVariable` are not `NULL`.
`title`	An overall title for the plot.
`margin`	The margin to have around the plot.
`rotate90`	Logical. IF `TRUE`, the plot is horizontal.
`showLegend`	If `TRUE`, a legend is plotted next to the plot. If FALSE, it is hidden.
`plot`	Logical. IF `TRUE`, a plot is produced on the current graphics device.

Details

If there are multiple values for a performance measure in a single result object, it is plotted as a violin plot, unless `aggregate` is `TRUE`, in which case the all predictions in a single result object are considered simultaneously, so that only one performance number is calculated, and a barchart is plotted.

Value

An object of class `ggplot` and a plot on the current graphics device, if `plot` is `TRUE`.

Author(s)

Dario Strbenac

Examples

```
 predicted <- data.frame(sample = sample(LETTERS[1:10], 80, replace = TRUE),
                         permutation = rep(1:2, each = 40),
                         class = factor(rep(c("Healthy", "Cancer"), 40)))
 actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validation"),
                  value = c("Example", "t-test", "Differential Expression", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], LETTERS[10:1], list(1:100, c(1:9, 11:101)),
                          list(c(1:3), c(2, 5, 6), 1:4, 5:8),
                          list(function(oracle){}), NULL, predicted, actual)
```

```
result1 <- calcCVperformance(result1, "Macro F1")

predicted <- data.frame(sample = sample(LETTERS[1:10], 80, replace = TRUE),
                        permutation = rep(1:2, each = 40),
                        class = factor(rep(c("Healthy", "Cancer"), 40)))

result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validation"),
                 value = c("Example", "Bartlett Test", "Differential Variability", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], LETTERS[10:1], list(1:100, c(1:5, 11:105)),
                          list(c(1:3), c(4:6), c(1, 6, 7, 9), c(5:8)),
                          list(function(oracle){}), NULL, predicted, actual)
result2 <- calcCVperformance(result2, "Macro F1")

performancePlot(list(result1, result2), performanceName = "Macro F1",
                title = "Comparison")
```

`plotFeatureClasses` *Plot Density, Scatterplot, Parallel Plot or Bar Chart for Features By Class*

Description

Allows the visualisation of measurements in the data set. If `targets` is of type `Pairs`, then a parallel plot is automatically drawn. If it's a single categorical variable, then a bar chart is automatically drawn.

Usage

```
## S4 method for signature 'matrix'
plotFeatureClasses(measurements, classes, targets, ...)

## S4 method for signature 'DataFrame'
plotFeatureClasses(
  measurements,
  classes,
  targets,
  groupBy = NULL,
  groupingName = NULL,
  whichNumericFeaturePlots = c("both", "density", "stripchart"),
  measurementLimits = NULL,
  lineWidth = 1,
  dotBinWidth = 1,
  xAxisLabel = NULL,
  yAxisLabels = c("Density", "Classes"),
  showXtickLabels = TRUE,
  showYtickLabels = TRUE,
  xLabelPositions = "auto",
  yLabelPositions = "auto",
  fontSizes = c(24, 16, 12, 12, 12),
  colours = c("#3F48CC", "#880015"),
  showDatasetName = TRUE,
  plot = TRUE
)

## S4 method for signature 'MultiAssayExperiment'
plotFeatureClasses(
  measurements,
  targets,
  classesColumn,
  groupBy = NULL,
  groupingName = NULL,
  showDatasetName = TRUE,
  ...
)
```

Arguments

`measurements`	A `matrix`, `DataFrame` or a `MultiAssayExperiment` object containing the data. For a matrix, the rows are for features and the columns are for samples. A column with name `"class"` must be present in the `DataFrame` stored in the `colData` slot.
`...`	Unused variables by the three top-level methods passed to the internal method which generates the plot(s).
`classes`	Either a vector of class labels of class `factor` or if the measurements are of class `DataFrame` a character vector of length 1 containing the column name in `measurement` is also permitted. Not used if `measurements` is a `MultiAssayExperiment` object.
`targets`	If `measurements` is a `matrix` or `DataFrame`, then a vector of numeric or character indices or the feature identifiers corresponding to the feature(s) to be plotted. If `measurements` is a `MultiAssayExperiment`, then a `DataFrame` of 2 columns must be specified. The first column contains the names of the tables and the second contains the names of the variables, thus each row unambiguously specifies a variable to be plotted.
`groupBy`	If `measurements` is a `DataFrame`, then a character vector of length 1, which contains the name of a categorical feature, may be specified. If `measurements` is a `MultiAssayExperiment`, then a character vector of length 2, which contains the name of a data table as the first element and the name of a categorical feature as the second element, may be specified. Additionally, the value `"sampleInfo"` may be used to refer to the column annotation stored in the `colData` slot of the of the `MultiAssayExperiment` object. A density plot will have additional lines of different line types for each category. A strip chart plot will have a separate strip chart created for each category and the charts will be drawn in a single column on the graphics device. A parallel plot and bar chart plot will similarly be laid out.

`groupingName`	A label for the grouping variable to be used in plots.
`whichNumericFeaturePlots`	If the feature is a single feature and has numeric measurements, this option specifies which types of plot(s) to draw. The default value is `"both"`, which draws a density plot and also a stip chart below the density plot. Other options are `"density"` for drawing only a density plot and `"stripchart"` for drawing only a strip chart.
`measurementLimits`	The minimum and maximum expression values to plot. Default: `NULL`. By default, the limits are automatically computed from the data values.
`lineWidth`	Numeric value that alters the line thickness for density plots. Default: 1.
`dotBinWidth`	Numeric value that alters the diameter of dots in the strip chart. Default: 1.
`xAxisLabel`	The axis label for the plot's horizontal axis. Default: `NULL`.
`yAxisLabels`	A character vector of length 1 or 2. If the feature's measurements are numeric an `whichNumericFeaturePlots` has the value `"both"`, the first value is the y-axis label for the density plot and the second value is the y-axis label for the strip chart. Otherwise, if the feature's measurements are numeric and only one plot is drawn, then a character vector of length 1 specifies the y-axis label for that particular plot. Ignored if the feature's measurements are categorical.
`showXtickLabels`	Logical. Default: `TRUE`. If set to `FALSE`, the x-axis labels are hidden.
`showYtickLabels`	Logical. Default: `TRUE`. If set to `FALSE`, the y-axis labels are hidden.
`xLabelPositions`	Either "auto" or a vector of values. The positions of labels on the x-axis. If "auto", the placement of labels is automatically calculated.
`yLabelPositions`	Either "auto" or a vector of values. The positions of labels on the y-axis. If "auto", the placement of labels is automatically calculated.
`fontSizes`	A vector of length 5. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the size of the legends' titles. The fifth number is the font size of the legend labels.
`colours`	The colours to plot data of each class in. The length of this vector must be as long as the distinct number of classes in the data set.
`showDatasetName`	Logical. Default: `TRUE`. If `TRUE` and the data is in a `MultiAssayExperiment` object, the the name of the table in which the feature is stored in is added to the plot title.
`plot`	Logical. Default: `TRUE`. If `TRUE`, a plot is produced on the current graphics device.
`classesColumn`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the class column in the table extracted by `colData(multiAssayExperiment)` that contains the samples' outcomes to use for prediction.

Value

Plots are created on the current graphics device and a list of plot objects is invisibly returned. The classes of the plot object are determined based on the type of data plotted and the number of plots per feature generated. If the plotted variable is discrete or if the variable is numeric and one plot type was specified, the list element is an object of class `ggplot`. Otherwise, if the variable is numeric and both the density and stripchart plot types were made, the list element is an object of class `TableGrob`.

Settling `lineWidth` and `dotBinWidth` to the same value doesn't result in the density plot and the strip chart having elements of the same size. Some manual experimentation is required to get similarly sized plot elements.

Author(s)

Dario Strbenac

Examples

```
# First 25 samples and first 5 genes are mixtures of two normals. Last 25 samples are
# one normal.
genesMatrix <- sapply(1:15, function(geneColumn) c(rnorm(5, 5, 1)))
genesMatrix <- cbind(genesMatrix, sapply(1:10, function(geneColumn) c(rnorm(5, 15, 1))))
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(geneColumn) c(rnorm(5, 9, 2))))
genesMatrix <- rbind(genesMatrix, sapply(1:50, function(geneColumn) rnorm(95, 9, 3)))
genesMatrix <- t(genesMatrix)
rownames(genesMatrix) <- paste("Sample", 1:50)
colnames(genesMatrix) <- paste("Gene", 1:100)
classes <- factor(rep(c("Poor", "Good"), each = 25), levels = c("Good", "Poor"))
plotFeatureClasses(genesMatrix, classes, targets = "Gene 4",
                   xAxisLabel = bquote(log[2]*'(expression)'), dotBinWidth = 0.5)



infectionResults <- c(rep(c("No", "Yes"), c(20, 5)), rep(c("No", "Yes"), c(5, 20)))
genders <- factor(rep(c("Male", "Female"), each = 10, length.out = 50))
clinicalData <- DataFrame(Gender = genders, Sugar = runif(50, 4, 10),
                          Infection = factor(infectionResults, levels = c("No", "Yes")),
                            row.names = rownames(genesMatrix))
plotFeatureClasses(clinicalData, classes, targets = "Infection")
plotFeatureClasses(clinicalData, classes, targets = "Infection", groupBy = "Gender")

genesMatrix <- t(genesMatrix) # MultiAssayExperiment needs features in rows.
dataContainer <- MultiAssayExperiment(list(RNA = genesMatrix),
                                      colData = cbind(clinicalData, class = classes))
targetFeatures <- DataFrame(dataset = "RNA", feature = "Gene 50")
plotFeatureClasses(dataContainer, targets = targetFeatures, classesColumn = "class",
                   groupBy = c("sampleInfo", "Gender"),
                   xAxisLabel = bquote(log[2]*'(expression)'), dotBinWidth = 0.5)
```

`PredictParams` *Parameters for Classifier Prediction*

Description

Collects the function to be used for making predictions and any associated parameters.

Details

The function specified must return either a factor vector of class predictions, or a numeric vector of scores for the second class, according to the levels of the class vector of the input data set, or a data frame which has two columns named class and score.

Constructor

`PredictParams()` Creates a default PredictParams object. This assumes that the object returned by the classifier has a list element named `"class"`.

`PredictParams(predictor, characteristics = DataFrame(), intermediate = character(0), ...)` Creates a PredictParams object which stores the function which will do the class prediction, if required, and parameters that the function will use. If the training function also makes predictions, this must be set to `NULL`.

- `predictor` Either `NULL` or a function to make predictions with. If it is a function, then the first argument must accept the classifier made in the training step. The second argument must accept a `DataFrame` of new data.
- `characteristics` A `DataFrame` describing the characteristics of the predictor function used. First column must be named `"charateristic"` and second column must be named `"value"`.
- `intermediate` Character vector. Names of any variables created in prior stages in `runTest` that need to be passed to the prediction function.
- `...` Other arguments that `predictor` may use.

Summary

`predictParams` is a `PredictParams` object.

`show(predictParams)`: Prints a short summary of what `predictParams` contains.

Author(s)

Dario Strbenac

Examples

```
predictParams <- PredictParams(predictor = DLDApredictInterface)
# For prediction by trained object created by DLDA training function.
PredictParams(predictor = NULL)
# For when the training function also does prediction and directly returns the
# predictions.
```

`PredictParamsOrNULL` *Union of A PredictParams Object and NULL*

Description

Allows a slot to be either a PredictParams class object or empty. No constructor. In other words, the training function specified also makes predictions with the test set.

Author(s)

Dario Strbenac

Examples

```
  ModellingParams(trainParams = TrainParams(kNNinterface, k = 5), predictParams = NULL)
```

`previousSelection` *Automated Selection of Previously Selected Features*

Description

Uses the feature selection of the same cross-validation iteration of a previous classification for the current classification task.

Usage

```
## S4 method for signature 'matrix'
previousSelection(measurementsTrain, ...)

## S4 method for signature 'DataFrame'
previousSelection(
  measurementsTrain,
  classesTrain,
  classifyResult,
  minimumOverlapPercent = 80,
  .iteration,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment'
previousSelection(measurementsTrain, ...)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`classesTrain`	Do not specify this variable. It is ignored and only used to create consistency of formal parameters with other feature selection methods.
`classifyResult`	An existing classification result from which to take the feature selections from.
`minimumOverlapPercent`	If at least this many selected features can't be identified in the current data set, then the selection stops with an error.
`.iteration`	Do not specify this variable. It is set by `runTests` if this function is being repeatedly called by `runTests`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.

Value

A vector of feature indices, from the most promising features in the first position to the least promising feature in the last position.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:50)
  colnames(genesMatrix) <- paste("Gene", 1:100)
  classes <- factor(rep(c("Poor", "Good"), each = 25))

  CVparams <- CrossValParams(permutations = 2, folds = 2)
  result <- runTests(genesMatrix, classes, CVparams, ModellingParams())
  chosenFeatureNames(result)

  # Genes 50 to 74 have differential expression in new data set.

  newDataset <- sapply(1:100, function(sample) c(rnorm(25, 9, 0.3)))
 newDataset <- rbind(newDataset, cbind(sapply(1:49, function(sample) rnorm(25, 9, 0.3)),
                                    sapply(1:25, function(sample) rnorm(25, 14, 0.3)),
                                    sapply(1:26, function(sample) rnorm(25, 9, 0.3))))
```

```
    rownames(newDataset) <- rownames(genesMatrix)
    colnames(newDataset) <- colnames(genesMatrix)

   selPars <- SelectParams(previousSelection, intermediate = ".iteration", classifyResult = result)
    previousParams <- ModellingParams(selectParams = selPars)
    newerResult <- runTests(newDataset, classes, CVparams, previousParams)

    # However, only genes 76 to 100 are chosen, because the feature selections are
    # carried over from the first cross-validated classification.
    chosenFeatureNames(newerResult)
  #}
```

`previousTrained` *Automated Usage of Previously Created Classifiers*

Description

Uses the trained classifier of the same cross-validation iteration of a previous classification for the current classification task.

Usage

```
## S4 method for signature 'ClassifyResult'
previousTrained(classifyResult, .iteration, verbose = 3)
```

Arguments

`classifyResult`	A `ClassifyResult` object which stores the models fitted previously.
`.iteration`	Do not specify this variable. It is set by `runTests` if this function is being repeatedly called by `runTests`.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.

Value

A trained classifier from a previously completed classification task.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:50)
  colnames(genesMatrix) <- paste("Gene", 1:100)
  classes <- factor(rep(c("Poor", "Good"), each = 25))

  CVparams <- CrossValParams(permutations = 2, folds = 2)
  result <- runTests(genesMatrix, classes, CVparams, ModellingParams())
  models(result)

  # Genes 50 to 74 have differential expression in new data set.
  newDataset <- sapply(1:100, function(sample) c(rnorm(25, 9, 0.3)))
 newDataset <- rbind(newDataset, cbind(sapply(1:49, function(sample) rnorm(25, 9, 0.3)),
                                  sapply(1:25, function(sample) rnorm(25, 14, 0.3)),
                                  sapply(1:26, function(sample) rnorm(25, 9, 0.3))))

  rownames(newDataset) <- rownames(genesMatrix)
  colnames(newDataset) <- colnames(genesMatrix)

 selPars <- SelectParams(previousSelection, intermediate = ".iteration", classifyResult = result)
 trPars <- TrainParams(previousTrained, intermediate = ".iteration", classifyResult = result)
  previousParams <- ModellingParams(selectParams = selPars, trainParams = trPars)
  newerResult <- runTests(newDataset, classes, CVparams, previousParams)
  models(newerResult)
#}
```

`randomForest-class` *Trained randomForest Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

randomForestInterfaces
An Interface for randomForest Package's randomForest Function

Description

A random forest classifier builds multiple decision trees and uses the predictions of the trees to determine a single prediction for each test sample.

Usage

```
## S4 method for signature 'matrix'
randomForestTrainInterface(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
randomForestTrainInterface(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
randomForestTrainInterface(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)

## S4 method for signature 'randomForest,matrix'
randomForestPredictInterface(forest, measurementsTest, ...)

## S4 method for signature 'randomForest,DataFrame'
randomForestPredictInterface(
  forest,
  measurementsTest,
  ...,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'randomForest,MultiAssayExperiment'
randomForestPredictInterface(
  forest,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method (e.g. `verbose`) or options which are accepted by the `randomForest` or `predict.randomForest` functions.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if `measurementsTrain` is a `MultiAssayExperiment`. If a column name, that column will be removed before training.
`verbose`	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that integer variables from the sample information data table will be used.
`forest`	A trained random forest which was created by `randomForest`.
`measurementsTest`	An object of the same class as `measurementsTrain` with no samples in common with `measurementsTrain` and the same number of features as it.
`returnType`	Default: `"both"`. Either `"class"`, `"score"` or `"both"`. Sets the return value from the prediction to either a vector of class labels, score for a sample belonging to the second class, as determined by the factor levels, or both labels and scores in a `data.frame`.

Value

For `randomForestTrainInterface`, the trained random forest. For `randomForestPredictInterface`, either a factor vector of predicted classes, a matrix of scores for each class, or a table of both the class labels and class scores, depending on the setting of `returnType`.

Author(s)

Dario Strbenac

See Also

`forestFeatures` for a function to extract the features used to build the trees.

Examples

```
if(require(randomForest))
{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                   c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
  colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))
  trainingSamples <- c(1:20, 26:45)
  testingSamples <- c(21:25, 46:50)

  trained <- randomForestTrainInterface(genesMatrix[trainingSamples, ],
                                       classes[trainingSamples])
  predicted <- randomForestPredictInterface(trained, genesMatrix[testingSamples, ])
}
```

`rankingPlot` *Plot Pair-wise Overlap of Ranked Features*

Description

Pair-wise overlaps can be done for two types of analyses. Firstly, each cross-validation iteration can be considered within a single classification. This explores the feature ranking stability. Secondly, the overlap may be considered between different classification results. This approach compares the feature ranking commonality between different results. Two types of commonality are possible to analyse. One summary is the average pair-wise overlap between all possible pairs of results. The second kind of summary is the pair-wise overlap of each level of the comparison factor that is not the reference level against the reference level. The overlaps are converted to percentages and plotted as lineplots.

Usage

```
## S4 method for signature 'list'
rankingPlot(
  results,
  topRanked = seq(10, 100, 10),
  comparison = "within",
  referenceLevel = NULL,
  characteristicsList = list(),
  orderingList = list(),
 sizesList = list(lineWidth = 1, pointSize = 2, legendLinesPointsSize = 1, fonts =
    c(24, 16, 12, 12, 12, 16)),
  lineColours = NULL,
  xLabelPositions = seq(10, 100, 10),
  yMax = 100,
  title = if (comparison[1] == "within") "Feature Ranking Stability" else
    "Feature Ranking Commonality",
  yLabel = if (is.null(referenceLevel)) "Average Common Features (%)" else
    paste("Average Common Features with", referenceLevel, "(%)"),
  margin = grid::unit(c(1, 1, 1, 1), "lines"),
  showLegend = TRUE,
  plot = TRUE,
  parallelParams = bpparam()
)
```

Arguments

`results`	A list of `ClassifyResult` objects.
`topRanked`	A sequence of thresholds of number of the best features to use for overlapping.
`comparison`	Default: within. The aspect of the experimental design to compare. Can be any characteristic that all results share or special value "within" to compared between all pairwise iterations of cross-validation.
`referenceLevel`	The level of the comparison factor to use as the reference to compare each non-reference level to. If `NULL`, then each level has the average pairwise overlap calculated to all other levels.
`characteristicsList`	A named list of characteristics. The name must be one of `"lineColour"`, `"pointType"`, `"row"` or `"column"`. The value of each element must be a characteristic name, as stored in the `"characteristic"` column of the results' characteristics table.
`orderingList`	An optional named list. Any of the variables specified to `characteristicsList` can be the name of an element of this list and the value of the element is the order in which the factor should be presented in.
`sizesList`	Default: `lineWidth = 1`, `pointSize = 2`,`legendLinesPointsSize = 1`, `fonts = c(24, 16, 12, 12, 12, 16)`. A list which must contain elements named `lineWidth`, `pointSize`, `legendLinesPointsSize` and `fonts`. The first three specify the size of lines and points in the graph, as well as in the plot legend. `fonts` is a vector of length 6. The first element is the size of the title text. The second element is the size of the axes titles. The third element is the size of the axes values. The fourth element is the size of the legends' titles. The fifth element is the font size of the legend labels. The sixth element is the font size of the titles of grouped plots, if any are produced. Each list element must numeric.
`lineColours`	A vector of colours for different levels of the line colouring parameter, if one is specified by `characteristicsList[["lineColour"]]`. If none are specified but, `characteristicsList[["lineColour"]]` is, an automatically-generated palette will be used.
`xLabelPositions`	Locations where to put labels on the x-axis.
`yMax`	The maximum value of the percentage to plot.
`title`	An overall title for the plot.
`yLabel`	Label to be used for the y-axis of overlap percentages.
`margin`	The margin to have around the plot.
`showLegend`	If `TRUE`, a legend is plotted next to the plot. If FALSE, it is hidden.
`plot`	Logical. If `TRUE`, a plot is produced on the current graphics device.
`parallelParams`	An object of class `MulticoreParam` or `SnowParam`.

Details

If `comparison` is `"within"`, then the feature selection overlaps are compared within a particular analysis. The result will inform how stable the selections are between different iterations of cross-validation for a particular analysis. Otherwise, the comparison is between different cross-validation runs, and this gives an indication about how common are the features being selected by different classifications.

Calculating all pair-wise set overlaps for a large cross-validation result can be time-consuming. This stage can be done on multiple CPUs by providing the relevant options to `parallelParams`.

Value

An object of class `ggplot` and a plot on the current graphics device, if `plot` is `TRUE`.

Author(s)

Dario Strbenac

Examples

```
 predicted <- data.frame(sample = sample(10, 100, replace = TRUE),
                         permutation = rep(1:2, each = 50),
                         class = rep(c("Healthy", "Cancer"), each = 50))
 actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
 allFeatures <- sapply(1:100, function(index) paste(sample(LETTERS, 3), collapse = ''))
 rankList <- list(allFeatures[1:100], allFeatures[c(15:6, 1:5, 16:100)],
               allFeatures[c(1:9, 11, 10, 12:100)], allFeatures[c(1:50, 61:100, 60:51)])
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validation"),
                 value = c("Melanoma", "t-test", "Diagonal LDA", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], allFeatures, rankList,
                          list(rankList[[1]][1:15], rankList[[2]][1:15],
                               rankList[[3]][1:10], rankList[[4]][1:10]),
                          list(function(oracle){}), NULL,
                          predicted, actual)

 predicted[, "class"] <- sample(predicted[, "class"])
 rankList <- list(allFeatures[1:100], allFeatures[c(sample(20), 21:100)],
               allFeatures[c(1:9, 11, 10, 12:100)], allFeatures[c(1:50, 60:51, 61:100)])
result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validations"),
                 value = c("Melanoma", "t-test", "Random Forest", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], allFeatures, rankList,
                          list(rankList[[1]][1:15], rankList[[2]][1:15],
                               rankList[[3]][1:10], rankList[[4]][1:10]),
                          list(function(oracle){}), NULL,
                          predicted, actual)

rankingPlot(list(result1, result2), characteristicsList = list(pointType = "Classifier Name"))
```

ROCplot *Plot Receiver Operating Curve Graphs for Classification Results*

Description

Creates one ROC plot or multiple ROC plots for a list of ClassifyResult objects. One plot is created if the data set has two classes and multiple plots are created if the data set has three or more classes.

Usage

```
## S4 method for signature 'list'
ROCplot(
  results,
  mode = c("merge", "average"),
  interval = 95,
  comparison = "Classifier Name",
  lineColours = NULL,
  lineWidth = 1,
  fontSizes = c(24, 16, 12, 12, 12),
  labelPositions = seq(0, 1, 0.2),
  plotTitle = "ROC",
  legendTitle = NULL,
  xLabel = "False Positive Rate",
  yLabel = "True Positive Rate",
  plot = TRUE,
  showAUC = TRUE
)
```

Arguments

`results`	A list of `ClassifyResult` objects.
`mode`	Default: "merge". Whether to merge all predictions of all iterations of cross-validation into one set or keep them separate. Keeping them separate will cause separate ROC curves to be computed for each iteration and confidence intervals to be drawn with the solid line being the averaged ROC curve.
`interval`	Default: 95 (percent). The percent confidence interval to draw around the averaged ROC curve, if mode is `"each"`.
`comparison`	The aspect of the experimental design to compare. Can be any characteristic that all results share. If the data set has two classes, then the slot name with factor levels to be used for colouring the lines. Otherwise, it specifies the variable used for plot facetting.
`lineColours`	A vector of colours for different levels of the comparison parameter, or if there are three or more classes, the classes. If `NULL`, a default colour palette is automatically generated.
`lineWidth`	A single number controlling the thickness of lines drawn.

`fontSizes`	A vector of length 5. The first number is the size of the title. The second number is the size of the axes titles and AUC text, if it is not part of the legend. The third number is the size of the axes values. The fourth number is the size of the legends' titles. The fifth number is the font size of the legend labels.
`labelPositions`	Default: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0. Locations where to put labels on the x and y axes.
`plotTitle`	An overall title for the plot.
`legendTitle`	A default name is used if the value is `NULL`. Otherwise a character name can be provided.
`xLabel`	Label to be used for the x-axis of false positive rate.
`yLabel`	Label to be used for the y-axis of true positive rate.
`plot`	Logical. If `TRUE`, a plot is produced on the current graphics device.
`showAUC`	Logical. If `TRUE`, the AUC value of each result is added to its legend text.

Details

The scores stored in the results should be higher if the sample is more likely to be from the class which the score is associated with. The score for each class must be in a column which has a column name equal to the class name.

For cross-validated classification, all predictions from all iterations are considered simultaneously, to calculate one curve per classification.

Value

An object of class `ggplot` and a plot on the current graphics device, if `plot` is `TRUE`.

Author(s)

Dario Strbenac

Examples

```
predicted <- do.call(rbind, list(data.frame(data.frame(sample = LETTERS[c(1, 8, 15, 3, 11, 20, 19, 18)],
                             Healthy = c(0.89, 0.68, 0.53, 0.76, 0.13, 0.20, 0.60, 0.25),
                              Cancer = c(0.11, 0.32, 0.47, 0.24, 0.87, 0.80, 0.40, 0.75),
                                fold = 1)),
                      data.frame(sample = LETTERS[c(11, 18, 15, 4, 6, 10, 11, 12)],
                             Healthy = c(0.45, 0.56, 0.33, 0.56, 0.33, 0.20, 0.60, 0.40),
                              Cancer = c(0.55, 0.44, 0.67, 0.44, 0.67, 0.80, 0.40, 0.60),
                                fold = 2)))
actual <- factor(c(rep("Healthy", 10), rep("Cancer", 10)), levels = c("Healthy", "Cancer"))
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                        "Cross-validation"),
                    value = c("Melanoma", "t-test", "Random Forest", "2 Permutations, 2 Folds")),
                             LETTERS[1:20], LETTERS[10:1],
                         list(1:100, c(1:9, 11:101)), list(sample(10, 10), sample(10, 10)),
                             list(function(oracle){}), NULL, predicted, actual)
```

```
  predicted[c(2, 6), "Healthy"] <- c(0.40, 0.60)
  predicted[c(2, 6), "Cancer"] <- c(0.60, 0.40)
 result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                         "Cross-validation"),
                    value = c("Example", "Bartlett Test", "Differential Variability", "2 Permutations, 2 Folds")),
                           LETTERS[1:20], LETTERS[10:1], list(1:100, c(1:5, 11:105)),
                         list(sample(10, 10), sample(10, 10)), list(function(oracle){}),
                           NULL, predicted, actual)
  ROCplot(list(result1, result2), plotTitle = "Cancer ROC")
```

runTest *Perform a Single Classification*

Description

For a data set of features and samples, the classification process is run. It consists of data transformation, feature selection, classifier training and testing.

Usage

```
## S4 method for signature 'matrix'
runTest(measurementsTrain, outcomesTrain, measurementsTest, outcomesTest, ...)

## S4 method for signature 'DataFrame'
runTest(
  measurementsTrain,
  outcomesTrain,
  measurementsTest,
  outcomesTest,
  crossValParams = CrossValParams(),
  modellingParams = ModellingParams(),
  characteristics = S4Vectors::DataFrame(),
  verbose = 1,
  .iteration = NULL
)

## S4 method for signature 'MultiAssayExperiment'
runTest(
  measurementsTrain,
  measurementsTest,
  targets = names(measurements),
  outcomesColumns,
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`outcomesTrain`	Either a factor vector of classes, a `Surv` object, or a character string, or vector of such strings, containing column name(s) of column(s) containing either classes or time and event information about survival.
`measurementsTest`	Same data type as `measurementsTrain`, but only the test samples.
`outcomesTest`	Same data type as `outcomesTrain`, but only the test samples.
`crossValParams`	An object of class `CrossValParams`, specifying the kind of cross-validation to be done, if nested cross-validation is used to tune any parameters.
`modellingParams`	An object of class `ModellingParams`, specifying the class rebalancing, transformation (if any), feature selection (if any), training and prediction to be done on the data set.
`characteristics`	A `DataFrame` describing the characteristics of the classification used. First column must be named `"charateristic"` and second column must be named `"value"`. Useful for automated plot annotation by plotting functions within this package. Transformation, selection and prediction functions provided by this package will cause the characteristics to be automatically determined and this can be left blank.
`verbose`	Default: 1. A number between 0 and 3 for the amount of progress messages to give. A higher number will produce more messages as more lower-level functions print messages.
`.iteration`	Not to be set by a user. This value is used to keep track of the cross-validation iteration, if called by `runTests`.
`targets`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.
`outcomesColumns`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the column (class) or columns (survival) in the table extracted by `colData(data)` that contain(s) the samples' outcomes to use for prediction.

Details

This function only performs one classification and prediction. See `runTests` for a driver function that enables a number of different cross-validation schemes to be applied and uses this function to perform each iteration.

Value

If called directly by the user rather than being used internally by runTests, a ClassifyResult object. Otherwise a list of different aspects of the result which is passed back to runTests.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  data(asthma)
  tuneList <- list(nFeatures = seq(5, 25, 5), performanceType = "Balanced Error")
  selectParams <- SelectParams(limmaRanking, tuneParams = tuneList)
  modellingParams <- ModellingParams(selectParams = selectParams)
  trainIndices <- seq(1, nrow(measurements), 2)
  testIndices <- seq(2, nrow(measurements), 2)

  runTest(measurements[trainIndices, ], classes[trainIndices],
      measurements[testIndices, ], classes[testIndices], modellingParams = modellingParams)
#}
```

---

runTests *Reproducibly Run Various Kinds of Cross-Validation*

Description

Enables doing classification schemes such as ordinary 10-fold, 100 permutations 5-fold, and leave one out cross-validation. Processing in parallel is possible by leveraging the package BiocParallel.

Usage

```
## S4 method for signature 'matrix'
runTests(measurements, outcomes, ...)

## S4 method for signature 'DataFrame'
runTests(
  measurements,
  outcomes,
  crossValParams = CrossValParams(),
  modellingParams = ModellingParams(),
  characteristics = S4Vectors::DataFrame(),
  verbose = 1
)

## S4 method for signature 'MultiAssayExperiment'
runTests(measurements, targets = names(measurements), outcomesColumns, ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing all of the data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`outcomes`	Either a factor vector of classes, a `Surv` object, or a character string, or vector of such strings, containing column name(s) of column(s) containing either classes or time and event information about survival.
`crossValParams`	An object of class `CrossValParams`, specifying the kind of cross-validation to be done.
`modellingParams`	An object of class `ModellingParams`, specifying the class rebalancing, transformation (if any), feature selection (if any), training and prediction to be done on the data set.
`characteristics`	A `DataFrame` describing the characteristics of the classification used. First column must be named `"charateristic"` and second column must be named `"value"`. Useful for automated plot annotation by plotting functions within this package. Transformation, selection and prediction functions provided by this package will cause the characteristics to be automatically determined and this can be left blank.
`verbose`	Default: 1. A number between 0 and 3 for the amount of progress messages to give. A higher number will produce more messages as more lower-level functions print messages.
`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"clinical"` is also a valid value and specifies that the clinical data table will be used.
`outcomesColumns`	If `measurementsTrain` is a `MultiAssayExperiment`, the names of the column (class) or columns (survival) in the table extracted by `colData(data)` that contain(s)s the samples' outcomes to use for prediction.

Value

An object of class `ClassifyResult`.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  data(asthma)
```

```
    CVparams <- CrossValParams(permutations = 5)
    tuneList <- list(nFeatures = seq(5, 25, 5), performanceType = "Balanced Error")
    selectParams <- SelectParams(differentMeansRanking, tuneParams = tuneList)
    modellingParams <- ModellingParams(selectParams = selectParams)
    runTests(measurements, classes, CVparams, modellingParams,
             DataFrame(characteristic = c("Dataset Name", "Classifier Name"),
                       value = c("Asthma", "Different Means"))
             )
  #}
```

---

samplesMetricMap *Plot a Grid of Sample Error Rates or Accuracies*

Description

A grid of coloured tiles is drawn. There is one column for each sample and one row for each classification result.

Usage

```
## S4 method for signature 'list'
samplesMetricMap(
  results,
  comparison = "Classifier Name",
  metric = c("Sample Error", "Sample Accuracy"),
  featureValues = NULL,
  featureName = NULL,
 metricColours = list(c("#3F48CC", "#6F75D8", "#9FA3E5", "#CFD1F2", "#FFFFFF"),
   c("#880015", "#A53F4F", "#C37F8A", "#E1BFC4", "#FFFFFF")),
  classColours = c("#3F48CC", "#880015"),
  groupColours = c("darkgreen", "yellow2"),
  fontSizes = c(24, 16, 12, 12, 12),
  mapHeight = 4,
  title = "Error Comparison",
  showLegends = TRUE,
  xAxisLabel = "Sample Name",
  showXtickLabels = TRUE,
  yAxisLabel = "Analysis",
  showYtickLabels = TRUE,
  legendSize = grid::unit(1, "lines"),
  plot = TRUE
)

## S4 method for signature 'matrix'
samplesMetricMap(
  results,
  classes,
  metric = c("Sample Error", "Sample Accuracy"),
  featureValues = NULL,
  featureName = NULL,
  metricColours = list(c("#3F48CC", "#6F75D8", "#9FA3E5", "#CFD1F2", "#FFFFFF"),
    c("#880015", "#A53F4F", "#C37F8A", "#E1BFC4", "#FFFFFF")),
  classColours = c("#3F48CC", "#880015"),
  groupColours = c("darkgreen", "yellow2"),
  fontSizes = c(24, 16, 12, 12, 12),
  mapHeight = 4,
  title = "Error Comparison",
  showLegends = TRUE,
  xAxisLabel = "Sample Name",
  showXtickLabels = TRUE,
  yAxisLabel = "Analysis",
  showYtickLabels = TRUE,
  legendSize = grid::unit(1, "lines"),
  plot = TRUE
)
```

Arguments

`results`	A list of `ClassifyResult` objects. Could also be a matrix of pre-calculated metrics, for backwards compatibility.
`comparison`	Default: Classifier Name. The aspect of the experimental design to compare. Can be any characteristic that all results share.
`metric`	The sample-wise metric to plot.
`featureValues`	If not NULL, can be a named factor or named numeric vector specifying some variable of interest to plot underneath the class bar.
`featureName`	A label describing the information in `featureValues`. It must be specified if `featureValues` is.
`metricColours`	A vector of colours for metric levels.
`classColours`	Either a vector of colours for class levels if both classes should have same colour, or a list of length 2, with each component being a vector of the same length. The vector has the colour gradient for each class.
`groupColours`	A vector of colours for group levels. Only useful if `groups` is not NULL.
`fontSizes`	A vector of length 5. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the size of the legends' titles. The fifth number is the font size of the legend labels.
`mapHeight`	Height of the map, relative to the height of the class colour bar.
`title`	The title to place above the plot.
`showLegends`	Logical. IF FALSE, the legend is not drawn.
`xAxisLabel`	The name plotted for the x-axis. NULL suppresses label.
`showXtickLabels`	Logical. IF FALSE, the x-axis labels are hidden.

`yAxisLabel`	The name plotted for the y-axis. NULL suppresses label.
`showYtickLabels`	Logical. IF FALSE, the y-axis labels are hidden.
`legendSize`	The size of the boxes in the legends.
`plot`	Logical. IF `TRUE`, a plot is produced on the current graphics device.
`classes`	If `results` is a matrix, this is a factor vector of the same length as the number of columns that `results` has.

Details

The names of `results` determine the row names that will be in the plot. The length of `metricColours` determines how many bins the metric values will be discretised to.

Value

A plot is produced and a grob is returned that can be saved to a graphics device.

Author(s)

Dario Strbenac

Examples

```
 predicted <- data.frame(sample = LETTERS[sample(10, 100, replace = TRUE)],
                          class = rep(c("Healthy", "Cancer"), each = 50))
actual <- factor(rep(c("Healthy", "Cancer"), each = 5), levels = c("Healthy", "Cancer"))
 features <- sapply(1:100, function(index) paste(sample(LETTERS, 3), collapse = ''))
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                        "Cross-validation"),
                value = c("Example", "t-test", "Differential Expression", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], features, list(1:100), list(sample(10, 10)),
                          list(function(oracle){}), NULL, predicted, actual)
 predicted[, "class"] <- sample(predicted[, "class"])
result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                        "Cross-validation"),
                value = c("Example", "Bartlett Test", "Differential Variability", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], features, list(1:100), list(sample(10, 10)),
                          list(function(oracle){}), NULL, predicted, actual)
 result1 <- calcCVperformance(result1, "Sample Error")
 result2 <- calcCVperformance(result2, "Sample Error")
 groups <- factor(rep(c("Male", "Female"), length.out = 10))
 names(groups) <- LETTERS[1:10]
 cholesterol <- c(4.0, 5.5, 3.9, 4.9, 5.7, 7.1, 7.9, 8.0, 8.5, 7.2)
 names(cholesterol) <- LETTERS[1:10]

 wholePlot <- samplesMetricMap(list(Gene = result1, Protein = result2))
 wholePlot <- samplesMetricMap(list(Gene = result1, Protein = result2),
                               featureValues = groups, featureName = "Gender")
 wholePlot <- samplesMetricMap(list(Gene = result1, Protein = result2),
                              featureValues = cholesterol, featureName = "Cholesterol")
```

`selectionPlot`	*Plot Pair-wise Overlap, Variable Importance or Selection Size Distribution of Selected Features*

Description

Pair-wise overlaps can be done for two types of analyses. Firstly, each cross-validation iteration can be considered within a single classification. This explores the feature selection stability. Secondly, the overlap may be considered between different classification results. This approach compares the feature selection commonality between different selection methods. Two types of commonality are possible to analyse. One summary is the average pair-wise overlap between all levels of the comparison factor and the other summary is the pair-wise overlap of each level of the comparison factor that is not the reference level against the reference level. The overlaps are converted to percentages and plotted as lineplots.

Usage

```
## S4 method for signature 'list'
selectionPlot(
  results,
  comparison = "within",
  referenceLevel = NULL,
  characteristicsList = list(x = "Classifier Name"),
  coloursList = list(),
  orderingList = list(),
  binsList = list(),
  yMax = 100,
  fontSizes = c(24, 16, 12, 16),
 title = if (comparison == "within") "Feature Selection Stability" else if (comparison
    == "size") "Feature Selection Size" else if (comparison == "importance")
    "Variable Importance" else "Feature Selection Commonality",
 yLabel = if (is.null(referenceLevel) && !comparison %in% c("size", "importance"))
   "Common Features (%)" else if (comparison == "size") "Set Size" else if (comparison
    == "importance") tail(names(results[[1]]@importance), 1) else
    paste("Common Features with", referenceLevel, "(%)"),
  margin = grid::unit(c(1, 1, 1, 1), "lines"),
  rotate90 = FALSE,
  showLegend = TRUE,
  plot = TRUE,
  parallelParams = bpparam()
)
```

Arguments

`results`	A list of `ClassifyResult` objects.
`comparison`	Default: within. The aspect of the experimental design to compare. Can be any characteristic that all results share or either one of the special values `"within"`

	to compare between all pairwise iterations of cross-validation. or `"size"`, to draw a bar chart of the frequency of selected set sizes, or `"importance"` to plot the variable importance scores of selected variables. `"importance"` only usable if `doImportance` was `TRUE` during cross-validation.
`referenceLevel`	The level of the comparison factor to use as the reference to compare each non-reference level to. If `NULL`, then each level has the average pairwise overlap calculated to all other levels.
`characteristicsList`	A named list of characteristics. Each element's name must be one of `"x"`, `"row"`, `"column"`, `"fillColour"`, or `"lineColour"`. The value of each element must be a characteristic name, as stored in the `"characteristic"` column of the results' characteristics table. Only `"x"` is mandatory.
`coloursList`	A named list of plot aspects and colours for the aspects. No elements are mandatory. If specified, each list element's name must be either `"fillColours"` or `"lineColours"`. If a characteristic is associated to fill or line by `characteristicsList` but this list is empty, a palette of colours will be automaticaly chosen.
`orderingList`	An optional named list. Any of the variables specified to `characteristicsList` can be the name of an element of this list and the value of the element is the order in which the factors should be presented in, in case alphabetical sorting is undesirable.
`binsList`	Used only if `comparison` is `"size"`. A list with elements named `"setSizes"` and `"frequencies"` Both elements are mandatory. `"setSizes"` specifies the bin boundaries for bins of interest of feature selection sizes (e.g. 0, 10, 20, 30). `"frequencies"` specifies the bin boundaries for the relative frequency percentages to plot (e.g. 0, 20, 40, 60, 80, 100).
`yMax`	Used only if `comparison` is not `"size"`. The maximum value of the percentage overlap to plot.
`fontSizes`	A vector of length 4. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the font size of the titles of grouped plots, if any are produced. In other words, when `rowVariable` or `columnVariable` are not `NULL`.
`title`	An overall title for the plot. By default, specifies whether stability or commonality is shown.
`yLabel`	Label to be used for the y-axis of overlap percentages. By default, specifies whether stability or commonality is shown.
`margin`	The margin to have around the plot.
`rotate90`	Logical. If `TRUE`, the boxplot is horizontal.
`showLegend`	If `TRUE`, a legend is plotted next to the plot. If FALSE, it is hidden.
`plot`	Logical. If `TRUE`, a plot is produced on the current graphics device.
`parallelParams`	An object of class `MulticoreParam` or `SnowParam`.

Details

Additionally, a heatmap of selection size frequencies can be made by specifying size as the comparison to make.

Lastly, a plot showing the distribution of performance metric changes when features are excluded from training can be made if variable importance calculation was turned on during cross-validation.

If `comparison` is `"within"`, then the feature selection overlaps are compared within a particular analysis. The result will inform how stable the selections are between different iterations of cross-validation for a particular analysis. Otherwise, the comparison is between different cross-validation runs, and this gives an indication about how common are the features being selected by different classifications.

Calculating all pair-wise set overlaps can be time-consuming. This stage can be done on multiple CPUs by providing the relevant options to `parallelParams`. The percentage is calculated as the intersection of two sets of features divided by the union of the sets, multiplied by 100.

For the feature selection size mode, `binsList` is used to create bins which include the lowest value for the first bin, and the highest value for the last bin using `cut`.

Value

An object of class `ggplot` and a plot on the current graphics device, if `plot` is `TRUE`.

Author(s)

Dario Strbenac

Examples

```
 predicted <- data.frame(sample = sample(10, 100, replace = TRUE),
                         class = rep(c("Healthy", "Cancer"), each = 50))
 actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
 allFeatures <- sapply(1:100, function(index) paste(sample(LETTERS, 3), collapse = ''))
 rankList <- list(allFeatures[1:100], allFeatures[c(5:1, 6:100)],
              allFeatures[c(1:9, 11, 10, 12:100)], allFeatures[c(1:50, 60:51, 61:100)])
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validations"),
                 value = c("Melanoma", "t-test", "Random Forest", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], allFeatures, rankList,
                          list(rankList[[1]][1:15], rankList[[2]][1:15],
                               rankList[[3]][1:10], rankList[[4]][1:10]),
                          list(function(oracle){}), NULL,
                          predicted, actual)

 predicted[, "class"] <- sample(predicted[, "class"])
 rankList <- list(allFeatures[1:100], allFeatures[c(sample(20), 21:100)],
              allFeatures[c(1:9, 11, 10, 12:100)], allFeatures[c(1:50, 60:51, 61:100)])
result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validation"),
                 value = c("Melanoma", "t-test", "Diagonal LDA", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], allFeatures, rankList,
                          list(rankList[[1]][1:15], rankList[[2]][1:25],
                               rankList[[3]][1:10], rankList[[4]][1:10]),
                          list(function(oracle){}), NULL,
                          predicted, actual)
 cList <- list(x = "Classifier Name", fillColour = "Classifier Name")
```

```
selectionPlot(list(result1, result2), characteristicsList = cList)

cList <- list(x = "Classifier Name", fillColour = "size")
selectionPlot(list(result1, result2), comparison = "size",
              characteristicsList = cList,
              binsList = list(frequencies = seq(0, 100, 10), setSizes = seq(0, 25, 5))
              )
```

`SelectParams` *Parameters for Feature Selection*

Description

Collects and checks necessary parameters required for feature selection. Either one function is specified or a list of functions to perform ensemble feature selection. The empty constructor is provided for convenience.

Constructor

`SelectParams()` Creates a default `SelectParams` object. This uses either an ordinary t-test or ANOVA (depending on the number of classes) and tries the top 10 to top 100 features in increments of 10, and picks the number of features with the best resubstitution balanced error rate. Users should create an appropriate `SelectParams` object for the characteristics of their data.

`SelectParams(featureSelection, characteristics = DataFrame(), minPresence = 1, intermediate = charact`
`subsetToSelections = TRUE, tuneParams = list(nFeatures = seq(10, 100, 10), performanceType = "Balar`
Creates a `SelectParams` object which stores the function(s) which will do the selection and parameters that the function will use.

- `featureRanking` Either a function which will rank the features from most promising to least promising or a list of such functions. For a particular function, the first argument must be an `DataFrame` object. The function's return value must be a vector of indices.
- `characteristics` A `DataFrame` describing the characteristics of feature selection to be done. First column must be named `"charateristic"` and second column must be named `"value"`. If using wrapper functions for feature selection in this package, the feature selection name will automatically be generated and therefore it is not necessary to specify it.
- `minPresence` If a list of functions was provided, how many of those must a feature have been selected by to be used in classification. 1 is equivalent to a set union and a number the same length as `featureSelection` is equivalent to set intersection.
- `intermediate` Character vector. Names of any variables created in prior stages by `runTest` that need to be passed to a feature selection function.
- `subsetToSelections` Whether to subset the data table(s), after feature selection has been done.
- `tuneParams` A list specifying tuning parameters required during feature selection. The names of the list are the names of the parameters and the vectors are the values of the parameters to try. All possible combinations are generated. Two elements named `nFeatures` and

`performanceType` are mandatory, to define the performance metric which will be used to select features and how many top-ranked features to try.

... Other named parameters which will be used by the selection function. If `featureSelection` was a list of functions, this must be a list of lists, as long as `featureSelection`.

Summary

`selectParams` is a `SelectParams` object.

`show(SelectParams)`: Prints a short summary of what `selectParams` contains.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  SelectParams(differentMeansRanking)

  # Ensemble feature selection.
  SelectParams(list(differentMeansRanking, pairsDifferencesRanking))
#}
```

`SelectParamsOrNULL-class`

Union of A SelectParams Object and NULL

Description

Allows a slot to be either a SelectParams class object or empty. No constructor.

Author(s)

Dario Strbenac

Examples

```
ModellingParams(selectParams = NULL)
ModellingParams(selectParams = SelectParams(differentMeansRanking))
```

`StageParams-class` *StageParams Virtual Class*

Description

A class for any one of `TransformParams`, `SelectParams`, `TrainParams` or `PredictParams`. Allows a method to dispatch on any of the parameter objects specifying any stage of cross-validation.

Author(s)

Dario Strbenac

`StageParamsOrMissing-class` *Union of A StageParams Object and NULL*

Description

StageParamsOrMissing: Allows a slot to be either a class that has StageParams as its virtual parent class or empty. No constructor. StageParamsOrMissingOrNULL: Allows a slot to be either a class that has StageParams as its virtual parent class or empty or NULL. No constructor.

Author(s)

Dario Strbenac

`subtractFromLocation` *Subtract Numeric Feature Measurements from a Location*

Description

For each numeric feature, calculates the location, and subtracts all measurements from that location.

Usage

```
## S4 method for signature 'matrix,matrix'
subtractFromLocation(
  measurementsTrain,
  measurementsTest,
  location = c("mean", "median"),
  absolute = TRUE,
  verbose = 3
)
```

```
## S4 method for signature 'DataFrame,DataFrame'
subtractFromLocation(
  measurementsTrain,
  measurementsTest,
  location = c("mean", "median"),
  absolute = TRUE,
  verbose = 3
)

## S4 method for signature 'MultiAssayExperiment,MultiAssayExperiment'
subtractFromLocation(
  measurementsTrain,
  measurementsTest,
  targets = names(measurementsTrain),
  location = c("mean", "median"),
  absolute = TRUE,
  verbose = 3
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`measurementsTest`	A data set of the same type as `measurementsTrain` with no samples in common with it. The subtraction will also be performed to it.
`location`	Character. Either "mean" or "median".
`absolute`	Logical. Default: `TRUE`. If `TRUE`, then absolute values of the differences are returned. Otherwise, they are signed.
`verbose`	Default: 3. A progress message is shown if this value is 3.
`targets`	If `measurements` is a `MultiAssayExperiment`, the names of the data tables to be used. `"sampleInfo"` is also a valid value and specifies that numeric variables from the sample information data table will be used.

Details

Only the samples specified by `measurementsTrain` are used in the calculation of the location.

Value

The same class of variable as the input variable `measurements` is, with the numeric features subtracted from the calculated location.

Author(s)

Dario Strbenac

Examples

```
aMatrix <- matrix(1:100, ncol = 10)
subtractFromLocation(aMatrix[1:5,], aMatrix[6:10, ], "median")
```

`Surv-class` *Survival Data Container*

Description

Enables S4 method dispatching on it.

`svm-class` *Trained svm Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

`SVMtrainInterface` *An Interface for e1071 Package's Support Vector Machine Classifier.*

Description

`SVMtrainInterface` generates a trained SVM classifier and `SVMpredictInterface` uses it to make predictions on a test data set.

Usage

```
## S4 method for signature 'matrix'
SVMtrainInterface(measurementsTrain, classesTrain, ...)

## S4 method for signature 'DataFrame'
SVMtrainInterface(measurementsTrain, classesTrain, ..., verbose = 3)

## S4 method for signature 'MultiAssayExperiment'
SVMtrainInterface(
  measurementsTrain,
  targets = names(measurementsTrain),
  classesTrain,
  ...
)

## S4 method for signature 'svm,matrix'
SVMpredictInterface(model, measurementsTest, ...)

## S4 method for signature 'svm,DataFrame'
SVMpredictInterface(
  model,
  measurementsTest,
  returnType = c("both", "class", "score"),
  verbose = 3
)

## S4 method for signature 'svm,MultiAssayExperiment'
SVMpredictInterface(
  model,
  measurementsTest,
  targets = names(measurementsTest),
  ...
)
```

Arguments

`measurementsTrain`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method (e.g. `verbose`) or options that are used by the `svm` function.
`classesTrain`	A vector of class labels of class `factor` of the same length as the number of samples in `measurementsTrain` if it is a `matrix` or a `DataFrame` or a character vector of length 1 containing the column name in `measurementsTrain` if it is a `DataFrame` or the column name in `colData(measurementsTrain)` if

	measurementsTrain is a MultiAssayExperiment. If a column name, that column will be removed before training.
verbose	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
targets	If measurements is a MultiAssayExperiment, the names of the data tables to be used. "sampleInfo" is also a valid value and specifies that numeric variables from the sample information data table will be used.
model	A fitted model as returned by SVMtrainInterface.
measurementsTest	An object of the same class as measurementsTrain with no samples in common with measurementsTrain and the same number of features as it.
returnType	Default: "both". Either "class", "score" or "both". Sets the return value from the prediction to either a vector of class labels, score for a sample belonging to the second class, as determined by the factor levels, or both labels and scores in a data.frame.

Value

For SVMtrainInterface, a trained SVM classifier of type svm. For SVMpredictInterface, either a factor vector of predicted classes, a vector of secores for the second class, or a table of both the class labels and second class scores, depending on the setting of returnType.

Author(s)

Dario Strbenac

Examples

```
if(require(e1071))
{
  # Genes 76 to 100 have differential expression.
  genesMatrix <- sapply(1:100, function(sample) rnorm(25, 9, 0.3))
  genesMatrix <- rbind(genesMatrix, t(sapply(1:25, function(sample)
                                  c(rnorm(75, 9, 0.3), rnorm(25, 14, 0.3)))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  rownames(genesMatrix) <- paste("Sample", 1:nrow(genesMatrix))
  colnames(genesMatrix) <- paste("Gene", 1:ncol(genesMatrix))
  trainingSamples <- c(1:20, 26:45)
  testingSamples <- c(21:25, 46:50)

  classifier <- SVMtrainInterface(genesMatrix[trainingSamples, ],
                                 classes[trainingSamples], kernel = "linear")
  SVMpredictInterface(classifier, genesMatrix[testingSamples, ])
}
```

`TrainParams` *Parameters for Classifier Training*

Description

Collects and checks necessary parameters required for classifier training. The empty constructor is provided for convenience.

Constructor

`TrainParams()` Creates a default `TrainParams` object. The classifier function is `dlda` for Diagonal LDA. Users should create an appropriate `TrainParams` object for the characteristics of their data, once they are familiar with this software.

```
TrainParams(classifier, characteristics = DataFrame(),
    intermediate = character(0), getFeatures = NULL, ...)
```

Creates a `TrainParams` object which stores the function which will do the classifier building and parameters that the function will use.

`classifier` A function which will construct a classifier, and also possibly make the predictions. The first argument must be a `DataFrame` object. The second argument must be a vector of classes. If the function also makes predictions and the value of the `predictor` setting of `PredictParams` is therefore `NULL`, the third argument must be a `DataFrame` of test data. The function must also accept a parameter named `verbose`. The function's return value can be either a trained classifier if the function only does training or a vector or data frame of class predictions if it also does prediction with the test set samples.

`characteristics` A `DataFrame` describing the characteristics of the classifier used. First column must be named `"charateristic"` and second column must be named `"value"`. If using wrapper functions for classifiers in this package, a classifier name will automatically be generated and therefore it is not necessary to specify it.

`intermediate` Character vector. Names of any variables created in prior stages by `runTest` that need to be passed to `classifier`.

`getFeatures` A function may be specified that extracts the selected features from the trained model. This is relevant if using a classifier that does feature selection within training (e.g. random forest). The function must return a list of two vectors. The first vector contains the ranked features (or empty if the training algorithm doesn't produce rankings) and the second vector contains the selected features.

`...` Other named parameters which will be used by the classifier.

Summary

`trainParams` is a `TrainParams` object.

`show(trainParams)`: Prints a short summary of what `trainParams` contains.

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
  trainParams <- TrainParams(DLDAtrainInterface)
```

`TransformParams` *Parameters for Data Transformation*

Description

Collects and checks necessary parameters required for transformation within CV. The empty constructor is for when no data transformation is desired. See `subtractFromLocation` for an example of such a function.

Constructor

`TransformParams(transform, characteristics = DataFrame(), intermediate = character(0), ...)` Creates a `TransformParams` object which stores the function which will do the transformation and parameters that the function will use.

`transform` A function which will do the transformation. The first argument must be a `DataFrame` object.

`characteristics` A `DataFrame` describing the characteristics of data transformation to be done. First column must be named `"charateristic"` and second column must be named `"value"`. If using wrapper functions for data transformation in this package, the data transformation name will automatically be generated and therefore it is not necessary to specify it.

`intermediate` Character vector. Names of any variables created in prior stages by `runTest` that need to be passed to a feature selection function.

`...` Other named parameters which will be used by the transformation function.

Summary

`transformParams` is a `TransformParams` object.

`show(transformParams)`: Prints a short summary of what `transformParams` contains.

Author(s)

Dario Strbenac

Examples

```
transformParams <- TransformParams(subtractFromLocation, location = "median")
# Subtract all values from training set median, to obtain absolute deviations.
```

TransformParamsOrNULL-class

Union of A TransformParams Object and NULL

Description

Allows a slot to be either a TransformParams class object or empty. No constructor.

Author(s)

Dario Strbenac

Examples

```
ModellingParams(transformParams = NULL)
ModellingParams(transformParams = TransformParams(subtractFromLocation),
                selectParams = SelectParams(leveneRanking))
```

Index