Package 'ClassifyR'

June 19, 2018

Type Package

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

Version 2.1.3

Date 2018-06-19

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

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

VignetteBuilder knitr

biocViews Classification, Survival

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

Imports locfit, grid, utils, plyr

Suggests limma, edgeR, car, Rmixmod, ggplot2 (>= 2.2.0), gridExtra (>= 2.0.0), BiocStyle, pamr, sparsediscrim, PoiClaClu, parathyroidSE, knitr, htmltools, gtable, scales, e1071, rmarkdown, IRanges, randomForest, robustbase, mlogit, mnlogit, glmnet

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.

Collate classes.R utilities.R bartlettSelection.R calcPerformance.R classifyInterface.R DLDAinterface.R DMDselection.R distribution.R edgeRselection.R edgesToHubNetworks.R elasticNetGLMinterface.R featureSetSummary.R fisherDiscriminant.R forestFeatures.R getLocationsAndScales.R interactorDifferences.R KolmogorovSmirnovSelection.R KullbackLeiblerSelection.R leveneSelection.R likelihoodRatioSelection.R limmaSelection.R logisticRegressionInterface.R mixmodels.R naiveBayesKernel.R

networkCorrelationsSelection.R NSCselectionInterface.R
NSCtrainInterface.R NSCpredictInterface.R performancePlot.R
plotFeatureClasses.R previousSelection.R
randomForestInterface.R rankingPlot.R ROCplot.R runTest.R
runTests.R samplesMetricMap.R selectionPlot.R
subtractFromLocation.R SVMinterface.R

License GPL-3

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

git_branch master

git_last_commit d16d087

git_last_commit_date 2018-06-19

Date/Publication 2018-06-19

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.

Usage

```
data(asthma)
```

Format

`measurements` has a row for each gene and a column for each sample. `classes` is a factor vector.

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

bartlettSelection *Selection of Differential Variability with Bartlett Statistic*

Description

Ranks features by largest Bartlett statistic and chooses the features which have best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
bartlettSelection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
bartlettSelection(measurements, classes, datasetName,
                  trainParams, predictParams, resubstituteParams,
                  selectionName = "Bartlett Test", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
bartlettSelection(measurements, targets, ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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 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.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name for the data set used. Stored in the result.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

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 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

An object of class SelectResult or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

Author(s)

Dario Strbenac

Examples

```
# Samples in one class with differential variability to other class.
# First 20 genes are DV.
genesRNAmatrix <- sapply(1:25, function(sample) c(rnorm(100, 9, 1)))
moreVariable <- sapply(1:25, function(sample) rnorm(20, 9, 5))
genesRNAmatrix <- cbind(genesRNAmatrix, rbind(moreVariable,
                        sapply(1:25, function(sample) rnorm(80, 9, 1))))
colnames(genesRNAmatrix) <- paste("Sample", 1:50)
rownames(genesRNAmatrix) <- paste("Gene", 1:100)
genesSNPmatrix <- matrix(sample(c("None", "Missense"), 250, replace = TRUE),
                         ncol = 50)
colnames(genesSNPmatrix) <- paste("Sample", 1:50)
rownames(genesSNPmatrix) <- paste("Gene", 1:5)
classes <- factor(rep(c("Poor", "Good"), each = 25))
names(classes) <- paste("Sample", 1:50)
genesDataset <- MultiAssayExperiment(list(RNA = genesRNAmatrix, SNP = genesSNPmatrix),
                                     colData = DataFrame(class = classes))
# Wait for update to MultiAssayExperiment wideFormat function.
trainIDs <- paste("Sample", c(1:20, 26:45))
genesDataset <- subtractFromLocation(genesDataset, training = trainIDs,
                                     targets = "RNA") # Exclude SNP data.

resubstituteParams <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                                         performanceType = "balanced error",
                                         better = "lower")
bartlettSelection(genesDataset, datasetName = "Example", targets = "RNA",
                  trainParams = TrainParams(fisherDiscriminant),
                  predictParams = PredictParams(NULL,
                                   getClasses = function(result) result),
                  resubstituteParams = resubstituteParams)
```

`calcPerformance` *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(actualClasses, predictedClasses,
                        performanceType = c("error", "balanced error", "average accuracy",
                                            "micro precision", "micro recall",
                                            "micro F1", "macro precision",
                                            "macro recall", "macro F1"))
  ## S4 method for signature 'ClassifyResult'
calcCVperformance(result, performanceType = c("error", "balanced error", "sample error",
                                              "sample accuracy", "average accuracy",
                                                "micro precision", "micro recall",
                                                "micro F1", "macro precision",
                                                "macro recall", "macro F1"))
```

Arguments

`result` An object of class `ClassifyResult`.

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

- `"balanced error"`: balanced error rate.
- `"error"`: ordinary error rate.
- `"sample error"`: error rate for each sample in the data set.
- `"sample accuracy"`: accuracy for each sample in the data set.
- `"average accuracy"`: Sum of accuracies of each class, divided by the number of classes.
- `"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.

`actualClasses` A factor vector specifying each sample's correct class.

`predictedClasses`
A factor vector of the same length as `actualClasses` specifying each sample's predicted class.

Details

All metrics 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

An updated ClassifyResult object, with new information in the performance slot.

Author(s)

Dario Strbenac

Examples

```
predictTable <- data.frame(sample = 1:10,
                           class = factor(sample(LETTERS[1:2], 50, replace = TRUE)))
actual <- factor(sample(LETTERS[1:2], 10, replace = TRUE))
result <- ClassifyResult("Example", "Differential Expression", "A Selection",
                         paste("A", 1:10, sep = ''), paste("Gene", 1:50, sep = ''),
                         50, list(1:50, 1:50), list(1:5, 6:15),
                         list(predictTable), actual, list("leave", 2))
result <- calcCVperformance(result, "balanced error")
performance(result)
```

---

characterOrDataFrame *Union of a Character and a DataFrame*

Description

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

Author(s)

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)
```

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

Description

More details of Poisson LDA are available in the documentation of `Classify`.

Usage

```
  ## S4 method for signature 'matrix'
classifyInterface(measurements, classes, test, ...)
  ## S4 method for signature 'DataFrame'
classifyInterface(measurements, classes, test, ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
classifyInterface(measurements, test, targets = names(measurements), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples. If of type `DataFrame`, the data set is subset to only those features of type `integer`.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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.
`test`	An object of the same class as `measurements` with no samples in common with `measurements` and the same number of features as it.
`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.
`...`	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.
`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.

Details

Data tables which consist entirely of non-integer 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 result list, the same as is returned by `Classify`.

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 <- t(readCounts[['x']])
  classes <- readCounts[['y']]
  testData <- t(readCounts[['xte']])
  storage.mode(trainData) <- storage.mode(testData) <- "integer"
  classified <- classifyInterface(trainData, classes, testData)

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

`ClassifyResult` *Container for Storing Classification Results*

Description

Contains a 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 error rates. This class is not intended to be created by the user, but could be used in another package. It is created by `runTests`.

Constructor

```
ClassifyResult(datasetName, classificationName, originalNames, originalFeatures,
               rankedFeatures, chosenFeatures, predictions, actualClasses,
               validation, tune = list(NULL))
```

`datasetName` A name associated with the dataset used.

`classificationName` A name associated with the classification.

`originalNames` All sample names.

`originalFeatures` All feature names. Character vector or DataFrame with one row for each feature if the data set is a `MultiAssayExperiment`.

`rankedFeatures` All features, from most to least important. Character vector or DataFrame if data set is a `MultiAssayExperiment`.

`chosenFeatures` Features selected at each fold. Character vector or DataFrame if data set is a `MultiAssayExperiment`.

`predictions` A `list` of `data.frame` containing information about samples, their actual class and predicted class.

`actualClasses` Factor of class of each sample.

`validation` List with first element being the name of the validation scheme, and other elements providing details about the scheme.

`tune` A description of the tuning parameters, and the value chosen of each parameter.

Summary

A method which summarises the results is available. `result` is a `ClassifyResult` object.

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

`totalPredictions(ClassifyResult)` Calculates the sum of the number of predictions.

Accessors

`result` is a `ClassifyResult` object.

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

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

`predictions(result)` Returns a `list` of `data.frame`. Each data.frame contains columns `sample`, `predicted`, and `actual`. For hold-out validation, only one data.frame is returned of all of the concatenated predictions.

`actualClasses(result)` Returns a `factor` class labels, one for each sample.

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

`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.

`sampleNames(result)` Returns a `character` vector of sample names.

Author(s)

Dario Strbenac

Examples

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

  resubstituteParams <- ResubstituteParams(nFeatures = seq(5, 25, 5),
                                           performanceType = "balanced error",
                                           better = "lower")
  classified <-
  runTests(measurements, classes, "Asthma", "Different Means",
           validation = "leaveOut", leave = 1,
           params = list(SelectParams(limmaSelection, "Moderated t Statistic",
                                      resubstituteParams = resubstituteParams),
                         TrainParams(DLDAtrainInterface),
                         PredictParams(DLDApredictInterface,
                                       getClasses = function(result) result[["class"]])))
  class(classified)
}
```

distribution *Get Frequencies of Feature Selection and Sample 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.

Usage

```
  ## S4 method for signature 'ClassifyResult'
distribution(result, dataType = c("features", "samples"),
             plotType = c("density", "histogram"), summaryType = c("percentage", "count"),
             plot = TRUE, xMax = NULL, xLabel = "Percentage of Cross-validations",
              yLabel = "Density", title = "Distribution of Feature Selections",
            fontSizes = c(24, 16, 12), ...)
```

Arguments

result	An object of class ClassifyResult.
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.
xLabel	The label for the x-axis of the plot.
yLabel	The label for the y-axis of the plot.
title	An overall title for the 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.
...	Further parameters, such as colour and fill, passed to geom_histogram or stat_density, depending on the value of plotType.

Value

If type 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 type 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)
  resubstituteParams <- ResubstituteParams(nFeatures = seq(5, 25, 5),
                                           performanceType = "balanced error",
                                           better = "lower")

  result <- runTests(measurements, classes, datasetName = "Asthma",
                     classificationName = "Different Means",
                     permutations = 5,
                    params = list(SelectParams(limmaSelection, "Moderated t Statistic",
                                               resubstituteParams = resubstituteParams),
                                  TrainParams(DLDAtrainInterface),
                                  PredictParams(DLDApredictInterface,
                                                getClasses = function(result)
                                                             result[["class"]]))
                     )
  featureDistribution <- distribution(result, "features", summaryType = "count",
                                      plotType = "histogram",
                                   xLabel = "Number of Cross-validations", yLabel = "Count",
                                      binwidth = 1)
  print(head(featureDistribution))
}
```

`dlda` *Trained dlda Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

`DLDAinterface` *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(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
DLDAtrainInterface(measurements, classes, verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
DLDAtrainInterface(measurements, targets = names(measurements), ...)
  ## S4 method for signature 'dlda,matrix'
DLDApredictInterface(model, test, ...)
  ## S4 method for signature 'dlda,DataFrame'
DLDApredictInterface(model, test, verbose = 3)
  ## S4 method for signature 'dlda,MultiAssayExperiment'
DLDApredictInterface(model, test, targets = names(test), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples. If of type DataFrame, the data set is subset to only those features of type integer.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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.
model	A fitted model as returned by DLDAtrainInterface.
test	An object of the same class as measurements with no samples in common with measurements and the same number of features as it. Also, if a DataFrame, the class column must be absent.
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.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method (e.g. verbose).
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.

Details

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

For DLDAtrainInterface, a trained DLDA classifier. For DLDApredictInterface, a result of type list with elements class, scores and posterior, as created by sparsediscrim's predict method.

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
  # 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))
  rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))
  selected <- rownames(genesMatrix)[91:100]
  trainingSamples <- c(1:20, 26:45)
  testingSamples <- c(21:25, 46:50)

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

`DMDselection` *Selection of Differential Distributions with Differences in Means or Medians and a Deviation Measure*

Description

Ranks features by largest Differences in Means/Medians and Deviations and chooses the features which have best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
DMDselection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
DMDselection(measurements, classes, datasetName, differences = c("both", "location", "scale"),
             trainParams, predictParams, resubstituteParams, ...,
             selectionName = "Differences of Medians and Deviations",
             verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
DMDselection(measurements, targets = names(measurements), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`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.
`...`	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`.

datasetName	Default: "Differences of Medians and Deviations". A name for the data set used. Stored in the result.
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.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

Details

DMD is defined as $|location_1 - location_2| + |scale_1 - scale_2|$. The subscripts denote the group which the parameter is calculated for.

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

An object of class SelectResult or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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(1:25, function(sample)
                           {
                             randomMeans <- sample(c(8, 12), 20, replace = TRUE)
                             c(rnorm(20, randomMeans, 1), rnorm(80, 10, 1))
                           }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

resubstituteParams <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                                         performanceType = "balanced error",
                                         better = "lower")
DMDselection(genesMatrix, classes, datasetName = "Example",
             trainParams = TrainParams(naiveBayesKernel),
            predictParams = PredictParams(NULL, getClasses = function(result) result),
             resubstituteParams = resubstituteParams)
```

edgeRselection *Feature Selection Based on Differential Expression for Count Data*

Description

Performs a differential expression analysis between classes and chooses the features which have best resubstitution performance. The data may have overdispersion and this is modelled.

Usage

```
  ## S4 method for signature 'matrix'
edgeRselection(counts, classes, ...)
  ## S4 method for signature 'DataFrame'
edgeRselection(counts, classes, datasetName,
               normFactorsOptions = NULL, dispOptions = NULL, fitOptions = NULL,
               trainParams, predictParams, resubstituteParams,
               selectionName = "edgeR LRT", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
edgeRselection(counts, targets = NULL, ...)
```

Arguments

counts	Either a matrix or MultiAssayExperiment containing the unnormalised counts.
classes	A vector of class labels of class factor of the same length as the number of samples in measurements. Not used if measurements is a MultiAssayExperiment object.
targets	If measurements is a MultiAssayExperiment, the names of the data tables of counts to be used.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name for the data set used. Stored in the result.
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.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

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. 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

An object of class `SelectResult` or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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]]
  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.

  getClasses <- function(result) result[["ytehat"]]
  selected <- edgeRselection(expression, classes, "DPN Treatment",
                trainParams = TrainParams(classifyInterface),
                predictParams = PredictParams(NULL, getClasses = getClasses),
                resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10),
                                performanceType = "balanced error", better = "lower"))

  head(selected@rankedFeatures[[1]])
  plotFeatureClasses(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 <- sapply(1:10000, function(index)
                     paste(c(sample(LETTERS, 3), sample(1:9, 1)), collapse = ''))
otherInteractor <- sapply(1:10000, function(index)
                     paste(c(sample(LETTERS, 3), sample(1:9, 1)), collapse = ''))
edges <- data.frame(interactor, otherInteractor, stringsAsFactors = FALSE)

edgesToHubNetworks(edges, minCardinality = 4)
```

elasticNetGLMinterface

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

```
  ## S4 method for signature 'matrix'
elasticNetGLMtrainInterface(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
elasticNetGLMtrainInterface(measurements, classes, lambda = NULL,
                            ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
elasticNetGLMtrainInterface(measurements, targets = names(measurements), ...)
  ## S4 method for signature 'multnet,matrix'
elasticNetGLMpredictInterface(model, test, ...)
  ## S4 method for signature 'multnet,DataFrame'
elasticNetGLMpredictInterface(model, test, classes = NULL, lambda, ..., verbose = 3)
  ## S4 method for signature 'multnet,MultiAssayExperiment'
elasticNetGLMpredictInterface(model, test, targets = names(test), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples. If of type DataFrame, the data set is subset to only those features of type integer.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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.
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.
test	An object of the same class as measurements with no samples in common with measurements and the same number of features as it.
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.
...	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.
model	A trained elastic net GLM, as created by the glmnet function.
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.

Details

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".

The value of the "family" parameter is fixed to "multinomial" so that classification with more than 2 classes is possible. During classifier training, if more than one lambda value is considered by specifying a vector of them as input, then the chosen value is determined based on classifier resubstitution performance. Leaving the default value of NULL, however, causes the glmnet function to determine a set of values to evaluate and then a single lambda value is chosen based on the fraction of (null) deviance explained by the fitted model.

Value

For elasticNetGLMtrainInterface, an object of type glmnet. For elasticNetGLMpredictInterface, a factor vector of class predictions.

Author(s)

Dario Strbenac

Examples

```
if(require(glmnet))
{
  # 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))
  rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))

  resubstituteParams <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                                           performanceType = "balanced error",
                                           better = "lower")

  # alpha is a user-specified tuning parameter.
  # lambda is automatically tuned, based on glmnet defaults, if not user-specified.
  trainParams <- TrainParams(elasticNetGLMtrainInterface,
                             tuneParams = list(alpha = c(0, 0.5, 1)),
                             resubstituteParams = resubstituteParams)
  predictParams <- PredictParams(elasticNetGLMpredictInterface,
                                 getClasses = function(result) result)
  classified <- runTests(genesMatrix, classes, datasetName = "Example",
                         classificationName = "Differential Expression",
                         validation = "leaveOut", leave = 1,
                         params = list(trainParams, predictParams))

  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

A method which summarises the results is available. `featureSets` is a `FeatureSetCollection` object.

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

Subsetting

The FeatureSetCollection may be subsetted 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` *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 or DataFrame containing the training data. For a matrix, the rows are features, and the columns are samples.
target	If the input is a MultiAssayExperiment, this specifies which data set will be transformed. Can either be an integer or a character string specifying the name of the table. Must have length 1.
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.

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)),
                        ncol = 10, byrow = TRUE)
rownames(genesMatrix) <- paste("Gene", 1:10)
colnames(genesMatrix) <- paste("Patient", 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.

Usage

```
  ## S4 method for signature 'matrix'
fisherDiscriminant(measurements, classes, test, ...)
  ## S4 method for signature 'DataFrame'
fisherDiscriminant(measurements, classes, test, returnType = c("class", "score", "both"),
                   verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
fisherDiscriminant(measurements, test, targets = names(measurements), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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.
`test`	An object of the same class as `measurements` with no samples in common with `measurements` and the same number of features as it.
`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.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`returnType`	Default: `"class"`. 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.

Details

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. 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 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), ncol = 10)
classes <- factor(rep(c("Poor", "Good"), each = 5))

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

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

# Make first 30 genes increased in value for sixth to tenth samples.
testMatrix[1:30, 6:10] <- testMatrix[1:30, 6:10] + 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 feautres which are those that were used in at least one tree of the forest.

Usage

```
  ## S4 method for signature 'randomForest'
forestFeatures(forest)
```

Arguments

forest	A trained random forest which was created by randomForest.

Value

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

Author(s)

Dario Strbenac

Examples

```
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))
rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))
selected <- rownames(genesMatrix)[91:100]
trainingSamples <- c(1:20, 26:45)
testingSamples <- c(21:25, 46:50)

classified <- randomForestInterface(genesMatrix[, trainingSamples],
                                    classes[trainingSamples],
                                    genesMatrix[, testingSamples], ntree = 10)

forestFeatures(classified)
```

---

functionOrList *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(limmaSelection)
SelectParams(list(limmaSelection, leveneSelection), "Ensemble Selection")
```

---

functionOrNULL *Union of A Function and NULL*

Description

Allows a slot to be either a function or empty.

Author(s)

Dario Strbenac

Examples

```
PredictParams(NULL, getClasses = function(result) result)
PredictParams(predict, getClasses = function(result) result[["classes"]])
```

`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 data. For a `matrix`, the rows are features, and the columns are samples.
`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.
`...`	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.

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), ncol = 10)
distributionInfo <- getLocationsAndScales(genesMatrix, "median", "MAD")

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

`integerOrNumeric` *Union of a Integer and a Numeric*

Description

Allows the same S4 subsetting function to be specified for `object[i]` and `object[i:j]`, where `i` and `j` are integers.

Author(s)

Dario Strbenac

Examples

```
setClass("Container", representation(scores = "numeric"))
setMethod("[", c("Container", "integerOrNumeric", "missing", "ANY"),
          function(x, i, j, ..., drop = TRUE)
          {
            new("Container", scores = x@scores[i])
          })

dataset <- new("Container", scores = 1:10)
dataset[1] # 1 is numeric.
dataset[4:6] # 4:6 is a sequence of integers.
```

`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, networkSets = NULL, verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
interactorDifferences(measurements, target = NULL, ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
networkSets	A object of type FeatureSetCollection. The sets slot must contain a list of two-column matrices with each row corresponding to a binary interaction. Such sub-networks may be determined by a community detection algorithm.
target	If measurements is a MultiAssayExperiment, the name of the data table to be used.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
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.

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

```
networksList <- list(`A Hub` = matrix(c('A', 'A', 'A', 'B', 'C', 'D'), ncol = 2),
                     `G Hub` = matrix(c('G', 'G', 'G', 'H', 'I', 'J'), ncol = 2))
netSets <- FeatureSetCollection(networksList)

# Differential correlation for sub-network with hub A.
measurements <- matrix(c(5.7, 10.1, 6.9, 7.7, 8.8, 9.1, 11.2, 6.4, 7.0, 5.5,
                         5.6, 9.6, 7.0, 8.4, 10.8, 12.2, 8.1, 5.7, 5.4, 12.1,
                         4.5, 9.0, 6.9, 7.0, 7.3, 6.9, 7.8, 7.9, 5.7, 8.7,
                         8.1, 10.6, 7.4, 7.15, 10.4, 6.1, 7.3, 2.7, 11.0, 9.1,
                         round(rnorm(60, 8, 1), 1)), ncol = 10, byrow = TRUE)

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

interactorDifferences(measurements, netSets)
```

KolmogorovSmirnovSelection *Selection of Differential Distributions with Kolmogorov-Smirnov Distance*

Description

Ranks features by largest Kolmogorov-Smirnov distance and chooses the features which have best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
KolmogorovSmirnovSelection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
KolmogorovSmirnovSelection(measurements, classes, datasetName,
                           trainParams, predictParams, resubstituteParams, ...,
                           selectionName = "Kolmogorov-Smirnov Test", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
KolmogorovSmirnovSelection(measurements, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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 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.
...	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.
datasetName	A name for the data set used. Stored in the result.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

Details

Features are sorted in order of biggest distance to smallest. The top number of features is used in a classifier, to determine which number of features has the best resubstitution performance.

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

An object of class `SelectResult` or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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(1:25, function(sample)
                           {
                             randomMeans <- sample(c(8, 12), 20, replace = TRUE)
                             c(rnorm(20, randomMeans, 1), rnorm(80, 10, 1))
                           }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))
classes <- factor(rep(c("Poor", "Good"), each = 25))

resubstituteParams <- ResubstituteParams(nFeatures = seq(5, 25, 5),
                                         performanceType = "balanced error",
                                         better = "lower")
selected <- KolmogorovSmirnovSelection(genesMatrix, classes, "Example",
                           trainParams = TrainParams(naiveBayesKernel),
                           predictParams = PredictParams(NULL,
                                             getClasses = function(result) result),
                           resubstituteParams = resubstituteParams)

head(selected[["weighted=unweighted"]]@chosenFeatures)
plotFeatureClasses(genesMatrix, classes, "Gene 13", dotBinWidth = 0.25,
                   xAxisLabel = expression(log[2](expression)))
```

KullbackLeiblerSelection

Selection of Differential Distributions with Kullback-Leibler Distance

---

Description

Ranks features by largest Kullback-Leibler distance and chooses the features which have best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
KullbackLeiblerSelection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
KullbackLeiblerSelection(measurements, classes, datasetName,
                         trainParams, predictParams, resubstituteParams, ...,
                         selectionName = "Kullback-Leibler Divergence", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
KullbackLeiblerSelection(measurements, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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 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.
...	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.
datasetName	A name for the data set used. Stored in the result.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

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.

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

An object of class SelectResult or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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(1:25, function(sample)
                           {
                             randomMeans <- sample(c(8, 12), 20, replace = TRUE)
                             c(rnorm(20, randomMeans, 1), rnorm(80, 10, 1))
                           }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
classes <- factor(rep(c("Poor", "Good"), each = 25))

resubstituteParams <- ResubstituteParams(nFeatures = seq(5, 25, 5),
                                         performanceType = "balanced error",
                                         better = "lower")
KullbackLeiblerSelection(genesMatrix, classes, "Example",
                         trainParams = TrainParams(naiveBayesKernel),
                         predictParams = PredictParams(NULL,
                                         getClasses = function(result) result),
                         resubstituteParams = resubstituteParams
                        )
```

---

leveneSelection *Selection of Differential Variability with Levene Statistic*

Description

Ranks features by largest Levene statistic and chooses the features which have best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
leveneSelection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
leveneSelection(measurements, classes, datasetName,
                trainParams, predictParams, resubstituteParams,
                selectionName = "Levene Test", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
leveneSelection(measurements, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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 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.

...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name for the data set used. Stored in the result.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

Details

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

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

An object of class SelectResult or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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(1:25, function(sample)
                           {
                             randomMeans <- sample(c(8, 12), 20, replace = TRUE)
                             c(rnorm(20, randomMeans, 1), rnorm(80, 10, 1))
                           }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))
classes <- factor(rep(c("Poor", "Good"), each = 25))
genesMatrix <- subtractFromLocation(genesMatrix, 1:ncol(genesMatrix))

resubstituteParams <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                                         performanceType = "balanced error",
                                         better = "lower")
selected <- leveneSelection(genesMatrix, classes, "Example",
                            trainParams = TrainParams(fisherDiscriminant),
                            predictParams = PredictParams(NULL,
                                              getClasses = function(result) result),
                            resubstituteParams = resubstituteParams)
selected@chosenFeatures
```

`likelihoodRatioSelection` *Selection of Differential Distributions with Likelihood Ratio Statistic*

Description

Ranks features by largest ratio and chooses the features which have the best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
likelihoodRatioSelection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
likelihoodRatioSelection(measurements, classes, datasetName,
                         trainParams, predictParams, resubstituteParams,
                         alternative = c(location = "different", scale = "different"),
                         ..., selectionName = "Likelihood Ratio Test (Normal)", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
likelihoodRatioSelection(measurements, targets = names(measurements), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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 `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.
`...`	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`.
`datasetName`	A name for the data set used. Stored in the result.
`trainParams`	A container of class `TrainParams` describing the classifier to use for training.
`predictParams`	A container of class `PredictParams` describing how prediction is to be done.
`resubstituteParams`	An object of class `ResubstituteParams` describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
`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".
`selectionName`	A name to identify this selection method by. Stored in the result.
`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.

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.

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

An object of class `SelectResult` or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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(1:25, function(sample)
                           {
                             randomMeans <- sample(c(8, 12), 20, replace = TRUE)
                             c(rnorm(20, randomMeans, 1), rnorm(80, 10, 1))
                           }
                     )
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))
classes <- factor(rep(c("Poor", "Good"), each = 25))

resubstituteParams <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                                         performanceType = "balanced error",
                                         better = "lower")
selected <- likelihoodRatioSelection(genesMatrix, classes, "Example",
                                     trainParams = TrainParams(naiveBayesKernel),
                         predictParams = PredictParams(NULL,
                                         getClasses = function(result) result),
                         resubstituteParams = resubstituteParams)

classifierVariety <- "weighted=weighted,weight=crossover distance"
head(selected[[classifierVariety]]@chosenFeatures[[1]])
```

`limmaSelection` *Selection of Differentially Abundant Features*

Description

Uses a moderated t-test with empirical Bayes shrinkage to select differentially expressed features.

Usage

```
  ## S4 method for signature 'matrix'
limmaSelection(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
limmaSelection(measurements, classes, datasetName,
               trainParams, predictParams, resubstituteParams, ...,
               selectionName = "Moderated t-test", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
limmaSelection(measurements, targets = NULL, ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	A vector of class labels of class factor of the same length as the number of samples in measurements. Not used if measurements is a MultiAssayExperiment object.
targets	Names of data tables to be combined into a single table and used in the analysis.
...	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.
datasetName	A name for the data set used. Stored in the result.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

Details

This selection method looks for changes in means and uses a moderated t-test.

Value

An object of class SelectResult or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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(1:25, 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))
  rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))

  resubstituteParams <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                          performanceType = "balanced error", better = "lower")
  selected <- limmaSelection(genesMatrix, classes, "Example",
                         trainParams = TrainParams(), predictParams = PredictParams(),
                            resubstituteParams = resubstituteParams)

  selected@chosenFeatures
}
```

---

logisticRegressionInterface

An Interface for mnlogit Package's mnlogit Function

Description

`logisticRegressionTrainInterface` generates a multinomial logistic regression model trained on some training data and `logisticRegressionPredictInterface` makes class predictions for samples in the test data set.

Usage

```
  ## S4 method for signature 'matrix'
logisticRegressionTrainInterface(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
logisticRegressionTrainInterface(measurements, classes, ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
logisticRegressionTrainInterface(measurements, targets = names(measurements), ...)
  ## S4 method for signature 'mnlogit,matrix'
logisticRegressionPredictInterface(model, test, ...)
  ## S4 method for signature 'mnlogit,DataFrame'
logisticRegressionPredictInterface(model, test, classes = NULL, verbose = 3)
  ## S4 method for signature 'mnlogit,MultiAssayExperiment'
logisticRegressionPredictInterface(model, test, targets = names(test), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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.

test	An object of the same class as measurements with no samples in common with measurements and the same number of features as it. Also, if a DataFrame, the class column must be absent.
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.
model	A fitted model as returned by logisticRegressionTrainInterface.
...	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 mnlogit.
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.

Details

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".

This wrapper works with individual-specific variables. If more a complex experimental design is utilised, such as a market research data set with both individual-specific and alternative-specific variables, then this wrapper is not suitable to classify it.

Value

For logisticRegressionTrainInterface, a fitted multinomial logistic regression model. For logisticRegressionPredictInterface, a factor vector with class predictions for the samples in the test set.

Author(s)

Dario Strbenac

Examples

```
if(require(mnlogit))
{
  variables <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
  trainSamples <- c(1:45, 51:95, 101:145)
  testSamples <- c(46:50, 96:100, 146:150)

 trained <- logisticRegressionTrainInterface(DataFrame(iris[trainSamples, variables]),
                                             iris[trainSamples, "Species"])
  predicted <- logisticRegressionPredictInterface(trained,
                                      DataFrame(iris[testSamples, variables]))
}
```

`mixmodels` *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(measurements, ...)
  ## S4 method for signature 'DataFrame'
mixModelsTrain(measurements, classes, ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
mixModelsTrain(measurements, targets = names(measurements), ...)
  ## S4 method for signature 'list,matrix'
mixModelsPredict(models, test, ...)
  ## S4 method for signature 'list,DataFrame'
mixModelsPredict(models, test, weighted = c("both", "unweighted", "weighted"),
                 weight = c("both", "height difference", "crossover distance"),
              densityXvalues = 1024, minDifference = 0,
              returnType = c("class", "score", "both"), verbose = 3)
  ## S4 method for signature 'list,MultiAssayExperiment'
mixModelsPredict(models, test, targets = names(test), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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.
`test`	An object of the same class as `measurements` with no samples in common with `measurements` and the same number of features as it. Also, if a `DataFrame`, the `class` column must be absent.
`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.
`...`	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.
`models`	A list of length 3 of models generated by the training function and training class information. The first element has mixture models the same length as the number of features in the expression data for one class. The second element has the same information for the other class. The third element has the class sizes of the classes in the training data.

`weighted` Default: `"both"`. Either `"both"`, `"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.

`weight` Default: `"both"`. Either `"both"`, `"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 two mixture densities cross 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. \itemdensityXvaluesDefault: 1024. Only relevant when `weight` is `"crossover distance"`. The number of equally-spaced locations at which to calculate y values for each mixture density. \itemminDifferenceDefault: 0. The minimum difference in sums of mixture densities within each class 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. \itemreturnTypeDefault: `"class"`. 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`. \itemverboseDefault: 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.

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.

For `mixModelsTrain`, a list of trained models of class `MixmodCluster`. For `mixModelsPredict`, a vector or list of class prediction information, as long as the number of samples in the test data, or lists of such information, if both weighted and unweighted voting or a range of `minDifference` values was provided.

Dario Strbenac

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

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

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

trained <- mixModelsTrain(genesMatrix[, trainSamples], classes[trainSamples], nbCluster = 1:3) mixModelsPredict(trained, genesMatrix[, testSamples], minDifference = 0:3)

mnlogit *Trained mnlogit Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

multnet *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(measurements, classes, test, ...)
  ## S4 method for signature 'DataFrame'
naiveBayesKernel(measurements, classes, test,
                 densityFunction = density,
                 densityParameters = list(bw = "nrd0", n = 1024,
                                          from = expression(min(featureValues)),
                                           to = expression(max(featureValues))),
                   weighted = c("both", "unweighted", "weighted"),
                  weight = c("both", "height difference", "crossover distance"),
                minDifference = 0, returnType = c("class", "score", "both"), verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
naiveBayesKernel(measurements, test, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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.
test	An object of the same class as measurements with no samples in common with measurements and the same number of features as it.
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.
...	Unused variables by the three top-level methods passed to the internal method which does the classification.
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,
weighted	Default: "both". Either "both", "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.
weight	Default: "both". Either "both", "height difference", or "crossover distance". The type of weight to calculate. For "height difference", the weight of each prediction is equal to the vertical distance between two densities, 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. \itemminDifferenceDefault: 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. \itemreturnTypeDefault: "class". 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. \itemverboseDefault: 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. 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. 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 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.

A vector or list of class prediction information, as long as the number of samples in the test data, or lists of such information, if a variety of predictions is generated.

Dario Strbenac, John Ormerod

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

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

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

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

naiveBayesKernel(trainMatrix, classes, testMatrix)

`networkCorrelationsSelection` *Selection of Differentially Correlated Hub Sub-networks*

Description

Ranks sub-networks by largest within-class to between-class correlation variability and chooses the sub-networks which have the best resubstitution performance.

Usage

```
  ## S4 method for signature 'matrix'
networkCorrelationsSelection(measurements, classes, metaFeatures = NULL, ...)
  ## S4 method for signature 'DataFrame'
networkCorrelationsSelection(measurements, classes, metaFeatures = NULL,
              networkSets, datasetName, trainParams, predictParams, resubstituteParams,
              selectionName = "Differential Correlation of Sub-networks", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
networkCorrelationsSelection(measurements, target = NULL, ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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.
`metaFeatures`	A `DataFrame` with the same number of samples as the numeric table of interest. The number of derived features in this table will be different to the original input data table. The command `mcols(metaFeatureMeasurements)` must return a `DataFrame` which has an "original" column with as many rows as there are meta-features and specifies the feature which the meta-feature is originally derived from (e.g. network name).

networkSets	A object of type FeatureSetCollection. The sets slot must contain a list of two-column matrices with each row corresponding to a binary interaction. Such sub-networks may be determined by a community detection algorithm. This will be used to determine which features belong to which sub-networks before calculating a statistic for each sub-network.
target	If measurements is a MultiAssayExperiment, the name of the data table to be used.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name for the data set used. Stored in the result.
trainParams	A container of class TrainParams describing the classifier to use for training.
predictParams	A container of class PredictParams describing how prediction is to be done.
resubstituteParams	An object of class ResubstituteParams describing the performance measure to consider and the numbers of top features to try for resubstitution classification.
selectionName	A name to identify this selection method by. Stored in the result.
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.

Details

The selection of sub-networks is based on the average difference in correlations between each pair of interactors, considering the samples within each class separately. Such differences of correlations within each of the two classes are scaled by the average difference of correlations within each class.

More formally, let $C_{i,j}$ be the correlation of the j^{th} edge using all samples belonging to to class i. Then, let $\overline{C}_{i\bullet}$ be defined as $\frac{C_{i,1}+C_{i,2}+\ldots+C_{i,e}}{n}$ where e is the number of edges in the sub-network being considered. Also, let $\overline{C}_{\bullet\bullet}$, the average overall correlation, be $\frac{C_{1\bullet}+C_{2\bullet}}{2}$. Then, the between-class sum-of-squares (BSS) is $\sum_{i=1}^{2} e(\overline{C}_{i\bullet} - \overline{C}_{\bullet\bullet})^2$. Also the within-class sum-of-squares (WSS) is $\sum_{i=1}^{2}\sum_{j=1}^{e}(C_{i,j} - \overline{C}_{i\bullet})^2$. The sub-networks are ranked in decreasing order of $\frac{BSS}{WSS}$.

The classifier specified by trainParams and predictParams is used to calculate resubtitution error rates using the transformation of the data set provided by metaFeatures. The set of top-ranked sub-networks which give the lowest resubstitution error rate are finally selected.

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

An object of class SelectResult or a list of such objects, if the classifier which was used for determining the specified performance metric made a number of prediction varieties.

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.

See Also

interactorDifferences for an example of a function which can turn the measurements into meta-features for classification.

Examples

```
networksList <- list(`A Hub` = matrix(c('A', 'A', 'A', 'B', 'C', 'D'), ncol = 2),
                     `G Hub` = matrix(c('G', 'G', 'G', 'H', 'I', 'J'), ncol = 2))
netSets <- FeatureSetCollection(networksList)

# Differential correlation for sub-network with hub A.
measurements <- matrix(c(5.7, 10.1, 6.9, 7.7, 8.8, 9.1, 11.2, 6.4, 7.0, 5.5,
                         5.6, 9.6, 7.0, 8.4, 10.8, 12.2, 8.1, 5.7, 5.4, 12.1,
                         4.5, 9.0, 6.9, 7.0, 7.3, 6.9, 7.8, 7.9, 5.7, 8.7,
                         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, byrow = TRUE)
classes <- factor(rep(c("Good", "Poor"), each = 5))

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

IHdifferences <- interactorDifferences(measurements, netSets)

# The features are sub-networks and there are only two in this example.
resubstituteParams <- ResubstituteParams(nFeatures = 1:2,
                                 performanceType = "balanced error", better = "lower")

predictParams <- PredictParams(NULL, weighted = "unweighted",
                                  weight = "height difference",
                                  getClasses = function(result) result)
networkCorrelationsSelection(measurements, classes, metaFeatures = IHdifferences,
                             networkSets = netSets, datasetName = "Example",
                             trainParams = TrainParams(naiveBayesKernel),
                             predictParams = predictParams,
                             resubstituteParams = resubstituteParams)
```

NSCpredictInterface *Interface for* pamr.predict *Function from* pamr *CRAN Package*

Description

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

Usage

```
  ## S4 method for signature 'pamrtrained,matrix'
NSCpredictInterface(trained, test, ...)
  ## S4 method for signature 'pamrtrained,DataFrame'
NSCpredictInterface(trained, test, classes = NULL, ..., verbose = 3)
  ## S4 method for signature 'pamrtrained,MultiAssayExperiment'
NSCpredictInterface(trained, test, targets = names(test), ...)
```

Arguments

trained	An object of class pamrtrained.
test	An object of the same class as measurements with no samples in common with measurements used in the training stage and the same number of features as it. Also, if a DataFrame, the class column must be absent.
classes	Either NULL or a character vector of length 1, specifying the column name to remove.
targets	If test 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.
...	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.
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.

Details

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.

Value

A factor of predicted classes for the test data.

Author(s)

Dario Strbenac

See Also

pamr.predict for the function that was interfaced to.

Examples

```
if(require(pamr))
{
  # Samples in one class with differential expression to other class.
  genesMatrix <- sapply(1:25, function(geneColumn) c(rnorm(100, 9, 1)))
  genesMatrix <- cbind(genesMatrix, sapply(1:25, function(geneColumn)
                                c(rnorm(75, 9, 1), 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)])
}
```

NSCselectionInterface *Interface for* pamr.listgenes *Function from* pamr *CRAN Package*

Description

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

Usage

```
  ## S4 method for signature 'matrix'
NSCselectionInterface(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
NSCselectionInterface(measurements, classes, datasetName,
                trained, ..., selectionName = "Shrunken Centroids", verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
NSCselectionInterface(measurements, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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 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.
datasetName	A name for the data set used. Stored in the result.
trained	The output of NSCtrainInterface, which is identical to the output of pamr.listgenes.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method or extra arguments passed to pamr.listgenes.
selectionName	A name to identify this selection method by. Stored in the result.
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.

Details

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

The set of features chosen is the obtained by considering the range of thresholds provided to NSCtrainInterface and using the threshold that obtains the lowest cross-validation error rate on the training set.

Value

An object of class SelectResult. The rankedFeatures slot will be empty.

Author(s)

Dario Strbenac

See Also

pamr.listgenes for the function that was interfaced to.

Examples

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

  trained <- NSCtrainInterface(genesMatrix, classes)
  selected <- NSCselectionInterface(genesMatrix, classes, "Example", trained)

  selected@chosenFeatures
}
```

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

Description

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

Usage

```
  ## S4 method for signature 'matrix'
NSCtrainInterface(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
NSCtrainInterface(measurements, classes, ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
NSCtrainInterface(measurements, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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 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.

...	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method or extra arguments passed to `pamr.train`.
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.

Details

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

Value

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

Author(s)

Dario Strbenac

See Also

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

Examples

```
if(require(pamr))
{
  # Samples in one class with differential expression to other class.
  genesMatrix <- sapply(1:25, function(geneColumn) c(rnorm(100, 9, 1)))
  genesMatrix <- cbind(genesMatrix, sapply(1:25, function(geneColumn)
                                c(rnorm(75, 9, 1), rnorm(25, 14, 1))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))

  NSCtrainInterface(genesMatrix, classes)
}
```

---

pamrtrained *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, aggregate = character(),
                xVariable = c("classificationName", "datasetName", "selectionName",
                              "validation"),
                performanceName = NULL,
                boxFillColouring = c("classificationName", "datasetName", "selectionName",
                                     "validation", "None"),
                boxFillColours = NULL,
                boxLineColouring = c("classificationName", "datasetName", "selectionName",
                                     "validation", "None"),
                boxLineColours = NULL,
                rowVariable = c("None", "validation", "datasetName", "classificationName",
                                "selectionName"),
                columnVariable = c("datasetName", "classificationName", "validation",
                                   "selectionName", "None"),
                yLimits = c(0, 1), fontSizes = c(24, 16, 12, 12), title = NULL,
                xLabel = "Analysis", yLabel = performanceName,
                margin = grid::unit(c(0, 0, 0, 0), "lines"), rotate90 = FALSE,
                showLegend = TRUE, plot = TRUE)
```

Arguments

results	A list of ClassifyResult objects.
aggregate	A character vector of the levels of xVariable 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.
xVariable	The factor to make separate boxes for.
performanceName	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.
boxFillColouring	A factor to colour the boxes by.
boxFillColours	A vector of colours, one for each level of boxFillColouring.
boxLineColouring	A factor to colour the box lines by.
boxLineColours	A vector of colours, one for each level of boxLineColouring.
rowVariable	The slot name that different levels of are plotted as separate rows of boxplots.
columnVariable	The slot name that different levels of are plotted as separate columns of boxplots.

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.
xLabel	Label to be used for the x-axis.
yLabel	Label to be used for the y-axis of overlap percentages.
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

Possible values for slot names are "datasetName", "classificationName", and "validation". If "None", then that graphic element is not used.

If there are multiple values for a performance measure in a single result object, it is plotted as a boxplot, 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 <- list(data.frame(sample = sample(10, 20, replace = TRUE),
                             class = rep(c("Healthy", "Cancer"), each = 10)),
                  data.frame(sample = sample(10, 20, replace = TRUE),
                             class = rep(c("Healthy", "Cancer"), each = 10)),
                  data.frame(sample = sample(10, 20, replace = TRUE),
                             class = rep(c("Healthy", "Cancer"), each = 10)),
                  data.frame(sample = sample(10, 20, replace = TRUE),
                             class = rep(c("Healthy", "Cancer"), each = 10)))
actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
result1 <- ClassifyResult("Example", "Differential Expression", "t-test",
                      LETTERS[1:10], LETTERS[10:1], 100, list(1:100, c(1:9, 11:101)),
                         list(c(1:3), c(2, 5, 6), c(1:4), c(5:8), 1:5),
                         predicted, actual, validation = list("resampleFold", 2, 2))
result1 <- calcCVperformance(result1, "macro F1")

predicted <- list(data.frame(sample = sample(10, 20, replace = TRUE),
                             class = rep(c("Healthy", "Cancer"), each = 10)),
                  data.frame(sample = sample(10, 20, replace = TRUE),
                             class = rep(c("Healthy", "Cancer"), each = 10)),
                  data.frame(sample = sample(10, 20, replace = TRUE),
```

```
                                class = rep(c("Healthy", "Cancer"), each = 10)),
                    data.frame(sample = sample(10, 20, replace = TRUE),
                                class = rep(c("Healthy", "Cancer"), each = 10)))
result2 <- ClassifyResult("Example", "Differential Variability", "Bartlett Test",
                        LETTERS[1:10], LETTERS[10:1], 100, list(1:100, c(1:5, 11:105)),
                           list(c(1:3), c(4:6), c(1, 6, 7, 9), c(5:8), c(1, 5, 10)),
                           predicted, actual, validation = list("resampleFold", 2, 2))
result2 <- calcCVperformance(result2, "macro F1")

performancePlot(list(result1, result2), performanceName = "Macro F1 Score",
              title = "Comparison", boxLineColouring = "None", columnVariable = "None")
```

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

Description

Allows the visualisation of measurements in the data set.

Usage

```
  ## S4 method for signature 'matrix'
plotFeatureClasses(measurements, classes, targets, ...)
  ## S4 method for signature 'DataFrame'
plotFeatureClasses(measurements, classes, targets, groupBy = NULL,
                   groupingName = NULL, whichNumericPlots = 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, 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.
`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 indicies 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 "clinical" 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 bar chart plot will similarly be laid out.

groupingName — A label for the grouping variable to be used in plots.

... — Unused variables by the three top-level methods passed to the internal method which generates the plot(s).

whichNumericPlots — If the feature 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 whichNumericPlots 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.

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)))
rownames(genesMatrix) <- paste("Gene", 1:100)
colnames(genesMatrix) <- paste("Sample", 1:50)
classes <- factor(rep(c("Poor", "Good"), each = 25), levels = c("Good", "Poor"))
plotFeatureClasses(genesMatrix, classes, targets = "Gene 4",
                   xAxisLabel = expression(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 = colnames(genesMatrix))
plotFeatureClasses(clinicalData, classes, targets = "Infection")
plotFeatureClasses(clinicalData, classes, targets = "Infection", groupBy = "Gender")

dataContainer <- MultiAssayExperiment(list(RNA = genesMatrix),
                                      colData = cbind(clinicalData, class = classes))
targetFeatures <- DataFrame(table = "RNA", feature = "Gene 50")
plotFeatureClasses(dataContainer, targets = targetFeatures,
                   groupBy = c("clinical", "Gender"),
                   xAxisLabel = expression(log[2](expression)))
```

PredictParams *Parameters for Classifier Prediction*

Description

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

Constructor

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

`PredictParams(predictor, intermediate = character(0), transform = NULL, getClasses, ...)` 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.

`intermediate` Character vector. Names of any variables created in prior stages in `runTest` that need to be passed to the prediction function.

`transform` Either NULL or a function. The function transforms the test data into a different format before predictions are made with it. For example, a set of features may be replaced by their median value. The set of features might be part of a network. If `predictor` is `NULL`, then it should also be `NULL`.

`getClasses` A `function` to extract the vector of class predictions from the result object created by `predictor`.

`...` Other arguments that `predictor` may use.

Author(s)

Dario Strbenac

Examples

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

---

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(measurements, ...)
  ## S4 method for signature 'DataFrame'
previousSelection(measurements, classes, datasetName,
                  classifyResult, minimumOverlapPercent = 80,
                   selectionName = "Previous Selection", .iteration, verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
previousSelection(measurements, ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Do not specify this variable. It is ignored and only used to create consistency of formal parameters with other feature selection methods.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name for the data set used. Stored in the result.
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.
selectionName	A name to identify this selection method by. Stored in the result.
.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

An object of class SelectResult.

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
  # 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))))
  colnames(genesMatrix) <- paste("Sample", 1:50)
  rownames(genesMatrix) <- paste("Gene", 1:100)
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  resubstitute <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                      performanceType = "error", better = "lower")
  result <- runTests(genesMatrix, classes, datasetName = "Example",
                     classificationName = "Differential Expression",
                     permutations = 2, fold = 2,
                     params = list(SelectParams(), TrainParams(),
                     PredictParams(predict,
                     getClasses = function(result) result[["class"]])))

  # Genes 50 to 74 have differential expression in new data set.
  newDataset <- sapply(1:25, function(sample) c(rnorm(100, 9, 2)))
 newDataset <- cbind(newDataset, rbind(sapply(1:25, function(sample) rnorm(49, 9, 2)),
                                       sapply(1:25, function(sample) rnorm(25, 14, 2)),
                                       sapply(1:25, function(sample) rnorm(26, 9, 2))))
```

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

  newerResult <- runTests(newDataset, classes, datasetName = "Latest Data Set",
                          classificationName = "Differential Expression",
                          permutations = 2, fold = 2,
                      params = list(SelectParams(previousSelection,
                                  intermediate = ".iteration",
                                  classifyResult = result),
                      TrainParams(DLDAtrainInterface),
                      PredictParams(DLDApredictInterface,
                                  getClasses = function(result) result[["class"]])))

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

`randomForest` *Trained randomForest Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

`randomForestInterface` *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'
randomForestInterface(measurements, classes, test, ...)
  ## S4 method for signature 'DataFrame'
randomForestInterface(measurements, classes, test, ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
randomForestInterface(measurements, targets = names(measurements), test, ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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.
test	An object of the same class as measurements with no samples in common with measurements and the same number of features as it.
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.
...	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 function.
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.

Details

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

An object of type randomForest. The predictions of the test set samples are stored in the list element named "predicted" of the "test" element.

Author(s)

Dario Strbenac

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))
  rownames(genesMatrix) <- paste("Gene", 1:nrow(genesMatrix))
  selected <- rownames(genesMatrix)[91:100]
  trainingSamples <- c(1:20, 26:45)
  testingSamples <- c(21:25, 46:50)

  classified <- randomForestInterface(genesMatrix[, trainingSamples],
                                      classes[trainingSamples],
                                      genesMatrix[, testingSamples], ntree = 10)
  classified[["test"]][["predicted"]]
}
```

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 methods. Two types of commonality are possible to analyse. One summary is the average pair-wise overlap between a level 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'
rankingPlot(results, topRanked = seq(10, 100, 10),
            comparison = c("within", "classificationName", "validation",
                           "datasetName", "selectionName"),
            referenceLevel = NULL,
           lineColourVariable = c("validation", "datasetName", "classificationName",
                                  "selectionName", "None"),
            lineColours = NULL, lineWidth = 1,
           pointTypeVariable = c("datasetName", "classificationName", "validation",
                                 "selectionName", "None"),
            pointSize = 2, legendLinesPointsSize = 1,
           rowVariable = c("None", "datasetName", "classificationName", "validation",
                           "selectionName"),
            columnVariable = c("classificationName", "datasetName", "validation",
                               "selectionName", "None"),
            yMax = 100, fontSizes = c(24, 16, 12, 12, 12, 16),
           title = if(comparison[1] == "within") "Feature Ranking Stability" else
                    "Feature Ranking Commonality",
            xLabelPositions = seq(10, 100, 10),
           yLabel = if(is.null(referenceLevel)) "Average Common Features (%)" else
            paste("Average Common Features with", referenceLevel, "(%)"),
            margin = grid::unit(c(0, 0, 0, 0), "lines"),
            showLegend = TRUE, plot = TRUE, parallelParams = bpparam())
```

Arguments

results	A list of ClassifyResult or SelectResult objects.
topRanked	A sequence of thresholds of number of the best features to use for overlapping.
comparison	The aspect of the experimental design to compare. See Details section for a detailed description.
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.

lineColourVariable
: The slot name that different levels of are plotted as different line colours.

lineColours
: A vector of colours for different levels of the line colouring parameter. If NULL, a default palette is used.

lineWidth
: A single number controlling the thickness of lines drawn.

pointTypeVariable
: The slot name that different levels of are plotted as different point shapes on the lines.

pointSize
: A single number specifying the diameter of points drawn.

legendLinesPointsSize
: A single number specifying the size of the lines and points in the legend, if a legend is drawn.

rowVariable
: The slot name that different levels of are plotted as separate rows of lineplots.

columnVariable
: The slot name that different levels of are plotted as separate columns of lineplots.

yMax
: The maximum value of the percentage to plot.

fontSizes
: A vector of length 6. 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. The sixth 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.

xLabelPositions
: Locations where to put labels on the x-axis.

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

Possible values for characteristics are "datasetName", "classificationName", "selectionName", and "validation". If "None", then that graphical element is not used.

If comparison is "within", then the feature rankings are compared within a particular analysis. The result will inform how stable the feature rankings are between different iterations of cross-validation for a particular analysis. If comparison is "classificationName", then the feature rankings are compared across different classification algorithm types, for each level of "datasetName", "selectionName" and "validation". The result will inform how stable the feature rankings are between different classification algorithms, for every cross-validation scheme, selection algorithm and data set. If comparison is "selectionName", then the feature rankings are compared across different feature selection algorithms, for each level of "datasetName", "classificationName" and "validation". The result will inform how stable the feature rankings are between feature selection classification algorithms, for every data set, classification algorithm, and cross-validation scheme. If comparison is "validation", then the feature rankings are compared across different cross-validation schemes, for each level of "classificationName", "selectionName" and "datasetName". The result will inform how stable the feature rankings

are between different cross-validation schemes, for every selection algorithm, classification algorithm and every data set. If comparison is "datasetName", then the feature rankings are compared across different data sets, for each level of "classificationName", "selectionName" and "validation". The result will inform how stable the feature rankings are between different data sets, for every classification algorithm and every data set. This could be used to consider if different experimental studies have a highly overlapping feature ranking pattern.

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),
                        class = rep(c("Healthy", "Cancer"), each = 50))
actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
features <- sapply(1:100, function(index) paste(sample(LETTERS, 3), collapse = ''))
rankList <- list(list(1:100, c(5:1, 6:100)), list(c(1:9, 11:101), c(1:50, 60:51, 61:100)))
result1 <- ClassifyResult("Example", "Differential Expression", "Example Selection",
                          LETTERS[1:10], features, 100, rankList,
                          list(list(rankList[[1]][[1]][1:15], rankList[[1]][[2]][1:15]),
                              list(rankList[[2]][[1]][1:10], rankList[[2]][[2]][1:10])),
                          list(predicted), actual, list("resampleFold", 2, 2))

predicted[, "class"] <- sample(predicted[, "class"])
rankList <- list(list(1:100, c(sample(20), 21:100)), list(c(1:9, 11:101), c(1:50, 60:51, 61:100)))
result2 <- ClassifyResult("Example", "Differential Variability", "Example Selection",
                          LETTERS[1:10], features, 100, rankList,
                          list(list(rankList[[1]][[1]][1:15], rankList[[1]][[2]][1:15]),
                              list(rankList[[2]][[1]][1:10], rankList[[2]][[2]][1:10])),
                        list(predicted), actual, validation = list("resampleFold", 2, 2))

rankingPlot(list(result1, result2), pointTypeVariable = "classificationName")

oneRanking <- c(10, 8, 1, 2, 3, 4, 7, 9, 5, 6)
otherRanking <- c(8, 2, 3, 4, 1, 10, 6, 9, 7, 5)
oneResult <- SelectResult("Example", "One Method", 10, list(oneRanking), list(oneRanking[1:5]))
otherResult <- SelectResult("Example", "Another Method", 10, list(otherRanking), list(otherRanking[1:2]))

rankingPlot(list(oneResult, otherResult), comparison = "selectionName",
          referenceLevel = "One Method", topRanked = seq(2, 8, 2),
          lineColourVariable = "selectionName", columnVariable = "None",
          pointTypeVariable = "None", xLabelPositions = 1:10)
```

ResubstituteParams *Parameters for Resubstitution Error Calculation*

Description

Some feature selection functions provided in the framework use resubstitution error rate to choose the best number of features for classification. This class stores parameters related to that process.

Constructor

`ResubstituteParams()` Creates a default ResubstituteParams object. The number of features tried is 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. The performance measure used is the balanced error rate.

`ResubstituteParams(nFeatures, performanceType, better = c("lower", "higher"))` Creates a ResubstituteParams object, storing information about the number of top features to calculate the performance measure for, the performance measure to use, and if higher or lower values of the measure are better.

`nFeatures` A vector for the top number of features to test the resubstitution error for.

`performanceType` One of the eleven types of performance metrics which can be calculated by `calcCVperformance`.

`better` Either `"lower"` or `"higher"`. Determines whether higher or lower values of the performance measure are desirable.

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

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

Author(s)

Dario Strbenac

Examples

```
ResubstituteParams(nFeatures = seq(25, 1000, 25), performanceType = "error", better = "lower")
```

ROCplot *Plot Receiver Operating Curve Graphs for Classification Results*

Description

The average pair-wise overlap is computed for every pair of cross-validations. The overlap is converted to a percentage and plotted as lineplots.

Usage

```
  ## S4 method for signature 'list'
ROCplot(results, nBins = sapply(results, totalPredictions),
        lineColourVariable = c("classificationName", "datasetName", "selectionName",
        "validation", "None"), lineColours = NULL,
        lineWidth = 1, fontSizes = c(24, 16, 12, 12, 12), labelPositions = seq(0.0, 1.0, 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.
nBins	The number of intervals to group the samples' scores into. By default, there are as many bins as there were predictions made, for each result object.
lineColourVariable	The slot name that different levels of are plotted as different line colours.
lineColours	A vector of colours for different levels of the line colouring parameter. If NULL, a default palette is used.
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	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

Possible values for slot names are "datasetName", "classificationName", and "validation". If "None", then any lines drawn will be black.

The scores stored in the results should be higher if the sample is more likely to be from the second class, based on the levels of the actual classes. The scores must be in a column named "score".

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

The number of bins determines how many pairs of TPR and FPR points will be used to draw the plot. A higher number will result in a smoother ROC curve.

The AUC is calculated using the trapezoidal rule.

Value

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

Author(s)

Dario Strbenac

Examples

```
predicted <- list(data.frame(sample = c(1, 8, 15, 3, 11, 20, 19, 18),
                             score = c(0.11, 0.32, 0.47, 0.24, 0.87, 0.80, 0.40, 0.75)),
                  data.frame(sample = c(11, 18, 15, 4, 6, 10, 11, 12),
                             score = c(0.55, 0.44, 0.67, 0.44, 0.67, 0.80, 0.40, 0.60)))
actual <- factor(c(rep("Healthy", 10), rep("Cancer", 10)), levels = c("Healthy", "Cancer"))
result1 <- ClassifyResult("Example", "Differential Expression", "t-test",
                          LETTERS[1:10], LETTERS[10:1], 100,
                      list(1:100, c(1:9, 11:101)), list(sample(10, 10), sample(10, 10)),
                          predicted, actual, list("resampleFold", 2, 1))
predicted[[1]][, "score"][c(2, 6)] <- c(0.60, 0.40)

result2 <- ClassifyResult("Example", "Differential Variability", "Bartlett Test",
                        LETTERS[1:10], LETTERS[10:1], 100, list(1:100, c(1:5, 11:105)),
                          list(sample(10, 10), sample(10, 10)),
                          predicted, actual, validation = list("resampleFold", 2, 1))
ROCplot(list(result1, result2), lineColourVariable = "classificationName",
        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 (prediction of samples not used in training).

Usage

```
  ## S4 method for signature 'matrix'
runTest(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
runTest(measurements, classes, metaFeatures = NULL, datasetName, classificationName,
         training, testing, params = list(SelectParams(), TrainParams(), PredictParams()),
        verbose = 1, .iteration = NULL)
  ## S4 method for signature 'MultiAssayExperiment'
runTest(measurements, targets = names(measurements), ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are features, and the columns are samples. The sample identifiers must be present as column names of the `matrix` or the row names of the `DataFrame`.
`classes`	Either a vector of class labels of class `factor` of the same length as the number of samples in `measurements` 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.
`metaFeatures`	Either `NULL` or a `DataFrame` which has meta-features of the numeric data of interest.

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.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name associated with the data set used.
classificationName	A name associated with the classification.
training	A vector which specifies the training samples.
testing	A vector which specifies the test samples.
params	A list of objects of class of TransformParams, SelectParams, TrainParams, or PredictParams. The order they are in the list determines the order in which the stages of classification are done in.
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.

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. datasetName and classificationName need to be provided.

Value

If called directly by the user rather than being used internally by runTests, a SelectResult object.

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
  data(asthma)
  resubstituteParams <- ResubstituteParams(nFeatures = seq(5, 25, 5),
                                           performanceType = "balanced error",
                                           better = "lower")
  runTest(measurements, classes, "Asthma", "Different Means",
          params = list(SelectParams(limmaSelection, "Moderated t Statistic",
                                     resubstituteParams = resubstituteParams),
                        TrainParams(DLDAtrainInterface),
                        PredictParams(DLDApredictInterface,
                                      getClasses = function(result) result[["class"]])),
          training = (1:ncol(measurements)) %% 2 == 0,
          testing = (1:ncol(measurements)) %% 2 != 0)
}
```

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, classes, ...)
  ## S4 method for signature 'DataFrame'
runTests(measurements, classes, metaFeatures = NULL, datasetName, classificationName,
         validation = c("permute", "leaveOut", "fold"),
         permutePartition = c("fold", "split"),
         permutations = 100, percent = 25, folds = 5, leave = 2,
         seed, parallelParams = bpparam(),
          params = list(SelectParams(), TrainParams(), PredictParams()), verbose = 1)
  ## S4 method for signature 'MultiAssayExperiment'
runTests(measurements, targets = names(measurements), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples. The sample identifiers must be present as column names of the matrix or the row names of the DataFrame.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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.
metaFeatures	Either NULL or a DataFrame which has meta-features of the numeric data of interest.
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.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
datasetName	A name associated with the data set used.
classificationName	A name associated with the classification.
validation	Default: "permute". "permute" for repeated permuting. "leaveOut" for leaving all possible combinations of k samples as test samples. "fold" for folding of the data set (no resampling).
permutePartition	Default: "fold". Either "fold" or "split". Only applicable if validation is "permute". If "fold", then the samples are split into folds and in each iteration one is used as the test set. If "split", the samples are split into two groups, the

sizes being based on the `percent` value. One group is used as the training set, the other is the test set.

`permutations` Default: 100. Relevant when permuting is used. The number of times to do reordering of the samples before splitting or folding them.

`percent` Default: 25. Used when permutation with the split method is chosen. The percentage of samples to be in the test set.

`folds` Default: 5. Relevant when repeated permutations are done and `permutePartition` is set to `"fold"` or when `validation` is set to `"fold"`. The number of folds to break the data set into. Each fold is used once as the test set.

`leave` Default: 2. Relevant when leave-k-out cross-validation is used. The number of samples to leave for testing.

`seed` The random number generator used for repeated resampling will use this seed, if it is provided. Allows reproducibility of repeated usage on the same input data.

`parallelParams` An object of class `MulticoreParam` or `SnowParam`.

`params` A `list` of objects of class of `TransformParams`, `SelectParams`, `TrainParams` or `PredictParams`. The order they are in the list determines the order in which the stages of classification are done in.

`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.

Value

If the predictor function made a single prediction, then an object of class `ClassifyResult`. If the predictor function made a set of predictions, then a list of such objects.

Author(s)

Dario Strbenac

Examples

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

  resubstituteParams <- ResubstituteParams(nFeatures = seq(5, 25, 5),
                                           performanceType = "balanced error",
                                           better = "lower")
  runTests(measurements, classes, "Asthma", "Different Means",
           permutations = 5,
           params = list(SelectParams(limmaSelection, "Moderated t Statistic",
                                      resubstituteParams = resubstituteParams),
                         TrainParams(DLDAtrainInterface),
                         PredictParams(DLDApredictInterface,
                                       getClasses = function(result) result[["class"]])))
}
```

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 = c("classificationName", "datasetName", "selectionName",
                                "validation"),
                 metric = c("error", "accuracy"),
                 metricColours = list(c("#3F48CC", "#6F75D8", "#9FA3E5", "#CFD1F2", "#FFFFFF"),
                                c("#880015", "#A53F4F", "#C37F8A", "#E1BFC4", "#FFFFFF")),
                 classColours = c("#3F48CC", "#880015"), 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.
comparison	The aspect of the experimental design to compare.
metric	The sample-wise metric to calculate and plot.
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.
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.
showYtickLabels	Logical. IF FALSE, the y-axis labels are hidden.
yAxisLabel	The name plotted for the y-axis. NULL suppresses label.
legendSize	The size of the boxes in the legends.
plot	Logical. IF TRUE, a plot is produced on the current graphics device.

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("Example", "Differential Expression", "t-test",
                      LETTERS[1:10], features, 100, list(1:100), list(sample(10, 10)),
                         list(predicted), actual, list("resampleFold", 100, 5))
predicted[, "class"] <- sample(predicted[, "class"])
result2 <- ClassifyResult("Example", "Differential Variability", "Bartlett Test",
                      LETTERS[1:10], features, 100, list(1:100), list(sample(10, 10)),
                         list(predicted), actual, validation = list("leave", 2))
result1 <- calcCVperformance(result1, "sample error")
result2 <- calcCVperformance(result2, "sample error")
wholePlot <- samplesMetricMap(list(Gene = result1, Protein = result2))
```

---

selectionPlot *Plot Pair-wise Overlap 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 a level 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.

Additionally, a heatmap of selection size frequencies can be made.

Usage

```
  ## S4 method for signature 'list'
selectionPlot(results,
              comparison = c("within", "size", "classificationName",
                             "validation", "datasetName", "selectionName"),
              referenceLevel = NULL,
```

```
                xVariable = c("classificationName", "datasetName", "validation", "selectionName"),
                   boxFillColouring = c("classificationName", "size", "datasetName",
                                        "validation", "selectionName", "None"),
                   boxFillColours = NULL,
                   boxFillBinBoundaries = NULL, setSizeBinBoundaries = NULL,
                   boxLineColouring = c("validation", "classificationName",
                                        "datasetName", "selectionName", "None"),
                   boxLineColours = NULL,
                   rowVariable = c("None", "validation", "datasetName",
                                   "classificationName", "selectionName"),
                   columnVariable = c("datasetName", "classificationName",
                                      "validation", "selectionName", "None"),
                   yMax = 100, fontSizes = c(24, 16, 12, 16),
                  title = if(comparison[1] == "within") "Feature Selection Stability"
                          else if(comparison == "size") "Feature Selection Size" else
                            "Feature Selection Commonality",
                   xLabel = "Analysis",
                yLabel = if(is.null(referenceLevel) && comparison != "size") "Common Features (%)"
                           else if(comparison == "size") "Set Size" else
                           paste("Common Features with", referenceLevel, "(%)"),
                   margin = grid::unit(c(0, 0, 0, 0), "lines"), rotate90 = FALSE,
                   showLegend = TRUE, plot = TRUE, parallelParams = bpparam())
```

Arguments

results	A list of ClassifyResult or SelectResult objects.
comparison	The aspect of the experimental design to compare. See Details section for a detailed description.
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.
xVariable	The factor to make separate boxes in the boxplot for.
boxFillColouring	A factor to colour the boxes by.
boxFillColours	A vector of colours, one for each level of boxFillColouring. If NULL, a default palette is used.
boxFillBinBoundaries	Used only if comparison is "size". A vector of integers, specifying the bin boundaries of percentages of size bins observed. e.g. 0, 10, 20, 30, 40, 50.
setSizeBinBoundaries	Used only if comparison is "size". A vector of integers, specifying the bin boundaries of set size bins. e.g. 50, 100, 150, 200, 250.
boxLineColouring	A factor to colour the box lines by.
boxLineColours	A vector of colours, one for each level of boxLineColouring. If NULL, a default palette is used.
rowVariable	The slot name that different levels of are plotted as separate rows of boxplots.
columnVariable	The slot name that different levels of are plotted as separate columns of boxplots.
yMax	The maximum value of the percentage 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.
xLabel	Label to be used for the x-axis.
yLabel	Label to be used for the y-axis of overlap percentages.
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

Possible values for characteristics are "datasetName", "classificationName", "size", "selectionName", and "validation". If "None", then that graphical element is not used.

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. If comparison is "classificationName", then the feature selections are compared across different classification algorithm types, for each level of "datasetName", "selectionName" and "validation". The result will inform how stable the feature selections are between different classification algorithms, for every cross-validation scheme, selection algorithm and data set. If comparison is "selectionName", then the feature selections are compared across different feature selection algorithms, for each level of "datasetName", "classificationName" and "validation". The result will inform how stable the feature selections are between feature selection algorithms, for every data set, classification algorithm, and cross-validation scheme. If comparison is "validation", then the feature selections are compared across different cross-validation schemes, for each level of "classificationName", "selectionName" and "datasetName". The result will inform how stable the feature selections are between different cross-validation schemes, for every selection algorithm, classification algorithm and every data set. If comparison is "datasetName", then the feature selections are compared across different data sets, for each level of "classificationName", "selectionName", and "validation". The result will inform how stable the feature selections are between different data sets, for every classification algorithm and every data set. This could be used to consider if different experimental studies have a highly overlapping feature selection pattern.

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 selection size mode, boxFillBins 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))
features <- sapply(1:100, function(index) paste(sample(LETTERS, 3), collapse = ''))
rankList <- list(list(1:100, c(5:1, 6:100)),
                 list(c(1:9, 11:101), c(1:50, 60:51, 61:100)))
result1 <- ClassifyResult("Example", "Differential Expression",
                          "Example Selection", LETTERS[1:10], features,
                          100, rankList,
                         list(list(rankList[[1]][[1]][1:15], rankList[[1]][[2]][1:15]),
                             list(rankList[[2]][[1]][1:10], rankList[[2]][[2]][1:10])),
                          list(predicted), actual, list("resampleFold", 2, 2))

predicted[, "class"] <- sample(predicted[, "class"])
rankList <- list(list(1:100, c(sample(20), 21:100)),
                 list(c(1:9, 11:101), c(1:50, 60:51, 61:100)))
result2 <- ClassifyResult("Example", "Differential Variability",
                          "Example Selection",
                          LETTERS[1:10], features, 100, rankList,
                         list(list(rankList[[1]][[1]][1:15], rankList[[1]][[2]][1:15]),
                             list(rankList[[2]][[1]][1:10], rankList[[2]][[2]][1:10])),
                        list(predicted), actual, validation = list("resampleFold", 2, 2))

selectionPlot(list(result1, result2), xVariable = "classificationName",
              xLabel = "Analysis", columnVariable = "None", rowVariable = "None",
              boxFillColouring = "classificationName")

selectionPlot(list(result1, result2), comparison = "size",
              xVariable = "classificationName", xLabel = "Analysis",
              columnVariable = "None", rowVariable = "None",
              boxFillColouring = "size", boxFillBinBoundaries = seq(0, 100, 10),
              setSizeBinBoundaries = seq(0, 25, 5), boxLineColouring = "None")

oneRanking <- c(10, 8, 1, 2, 3, 4, 7, 9, 5, 6)
otherRanking <- c(8, 2, 3, 4, 1, 10, 6, 9, 7, 5)
oneResult <- SelectResult("Example", "One Method", 10, list(oneRanking), list(oneRanking[1:5]))
otherResult <- SelectResult("Example", "Another Method", 10, list(otherRanking), list(otherRanking[1:2]))

selectionPlot(list(oneResult, otherResult), comparison = "selectionName",
              xVariable = "selectionName", xLabel = "Selection Method",
              columnVariable = "None", rowVariable = "None",
              boxFillColouring = "selectionName", boxLineColouring = "None")
```

SelectParams *Parameters for Feature Selection*

Description

Collects and checks necessary parameters required for feature selection. The empty constructor is provided for convenience.

Constructor

`SelectParams()` Creates a default SelectParams object. This uses a limma t-test 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, once they are familiar with this software.

```
SelectParams(featureSelection, selectionName, minPresence = 1, intermediate = character(0),
             subsetToSelections = TRUE, ...)
```

Creates a SelectParams object which stores the function which will do the selection and parameters that the function will use.

`featureSelection` Either a function which will do the selection 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 `SelectResult` object.

`selectionName` A name to identify this selection method by.

`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.

`...` 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`.

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
  SelectParams(limmaSelection, "t-test",
               trainParams = TrainParams(), predictParams = PredictParams(),
               resubstituteParams = ResubstituteParams())

  # For pamr shrinkage selection.
  SelectParams(NSCselectionInterface, datasetName = "Cancer",
               intermediate = "trained", subsetToSelections = FALSE)
}
```

SelectResult *Container for Storing Feature Selection Results*

Description

Contains a list of ranked names of features, from most discriminative to least discriminative, and a list of features selected for use in classification. The names will be in a data frame if the input data set is a `MultiAssayExperiment`, with the first column containing the name of the data table the feature is from and the second column containing the name of the feature. Each vector or data frame element in the list corresponds to a particular iteration of classifier training. Nested lists will be present if the permutation and folding cross-validation scheme was used. This class is not intended to be created by the user, but could be used in another software package.

Constructor

```
SelectResult(datasetName, selectionName, totalFeatures, rankedFeatures, chosenFeatures)
```

`datasetName` A name associated with the data set used.

`selectionName` A name associated with the classification.

`totalFeatures` The total number of features in the data set.

`rankedFeatures` Identifiers of all features or meta-features if meta-features were used by the classifier, from most to least discriminative.

`chosenFeatures` Identifiers of features or meta-features if meta-features were used by the classifier selected at each fold.

Summary

A method which summarises the results is available. `result` is a `SelectResult` object.

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

Author(s)

Dario Strbenac

Examples

```
SelectResult("Asthma", "Moderated t-test", 50, list(1:50), list(1:10))
```

---

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

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.
training	A vector specifying which samples are in the training set.
location	Character. Either "mean" or "median".
absolute	Logical. Default: TRUE. If TRUE, then absolute values of the differences are returned. Otherwise, they are signed.
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.
verbose	Default: 3. A progress message is shown if this value is 3.

Details

Only the samples specified by training are used in the calculation of the location. To use all samples for calculation of the location, simply provide indices of all the samples.

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, training = 1:5, "median")
```

---

svm *Trained svm Object*

Description

Enables S4 method dispatching on it.

Author(s)

Dario Strbenac

SVMinterface *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(measurements, classes, ...)
  ## S4 method for signature 'DataFrame'
SVMtrainInterface(measurements, classes, ..., verbose = 3)
  ## S4 method for signature 'MultiAssayExperiment'
SVMtrainInterface(measurements, targets = names(measurements), ...)
  ## S4 method for signature 'svm,matrix'
SVMpredictInterface(model, test, ...)
  ## S4 method for signature 'svm,DataFrame'
SVMpredictInterface(model, test, classes = NULL, verbose = 3)
  ## S4 method for signature 'svm,MultiAssayExperiment'
SVMpredictInterface(model, test, targets = names(test), ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are features, and the columns are samples. If of type DataFrame, the data set is subset to only those features of type integer.
classes	Either a vector of class labels of class factor of the same length as the number of samples in measurements 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.
test	An object of the same class as measurements with no samples in common with measurements and the same number of features as it. Also, if a DataFrame, the class column must be absent.
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.
model	A fitted model as returned by SVMtrainInterface.
...	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.
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.

Details

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

For SVMtrainInterface, a trained SVM classifier of type svm. For SVMpredictInterface, a result of type factor, as created by e1071's predict method for trained SVM models.

Author(s)

Dario Strbenac

Examples

```
if(require(e1071))
{
  # 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))
  rownames(genesMatrix) <- paste("Gene", 1:nrow(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. Users should create an appropriate `TrainParams` object for the characteristics of their data, once they are familiar with this software.

```
TrainParams(classifier, intermediate = character(0), transform = NULL,
            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.

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

`transform` Either NULL or a function. The function transforms the training data into a different format before it is passed into the classifier. For example, a set of features may be replaced by their median value. The set of features might be part of a network.

`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.

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. The empty constructor is for when no data transformation is desired. One data transformation function is distributed. See `subtractFromLocation`.

Constructor

`TransformParams(transform, 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.

`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.

Author(s)

Dario Strbenac

Examples

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

Index