Package 'ClassifyR'

April 22, 2016

Type Package

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

Version 1.4.15

Date 2016-03-08

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

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

VignetteBuilder knitr

biocViews Classification, Survival

Depends R (>= 3.0.3), methods, Biobase, BiocParallel

Imports locfit, ROCR, grid

Suggests limma, edgeR, car, Rmixmod, ggplot2 (>= 2.0.0), gridExtra (>= 2.0.0), BiocStyle, pamr, sparsediscrim, PoiClaClu, curatedOvarianData, parathyroidSE, knitr, klaR, gtable, scales, e1071, rmarkdown, IRanges

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 bartlettSelection.R classes.R utilities.R calcPerformance.R classifyInterface.R DMDselection.R edgeRselection.R errorMap.R fisherDiscriminant.R distribution.R getLocationsAndScales.R KolmogorovSmirnovSelection.R KullbackLeiblerSelection.R leveneSelection.R likelihoodRatioSelection.R limmaSelection.R mixmodels.R naiveBayesKernel.R nearestShrunkenCentroidSelectionInterface.R

nearestShrunkenCentroidTrainInterface.R
nearestShrunkenCentroidPredictInterface.R performancePlot.R
plotFeatureClasses.R previousSelection.R rankingPlot.R
ROCplot.R runTest.R runTests.R selectionPlot.R
subtractFromLocation.R

License GPL-3

NeedsCompilation no

R topics documented:

`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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
bartlettSelection(expression, datasetName,
                                   trainParams, predictParams, resubstituteParams,
                                   selectionName = "Bartlett Test", verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	For the `matrix` method, variables passed to the `ExpressionSet` method.
`datasetName`	A name for the dataset 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`	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.

Value

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

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
  # Samples in one class with differential variability to other class.
  # First 20 genes are DV.
  genesMatrix <- sapply(1:25, function(sample) c(rnorm(100, 9, 1)))
 genesMatrix <- cbind(genesMatrix, rbind(sapply(1:25, function(sample) rnorm(20, 9, 5)),
                                    sapply(1:25, function(sample) rnorm(80, 9, 1))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  genesMatrix <- exprs(subtractFromLocation(genesMatrix, 1:ncol(genesMatrix)))
  bartlettSelection(genesMatrix, classes, datasetName = "Example",
                  trainParams = TrainParams(fisherDiscriminant, FALSE, TRUE),
             predictParams = PredictParams(function(){}, FALSE, getClasses = function(result) result),
                resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10),
                                            performanceType = "balanced", better = "lower"))
}
```

calcPerformance *Add Performance Calculations to a ClassifyResult object*

Description

Annotates the results of calling `runTests` with different kinds of performance measures.

Usage

```
  ## S4 method for signature 'ClassifyResult'
calcPerformance(result, performanceType, ...)
```

Arguments

`result`	An object of class `ClassifyResult`.
`performanceType`	Either `"balanced"` or one of the options provided by `performance`.
`...`	Further arguments that may be used by `performance`.

Details

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

Because ROCR only provides calculations for two-class classification, this function is only suitable for two-class classification performance measures.

Value

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

Author(s)

Dario Strbenac

Examples

```
predictTable <- data.frame(sample = 1:5,
                           label = factor(sample(LETTERS[1:2], 50, replace = TRUE)))
actual <- factor(sample(LETTERS[1:2], 50, replace = TRUE))
result <- ClassifyResult("Example", "Differential Expression", "A Selection",
                         paste("A", 1:10, sep = ''), paste("Gene", 1:50, sep = ''),
                         list(1:100, 1:100), list(1:5, 6:15),
                         list(predictTable), actual, list("leave", 2))
result <- calcPerformance(result, "balanced")
performance(result)
```

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

Description

Passes along all parameters except `verbose`, from the framework to `Classify`.

Usage

```
classifyInterface(..., verbose = 3)
```

Arguments

`...`	All parameters that `Classify` can accept and also `verbose`.
`verbose`	A number between 0 and 3 for the amount of progress messages to give. This function only prints a progress message if the value is 3.

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, 1)
 classifyInterface(readCounts[["x"]], readCounts[["y"]], readCounts[["xte"]], verbose = TRUE)
}
```

`ClassifyResult` *Container for Storing Classification Results*

Description

Contains a table of actual sample classes and predicted classes, the indices of features selected for each fold of each bootstrap resampling 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,
```

`datasetName` A name associated with the dataset used.

`classificationName` A name associated with the classification.

`originalNames` Sample names.

`originalFeatures` Feature names.

`rankedFeatures` Indices or names of all features, from most to least important.

`chosenFeatures` Indices or names of features selected at each fold.

`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 elment being name of the validation scheme, and other elements providing details about scehme.

`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.

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 calcPerformance 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 validation.

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

Author(s)

Dario Strbenac

Examples

```
if(require(curatedOvarianData) && require(sparsediscrim))
{
  data(TCGA_eset)
 badOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "deceased" & pData(TCGA_eset)[, "days_to_death"] <=
 goodOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "living" & pData(TCGA_eset)[, "days_to_death"] >=
  TCGA_eset <- TCGA_eset[, c(badOutcome, goodOutcome)]
 classes <- factor(rep(c("Poor", "Good"), c(length(badOutcome), length(goodOutcome))))
  pData(TCGA_eset)[, "class"] <- classes
 results <- runTests(TCGA_eset, "Ovarian Cancer", "Differential Expression", resamples = 2, folds = 2)
  show(results)
  predictions(results)
  actualClasses(results)
}
```

---

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(curatedOvarianData) && require(sparsediscrim))
{
  data(TCGA_eset)
 badOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "deceased" & pData(TCGA_eset)[, "days_to_death"] <=
 goodOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "living" & pData(TCGA_eset)[, "days_to_death"] >=
  TCGA_eset <- TCGA_eset[, c(badOutcome, goodOutcome)]
 classes <- factor(rep(c("Poor", "Good"), c(length(badOutcome), length(goodOutcome))))
```

```
  pData(TCGA_eset)[, "class"] <- classes
 result <- runTests(TCGA_eset, "Ovarian Cancer", "Differential Expression", resamples = 2, fold = 2)
  sampleDistribution <- distribution(result, "samples", xLabel = "Sample Error Rate",
                                     title = "Distribution of Error Rates")
 featureDistribution <- distribution(result, "features", summaryType = "count", plotType = "histogram",
                              xLabel = "Number of Cross-validations", yLabel = "Count",
                                      binwidth = 1)
  print(head(sampleDistribution))
  print(head(featureDistribution))
}
```

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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
DMDselection(expression, datasetName,
                                trainParams, predictParams, resubstituteParams, ...,
                                       selectionName, verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`datasetName`	A name for the dataset 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.
`...`	Either variables passed from the `matrix` method to the `ExpressionSet` method or variables passed to `getLocationsAndScales` from the `ExpressionSet` method.
`selectionName`	A name to identify this selection method by. Stored in the result.
`verbose`	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.

Value

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

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
 # First 20 features have bimodal distribution for Poor class. Other 80 features have normal distribution for
  # both classes.
 genesMatrix <- sapply(1:25, function(sample) c(rnorm(20, sample(c(8, 12), 20, replace = TRUE), 1), rnorm(80, 1
  genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  DMDselection(genesMatrix, classes, datasetName = "Example",
                 trainParams = TrainParams(naiveBayesKernel, FALSE, doesTests = TRUE),
            predictParams = PredictParams(function(){}, FALSE, getClasses = function(result) result),
            resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10), performanceType = "balanced"
}
```

edgeRselection *Feature Selection Based on Differential Expression for RNA-seq*

Description

Performs a differential expression analysis between classes and chooses the features which have best resubstitution performance.

Usage

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

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the expression values.
`classes`	A vector of class labels.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`datasetName`	A name for the dataset 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`	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.

Value

An object of class `SelectResult` or a list of such objects, if the classifier which was used for determining resubstitution error rate 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, `bioinformatics.oxfordjournals.org/content/26/1/139`.

Examples

```
if(require(parathyroidSE) && require(sparsediscrim) && require(PoiClaClu))
{
  data(parathyroidGenesSE)
  expression <- assays(parathyroidGenesSE)[[1]]
  DPN <- which(colData(parathyroidGenesSE)[, "treatment"] == "DPN")
```

```
    control <- which(colData(parathyroidGenesSE)[, "treatment"] == "Control")
    expression <- expression[, c(control, DPN)]
    classes <- rep(c("Contol", "DPN"), c(length(control), length(DPN)))
    expression <- expression[rowSums(expression > 1000) > 8, ] # Make small dataset.
    edgeRselection(expression, classes, "DPN Treatment",
                   trainParams = TrainParams(classifyInterface, TRUE, TRUE),
               predictParams = PredictParams(function(){}, TRUE, getClasses = function(result) result[["ytehat"
                   resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10),
                                            performanceType = "balanced", better = "lower"))
  }
```

---

errorMap *Plot a Grid of Sample Error Rates*

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'
errorMap(results,
                comparison = c("classificationName", "datasetName", "selectionName", "validation"),
                errorColours = list(c("#0000FF", "#3F3FFF", "#7F7FFF", "#BFBFFF", "#FFFFFF"),
                                 c("#FF0000", "#FF3F3F", "#FF7F7F", "#FFBFBF", "#FFFFFF")),
                classColours = c("blue", "red"), fontSizes = c(24, 16, 12, 12, 12),
                mapHeight = 4, title = "Error Comparison", showLegends = TRUE, xAxisLabel = "Sample Nam
                showXtickLabels = TRUE, showYtickLabels = TRUE, yAxisLabel = "Analysis",
                    legendSize = grid::unit(1, "lines"), plot = TRUE)
```

Arguments

`results`	A list of `ClassifyResult` objects.
`comparison`	The aspect of the experimental design to compare.
`errorColours`	A vector of colours for error 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 `errorColours` determines how many bins the error rates 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 = sample(10, 100, replace = TRUE),
                        label = rep(c("Healthy", "Cancer"), each = 50))
actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
result1 <- ClassifyResult("Example", "Differential Expression", "t-test",
                      LETTERS[1:10], LETTERS[10:1], list(1:100), list(sample(10, 10)),
                         list(predicted), actual, list("fold", 100, 5))
predicted[, "label"] <- sample(predicted[, "label"])
result2 <- ClassifyResult("Example", "Differential Variability", "F-test",
                      LETTERS[1:10], LETTERS[10:1], list(1:100), list(sample(10, 10)),
                         list(predicted), actual, validation = list("leave", 1))
wholePlot <- errorMap(list(Gene = result1, Protein = result2))
# if(require(ggplot2))
  # ggsave("wholePlot.png", wholePlot)
```

`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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
fisherDiscriminant(expression, test, returnType = c("label", "score", "both"), verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`test`	Either a `matrix` or `ExpressionSet` containing the test data.
`returnType`	Either `"label"`, `"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`	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.

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)
trainMatrix[1:30, 1:5] <- trainMatrix[1:30, 1:5] + 5 # Make first 30 genes D.E.
testMatrix <- matrix(rnorm(1000, 8, 2), ncol = 10)
testMatrix[1:30, 6:10] <- testMatrix[1:30, 6:10] + 5 # Make first 30 genes D.E.
classes <- factor(rep(c("Poor", "Good"), each = 5))
fisherDiscriminant(trainMatrix, classes, testMatrix)
```

`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")
```

`getLocationsAndScales` *Calculate Location and Scale*

Description

Calculates the location and scale for each feature.

Usage

```
  ## S4 method for signature 'matrix'
getLocationsAndScales(expression, ...)
  ## S4 method for signature 'ExpressionSet'
getLocationsAndScales(expression, location = c("mean", "median"),
                                  scale = c("SD", "MAD", "Qn"))
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing data. For a matrix, the rows are features, and the columns are samples.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`location`	The location to be calculated.
`scale`	The scale to be calculated.

Details

Location can be either "mean" or "median". Scale can be standard deviation, median absolute deviation, or Q_n.

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)
getLocationsAndScales(genesMatrix, "median", "MAD")
```

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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
KolmogorovSmirnovSelection(expression, datasetName, trainParams,
                                      predictParams, resubstituteParams, ...,
                                      selectionName, verbose = 3)
```

Arguments

expression	Either a matrix or ExpressionSet containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	A vector of class labels.
datasetName	A name for the dataset 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.

`...`	For the `matrix` method, variables passed to the `ExpressionSet` method. For the `ExpressionSet` method, the options to be passed to function `ks.test`.
`selectionName`	A name to identify this selection method by. Stored in the result.
`verbose`	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.

Value

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

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
 # First 20 features have bimodal distribution for Poor class. Other 80 features have normal distribution for
  # both classes.
 genesMatrix <- sapply(1:25, function(sample) c(rnorm(20, sample(c(8, 12), 20, replace = TRUE), 1), rnorm(80, 1
  genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  KolmogorovSmirnovSelection(genesMatrix, classes, "Example",
                       trainParams = TrainParams(naiveBayesKernel, FALSE, doesTests = TRUE),
                       predictParams = PredictParams(function(){}, FALSE, getClasses = function(result) result
                       resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10), performanceType =
}
```

`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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
KullbackLeiblerSelection(expression, datasetName,
                              trainParams, predictParams, resubstituteParams, ...,
                                     selectionName, verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`datasetName`	A name for the dataset 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.
`...`	Variables passed to `getLocationsAndScales`.
`selectionName`	A name to identify this selection method by. Stored in the result.
`verbose`	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 $1/2 * (location_1 - location_2)^2$

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

Value

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

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
 # First 20 features have bimodal distribution for Poor class. Other 80 features have normal distribution for
  # both classes.
 genesMatrix <- sapply(1:25, function(sample) c(rnorm(20, sample(c(8, 12), 20, replace = TRUE), 1), rnorm(80, 1
  genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
```

```
    KullbackLeiblerSelection(genesMatrix, classes, "Example",
                       trainParams = TrainParams(naiveBayesKernel, FALSE, doesTests = TRUE),
                       predictParams = PredictParams(function(){}, FALSE, getClasses = function(result) result)
                       resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10), performanceType = "
                            )
  }
```

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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
leveneSelection(expression, datasetName,
                       trainParams, predictParams, resubstituteParams, selectionName = "Levene Test"
                             verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	For the `matrix` method, variables passed to the `ExpressionSet` method.
`datasetName`	A name for the dataset 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`	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.

Value

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

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
  # Samples in one class with differential variability to other class.
  # First 20 genes are DV.
  genesMatrix <- sapply(1:25, function(sample) c(rnorm(100, 9, 1)))
 genesMatrix <- cbind(genesMatrix, rbind(sapply(1:25, function(sample) rnorm(20, 9, 5)),
                                     sapply(1:25, function(sample) rnorm(80, 9, 1))))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  genesMatrix <- exprs(subtractFromLocation(genesMatrix, 1:ncol(genesMatrix)))
  leveneSelection(genesMatrix, classes, "Example",
                  trainParams = TrainParams(fisherDiscriminant, FALSE, TRUE),
              predictParams = PredictParams(function(){}, FALSE, getClasses = function(result) result),
                 resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10),
                                     performanceType = "balanced", better = "lower"))
}
```

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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
likelihoodRatioSelection(expression, datasetName, trainParams, predictParams,
                              resubstituteParams, alternative = c(location = "different", scale = "d
                              ..., selectionName = "Likelihood Ratio Test (Normal)", verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`datasetName`	A name for the dataset 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`	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. Acceptable values are "same" or "different".
`...`	Either variables passed from the `matrix` method to the `ExpressionSet` method or variables passed to `getLocationsAndScales` from the `ExpressionSet` method.
`selectionName`	A name to identify this selection method by. Stored in the result.
`verbose`	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 calucated by `getLocationsAndScales`. The distribution fitted in the normal distribution.

Value

A list of length 2. The first element has the features ranked from most important to least important. The second element has the features that were selected to be used for classification.

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
{
 # First 20 features have bimodal distribution for Poor class. Other 80 features have normal distribution for
  # both classes.
 genesMatrix <- sapply(1:25, function(sample) c(rnorm(20, sample(c(8, 12), 20, replace = TRUE), 1), rnorm(80, 1
  genesMatrix <- cbind(genesMatrix, sapply(1:25, function(sample) rnorm(100, 10, 1)))
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  likelihoodRatioSelection(genesMatrix, classes, "Example",
                           trainParams = TrainParams(naiveBayesKernel, FALSE, TRUE),
                     predictParams = PredictParams(function(){}, FALSE, getClasses = function(result) result)
                     resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10), performanceType = "
}
```

`limmaSelection` *Selection of Differentially Expressed Features*

Description

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

Usage

```
  ## S4 method for signature 'matrix'
limmaSelection(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
limmaSelection(expression, datasetName, trainParams, predictParams,
                                 resubstituteParams, ..., selectionName = "Moderated t-test", verbo
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`datasetName`	A name for the dataset 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.
`...`	For the `matrix` method, variables passed to the `ExpressionSet` method. For the `ExpressionSet` method, extra parameters that are passed to `lmFit`.
`selectionName`	A name to identify this selection method by. Stored in the result.
`verbose`	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 differential expression. It 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 resubstitution error rate 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))

  limmaSelection(genesMatrix, classes, "Example",
                 trainParams = TrainParams(), predictParams = PredictParams(),
             resubstituteParams = ResubstituteParams(nFeatures = seq(10, 100, 10), performanceType = "balance
}
```

---

mixmodels *Selection of Differential Distributions with Mixtures of Normals*

Description

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

Usage

```
  ## S4 method for signature 'matrix'
mixModelsTrain(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
mixModelsTrain(expression, ..., verbose = 3)
  ## S4 method for signature 'list,matrix'
mixModelsTest(models, test, ...)
  ## S4 method for signature 'list,ExpressionSet'
mixModelsTest(models, test,
            weighted = c("both", "unweighted", "weighted"),
          weight = c("all", "height difference", "crossover distance", "sum differences"),
            densityXvalues = 1024, minDifference = 0,
            returnType = c("label", "score", "both"), verbose = 3)
```

Arguments

expression	Either a matrix or ExpressionSet containing the training data. For a matrix, the rows are features, and the columns are samples.
test	Either a matrix or ExpressionSet containing the test data. For a matrix, the rows are features, and the columns are samples.

`classes`	A vector of class labels.
`weighted`	In weighted mode, the difference in densities is summed over all features. If unweighted mode, each features's vote is worth the same. To save computational time, both can be calculated simultaneously.
`weight`	The type of weight to calculate. For `"height difference"`, the weight of each prediction is equal to the sum of the verical 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.
`densityXvalues`	Only relevant when `weight` is `"crossover distance"`. The number of equally-spaced locations at which to calculate y values for each mixture density.
`minDifference`	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.
`...`	For the training or testing function with `matrix` dispatch, arguments passed to the function with `ExpressionSet` dispatch. For the training function with `ExpressionSet` dispatch, extra arguments passed to `mixmodCluster`. The argument `nbCluster` is mandatory.
`models`	A list of length 2 of models generated by the training function. 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.
`returnType`	Either `"label"`, `"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`	A number between 0 and 3 for the amount of progress messages to give. A higher number will produce more messages.

Details

If `weighted` is `TRUE`, then a sample's predicted class is the class with the largest sum of weights, 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. Setting weight to `"sum differences"` is intended to find a mix of features which are strongly differentially expressed and differentially variable.

Value

For `mixModelsTrain`, a list of trained models of class `MixmodCluster`. 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.

Author(s)

Dario Strbenac

Examples

```
# First 25 samples are mixtures of two normals. Last 25 samples are one normal.
genesMatrix <- sapply(1:25, function(geneColumn) c(rnorm(50, 5, 1), rnorm(50, 15, 1)))
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(geneColumn) rnorm(100, 9, 3)))
classes <- factor(rep(c("Poor", "Good"), each = 25))
trained <- mixModelsTrain(genesMatrix, classes, nbCluster = 1:3)
mixModelsTest(trained, genesMatrix, minDifference = 1:3)
```

`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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
naiveBayesKernel(expression, test, densityFunction = density,
         densityParameters = list(bw = "nrd0", n = 1024, from = expression(min(featureValues)),
                                  to = expression(max(featureValues))),
           weighted = c("both", "unweighted", "weighted"),
         weight = c("all", "height difference", "crossover distance", "sum differences"),
         minDifference = 0, returnType = c("label", "score", "both"), verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`test`	Either a `matrix` or `ExpressionSet` containing the test data.
`densityFunction`	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`.

`weighted`	In weighted mode, the difference in densities is summed over all features. If unweighted mode, each feature's vote is worth the same. To save computational time, both can be calculated simultaneously.
`weight`	The type of weight to calculate. For `"height difference"`, the weight of each prediction is equal to the verical 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. For `"sum differences"`, the weight is the sum of the weights calculated by both types of distances.
`minDifference`	The minimum difference in densities for a feature to be allowed to vote. Can be a vector of cutoffs. If no features for a particular sample have a difference large enough, the class predicted is simply the largest class.
`returnType`	Either `"label"`, `"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`	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 `weighted` is `TRUE`, then a sample's predicted class is the class with the largest sum of weights, 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.

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.

Setting weight to `"sum differences"` is intended to find a mix of features which are strongly differentially expressed and differentially variable.

Value

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.

Author(s)

Dario Strbenac, John Ormerod

Examples

```
trainMatrix <- matrix(rnorm(1000, 8, 2), ncol = 10)
trainMatrix[1:30, 1:5] <- trainMatrix[1:30, 1:5] + 5 # Make first 30 genes D.E.
testMatrix <- matrix(rnorm(1000, 8, 2), ncol = 10)
testMatrix[1:30, 6:10] <- testMatrix[1:30, 6:10] + 5 # Make first 30 genes D.E.
classes <- factor(rep(c("Poor", "Good"), each = 5))
# Expected: Good Good Good Good Good Poor Poor Poor Poor Poor
naiveBayesKernel(trainMatrix, classes, testMatrix)
```

`nearestShrunkenCentroidPredictInterface`
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'
nearestShrunkenCentroidPredictInterface(trained, test, ...)
  ## S4 method for signature 'pamrtrained,ExpressionSet'
nearestShrunkenCentroidPredictInterface(trained, test, ..., verbose = 3)
```

Arguments

`trained`	An object of class `pamrtrained`.
`test`	Either a `matrix` or `ExpressionSet` containing the test data. For a matrix, the rows are features, and the columns are samples.
`...`	For the function with `matrix` dispatch, arguments passed to the function with `ExpressionSet` dispatch. For the function with `ExpressionSet` dispatch, arguemnts passed to `pamr.predict`.
`verbose`	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`.

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

---

nearestShrunkenCentroidSelectionInterface

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'
nearestShrunkenCentroidSelectionInterface(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
nearestShrunkenCentroidSelectionInterface(expression, datasetName, trained, ...,
                                           selectionName = "Shrunken Centroids", verbose = 3)
```

Arguments

expression	Either a matrix or ExpressionSet containing the training data. For a matrix, the rows are features, and the columns are samples.
datasetName	A name for the dataset used. Stored in the result.
classes	A vector of class labels.
trained	The output of nearestShrunkenCentroidTrainInterface, which is identical to the output of pamr.listgenes.
...	Extra arguments passed to pamr.listgenes or parameters not used by the matrix method that are passed to the ExpressionSet method.
selectionName	A name to identify this selection method by. Stored in the result.
verbose	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 `nearestShrunkenCentroidTrainInterface` 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))))
    classes <- factor(rep(c("Poor", "Good"), each = 25))

    trained <- nearestShrunkenCentroidTrainInterface(genesMatrix, classes)
    nearestShrunkenCentroidSelectionInterface(genesMatrix, classes, "Example", trained)
  }
```

`nearestShrunkenCentroidTrainInterface`
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'
nearestShrunkenCentroidTrainInterface(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
nearestShrunkenCentroidTrainInterface(expression, ..., verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	Extra arguments passed to `pamr.train`.
`verbose`	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))

  nearestShrunkenCentroidTrainInterface(genesMatrix, classes)
}
```

`pamrtrained` *Trained pamr Object*

Description

Enables dispatching on it.

Summary

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

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

Author(s)

Dario Strbenac

Examples

```
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))

result <- nearestShrunkenCentroidTrainInterface(genesMatrix, classes)
class(result)
```

---

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", "validatio
                    boxFillColours = NULL,
                boxLineColouring = c("classificationName", "datasetName", "selectionName", "validatio
                    boxLineColours = NULL,
                rowVariable = c("None", "validation", "datasetName", "classificationName", "selection
                columnVariable = c("datasetName", "classificationName", "validation", "selectionName"
                  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
```

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 partciularly 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),
                               label = rep(c("Healthy", "Cancer"), each = 10)),
                    data.frame(sample = sample(10, 20, replace = TRUE),
                                label = rep(c("Healthy", "Cancer"), each = 10)),
                    data.frame(sample = sample(10, 20, replace = TRUE),
                                label = rep(c("Healthy", "Cancer"), each = 10)),
                   data.frame(sample = sample(10, 20, replace = TRUE),
                              label = 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], list(1
                           predicted, actual, list("fold", 2, 2))
  result1 <- calcPerformance(result1, "f")
  predicted <- data.frame(sample = sample(10, 100, replace = TRUE),
                          label = rep(c("Healthy", "Cancer"), each = 50))
  result2 <- ClassifyResult("Example", "Differential Variability", "F-test", LETTERS[1:10], LETTERS[10:1], list(
                           list(predicted), actual, validation = list("leave", 1))
  result2 <- calcPerformance(result2, "f")
  performancePlot(list(result1, result2), performanceName = "Precision-Recall F measure", title = "Comparison", b
```

plotFeatureClasses *Plot Density and Scatterplot for Genes By Class*

Description

Allows the visualisation of genes which were selected by a feature selection method.

Usage

```
  ## S4 method for signature 'matrix'
plotFeatureClasses(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
plotFeatureClasses(expression, rows, whichPlots = c("both", "density", "stripchart"),
                        xAxisLabel = expression(log[2](expression)), expressionLimits = c(2, 16),
                        yAxisLabels = c("Density", "Classes"), showXtickLabels = TRUE,
                              showYtickLabels = TRUE, xLabelPositions = "auto",
                        yLabelPositions = "auto", fontSizes = c(24, 16, 12, 12, 12),
                              colours = c("blue", "red"), plot = TRUE)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`rows`	A vector specifying which rows of the matrix to plot.
`whichPlots`	Which plots to draw. Can draw either a density plot, stripchart, or both.
`xAxisLabel`	The axis label for the expression axis.
`yAxisLabels`	A character vector of length 2. The first value is the y-axis label for the density plot. The second value is the y-axis labels for the stripchart. Provide both labels, even if only plotting one kind of plot.
`expressionLimits`	The minimum and maximum expression values to plot. Set to NULL to use range of data.

`showXtickLabels`	Logical. IF FALSE, the x-axis labels are hidden.
`showYtickLabels`	Logical. IF 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.
`plot`	Logical. If `TRUE`, a plot is produced on the current graphics device.

Value

Plots.

Author(s)

Dario Strbenac

Examples

```
# First 25 samples are mixtures of two normals. Last 25 samples are one normal.
genesMatrix <- sapply(1:25, function(geneColumn) c(rnorm(50, 5, 1), rnorm(50, 15, 1)))
genesMatrix <- cbind(genesMatrix, sapply(1:25, function(geneColumn) rnorm(100, 9, 3)))
classes <- factor(rep(c("Poor", "Good"), each = 25), levels = c("Good", "Poor"))
chosen <- 1:5 # First five genes in the data were chosen.

plotFeatureClasses(genesMatrix, classes, chosen, expressionLimits = NULL)
```

`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, transposeExpression, intermediate = character(0), getClasses, ...)` Creates a PredictParams object which stores the function which will do the class prediction and parameters that the function will use.

- `predictor` A `function` to make predictions with. The first argument must accept the classifier made in the training step. The second argument must accept a `matrix` of new data.
- `transposeExpression` Set to `TRUE` if `classifier` expects features as columns.
- `intermediate` Character vector. Names of any variables created in prior stages by `runTest` that need to be passed to the prediction function.
- `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 = predict, TRUE, getClasses = function(result) result)
# For prediction by trained object created by dlda function.
PredictParams(predictor = function(){}, TRUE, getClasses = function(result) result)
# For when the training function also does prediction and directly returns vector of 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(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
previousSelection(expression, datasetName, classifyResult,
                                              minimumOverlapPercent = 80,
                                     selectionName = "Previous Selection", .iteration, verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`...`	For the `matrix` method, variables passed to the `ExpressionSet` method.
`datasetName`	A name for the dataset 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 dataset, then the selection stops with an error.
`selectionName`	A name to identify this selection method by. Stored in the result.
`.iteration`	Not to be set by the user.
`verbose`	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))))
  rownames(genesMatrix) <- paste("Gene", 1:100)
  classes <- factor(rep(c("Poor", "Good"), each = 25))
  resubstitute <- ResubstituteParams(nFeatures = seq(10, 100, 10),
                                     performanceType = "err", better = "lower")
 result <- runTests(genesMatrix, classes, "Ovarian Cancer", "Differential Expression",
                    resamples = 2, fold = 2,
               params = list(SelectParams(limmaSelection, resubstituteParams = resubstitute),
                    TrainParams(dlda, TRUE, FALSE),
               PredictParams(predict, TRUE, getClasses = function(result) result[["class"]])))

  # Genes 74 to 98 have differential expression in new dataset.
  newDataset <- sapply(1:25, function(sample) c(rnorm(100, 9, 2)))
 newDataset <- cbind(newDataset, rbind(sapply(1:25, function(sample) rnorm(73, 9, 2)),
                                      sapply(1:25, function(sample) rnorm(25, 14, 2)),
                                      sapply(1:25, function(sample) rnorm(2, 14, 2))))

 newerResult <- runTests(newDataset, classes, "Ovarian Cancer Updated", "Differential Expression",
                        resamples = 2, fold = 2,
                    params = list(SelectParams(previousSelection, intermediate = ".iteration",
                                               classifyResult = result),
                        TrainParams(dlda, TRUE, FALSE),
                    PredictParams(predict, TRUE, getClasses = function(result) result[["class"]])))
  }
```

`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", "s
                           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", "se
                      columnVariable = c("classificationName", "datasetName", "validation", "selecti
                           yMax = 100, fontSizes = c(24, 16, 12, 12, 12, 16),
                      title = if(comparison[1] == "within") "Feature Ranking Stability" else "Feature
                           xLabelPositions = seq(10, 100, 10),
                      yLabel = if(is.null(referenceLevel)) "Average Common Features (%)" else paste("
                           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, selction algorithm and dataset. 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 dataset, 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 dataset. If comparison is "datasetName", then the feature rankings are compared across different datasets, for each level of "classificationName", "selectionName" and "validation". The result will inform how stable the feature rankings are between different datasets, for every classification algorithm and every dataset. 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),
                         label = rep(c("Healthy", "Cancer"), each = 50))
 actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
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], LETTERS[10
                          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("fold", 2, 2))

 predicted[, "label"] <- sample(predicted[, "label"])
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], LETTERS[1
                          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("fold", 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", list(oneRanking), list(oneRanking[1:5]))
```

```
otherResult <- SelectResult("Example", "Another Method", 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 100, 200, 300, 400, 500. 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` Either `"balanced"` or one of the options provided by `performance`.

`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 `classifier`.

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

Author(s)

Dario Strbenac

Examples

```
ResubstituteParams(nFeatures = seq(25, 1000, 25), performanceType = "err", 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", "validat
                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 F
                    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, 0.8
                  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,
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], list(1
                          predicted, actual, list("fold", 2, 1))
 predicted[[1]][, "score"][c(2, 6)] <- c(0.60, 0.40)
result2 <- ClassifyResult("Example", "Differential Variability", "F-test", LETTERS[1:10], LETTERS[10:1], list(
                          predicted, actual, validation = list("fold", 2, 1))
ROCplot(list(result1, result2), lineColourVariable = "classificationName", plotTitle = "Ovarian Cancer ROC")
```

`runTest` *Perform a Single Classification*

Description

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

Usage

```
  ## S4 method for signature 'matrix'
runTest(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
runTest(expression, datasetName, classificationName,
                training, testing, params = list(SelectParams(), TrainParams(), PredictParams()),
                    verbose = 1, .iteration = NULL)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the training data. For a matrix, the rows are features, and the columns are samples.
`classes`	A vector of class labels.
`datasetName`	A name associated with the dataset 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.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`verbose`	A number between 0 and 3 for the amount of progress messages to give. A higher number will produce more 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 does cross validation and uses this function. `datasetName` and `classificationName` need to be provided.

Value

A named list with five elements. The first element contains all of the features, ranked from most important to least important. The second element contains the indices of genes that were selected by the feature selection step. The third element contains the indices of the samples that were in the test set. The fourth element contains a vector of the classes predicted by the classifer. The fifth element contains the value of any tuning parameters tried and chosen.

Author(s)

Dario Strbenac

Examples

```
if(require(curatedOvarianData) && require(sparsediscrim))
{
  data(TCGA_eset)
 badOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "deceased" & pData(TCGA_eset)[, "days_to_death"] <=
 goodOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "living" & pData(TCGA_eset)[, "days_to_death"] >=
  TCGA_eset <- TCGA_eset[, c(badOutcome, goodOutcome)]
 classes <- factor(rep(c("Poor", "Good"), c(length(badOutcome), length(goodOutcome))))
  pData(TCGA_eset)[, "class"] <- classes
  runTest(TCGA_eset, "Ovarian Cancer", "Differential Expression",
          training = (1:ncol(TCGA_eset)) %% 2 == 0,
          testing = (1:ncol(TCGA_eset)) %% 2 != 0)
}
```

---

runTests *Reproducibly Do Resampling or Leave Out and Cross Validation*

Description

Enables doing classification schemes such as 100 resamples 5-fold cross validation or leave one out cross validaion. Processing in parallel is possible by leveraging the package BiocParallel.

Usage

```
  ## S4 method for signature 'matrix'
runTests(expression, classes, ...)
  ## S4 method for signature 'ExpressionSet'
runTests(expression, datasetName, classificationName,
                 validation = c("bootstrap", "leaveOut"), bootMode = c("fold", "split"),
                 resamples = 100, percent = 25, folds = 5, leave = 2, seed, parallelParams = bpparam()
                  params = list(SelectParams(), TrainParams(), PredictParams()),
                     verbose = 1)
```

Arguments

expression	Either a matrix or ExpressionSet containing the training data. For a matrix, the rows are features, and the columns are samples.
classes	A vector the same length as the number of columns of expression data specifying the class that the samples belong to.
datasetName	A name associated with the dataset used.
classificationName	A name associated with the classification.
validation	"bootstrap" for repeated resampling or "leaveOut" for leaving all combinations of k samples as test samples.
bootMode	Character. Either "fold" or "split". 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. One is used as the training set, the other is the test set.
resamples	Relevant when repeated resampling is used. The number of times to do sampling with replacement.
percent	Used when bootstrap resampling with split method is chosen. The percentage of samples to be in the test set.
folds	Relevant when repeated resampling is used with fold mode. The number of folds to break each resampling into. Each fold is used once as the test set.
leave	Relevant when leave k out 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.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`verbose`	A number between 0 and 3 for the amount of progress messages to give. A higher number will produce more 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(curatedOvarianData) && require(sparsediscrim))
{
  data(TCGA_eset)
 badOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "deceased" & pData(TCGA_eset)[, "days_to_death"] <=
 goodOutcome <- which(pData(TCGA_eset)[, "vital_status"] == "living" & pData(TCGA_eset)[, "days_to_death"] >=
  TCGA_eset <- TCGA_eset[, c(badOutcome, goodOutcome)]
 classes <- factor(rep(c("Poor", "Good"), c(length(badOutcome), length(goodOutcome))))
  pData(TCGA_eset)[, "class"] <- classes
 runTests(TCGA_eset, "Ovarian Cancer", "Differential Expression", resamples = 2, fold = 2)
}
```

`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", "s
                   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", "selectionNam
                   boxLineColours = NULL,
               rowVariable = c("None", "validation", "datasetName", "classificationName", "selection
               columnVariable = c("datasetName", "classificationName", "validation", "selectionName"
                   yMax = 100, fontSizes = c(24, 16, 12, 16),
               title = if(comparison[1] == "within") "Feature Selection Stability" else if(comparisor
                   xLabel = "Analysis",
               yLabel = if(is.null(referenceLevel) && comparison != "size") "Common Features (%)" els
                  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, selction algorithm and dataset. 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 dataset, 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 dataset. If `comparison` is `"datasetName"`, then the feature selections are compared across different datasets, for each level of `"classificationName"`, `"selectionName"`, and `"validation"`. The result will inform how stable the feature selections are between different datasets, for every classification algorithm and every dataset. 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),
                          label = rep(c("Healthy", "Cancer"), each = 50))
 actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
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], LETTERS[10
                            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("fold", 2, 2))

 predicted[, "label"] <- sample(predicted[, "label"])
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], LETTERS[1
                            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("fold", 2, 2))

selectionPlot(list(result1, result2), xVariable = "classificationName", xLabel = "Analysis", columnVariable = "

selectionPlot(list(result1, result2), comparison = "size", xVariable = "classificationName", xLabel = "Analysis
 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", list(oneRanking), list(oneRanking[1:5]))
otherResult <- SelectResult("Example", "Another Method", list(otherRanking), list(otherRanking[1:2]))

selectionPlot(list(oneResult, otherResult), comparison = "selectionName", xVariable = "selectionName", xLabel =
```

`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 100, 200, 300, 400, 500 features, and picks the number of features with the best resubstitution 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),` 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 `ExpressionSet` object. The function's return value must be a vector of row indices of genes that were selected.
- `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.
- `subsetExpressionData` Whether to subset the expression data, after 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(nearestShrunkenCentroidSelectionInterface, datasetName = "Ovarian Cancer",
               intermediate = "trained", subsetExpressionData = FALSE)
}
```

`SelectResult` *Container for Storing Feature Selection Results*

Description

Contains the ranked indices or names of features, from most discriminative to least discriminative and a list of indicies of feature selected for use in classification. This class is not intended to be created by the user, but could be used in another package.

Constructor

`SelectResult(datasetName, selectionName, rankedFeatures, chosenFeatures`

`datasetName`	A name associated with the dataset used.
`selectionName`	A name associated with the classification.
`rankedFeatures`	Indices or names of all features, from most to least discriminative.
`chosenFeatures`	Indices or names of features 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("Melanoma", "Moderated t-test", list(1:50), list(1:10))
```

`subtractFromLocation` *Subtract All Feature Measurements from Location*

Description

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

Usage

```
  ## S4 method for signature 'matrix'
subtractFromLocation(expression, ...)
  ## S4 method for signature 'ExpressionSet'
subtractFromLocation(expression, training, location = c("mean", "median"),
                                  absolute = TRUE, verbose = 3)
```

Arguments

`expression`	Either a `matrix` or `ExpressionSet` containing the data. For a matrix, the rows are features, and the columns are samples.
`...`	Unused variables from the `matrix` method passed to the `ExpressionSet` method.
`training`	A vector specifying which samples are in the training set.
`location`	Character. Either "mean" or "median".
`absolute`	If `TRUE`, then absolute values of the differences are returned.
`verbose`	A number between 0 and 3 for the amount of progress messages to give. A higher number will produce more messages.

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

An `ExpressionSet` of the same dimension that was input, with values subtracted from the location specified.

Author(s)

Dario Strbenac

Examples

```
subtractFromLocation(matrix(1:100, ncol = 10), training = 1:5, "median")
```

`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, transposeExpression, doesTests, ...)` 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 `matrix` object. The second argument must be a vector of classes. The third argument must be `verbose`. If doesTests is `TRUE`, the third argument must be a `matrix` of test data and the fourth argument is `verbose`. The function's return value can be either a trained classifier when `doesTests` is `FALSE` or a vector of class predictions if `doesTests` is `TRUE`.
- `transposeExpression` Set to `TRUE` if `classifier` expects features as columns.
- `doesTests` Set to `TRUE` if `classifier` also performs and returns predictions.
- `intermediate` Character vector. Names of any variables created in prior stages by `runTest` that need to be passed to `classifier`.
- `...` Other named parameters which will be used by the classifier.

Author(s)

Dario Strbenac

Examples

```
if(require(sparsediscrim))
  trainParams <- TrainParams(dlda, transposeExpression = TRUE, doesTests = FALSE)
# sparsediscrim has a separate predict method for trained DLDA objects.
# dlda expects features in columns, and samples in rows.
```

`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 an `ExpressionSet` 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