Package 'ClassifyR'

April 10, 2023

Type Package

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

Version 3.2.7

Date 2022-02-03

Author Dario Strbenac, Ellis Patrick, Sourish Iyengar, Harry Robertson, Andy Tran, John Ormerod, Graham Mann, Jean Yang

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

VignetteBuilder knitr

Encoding UTF-8

biocViews Classification, Survival

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

Imports grid, genefilter, utils, dplyr, tidyr, rlang, ranger

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

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

License GPL-3

RoxygenNote 7.2.2

NeedsCompilation yes

Collate 'ROCplot.R' 'available.R' 'classes.R' 'calcPerformance.R'
'constants.R' 'crossValidate.R' 'data.R' 'distribution.R'
'edgesToHubNetworks.R' 'featureSetSummary.R'
'getLocationsAndScales.R' 'interactorDifferences.R'
'interfaceClassify.R' 'interfaceCoxPH.R' 'interfaceCoxnet.R'
'interfaceDLDA.R' 'interfaceElasticNetGLM.R'
'interfaceFisherDiscriminant.R' 'interfaceGLM.R'
'interfaceKNN.R' 'interfaceKTSPclassifier.R' 'interfaceMerge.R'
'interfaceMixModels.R' 'interfaceNSC.R'
'interfaceNaiveBayesKernel.R' 'interfacePCA.R'
'interfacePrevalidation.R' 'interfaceRandomForest.R'
'interfaceRandomForestSurvival.R' 'interfaceSVM.R'
'interfaceXGB.R' 'performancePlot.R' 'plotFeatureClasses.R'
'prepareData.R' 'previousSelection.R' 'previousTrained.R'
'rankingBartlett.R' 'rankingCoxPH.R' 'rankingDMD.R'
'rankingDifferentMeans.R' 'rankingEdgeR.R'
'rankingKolmogorovSmirnov.R' 'rankingKullbackLeibler.R'
'rankingLevene.R' 'rankingLikelihoodRatio.R' 'rankingLimma.R'
'rankingPairsDifferences.R' 'rankingPlot.R' 'runTest.R'
'runTests.R' 'samplesMetricMap.R' 'selectMulti.R'
'selectionPlot.R' 'simpleParams.R' 'subtractFromLocation.R'
'utilities.R'

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

git_branch RELEASE_3_16

git_last_commit 04d9a3e

git_last_commit_date 2023-02-03

Date/Publication 2023-04-10

R topics documented:

`asthma` *Asthma RNA Abundance and Patient Classes*

Description

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

Format

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

Source

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

`available` *List Available Feature Selection and Classification Approaches*

Description

Prints a list of keywords to use with `crossValidate`

Usage

```
available(what = c("classifier", "selectionMethod", "multiViewMethod"))
```

Arguments

`what`	Default: `"classifier"`. Either `"classifier"`, `"selectionMethod"` or `"multiViewMethod"`.

Author(s)

Dario Strbenac

Examples

```
available()
```

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

Description

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

Usage

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

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

## S4 method for signature 'factor,tabular'
calcExternalPerformance(
  actualOutcome,
  predictedOutcome,
  performanceType = "AUC"
)

## S4 method for signature 'ClassifyResult'
calcCVperformance(
```

```
  result,
  performanceType = c("Balanced Accuracy", "Balanced Error", "Error", "Accuracy",
   "Sample Error", "Sample Accuracy", "Micro Precision", "Micro Recall", "Micro F1",
   "Macro Precision", "Macro Recall", "Macro F1", "Matthews Correlation Coefficient",
    "AUC", "C-index", "Sample C-index")
)
```

Arguments

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

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

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

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

`result` An object of class `ClassifyResult`.

Details

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

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

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

Value

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

Author(s)

Dario Strbenac

Examples

```
predictTable <- DataFrame(sample = paste("A", 1:10, sep = ''),
                          class = factor(sample(LETTERS[1:2], 50, replace = TRUE)))
actual <- factor(sample(LETTERS[1:2], 10, replace = TRUE))
result <- ClassifyResult(DataFrame(characteristic = "Data Set", value = "Example"),
                paste("A", 1:10, sep = ''), paste("Gene", 1:50), list(paste("Gene", 1:50), paste("Gene", 1:50)),
                         list(function(oracle){}), NULL, predictTable, actual)
result <- calcCVperformance(result)
performance(result)
```

ClassifyResult *Container for Storing Classification Results*

Description

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

Constructor

```
ClassifyResult(characteristics, originalNames, originalFeatures,
          rankedFeatures, chosenFeatures, models, tunedParameters, predictions, actualOutcome, importa
```

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

`originalNames` All sample names.

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

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

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

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

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

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

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

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

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

Summary

`result` is a `ClassifyResult` object.

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

Accessors

`result` is a `ClassifyResult` object.

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

`actualOutcome(result)` Returns the known outcome of each sample.

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

`finalModel(result)` A deployable model fitted on all of the data for use on future data.

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

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

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

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

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

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  data(asthma)
  classified <- crossValidate(measurements, classes, nRepeats = 5)
  class(classified)
#}
```

`colCoxTests` *A function to perform fast or standard Cox proportional hazard model tests.*

Description

A function to perform fast or standard Cox proportional hazard model tests.

Usage

```
colCoxTests(measurements, outcome, option = c("fast", "slow"), ...)
```

Arguments

`measurements`	matrix with variables as columns.
`outcome`	matrix with first column as time and second column as event.
`option`	Default: `"fast"`. Whether to use the fast or slow method.
`...`	Not currently used.

Value

CrossValParams object

Examples

```
data(asthma)
time <- rpois(nrow(measurements), 100)
status <- sample(c(0,1), nrow(measurements), replace = TRUE)
outcome <- cbind(time, status)
output <- colCoxTests(measurements, outcome, "fast")
```

crossValidate *Cross-validation to evaluate classification performance.*

Description

This function has been designed to facilitate the comparison of classification methods using cross-validation. A selection of typical comparisons are implemented. The `train` function is a convenience method for training on one data set and predicting on an independent validation data set.

Usage

```
crossValidate(measurements, outcome, ...)

## S4 method for signature 'DataFrame'
crossValidate(
  measurements,
  outcome,
  nFeatures = 20,
  selectionMethod = "auto",
  selectionOptimisation = "Resubstitution",
  performanceType = "auto",
  classifier = "auto",
  multiViewMethod = "none",
  assayCombinations = "all",
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL,
  ...
)

## S4 method for signature 'MultiAssayExperiment'
crossValidate(
  measurements,
  outcome,
  nFeatures = 20,
  selectionMethod = "auto",
  selectionOptimisation = "Resubstitution",
  performanceType = "auto",
  classifier = "auto",
  multiViewMethod = "none",
  assayCombinations = "all",
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL,
  ...
```

```
)

## S4 method for signature 'data.frame'
crossValidate(
  measurements,
  outcome,
  nFeatures = 20,
  selectionMethod = "auto",
  selectionOptimisation = "Resubstitution",
  performanceType = "auto",
  classifier = "auto",
  multiViewMethod = "none",
  assayCombinations = "all",
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL,
  ...
)

## S4 method for signature 'matrix'
crossValidate(
  measurements,
  outcome,
  nFeatures = 20,
  selectionMethod = "auto",
  selectionOptimisation = "Resubstitution",
  performanceType = "auto",
  classifier = "auto",
  multiViewMethod = "none",
  assayCombinations = "all",
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL,
  ...
)

## S4 method for signature 'list'
crossValidate(
  measurements,
  outcome,
  nFeatures = 20,
  selectionMethod = "auto",
  selectionOptimisation = "Resubstitution",
  performanceType = "auto",
  classifier = "auto",
  multiViewMethod = "none",
```

```
  assayCombinations = "all",
  nFolds = 5,
  nRepeats = 20,
  nCores = 1,
  characteristicsLabel = NULL,
  ...
)

## S3 method for class 'matrix'
train(x, outcomeTrain, ...)

## S3 method for class 'data.frame'
train(x, outcomeTrain, ...)

## S3 method for class 'DataFrame'
train(
  x,
  outcomeTrain,
  selectionMethod = "auto",
  nFeatures = 20,
  classifier = "auto",
  performanceType = "auto",
  multiViewMethod = "none",
  assayIDs = "all",
  ...
)

## S3 method for class 'list'
train(x, outcomeTrain, ...)

## S3 method for class 'MultiAssayExperiment'
train(x, outcome, ...)

## S3 method for class 'trainedByClassifyR'
predict(object, newData, ...)
```

Arguments

measurements	Either a DataFrame, data.frame, matrix, MultiAssayExperiment or a list of these objects containing the data.
outcome	A vector of class labels of class factor of the same length as the number of samples in measurements or a character vector of length 1 containing the column name in measurements if it is a DataFrame. Or a Surv object or a character vector of length 2 or 3 specifying the time and event columns in measurements for survival outcome. If measurements is a MultiAssayExperiment, the column name(s) in colData(measurements) representing the outcome. If column names of survival information, time must be in first column and event status in the second.

`...` Parameters passed into `prepareData` which control subsetting and filtering of input data.

`nFeatures` The number of features to be used for classification. If this is a single number, the same number of features will be used for all comparisons or assays. If a numeric vector these will be optimised over using `selectionOptimisation`. If a named vector with the same names of multiple assays, a different number of features will be used for each assay. If a named list of vectors, the respective number of features will be optimised over. Set to NULL or "all" if all features should be used.

`selectionMethod`
Default: `"auto"`. A character vector of feature selection methods to compare. If a named character vector with names corresponding to different assays, and performing multiview classification, the respective classification methods will be used on each assay. If `"auto"`, t-test (two categories) / F-test (three or more categories) ranking and top `nFeatures` optimisation is done. Otherwise, the ranking method is per-feature Cox proportional hazards p-value.

`selectionOptimisation`
A character of "Resubstitution", "Nested CV" or "none" specifying the approach used to optimise `nFeatures`.

`performanceType`
Performance metric to optimise if classifier has any tuning parameters.

`classifier` Default: `"auto"`. A character vector of classification methods to compare. If a named character vector with names corresponding to different assays, and performing multiview classification, the respective classification methods will be used on each assay. If `"auto"`, then a random forest is used for a classification task or Cox proportional hazards model for a survival task.

`multiViewMethod`
A character vector specifying the multiview method or data integration approach to use.

`assayCombinations`
A character vector or list of character vectors proposing the assays or, in the case of a list, combination of assays to use with each element being a vector of assays to combine. Special value `"all"` means all possible subsets of assays.

`nFolds` A numeric specifying the number of folds to use for cross-validation.

`nRepeats` A numeric specifying the the number of repeats or permutations to use for cross-validation.

`nCores` A numeric specifying the number of cores used if the user wants to use parallelisation.

`characteristicsLabel`
A character specifying an additional label for the cross-validation run.

`x` Same as `measurements` but only training samples.

`outcomeTrain` For the `train` function, either a factor vector of classes, a `Surv` object, or a character string, or vector of such strings, containing column name(s) of column(s) containing either classes or time and event information about survival. If column names of survival information, time must be in first column and event status in the second.

assayIDs — A character vector for assays to train with. Special value "all" uses all assays in the input object.

object — A fitted model or a list of such models.

newData — For the predict function, an object of type matrix, data.frame DataFrame, list (of matrices or data frames) or MultiAssayExperiment containing the data to make predictions with with either a fitted model created by train or the final model stored in a ClassifyResult object.

Details

classifier can be any a keyword for any of the implemented approaches as shown by available().
selectionMethod can be a keyword for any of the implemented approaches as shown by available("selectionMethod").
multiViewMethod can be a keyword for any of the implemented approaches as shown by available("multiViewMethod").

Value

An object of class ClassifyResult

Examples

```
data(asthma)

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


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

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

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


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

CrossValParams *Parameters for Cross-validation Specification*

Description

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

Usage

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

Arguments

samplesSplits	Default: "Permute k-Fold". A character value specifying what kind of sample splitting to do.
permutations	Default: 100. Number of times to permute the data set before it is split into training and test sets. Only relevant if samplesSplits is either "Permute k-Fold" or "Permute Percentage Split".
percentTest	The percentage of the data set to assign to the test set, with the remainder of the samples belonging to the training set. Only relevant if samplesSplits is "Permute Percentage Split".
folds	The number of approximately equal-sized folds to partition the samples into. Only relevant if samplesSplits is "Permute k-Fold" or "k-Fold".
leave	The number of samples to generate all possible combination of and use as the test set. Only relevant if samplesSplits is "Leave-k-Out". If set to 1, it is the traditional leave-one-out cross-validation, sometimes written as LOOCV.
tuneMode	Default: Resubstitution. The scheme to use for selecting any tuning parameters.
adaptiveResamplingDelta	Default: NULL. If not null, adaptive resampling of training samples is performed and this number is the difference in consecutive iterations that the class probability or risk of all samples must change less than for the iterative process to stop. 0.01 was used in the original publication.
parallelParams	An instance of BiocParallelParam specifying the kind of parallelisation to use. Default is to use two cores less than the total number of cores the computer has, if it has four or more cores, otherwise one core, as is the default of bpparam. To

make results fully reproducible, please choose a specific back-end depending on your operating system and also set RNGseed to a number.

Author(s)

Dario Strbenac

Examples

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

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

Description

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

Arguments

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

Value

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

Author(s)

Dario Strbenac

Examples

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

---

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

Description

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

Usage

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

Arguments

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

Value

An object of type `FeatureSetCollection`.

Author(s)

Dario Strbenac

References

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

Examples

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

edgesToHubNetworks(edges, minCardinality = 4)
```

FeatureSetCollection-class

Container for Storing A Collection of Sets

Description

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

Constructor

`FeatureSetCollection(sets)`

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

Summary

`featureSets` is a `FeatureSetCollection` object.

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

`length(featureSets)`: Prints how many sets of features there are.

Subsetting

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

`featureSets[i:j]`: Reduces the object to a subset of the feature sets between elements `i` and `j` of the collection.

`featureSets[[i]]`: Extract the feature set identified by `i`. `i` may be a numeric index or the character name of a feature set.

Author(s)

Dario Strbenac

Examples

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

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

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

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

Description

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

Usage

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

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

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

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing the training data. For a `matrix`, the rows are samples, and the columns are features. If of type `DataFrame` or `MultiAssayExperiment`, the data set is subset to only those features of type `numeric`.
`location`	Default: The median. The type of location to summarise a set of features belonging to a feature set by.

`featureSets` An object of type `FeatureSetCollection` which defines the feature sets.

`minimumOverlapPercent`
The minimum percentage of overlapping features between the data set and a feature set defined in `featureSets` for that feature set to not be discarded from the anaylsis.

`verbose` Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.

`target` If the input is a `MultiAssayExperiment`, this specifies which data set will be transformed. Can either be an integer index or a character string specifying the name of the table. Must have length 1.

Details

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

Value

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

Author(s)

Dario Strbenac

References

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

Examples

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

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

featureSetSummary(genesMatrix, featureSets = featureSets)
```

`HuRI` *Human Reference Interactome*

Description

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

Format

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

Source

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

---

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

Description

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

Usage

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

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

## S4 method for signature 'MultiAssayExperiment'
interactorDifferences(measurements, useFeatures = "all", ...)
```

Arguments

measurements	Either a matrix, DataFrame or MultiAssayExperiment containing the training data. For a matrix, the rows are samples, and the columns are features.
...	Variables not used by the matrix nor the MultiAssayExperiment method which are passed into and used by the DataFrame method.
featurePairs	A object of type Pairs.
absolute	If TRUE, then the absolute values of the differences are returned.
verbose	Default: 3. A number between 0 and 3 for the amount of progress messages to give. This function only prints progress messages if the value is 3.
useFeatures	If measurements is a MultiAssayExperiment, "all" or a two-column table of features to use. If a table, the first column must have assay names and the second column must have feature names found for that assay. "clinical" is also a valid assay name and refers to the clinical data table.

Details

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

Value

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

Author(s)

Dario Strbenac

References

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

Examples

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

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

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

```
interactorDifferences(measurements, pairs)
```

ModellingParams *Parameters for Data Modelling Specification*

Description

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

Usage

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

Arguments

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

Author(s)

Dario Strbenac

Examples

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

`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 'ClassifyResult'
performancePlot(results, ...)

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

Arguments

`results`	A list of `ClassifyResult` objects.
`...`	Not used by end user.
`metric`	Default: `"auto"`. The name of the performance measure or "auto". If the results are classification then balanced accuracy will be displayed. Otherwise, the results would be survival risk predictions and then C-index will be displayed. This is one of the names printed in the Performance Measures field when a `ClassifyResult` object is printed, or if none are stored, the performance metric will be calculated automatically.

`characteristicsList`
: A named list of characteristics. Each element's name must be one of `"x"`, `"row"`, `"column"`, `"fillColour"`, or `"fillLine"`. The value of each element must be a characteristic name, as stored in the `"characteristic"` column of the results' characteristics table. Only `"x"` is mandatory. It is `"auto"` by default, which will identify a characteristic that has a unique value for each element of `results`.

`aggregate`
: A character vector of the levels of `characteristicsList['x']` to aggregate to a single number by taking the mean. This is particularly meaningful when the cross-validation is leave-k-out, when k is small.

`coloursList`
: A named list of plot aspects and colours for the aspects. No elements are mandatory. If specified, each list element's name must be either `"fillColours"` or `"lineColours"`. If a characteristic is associated to fill or line by `characteristicsList` but this list is empty, a palette of colours will be automatically chosen.

`orderingList`
: An optional named list. Any of the variables specified to `characteristicsList` can be the name of an element of this list and the value of the element is the order in which the factors should be presented in, in case alphabetical sorting is undesirable.

`densityStyle`
: Default: "box". Either `"violin"` for violin plot or `"box"` for box plot.

`yLimits`
: The minimum and maximum value of the performance metric to plot.

`fontSizes`
: A vector of length 4. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the font size of the titles of grouped plots, if any are produced. In other words, when `rowVariable` or `columnVariable` are not `NULL`.

`title`
: An overall title for the plot.

`margin`
: The margin to have around the plot.

`rotate90`
: Logical. IF `TRUE`, the plot is horizontal.

`showLegend`
: If `TRUE`, a legend is plotted next to the plot. If FALSE, it is hidden.

Details

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

Value

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

Author(s)

Dario Strbenac

Examples

```
predicted <- DataFrame(sample = sample(LETTERS[1:10], 80, replace = TRUE),
                       permutation = rep(1:2, each = 40),
                       class = factor(rep(c("Healthy", "Cancer"), 40)))
actual <- factor(rep(c("Healthy", "Cancer"), each = 5))
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validation"),
                 value = c("Example", "t-test", "Differential Expression", "2 Permutations, 2 Folds")),
                 LETTERS[1:10], paste("Gene", 1:100), list(paste("Gene", 1:100), paste("Gene", c(10:1, 11:100)),
                 list(paste("Gene", 1:3), paste("Gene", c(2, 5, 6)), paste("Gene", 1:4), paste("Gene", 5:8)),
                          list(function(oracle){}), NULL, predicted, actual)
result1 <- calcCVperformance(result1, "Macro F1")

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

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

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

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

Description

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

Usage

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

## S4 method for signature 'DataFrame'
plotFeatureClasses(
  measurements,
  classes,
  useFeatures,
```

```
  groupBy = NULL,
  groupingName = NULL,
  whichNumericFeaturePlots = c("both", "density", "stripchart"),
  measurementLimits = NULL,
  lineWidth = 1,
  dotBinWidth = 1,
  xAxisLabel = NULL,
  yAxisLabels = c("Density", "Classes"),
  showXtickLabels = TRUE,
  showYtickLabels = TRUE,
  xLabelPositions = "auto",
  yLabelPositions = "auto",
  fontSizes = c(24, 16, 12, 12, 12),
  colours = c("#3F48CC", "#880015"),
  showAssayName = TRUE,
  plot = TRUE
)

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

Arguments

measurements	A matrix, DataFrame or a MultiAssayExperiment object containing the data. For a matrix, the rows are for features and the columns are for samples. A column with name "class" must be present in the DataFrame stored in the colData slot.
...	Unused variables by the three top-level methods passed to the internal method which generates the plot(s).
classes	Either a vector of class labels of class factor or if the measurements are of class DataFrame a character vector of length 1 containing the column name in measurement is also permitted. Not used if measurements is a MultiAssayExperiment object.
useFeatures	If measurements is a matrix or DataFrame, then a vector of numeric or character indices or the feature identifiers corresponding to the feature(s) to be plotted. If measurements is a MultiAssayExperiment, then a DataFrame of 2 columns must be specified. The first column contains the names of the assays and the second contains the names of the variables, thus each row unambiguously specifies a variable to be plotted.

groupBy If measurements is a DataFrame, then a character vector of length 1, which contains the name of a categorical feature, may be specified. If measurements is a MultiAssayExperiment, then a character vector of length 2, which contains the name of a data table as the first element and the name of a categorical feature as the second element, may be specified. Additionally, the value "clinical" may be used to refer to the column annotation stored in the colData slot of the of the MultiAssayExperiment object. A density plot will have additional lines of different line types for each category. A strip chart plot will have a separate strip chart created for each category and the charts will be drawn in a single column on the graphics device. A parallel plot and bar chart plot will similarly be laid out.

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

whichNumericFeaturePlots
If the feature is a single feature and has numeric measurements, this option specifies which types of plot(s) to draw. The default value is "both", which draws a density plot and also a stip chart below the density plot. Other options are "density" for drawing only a density plot and "stripchart" for drawing only a strip chart.

measurementLimits
The minimum and maximum expression values to plot. Default: NULL. By default, the limits are automatically computed from the data values.

lineWidth Numeric value that alters the line thickness for density plots. Default: 1.

dotBinWidth Numeric value that alters the diameter of dots in the strip chart. Default: 1.

xAxisLabel The axis label for the plot's horizontal axis. Default: NULL.

yAxisLabels A character vector of length 1 or 2. If the feature's measurements are numeric an whichNumericFeaturePlots has the value "both", the first value is the y-axis label for the density plot and the second value is the y-axis label for the strip chart. Otherwise, if the feature's measurements are numeric and only one plot is drawn, then a character vector of length 1 specifies the y-axis label for that particular plot. Ignored if the feature's measurements are categorical.

showXtickLabels
Logical. Default: TRUE. If set to FALSE, the x-axis labels are hidden.

showYtickLabels
Logical. Default: TRUE. If set to FALSE, the y-axis labels are hidden.

xLabelPositions
Either "auto" or a vector of values. The positions of labels on the x-axis. If "auto", the placement of labels is automatically calculated.

yLabelPositions
Either "auto" or a vector of values. The positions of labels on the y-axis. If "auto", the placement of labels is automatically calculated.

fontSizes A vector of length 5. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the size of the legends' titles. The fifth number is the font size of the legend labels.

colours The colours to plot data of each class in. The length of this vector must be as long as the distinct number of classes in the data set.

showAssayName	Logical. Default: TRUE. If TRUE and the data is in a MultiAssayExperiment object, the the name of the table in which the feature is stored in is added to the plot title.
plot	Logical. Default: TRUE. If TRUE, a plot is produced on the current graphics device.
classesColumn	If measurementsTrain is a MultiAssayExperiment, the names of the class column in the table extracted by colData(multiAssayExperiment) that contains each sample's outcome to use for prediction.

Value

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

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

Author(s)

Dario Strbenac

Examples

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



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

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

PredictParams *Parameters for Classifier Prediction*

Description

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

Details

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

Constructor

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

`predictor` A character keyword referring to a registered classifier. See `available` for valid keywords.

`characteristics` A `DataFrame` describing the characteristics of the predictor function used. First column must be named `"charateristic"` and second column must be named `"value"`.

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

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

Summary

`predictParams` is a `PredictParams` object.

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

Author(s)

Dario Strbenac

Examples

```
# For prediction by trained object created by DLDA training function.
predictParams <- PredictParams("DLDA")
```

`prepareData` *Convert Different Data Classes into DataFrame and Filter Features*

Description

Input data could be of matrix, MultiAssayExperiment, or DataFrame format and this function will prepare a DataFrame of features and a vector of outcomes and help to exclude nuisance features such as dates or unique sample identifiers from subsequent modelling.

Usage

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

## S4 method for signature 'data.frame'
prepareData(measurements, outcome, ...)

## S4 method for signature 'DataFrame'
prepareData(
  measurements,
  outcome,
  useFeatures = "all",
  maxMissingProp = 0,
  topNvariance = NULL
)

## S4 method for signature 'MultiAssayExperiment'
prepareData(measurements, outcomeColumns = NULL, useFeatures = "all", ...)
```

Arguments

`measurements`	Either a `matrix`, `DataFrame` or `MultiAssayExperiment` containing all of the data. For a `matrix` or `DataFrame`, the rows are samples, and the columns are features.
`...`	Variables not used by the `matrix` nor the `MultiAssayExperiment` method which are passed into and used by the `DataFrame` method.
`outcome`	Either a factor vector of classes, a `Surv` object, or a character string, or vector of such strings, containing column name(s) of column(s) containing either classes or time and event information about survival. If column names of survival information, time must be in first column and event status in the second.

useFeatures	If measurements is a MultiAssayExperiment, a two-column table of features to use. The first column must have assay names and the second column must have feature names found for that assay. "clinical" is also a valid assay name and refers to the clinical data table. "all" is a special keyword that means all features (passing any other filters) of that assay will be used for modelling. Otherwise, a character vector of feature names to use suffices.
maxMissingProp	Default: 0.0. A proportion less than 1 which is the maximum tolerated proportion of missingness for a feature to be retained for modelling.
topNvariance	Default: NULL. An integer number of most variable features to subset to.
outcomeColumns	If measurements is a MultiAssayExperiment, the names of the column (class) or columns (survival) in the table extracted by colData(data) that contain(s) the each individual's outcome to use for prediction.

Value

A list of length two. The first element is a DataFrame of features and the second element is the outcomes to use for modelling.

Author(s)

Dario Strbenac

---

rankingPlot *Plot Pair-wise Overlap of Ranked Features*

Description

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

Usage

```
## S4 method for signature 'ClassifyResult'
rankingPlot(results, ...)

## S4 method for signature 'list'
rankingPlot(
  results,
  topRanked = seq(10, 100, 10),
  comparison = "within",
```

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

Arguments

`results`	A list of `ClassifyResult` objects.
`...`	Not used by end user.
`topRanked`	A sequence of thresholds of number of the best features to use for overlapping.
`comparison`	Default: within. The aspect of the experimental design to compare. Can be any characteristic that all results share or special value "within" to compared between all pairwise iterations of cross-validation.
`referenceLevel`	The level of the comparison factor to use as the reference to compare each non-reference level to. If `NULL`, then each level has the average pairwise overlap calculated to all other levels.
`characteristicsList`	A named list of characteristics. The name must be one of `"lineColour"`, `"pointType"`, `"row"` or `"column"`. The value of each element must be a characteristic name, as stored in the `"characteristic"` column of the results' characteristics table.
`orderingList`	An optional named list. Any of the variables specified to `characteristicsList` can be the name of an element of this list and the value of the element is the order in which the factor should be presented in.
`sizesList`	Default: `lineWidth = 1, pointSize = 2,legendLinesPointsSize = 1, fonts = c(24, 16, 12, 12, 12, 16)`. A list which must contain elements named `lineWidth`, `pointSize`, `legendLinesPointsSize` and `fonts`. The first three specify the size of lines and points in the graph, as well as in the plot legend. `fonts` is a vector of length 6. The first element is the size of the title text. The second element is the size of the axes titles. The third element is the size of the axes values. The fourth element is the size of the legends' titles. The fifth element is the font size of the legend labels. The sixth element is the font size of the titles of grouped plots, if any are produced. Each list element must numeric.
`lineColours`	A vector of colours for different levels of the line colouring parameter, if one is specified by `characteristicsList[["lineColour"]]`. If none are specified

	but, `characteristicsList[["lineColour"]]` is, an automatically-generated palette will be used.
`xLabelPositions`	Locations where to put labels on the x-axis.
`yMax`	The maximum value of the percentage to plot.
`title`	An overall title for the plot.
`yLabel`	Label to be used for the y-axis of overlap percentages.
`margin`	The margin to have around the plot.
`showLegend`	If `TRUE`, a legend is plotted next to the plot. If FALSE, it is hidden.
`parallelParams`	An object of class `MulticoreParam` or `SnowParam`.

Details

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

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

Value

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

Author(s)

Dario Strbenac

Examples

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

predicted[, "class"] <- sample(predicted[, "class"])
rankList <- list(allFeatures[1:100], allFeatures[c(sample(20), 21:100)],
```

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

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

ROCplot *Plot Receiver Operating Curve Graphs for Classification Results*

Description

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

Usage

```
## S4 method for signature 'ClassifyResult'
ROCplot(results, ...)

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

Arguments

results	A list of ClassifyResult objects.
...	Parameters not used by the ClassifyResult method but passed to the list method.

mode	Default: "merge". Whether to merge all predictions of all iterations of cross-validation into one set or keep them separate. Keeping them separate will cause separate ROC curves to be computed for each iteration and confidence intervals to be drawn with the solid line being the averaged ROC curve.
interval	Default: 95 (percent). The percent confidence interval to draw around the averaged ROC curve, if mode is "each".
comparison	Default: "auto". The aspect of the experimental design to compare. Can be any characteristic that all results share. If the data set has two classes, then the slot name with factor levels to be used for colouring the lines. Otherwise, it specifies the variable used for plot facetting.
lineColours	Default: "auto". A vector of colours for different levels of the comparison parameter, or if there are three or more classes, the classes. If "auto", a default colour palette is automatically generated.
lineWidth	A single number controlling the thickness of lines drawn.
fontSizes	A vector of length 5. The first number is the size of the title. The second number is the size of the axes titles and AUC text, if it is not part of the legend. The third number is the size of the axes values. The fourth number is the size of the legends' titles. The fifth number is the font size of the legend labels.
labelPositions	Default: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0. Locations where to put labels on the x and y axes.
plotTitle	An overall title for the plot.
legendTitle	A default name is used if the value is NULL. Otherwise a character name can be provided.
xLabel	Label to be used for the x-axis of false positive rate.
yLabel	Label to be used for the y-axis of true positive rate.
showAUC	Logical. If TRUE, the AUC value of each result is added to its legend text.

Details

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

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

Value

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

Author(s)

Dario Strbenac

Examples

```
predicted <- do.call(rbind, list(DataFrame(data.frame(sample = LETTERS[seq(1, 20, 2)],
                  Healthy = c(0.89, 0.68, 0.53, 0.76, 0.13, 0.20, 0.60, 0.25, 0.10, 0.30),
                  Cancer = c(0.11, 0.32, 0.47, 0.24, 0.87, 0.80, 0.40, 0.75, 0.90, 0.70),
                            fold = 1)),
                 DataFrame(sample = LETTERS[seq(2, 20, 2)],
                  Healthy = c(0.45, 0.56, 0.33, 0.56, 0.65, 0.33, 0.20, 0.60, 0.40, 0.80),
                  Cancer = c(0.55, 0.44, 0.67, 0.44, 0.35, 0.67, 0.80, 0.40, 0.60, 0.20),
                            fold = 2)))
actual <- factor(c(rep("Healthy", 10), rep("Cancer", 10)), levels = c("Healthy", "Cancer"))
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name", "Cross-v
                          value = c("Melanoma", "t-test", "Random Forest", "2-fold")),
                LETTERS[1:20], paste("Gene", LETTERS[1:10]), list(paste("Gene", LETTERS[1:10]), paste("Gene",
                      list(paste("Gene", LETTERS[1:3]), paste("Gene", LETTERS[1:5])),
                         list(function(oracle){}), NULL, predicted, actual)

predicted[c(2, 6), "Healthy"] <- c(0.40, 0.60)
predicted[c(2, 6), "Cancer"] <- c(0.60, 0.40)
result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name", "Cross-v
                          value = c("Melanoma", "Bartlett Test", "Differential Variability", "2-fold")),
                LETTERS[1:20], paste("Gene", LETTERS[1:10]), list(paste("Gene", LETTERS[1:10]), paste("Gene",
                      list(paste("Gene", LETTERS[1:3]), paste("Gene", LETTERS[1:5])),
                         list(function(oracle){}), NULL, predicted, actual)
ROCplot(list(result1, result2), plotTitle = "Cancer ROC")
```

---

runTest *Perform a Single Classification*

Description

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

Usage

```
## S4 method for signature 'matrix'
runTest(measurementsTrain, outcomeTrain, measurementsTest, outcomeTest, ...)

## S4 method for signature 'DataFrame'
runTest(
  measurementsTrain,
  outcomeTrain,
  measurementsTest,
  outcomeTest,
  crossValParams = CrossValParams(),
  modellingParams = ModellingParams(),
  characteristics = S4Vectors::DataFrame(),
```

```
  ...,
  verbose = 1,
  .iteration = NULL
)

## S4 method for signature 'MultiAssayExperiment'
runTest(measurementsTrain, measurementsTest, outcomeColumns, ...)
```

Arguments

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

Details

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

Value

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

Author(s)

Dario Strbenac

Examples

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

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

---

runTests *Reproducibly Run Various Kinds of Cross-Validation*

Description

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

Usage

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

## S4 method for signature 'DataFrame'
runTests(
  measurements,
  outcome,
  crossValParams = CrossValParams(),
```

```
  modellingParams = ModellingParams(),
  characteristics = S4Vectors::DataFrame(),
  ...,
  verbose = 1
)

## S4 method for signature 'MultiAssayExperiment'
runTests(measurements, outcome, ...)
```

Arguments

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

Value

An object of class ClassifyResult.

Author(s)

Dario Strbenac

Examples

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

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

---

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

Description

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

Usage

```
## S4 method for signature 'ClassifyResult'
samplesMetricMap(results, ...)

## S4 method for signature 'list'
samplesMetricMap(
  results,
  comparison = "auto",
  metric = "auto",
  featureValues = NULL,
  featureName = NULL,
  metricColours = list(c("#FFFFFF", "#CFD1F2", "#9FA3E5", "#6F75D8", "#3F48CC"),
    c("#FFFFFF", "#E1BFC4", "#C37F8A", "#A53F4F", "#880015")),
  classColours = c("#3F48CC", "#880015"),
  groupColours = c("darkgreen", "yellow2"),
  fontSizes = c(24, 16, 12, 12, 12),
  mapHeight = 4,
  title = switch(metric, `Sample Error` = "Error Comparison", `Sample Accuracy` =
    "Accuracy Comparison", `Sample C-index` = "Risk Score Comparison"),
  showLegends = TRUE,
  xAxisLabel = "Sample Name",
  showXtickLabels = TRUE,
  yAxisLabel = "Analysis",
```

```
  showYtickLabels = TRUE,
  legendSize = grid::unit(1, "lines"),
  plot = TRUE
)

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

Arguments

results	A list of ClassifyResult objects. Could also be a matrix of pre-calculated metrics, for backwards compatibility.
...	Parameters not used by the ClassifyResult method that does list-packaging but used by the main list method.
comparison	Default: "auto". The aspect of the experimental design to compare. Can be any characteristic that all results share.
metric	Default: "auto". The name of the performance measure or "auto". If the results are classification then sample accuracy will be displayed. Otherwise, the results would be survival risk predictions and then a sample C-index will be displayed. Valid values are "Sample Error", "Sample Error" or "Sample C-index". If the metric is not stored in the results list, the performance metric will be calculated automatically.
featureValues	If not NULL, can be a named factor or named numeric vector specifying some variable of interest to plot above the heatmap.
featureName	A label describing the information in featureValues. It must be specified if featureValues is.

metricColours	If the outcome is categorical, a list of vectors of colours for metric levels for each class. If the outcome is numeric, such as a risk score, then a single vector of colours for the metric levels for all samples.
classColours	Either a vector of colours for class levels if both classes should have same colour, or a list of length 2, with each component being a vector of the same length. The vector has the colour gradient for each class.
groupColours	A vector of colours for group levels. Only useful if featureValues is not NULL.
fontSizes	A vector of length 5. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the size of the legends' titles. The fifth number is the font size of the legend labels.
mapHeight	Height of the map, relative to the height of the class colour bar.
title	The title to place above the plot.
showLegends	Logical. IF FALSE, the legend is not drawn.
xAxisLabel	The name plotted for the x-axis. NULL suppresses label.
showXtickLabels	Logical. IF FALSE, the x-axis labels are hidden.
yAxisLabel	The name plotted for the y-axis. NULL suppresses label.
showYtickLabels	Logical. IF FALSE, the y-axis labels are hidden.
legendSize	The size of the boxes in the legends.
plot	Logical. IF TRUE, a plot is produced on the current graphics device.
classes	If results is a matrix, this is a factor vector of the same length as the number of columns that results has.

Details

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

Value

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

Author(s)

Dario Strbenac

Examples

```
predicted <- DataFrame(sample = LETTERS[sample(10, 100, replace = TRUE)],
                       class = rep(c("Healthy", "Cancer"), each = 50))
actual <- factor(rep(c("Healthy", "Cancer"), each = 5), levels = c("Healthy", "Cancer"))
features <- sapply(1:100, function(index) paste(sample(LETTERS, 3), collapse = ''))
result1 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
```

```
                                                    "Cross-validation"),
                  value = c("Example", "t-test", "Differential Expression", "2 Permutations, 2 Folds")),
                            LETTERS[1:10], features, list(1:100), list(sample(10, 10)),
                            list(function(oracle){}), NULL, predicted, actual)
 predicted[, "class"] <- sample(predicted[, "class"])
result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                    "Cross-validation"),
                  value = c("Example", "Bartlett Test", "Differential Variability", "2 Permutations, 2 Folds")),
                            LETTERS[1:10], features, list(1:100), list(sample(10, 10)),
                            list(function(oracle){}), NULL, predicted, actual)
 result1 <- calcCVperformance(result1)
 result2 <- calcCVperformance(result2)
 groups <- factor(rep(c("Male", "Female"), length.out = 10))
 names(groups) <- LETTERS[1:10]
 cholesterol <- c(4.0, 5.5, 3.9, 4.9, 5.7, 7.1, 7.9, 8.0, 8.5, 7.2)
 names(cholesterol) <- LETTERS[1:10]

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

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

Description

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

Usage

```
## S4 method for signature 'ClassifyResult'
selectionPlot(results, ...)

## S4 method for signature 'list'
selectionPlot(
  results,
  comparison = "within",
  referenceLevel = NULL,
```

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

Arguments

results	A list of ClassifyResult objects.
...	Not used by end user.
comparison	Default: within. The aspect of the experimental design to compare. Can be any characteristic that all results share or either one of the special values "within" to compare between all pairwise iterations of cross-validation. or "size", to draw a bar chart of the frequency of selected set sizes, or "importance" to plot the variable importance scores of selected variables. "importance" only usable if doImportance was TRUE during cross-validation.
referenceLevel	The level of the comparison factor to use as the reference to compare each non-reference level to. If NULL, then each level has the average pairwise overlap calculated to all other levels.
characteristicsList	A named list of characteristics. Each element's name must be one of "x", "row", "column", "fillColour", or "lineColour". The value of each element must be a characteristic name, as stored in the "characteristic" column of the results' characteristics table. Only "x" is mandatory. It is "auto" by default, which will identify a characteristic that has a unique value for each element of results.
coloursList	A named list of plot aspects and colours for the aspects. No elements are mandatory. If specified, each list element's name must be either "fillColours" or "lineColours". If a characteristic is associated to fill or line by characteristicsList but this list is empty, a palette of colours will be automaticaly chosen.
orderingList	An optional named list. Any of the variables specified to characteristicsList can be the name of an element of this list and the value of the element is the order in which the factors should be presented in, in case alphabetical sorting is undesirable.

binsList	Used only if comparison is "size". A list with elements named "setSizes" and "frequencies" Both elements are mandatory. "setSizes" specifies the bin boundaries for bins of interest of feature selection sizes (e.g. 0, 10, 20, 30). "frequencies" specifies the bin boundaries for the relative frequency percentages to plot (e.g. 0, 20, 40, 60, 80, 100).
yMax	Used only if comparison is not "size". The maximum value of the percentage overlap to plot.
fontSizes	A vector of length 4. The first number is the size of the title. The second number is the size of the axes titles. The third number is the size of the axes values. The fourth number is the font size of the titles of grouped plots, if any are produced. In other words, when rowVariable or columnVariable are not NULL.
title	An overall title for the plot. By default, specifies whether stability or commonality is shown.
yLabel	Label to be used for the y-axis of overlap percentages. By default, specifies whether stability or commonality is shown.
margin	The margin to have around the plot.
rotate90	Logical. If TRUE, the boxplot is horizontal.
showLegend	If TRUE, a legend is plotted next to the plot. If FALSE, it is hidden.
plot	Logical. If TRUE, a plot is produced on the current graphics device.
parallelParams	An object of class MulticoreParam or SnowParam.

Details

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

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

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

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

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

Value

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

Author(s)

Dario Strbenac

Examples

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

predicted[, "class"] <- sample(predicted[, "class"])
rankList <- list(allFeatures[1:100], allFeatures[c(sample(20), 21:100)],
             allFeatures[c(1:9, 11, 10, 12:100)], allFeatures[c(1:50, 60:51, 61:100)])
result2 <- ClassifyResult(DataFrame(characteristic = c("Data Set", "Selection Name", "Classifier Name",
                                                       "Cross-validation"),
                value = c("Melanoma", "t-test", "Diagonal LDA", "2 Permutations, 2 Folds")),
                          LETTERS[1:10], allFeatures, rankList,
                          list(rankList[[1]][1:15], rankList[[2]][1:25],
                               rankList[[3]][1:10], rankList[[4]][1:10]),
                          list(function(oracle){}), NULL,
                          predicted, actual)
cList <- list(x = "Classifier Name", fillColour = "Classifier Name")
selectionPlot(list(result1, result2), characteristicsList = cList)

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

`SelectParams` *Parameters for Feature Selection*

Description

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

Constructor

```
SelectParams(featureRanking, characteristics = DataFrame(), minPresence = 1, intermediate = character
   subsetToSelections = TRUE, tuneParams = list(nFeatures = seq(10, 100, 10), performanceType = "Balan
```

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

featureRanking A character keyword referring to a registered feature ranking function. See available for valid keywords.

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

minPresence If a list of functions was provided, how many of those must a feature have been selected by to be used in classification. 1 is equivalent to a set union and a number the same length as featureSelection is equivalent to set intersection.

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

subsetToSelections Whether to subset the data table(s), after feature selection has been done.

tuneParams A list specifying tuning parameters required during feature selection. The names of the list are the names of the parameters and the vectors are the values of the parameters to try. All possible combinations are generated. Two elements named nFeatures and performanceType are mandatory, to define the performance metric which will be used to select features and how many top-ranked features to try.

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

Summary

selectParams is a SelectParams object.

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

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
#{
  SelectParams("KS")

  # Ensemble feature selection.
  SelectParams(list("Bartlett", "Levene"))
#}
```

TrainParams *Parameters for Classifier Training*

Description

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

Constructor

`TrainParams(classifier, balancing = c("downsample", "upsample", "none"), characteristics = DataFrame(`
`intermediate = character(0), tuneParams = NULL, getFeatures = NULL, ...)`

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

`classifier` A character keyword referring to a registered classifier. See `available` for valid keywords.

`balancing` Default: `"downsample"`. A keyword specifying how to handle class imbalance for data sets with categorical outcome. Valid values are `"downsample"`, `"upsample"` and `"none"`.

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

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

`tuneParams` A list specifying tuning parameters required during feature selection. The names of the list are the names of the parameters and the vectors are the values of the parameters to try. All possible combinations are generated.

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

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

Summary

`trainParams` is a `TrainParams` object.

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

Author(s)

Dario Strbenac

Examples

```
#if(require(sparsediscrim))
  trainParams <- TrainParams("DLDA")
```

`TransformParams` *Parameters for Data Transformation*

Description

Collects and checks necessary parameters required for transformation within CV.

Constructor

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

`transform` A character keyword referring to a registered transformation function. See `available` for valid keywords.

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

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

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

Summary

`transformParams` is a `TransformParams` object.

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

Author(s)

Dario Strbenac

Examples

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

Index