Package: exreport (via r-universe)

October 9, 2024

Title Fast, Reliable and Elegant Reproducible Research

Version 0.4.1

Description Analysis of experimental results and automatic report generation in both interactive HTML and LaTeX. This package ships with a rich interface for data modeling and built in functions for the rapid application of statistical tests and generation of common plots and tables with publish-ready quality.

Depends R (>= 3.1.1)

Imports ggplot2, grDevices, methods, reshape2, stats, tools, utils

License GPL-2

LazyData true

RoxygenNote 5.0.1

Repository https://jacintoarias.r-universe.dev

RemoteUrl https://github.com/jacintoarias/exreport

RemoteRef HEAD

RemoteSha ac61cb5a88a64458ced16b47f28f3b1abc97f0e8

Contents

`expCombine` *Combine two experiments with different outputs*

Description

This fuctions joints two experiments sharing the same configuration of methods, problems and parameters but different outputs. The resulting experiment includes the common rows for both experiments with all the output columns.

Usage

```
expCombine(e1, e2, name = NULL)
```

Arguments

`e1`	First experiment to combine.
`e2`	An second experiment to combine, must share the same config as e1.
`name`	Optional name for the resulting experiment. If not specified the new experiment will be called "e1_name U e2_name"

Value

An new experiment with common rows and all columns.

Examples

```
# In this example we turn the wekaExperiment into two different experiments,
# with different outputs to combine them:

df_acc  <- wekaExperiment[,
           c("method", "problem", "fold", "featureSelection", "accuracy")]
df_time <- wekaExperiment[,
           c("method", "problem", "fold", "featureSelection", "trainingTime")]

exp_acc <- expCreate(df_acc, name="acc", parameter="fold")
exp_time <- expCreate(df_time, name="time", parameter="fold")

# With expCombine we can mix the two experiments:
expCombine(exp_acc, exp_time)
```

`expConcat` *Concatenate rows of matching experiments*

Description

This function concatenates two experiments with the same configuration of parameter an outputs. At least one common output must be present, the rest of them will be removed from the resulting experiment. Different methods and problems can be present.

Usage

```
expConcat(e1, e2, name = NULL, tol = 1e-09)
```

Arguments

`e1`	First experiment object to concat.
`e2`	Second experiment object to concat. Must have the same configuration than e1.
`name`	Optional name, if not provided the new experiment will be called "e1_name + e2_name"
`tol`	Tolerance value for duplicate checking.

Value

An experiment object having all the rows of e1 and e2

Examples

```
# In this example we turn the wekaExperiment into two different experiments,
# with different parameter values to combine them:

df_no  <- wekaExperiment[wekaExperiment$featureSelection=="no",]
df_yes <- wekaExperiment[wekaExperiment$featureSelection=="yes",]
```

```
exp_yes <- expCreate(df_yes, name="fss-yes", parameter="fold")
exp_no <- expCreate(df_no, name="fss-no", parameter="fold")

expConcat(exp_yes, exp_no)
```

`expCreate` *Load data and create an exreport experiment*

Description

This function loads a data.frame, checks its properties and formats an exreport experiment object. The columns of an experiments must contain at least two categorical columns to be identified as the method and problem variables and a thrid numerical column to be identified as an output variable. Additional columns can be added as parameters or additional outputs.

Usage

```
expCreate(data, methods = "method", problems = "problem",
  parameters = c(), respectOrder = FALSE, name, tol = 1e-09)
```

Arguments

`data`	A data.frame object satisfying the experiment format
`methods`	The name of the variable which contains the methods, by default is searches for a column named "method".
`problems`	The name of the variable which contains the problems, by default is searches for a column named "problem".
`parameters`	A list of the columns names to be identified as parameters. By default the remaining categorical columns are identified as parameters, so this list is useful only to identify numeric columns.
`respectOrder`	A logical parameter which indicates if the order of the elements of the method and problem columns must be respected by appearance or ordered alphabeticaly. It affects to the look of data representations.
`name`	A string which will identify the experiment in the report.
`tol`	Tolerance factor to identify repeated experiments for duplicated rows.

Value

A new exreport experiment object.

See Also

expCreateFromTable

Examples

```
# Creates experiment specifying column names and the numerical variables that
# are parameters

expCreate(wekaExperiment,
methods="method",
problems="problem",
parameters="fold",
name="Test Experiment")
```

`expCreateFromTable` *Create an exreport experiment from a tabular representation*

Description

Create an exreport experiment object from a tabular representation. The input data must be a table having methods as rows and problems as columns. The values in such table correspond to a particular output. The resulting experiment can be characterized with static parameters.

Usage

```
expCreateFromTable(data, output, name, parameters = list(),
  respectOrder = FALSE)
```

Arguments

`data`	Input tabular data satisfying the previous constraints.
`output`	String indicating the name of the output that the table values represent.
`name`	A string which will identify the experiment in the report.
`parameters`	A list of strings containing the names and values for the static configuration of the algorithm. The name of each element of the list will correspond with the name of a parameter and the element with the value asigned.
`respectOrder`	A logical parameter which indicates if the order of the elements of the method and problem columns must be respected by appearance or ordered alphabeticaly. It affects to the look of data representations.

Value

A new exreport experiment object.

See Also

expCreate

Examples

```
# We generate a data frame where the methods are rows and the problems columns
# from the wekaExperiment problem. (This is only an example, normally you
# would prefer to load a proper experiment and process it.)

library(reshape2)
df <- dcast(wekaExperiment[wekaExperiment$featureSelection=="no",],
method ~ problem,
value.var="accuracy",
fun.aggregate = mean)

# We can create it and parametrice accordingly:
expCreateFromTable(df, output="accuracy", name="weka")

# Optionally we can set a fixed value for parameters, and ordered by appearance:
expCreateFromTable(df, output="accuracy", name="weka",
parameters=list(featureSelection = "no"), respectOrder=TRUE)
```

---

expExtend *Extend an experiment by adding new parameters*

Description

This function extends an existing exreport experiment object by adding new parameters with fixed values.

Usage

```
expExtend(e, parameters)
```

Arguments

`e`	Input experiment
`parameters`	A list of strings containing the values of the new parameters, the name for each one of them will be given by the name of the corresponding object in the list.

Value

A modified exreport experiment object with additional parameters.

Examples

```
# We load the wekaExperiment problem as an experiment and then add a new param
# with a default value.

experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
expExtend(experiment, list(discretization = "no"))
```

expExtract — *Extract statistically equivalent methods from a multiple comparison test*

Description

This functions generates a new experiment incluing the methods that obtained an equivalent performance with statisticall significance in the multiple comparison test i.e. those whose hypotheses were not rejected

Usage

```
expExtract(ph)
```

Arguments

`ph`	A testMultipleControl test object

Value

an experiment object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a testMultiplePairwise test procedure
test <- testMultipleControl(experiment, "trainingTime", "min")

expExtract(test)
```

expGetDuplicated — *Create a new experiment with only the duplicated rows*

Description

This function computes the duplicated rows attending to the method, problem and input parameters (but not the outputs). The resulting experiment will contain these duplicated rows.

Usage

```
expGetDuplicated(e, tol = 1e-09)
```

Arguments

e	The experiment to check for duplicated rows
tol	The tolerance for numeric values to check if two outputs are numerically equal or not.

Details

If duplicated rows show different outputs the function will launch a a warning message indicating how many of them differ in the outputs from the original row, the extent to what two rows are divergent in their output can be parametrized.

This function is useful to determine the consistency of the experiment, as a measure to sanitice the original data source if needed,

Value

A new experiment containing the duplicated rows

Examples

```
# We duplicate some of the rows of a given experiment:
e <- expCreate(wekaExperiment, parameters="fold", name="Test Experiment")
redundant <- expCreate(wekaExperiment[wekaExperiment$method=="NaiveBayes",],
                       parameters="fold", name="Test Experiment")
e2 <- expConcat(e,redundant)

# Now we check for duplicates:
expGetDuplicated(e2)
```

---

expInstantiate *Instatiate the methods in the experiment for each one of the different parameter configurations.*

Description

When performing statistical tests or summarizing an experiment for a given output variable there can be different parameter configuration for each interaction of method and problem. Once applied the desired transformations this function can be used to remove unary parameters from the experiment or to instantiate the methods for each configuration.

Usage

```
expInstantiate(e, parameters = NULL, removeUnary = TRUE)
```

Arguments

`e`	The experiment object to be instantiated
`parameters`	A vector indicating the parameters to be instantiaded. If NULL or default all parameters would be considered.
`removeUnary`	Boolean value indicating if the unary parameters will be used in an instantiation or if the column can be erased.

Details

If any method is instantiated the cartesian product of the method and the selected parameters is performed and included in the resulting experiment as the methods variable. The name of the corresponding value will indicate the name of the former method and the value of each parameter instantiated.

Value

an experiment object

Examples

```
# Create an experiment from the wekaExperiment
experiment <- expCreate(wekaExperiment, name="test-exp", parameter="fold")

# We would like to reduce the fold parameter by its mean value. It becomes an
# unary parameter.
experiment <- expReduce(experiment, "fold", mean)

# Now we instantiate the experiment by the featureSelection parameter and
# remove the unary fold parameter
expInstantiate(experiment, removeUnary=TRUE)
```

`expReduce` *Reduce a parameter by a function for each method, problem and remaining parameter configuration interaction*

Description

This functions reduces a parameter by aggregating the outputs variables for each value and for each configuration of method, problem and remaining parameters. By default it computes the mean of the variables.

Usage

```
expReduce(e, parameters = NULL, FUN = mean)
```

Arguments

`e`	An input experiment object.
`parameters`	The parameter or parameters to be reduced, if NULL or default all parameters are considered.
`FUN`	The function used to agregate the ouput values

Value

An experiment object.

Examples

```
# Create an experiment from the wekaExperiment
experiment <- expCreate(wekaExperiment, name="test-exp", parameter="fold")

# We would like to reduce the fold parameter by its mean value. This way
expReduce(experiment, "fold", mean)
```

`expRemoveDuplicated` *Remove duplicated rows from an experiment*

Description

This function removes duplicated rows of a given experiment attending to the interaction of methods, problems and parameters (but no outputs).

Usage

```
expRemoveDuplicated(e, tol = 1e-09)
```

Arguments

`e`	The experiment to be analised
`tol`	The tolerance for numeric values to check if two outputs are numerically equal or not.

Details

The duplicated rows found are compared among themselves to determine if there is divergence between the outputs, if the rows are not consistent a warning is raised to note this difference.

Value

an experiment object

Examples

```
# We duplicate some of the rows of a given experiment:
e <- expCreate(wekaExperiment, parameters="fold", name="Test Experiment")
redundant <- expCreate(wekaExperiment[wekaExperiment$method=="NaiveBayes",],
                       parameters="fold", name="Test Experiment")
e2 <- expConcat(e,redundant)

# Now we remove those duplicates:
expRemoveDuplicated(e2)
```

`expRename` *Change the name of elements that an experiment contains*

Description

This function change the name of problems, methods or parameter values that an existing experiment object contains.

Usage

```
expRename(e, elements = list(), name = NULL)
```

Arguments

`e`	Input experiment
`elements`	A list of arrays of strings containing the new names. The old name will be specified as the name of the element in such array, and the name for the parameter, method or problem will be given by the name of the corresponding object in the list. If a name is not present in the set of parameter names or parameter values, it will be ignored.
`name`	The name of the new experiment. If NULL, the previous name will be used.

Value

A modified exreport experiment object with some changes on the name of the elements.

Examples

```
# We load the wekaExperiment problem as an experiment and then change the name
# of one value for the parameter discretization and for one method.

experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
expRename(experiment, list(featureSelection = c("no"="false"),
                           method=c("RandomForest"="RndForest")))
```

expReorder *Change the order of elements that an experiment contains*

Description

This function change the order of problems, methods or parameter values that an existing experiment object contains. The order affects the look of the data representation (as tables and plots).

Usage

```
expReorder(e, elements, placeRestAtEnd = TRUE)
```

Arguments

e	Input experiment
elements	A list of arrays of strings containing the ordered names. The name for the parameter, method or problem will be given by the name of the corresponding object in the list. The names which have not been specified will be placed at the begining or at the end (depending on the parameter placeRestAtEnd). If a name is not present in the set of parameter values, it will be ignored.
placeRestAtEnd	Logical value which indicates if the non specified value names have to be placed after the specified ones (TRUE) or before (FALSE).

Value

A modified exreport experiment object with some changes on the name of the elements.

Examples

```
# We load the wekaExperiment problem as an experiment and then change the order
# of the values for the parameter featureSelection and for one valoue for the method.

experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
expReorder(experiment, list(featureSelection = c("yes","no"),
                            method=c("OneR")))
```

expSubset *Obtains a subset of an experiment matching the given conditions*

Description

This function receives a named list indicating variables and values to filter the input experiment.

Usage

```
expSubset(e, columns, invertSelection = FALSE)
```

Arguments

`e`	The experiment to be subsetted
`columns`	A named list containing the variables to be filtered and the valid values.
`invertSelection`	If the filtering must match the inversion of the specified conditions.

Details

The names of the elements in the list correspond with the variables to be filtered, indicating either the methos or problem variables as well as parameters. The values of the list correspond with the valid states for the filtering.

Value

a filtered experiment object

Examples

```
# We create a new experiment from the wekaExperiment problem
e <- expCreate(wekaExperiment, parameters="fold", name="Test Experiment")

# We can filter the experiment to reduce the number of methods.
e <- expSubset(e, list(method = c("J48", "NaiveBayes")))
e

# We can filter the experiment to remove a given problem
e <- expSubset(e, list(problem = "iris"), invertSelection=TRUE)
e

# We can subset the experiment to obtain a specific parameter configuration
e <- expSubset(e, list("featureSelection" = "no"))
e
```

`exreport` *Create a new exreport document*

Description

This function inits a new exreport document to start adding elements for later rendering.

Usage

```
exreport(title)
```

Arguments

`title`	A string representing a short title for this document

Value

an empty exreport document

See Also

exreportRender, exreportAdd

`exreportAdd` *Add elements to an existing exreport document*

Description

This function allows to add one or more reportable objects to an exisiting exreport document.

Usage

```
exreportAdd(rep, elem)
```

Arguments

`rep`	an exreport object in which the elem will be added
`elem`	a reportable object or a list of them

Value

an extended exreport document

Examples

```
# Create an empty document:
report <- exreport("Test document")

# Create a reportable object (an experiment)
experiment <- expCreate(wekaExperiment, name="test-exp", parameter="fold")

# Add this object to the document
exreportAdd(report, experiment)
```

`exreportRender` *Render an exreport document*

Description

This function renders an existing exreport object to a given file and format.

Usage

```
exreportRender(rep, destination = NULL, target = "html", safeMode = TRUE,
  visualize = TRUE)
```

Arguments

`rep`	The exreport object to be rendered
`destination`	Path to the rendered file. If NULL, it uses a temporary directory
`target`	The format of the target rendering. HTML and PDF are allowed.
`safeMode`	Denies or allows (TRUE or FALSE) output files overwriting
`visualize`	Visualize the generated output or not

Value

an experiment object

`ILsMultiple` *Problem: Comparison between three Ionic Liquids packs to capture and reduce the emission of CO2 in industrial fuel combustion processes.*

Description

A problem containing the percentage of the CO2 reduction in the emission of 20 industrial fuel combustion processes. It has been used three different Ionic Liquids (ILs) pills with different properties. The pills has been reused up to three times, and each experiment has been repeated three times under the same conditions. The variables of the problem are as follows:

Usage

```
data(ILsMultiple)
```

Format

A data frame with the data detailed in the Description.

Details

- IL The name of the IL pills (IL1, IL2 and IL3).
- Scenario The name of the industrial fuel combustion process (from Scenario 1 to Scenario20).
- Execution The number of the execution for each experiment under the same conditions.
- Reused The number of experiments which the IL pill has been used previously (from 0 to 2).
- CO2 The percentage of CO2 which has been reduced from the emission.

`ILsPaired` *Problem: Comparison between two Ionic Liquids packs to capture and reduce the emission of CO2 in industrial fuel combustion processes.*

Description

A problem containing the percentage of the CO2 reduction in the emission of 20 industrial fuel combustion processes. It has been used two different Ionic Liquids (ILs) pills with different properties. The pills has been reused up to three times, and each experiment has been repeated three times under the same conditions. The variables of the problem are as follows:

Usage

```
data(ILsPaired)
```

Format

A data frame with the data detailed in the Description.

Details

- IL The name of the IL pills (IL1 and IL2).
- Scenario The name of the industrial fuel combustion process (from Scenario 1 to Scenario20).
- Execution The number of the execution for each experiment under the same conditions.
- Reused The number of experiments which the IL pill has been used previously (from 0 to 2).
- CO2 The percentage of CO2 which has been reduced from the emission.

`plotCumulativeRank` *Area plot for the rank distribution from a multiple test*

Description

This function builds an area plot from a testMultiple object displaying the cumulative value for each method for all the evaluated problems. The value for the rankings is obtained from the Friedman test independently of the scope of the test (control or pairwise).

Usage

```
plotCumulativeRank(testMultiple, grayscale = FALSE)
```

Arguments

`testMultiple`	Statistical test from which the plot is generated. The rankings are obtained from the Friedman test.
`grayscale`	Configure the plot using a grayscale palette.

Value

an exPlot object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expSubset(experiment, list(featureSelection = "no"))
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a Friedman test included ina a testMultipleControl
# test procedure
test <- testMultipleControl(experiment, "accuracy")

# Finally we obtain the plot
plotCumulativeRank(test)
cat()
```

plotExpSummary *Barplot for summarizing an experiment output variable*

Description

This function builds a barplot for a given experiment output variable, summarizing its distribution according to the different methods and problems. The aspect of the plot can be parametrized in several ways.

Usage

```
plotExpSummary(exp, output, columns = 0, freeScale = FALSE,
  fun = identity, grayscale = FALSE)
```

Arguments

`exp`	- The experiment object to take the data from
`output`	- A string identifying the name of the output variable to be plotted
`columns`	- Integer number, 0 for a wide aspect plot and any other value to include n columns of facets separating the problems.
`freeScale`	- Boolean, if using facets sets the scale of each one independent or not
`fun`	- A function to be applied to the selected output variables before being plotted.
`grayscale`	- Defaulted to False. True for a plot in grayscale.

Details

Please notice that the plot function requires that an unique configuration of parameters is present in the experiment. So the user must have processed and instantiated the experiment before.

Value

an exPlot object

Examples

```
# This example plots the distribution of the trainingTime variable in the
# wekaExperiment problem.

# First we create the experiment from the problem.
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")

# Next we must process it to have an unique parameter configuration:
# We select a value for the parameter featureSelection:
experiment <- expSubset(experiment, list(featureSelection = "yes"))
# Then we reduce the fold parameter:
experiment <- expReduce(experiment, "fold", mean)
# Finally we remove unary parameters by instantiation:
```

```
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Now we can generate several plots:

# Default plot:
plotExpSummary(experiment, "accuracy")

# We can include faceting in the plot by dividing it into columns:
plotExpSummary(experiment, "accuracy", columns=3)

# If we want to show the independent interaction for the output variable
# in each experiment we can make the scales for example, remark the difference
# in :
plotExpSummary(experiment, "trainingTime", columns=3, freeScale=FALSE)
plotExpSummary(experiment, "trainingTime", columns=3, freeScale=TRUE)
```

`plotRankDistribution` *Boxplot for the ranks distribution and control hypotheses from multiple test*

Description

This function generates a boxplot from a testMultiple statistical test showing the ordered distrubution of rankings for each method computed for the Friedman test. If the input test features a control multiple comparison then the rejected hypotheses by the Holm methd are also indicates in the plot.

Usage

```
plotRankDistribution(testMultiple)
```

Arguments

`testMultiple`	The statistical test from which the plot is generated. The functions accepts either control and pairwise multiple tests.

Value

an experiment object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expSubset(experiment, list(featureSelection = "yes"))
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a Friedman test included ina a testMultipleControl
# test procedure
test <- testMultipleControl(experiment, "accuracy")

# Finally we obtain the plot
plotRankDistribution(test)
cat()
```

`tabularExpSummary` *Summarize the experiment with a table for given outputs*

Description

This function generates a table for the given outputs of the experiment, comparing all methods for each one of the problems. In addition the function can highlight the best results for each problem as well as display a range of parameters for the posterior renderization.

Usage

```
tabularExpSummary(exp, outputs, boldfaceColumns = "none", format = "f",
  digits = 4, tableSplit = 1, rowsAsMethod = TRUE)
```

Arguments

`exp`	The ource experiment to generate the table from
`outputs`	A given variable or list of them to be the target of the table
`boldfaceColumns`	Indicate ("none","max" or "min") to highlight the method optimizing the variables for each problem.
`format`	Indicates the format of the numeric output using C formatting styles. Defaults to 'f'
`digits`	The number of decimal digits to include for the numeric output.
`tableSplit`	Indicates the number of paritition of the table that will be rendered. Usefull when the the table is excessivelly wide.
`rowsAsMethod`	Display the methods as the rows of the table, indicate FALSE for a transpose table.

Value

An extabular object

Examples

```
# This example plots the distribution of the trainingTime variable in the
# wekaExperiment problem.

# First we create the experiment from the problem.
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")

# Next we must process it to have an unique parameter configuration:
# We select a value for the parameter featureSelection:
experiment <- expSubset(experiment, list(featureSelection = "yes"))
# Then we reduce the fold parameter:
experiment <- expReduce(experiment, "fold", mean)
# Finally we remove unary parameters by instantiation:
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Generate the default table:
tabularExpSummary(experiment, "accuracy")
```

`tabularTestPairwise` *Display pairwise information about a multiple test between the methods*

Description

This function obtain a pairwise table comparing the methods among themselves for the specified metrics. It takes an testMultiplePairwise object as an input.

Usage

```
tabularTestPairwise(ph, value = "pvalue", charForNAs = "-")
```

Arguments

`ph`	The input testMultiplePairwise object
`value`	Indicates the metric to be displayed ("pvalue", "wtl")
`charForNAs`	Indicates the character included when there is not comparison available

Value

An extabular object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a a testMultiplePairwise test procedure
test <- testMultiplePairwise(experiment, "accuracy", "max")

# Different tables can be obtained by using a range of metrics
tabularTestPairwise(test, "pvalue")

tabularTestPairwise(test, "wtl")
```

`tabularTestSummary` *Summarize the result of a multiple comparison statistical test in a table*

Description

This function builds a table from a testMultiple object, either control or pairwise. The htpotheses are added and compared in the table showing the methods and a range of different metrics than can be added to the table. Also the table shows information about rejected hypotheses.

Usage

```
tabularTestSummary(ph, columns = c("pvalue"))
```

Arguments

`ph`	The input testMultiple from which the table is generated
`columns`	A vector indicating the metrics that will be shown in the table

Value

an extabular object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a a testMultiplePairwise test procedure
test <- testMultipleControl(experiment, "accuracy", "min")

# Different tables can be obtained by using a range of metrics
tabularTestSummary(test, c("pvalue"))

tabularTestSummary(test, c("rank", "pvalue", "wtl"))
```

`testMultipleControl` *Multiple Comparison Statistical Test (Friedman + Control Holm PostHoc)*

Description

This function perfoms a multiple comparison statistical test for the given experiment. First of all it performs a Friedman Test over all methods. In the case this test is rejected, meaning that significant differences are present among the methods a post-hoc test is then executed. For that, a comparison using the best method as a control is performed for each other method, finally a Holm familywise error correction is applied to the resulting p-values.

Usage

```
testMultipleControl(e, output, rankOrder = "max", alpha = 0.05)
```

Arguments

`e`	Input experiment
`output`	The output for which the tet will be performed.
`rankOrder`	The optimization strategy, can be either maximizing "max" or minimizing "min" the target output variable.
`alpha`	The significance level used for the whole testing procedure.

Value

an testMultipleControl object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expSubset(experiment, list(featureSelection = "yes"))
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a testMultiplePairwise test procedure
test <- testMultipleControl(experiment, "accuracy", "max")

summary(test)
```

testMultiplePairwise *Multiple Comparison Statistical Test (Friedman + Pairwise Shaffer PostHoc)*

Description

This function perfoms a multiple comparison statistical test for the given experiment. First of all it performs a Friedman Test over all methods. In the case this test is rejected, meaning that significant differences are present among the methods a post-hoc test is then executed. For that, each pair of methods are compared between each other, and finally a Shaffer familywise error correction is applied to the resulting p-values.

Usage

```
testMultiplePairwise(e, output, rankOrder = "max", alpha = 0.05)
```

Arguments

`e`	Input experiment
`output`	The output for which the tet will be performed.
`rankOrder`	The optimization strategy, can be either maximizing "max" or minimizing "min" the target output variable.
`alpha`	The significance level used for the whole testing procedure.

Value

an testMultiplePairwise object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expReduce(experiment, "fold", mean)
experiment <- expSubset(experiment, list(featureSelection = "yes"))
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a testMultiplePairwise test procedure
test <- testMultiplePairwise(experiment, "accuracy", "max")

summary(test)
```

testPaired *Paired Wilcoxon statistical test*

Description

This function performs a Wilcoxon paired test to compare the methods of an experiment consisting exactly on two of them. If more methods are present, then a multiple comparison test must be applied.

Usage

```
testPaired(e, output, rankOrder = "max", alpha = 0.05)
```

Arguments

`e`	Input experiment
`output`	The output for which the tet will be performed.
`rankOrder`	The optimization strategy, can be either maximizing "max" or minimizing "min" the target output variable.
`alpha`	The significance level used for the whole testing procedure.

Value

a testPaired object

Examples

```
# First we create an experiment from the wekaExperiment problem and prepare
# it to apply the test, we must subset it to only two methods:
experiment <- expCreate(wekaExperiment, name="test", parameter="fold")
experiment <- expSubset(experiment, list(method = c("J48", "NaiveBayes")))
experiment <- expSubset(experiment, list(featureSelection = c("no")))
experiment <- expReduce(experiment, "fold", mean)
experiment <- expInstantiate(experiment, removeUnary=TRUE)

# Then we perform a Wilcoxon test procedure
test <- testPaired(experiment, "accuracy", "max")

summary(test)
```

`wekaExperiment` *Problem: Comparison between several Machine Learning algorithms from the Weka library*

Description

A problem containing experimental data obtaining by comparing several instances of Machine Algorithms from the Weka library. The variables are as follows:

Usage

```
data(wekaExperiment)
```

Format

A data frame with the data detailed in the Description.

Details

- method. Classification algorithms used in the experimen (NaiveBayes, J48, IBk)
- problem. Problems used as benchmark in the comparison, up to 12.
- featureSelection. Boolean parameter indicating if the data was preprocessed
- fold. For each configuration a 10-fold cross validation was performed. This variable is a numeric value ranging from 1 to 10.
- accuracy. This is a measure of the performance of each algorithm. Representing the percentage of correctly classified instances.
- trainingTime. A second measure of performance. This one indicates the time in seconds that took the algorithm to build the model.

Index