Package ‘crtests’

October 12, 2022

Type Package
Title Classification and Regression Tests
Version 0.2.1
Date 2016-05-13
Description Provides wrapper functions for running classification and regression tests using different machine learning techniques, such as Random Forests and decision trees. The package provides standardized methods for preparing data to suit the algorithm's needs, training a model, making predictions, and evaluating results. Also, some functions are provided to run multiple instances of a test.
License GPL-3 | file LICENSE
Imports caret (>= 6.0.41), plyr (>= 1.8.1), stringr (>= 0.6.2)
Suggests testthat, knitr, e1071, randomForest, rpart, adabag, rmarkdown
LazyData true
VignetteBuilder knitr
RoxygenNote 5.0.1
NeedsCompilation no
Author Sjoerd van der Spoel [aut, cre]
Maintainer Sjoerd van der Spoel <r@sjoerdvanderspoel.nl>
Repository CRAN
Date/Publication 2016-05-20 22:36:56

R topics documented:

apply_levels .................................................. 2
argument_match_test ....................................... 3
capitalize_first ............................................ 3
classification_model ........................................ 4
createtest ................................................... 5
create_and_run_test ....................................... 6
apply_levels.

Converts the column factor levels in df to those in df_reference.

Description

Each column in the result is a factor with the values of df and the levels of df_reference. This means that if there are levels in df_to_change that are not in df_reference, NAs will be introduced. The main use of this function is in classifier problems, where the training and the test set need to have equal factors. To work, all names(df_reference) need to be

Usage

apply_levels(df, df_reference)
Argument_match_test

Arguments

- **df**
  - The df that is to be relabeled
- **df_reference**
  - A reference df, which column levels will be applied to df if that column is a factor

Value

A data.frame where all factor’s levels where changed. Through applying new levels, NAs could have been introduced

---

**argument_match_test**  
*Test function with non-matching arguments*

**Description**

Goes through all the parameters of fun which have a vector of default values. It then calls *fun* with a different value.

**Usage**

`argument_match_test(fun, args)`

**Arguments**

- **fun**
  - Function to test
- **args**
  - Complete list of required arguments to *fun*

---

**capitalize_first**  
*Capitalize the first letter of a word*

**Description**

Takes a string, and converts its first letter to upper case

**Usage**

`capitalize_first(word)`

**Arguments**

- **word**
  - A string

**Value**

String with first letter converted to capital
classification_model  
Generic function for creating a classification model

Description
Generic function for creating a classification model

Usage
classification_model(method, test, x, y, training_data, ...)

## Default S3 method:
classification_model(method, test, x, y, training_data, ...)

## S3 method for class 'rpart'
classification_model(method, test, x, y, training_data, ...)

## S3 method for class 'boosting'
classification_model(method, test, x, y, training_data, ...)

Arguments

method  The method for classification.
test  The test being conducted
x  The independent variables.
y  The dependent (class) variable. Should be a factor for most algorithms
training_data  The complete data set for training
...  Extra arguments to pass to the classification algorithm

Value
The produced model

Methods (by class)

- default: Default function for creating a classification model
  gets the method and calls it using x, y, and data
- rpart: Rpart specific function for creating a classification model
  RPart requires a formula for classification, which is not provided by the default function
- boosting: Create a classification model using Freund & Schapire's adaboost.M1
Create a classification or regression test case

Description

Create a test, which can be run using any of the available runtest functions

Usage

createtest(data, problem = c("classification", "regression"), dependent,
    data_transform = identity, train_index, method, name, description = "",
    ...)  

Arguments

data A data frame
problem Either classification or regression. This influences how the algorithms are trained
    and what method is used to determine performance
dependent The dependent variable: the name of the column containing the prediction goal
data_transform A quoted function name that transforms the data. It should maintain it in data
    frame form and maintain the dependent variable.
train_index A vector of the rows to be used as training set. All other rows will form the
    holdout set
method The regression or classification method
name The name of the test. Printed in the test results
description Optional. A more elaborate description of the test
... Extra arguments used while running the test.

Value

An object of class 'classification' or 'regression', which holds the data, method, etc. for executing
the test case.

Examples

data(iris)
# A classification test
test <- createtest(data = iris,
    dependent = "Species",
    problem = "classification",
    method = "randomForest",
    name = "An example classification test",
    train_index = sample(150, 100)
)

# A regression test
test <- createtest(data = iris,
  dependent = "Sepal.Width",
  problem = "regression",
  method = "randomForest",
  name = "An example regression test",
  train_index = sample(150, 100)
)

create_and_run_test     Create test and run it

Description
A convenience function calling createtest first, then runs the test using runtest.

Usage
create_and_run_test(train_index, data, dependent,
  problem = c("classification", "regression"), method = c("randomForest",
  "rpart"), name, description, data_transform = identity,
  data_transform_name = "identity")

Arguments
  train_index     A vector containing the rows from data to be used as the training
  data            A data frame
  dependent       The dependent variable: the name of the column containing the prediction goal
  problem         Either classification or regression. This influences how the algorithms are trained and what method is used to determine performance
  method          The regression or classification method
  name            The name of the test. Printed in the test results
  description     Optional. A more elaborate description of the test
  data_transform  A quoted function name that transforms the data. It should maintain it in data frame form and maintain the dependent variable.
  data_transform_name
      The name of the data transformation function

Value
An object of class 'evaluation', containing the evaluated test
crtests: A package for creating and executing classification and regression tests

Description
The crtests package provides functions to prepare data, train models, test models, and evaluate outcomes. Its goal is to provide a generic and extendable API to machine learning and statistics functions.

drop_na: Remove NAs according to a strategy

Description
Remove NAs according to a strategy

Usage
```r
drop_na(strategy = c("dependent", "predictors", "all", "none"), df, dependent)
```

Arguments
- strategy: Character string denoting how NAs should be dealt with. "dependent" means rows with NA in the dependent variable are dropped. "predictors" means rows with NA in an independent variable are dropped. "all" means rows with NA in any column are dropped. "none" means NAs are ignored.
- df: Data frame to remove NAs from
- dependent: Dependent variable of the data frame

Value
A data.frame where strategy has been applied to remove data
evaluate

Evaluate the performance of a prediction.

Description
Wraps the problem-specific evaluation functions by calling evaluate_problem. This wrapper is desirable, as it can perform the extraction of the holdout set (observations)

Usage
evaluate(prediction, data, test, ...)

Arguments
- prediction: A vector of predictions for each row in the holdout set
- data: The data list containing train and holdout data sets
- test: The test object being evaluated
- ...: Extra arguments to evaluate

Value
An object of class 'evaluation', which contains a list of performance measures and a test object.

evaluate_problem

Generic function for evaluation of test results

Description
Generic function for evaluation of test results

Usage
evaluate_problem(test, prediction, observations)

## S3 method for class 'classification'
evaluate_problem(test, prediction, observations)

## S3 method for class 'regression'
evaluate_problem(test, prediction, observations)

Arguments
- test: The test that was run
- prediction: A vector of predictions for each row in the holdout set
- observations: The true observations for the dependent value in the holdout set
**evaluation**

**Value**

An object of class 'evaluation', which contains a list of performance measures and a test object.

**Methods (by class)**

- **classification**: Evaluate a classification test’s results. Uses `confusionMatrix` to determine accuracy and other performance measures.
- **regression**: Evaluate a regression test’s results.

---

**evaluation**

*Create an evaluation object*

---

**Description**

Creates an evaluation object from the test and measures. Reads out the attributes of the test

**Usage**

`evaluation(test, measures)`

**Arguments**

- **test**: Object of class 'regression' or 'classification'
- **measures**: List of test measures and their values

**Value**

Object of class 'evaluation', with attributes: 'test_attributes', 'measures' and 'test'

---

**extract_formula**

*Extract a formula from a test*

---

**Description**

Extracts a formula of the form dependent ~ . from the test object

**Usage**

`extract_formula(test)`

**Arguments**

- **test**: An object of class 'regression' or 'classification'
factor_length

*Determine the length of the factors in a data.frame*

**Description**

Goes through every column in the data.frame, and return the length of its levels

**Usage**

`factor_length(df)`

**Arguments**

- `df`: A data.frame

**Value**

A vector of length n, with n the number of factor columns in the data frame, containing the length of the levels of those factors

---

group_levels

*Group infrequent levels in data, either a factor or a data.frame*

**Description**

Group infrequent levels in data, either a factor or a data.frame
Group infrequent factor levels in a data.frame
Group infrequent factor levels in a list of data.frames

**Usage**

`group_levels(data, maximum_levels = 32)`

```r
## S3 method for class 'factor'
group_levels(data, maximum_levels = 32)
```

```r
## S3 method for class 'data.frame'
group_levels(data, maximum_levels = 32)
```

```r
## S3 method for class 'list'
group_levels(data, maximum_levels = 32)
```
Arguments

- `data` A data.frame or factor. In the first case, `group_levels` is applied to each factor in the data.frame.

- `maximum_levels` Numeric. The maximum number of levels allowed per factor

Value

A factor with at most `maximum_levels`, or a data.frame where each factor matches that requirement

Methods (by class)

- `factor`: Group infrequent levels in a factor. Takes a factor, and if that factor has more than `maximum_levels`, it makes a table of level frequencies. The top (maximum_levels-1) are left unchanged, all less frequent levels are grouped into the level "other".

- `data.frame`: Takes a data.frame, and applies `group_levels.factor` to each column

- `list`: Takes a list of data.frames and applies `group_levels.data.frame` to each

---

**group_levels.default**  
*Group infrequent factor levels*

---

Description

The default `group_levels` does nothing. This is desirable behavior for any structure that is not a list, data.frame or factor: there is no meaningful way apply `group_levels` to this type of structure.

Usage

```r
## Default S3 method:
group_levels(data, maximum_levels = 32)
```

Arguments

- `data` A data.frame or factor. In the first case, `group_levels` is applied to each factor in the data.frame.

- `maximum_levels` Numeric. The maximum number of levels allowed per factor
is_complete_row: Determine if the rows in a data.frame have NAs

**Description**
Determine if the rows in a data.frame have NAs

**Usage**
```r
is_complete_row(data)
```

**Arguments**
- `data`: A data.frame

**Value**
A vector of length `nrow(data)` containing whether that row has NAs.

make_predictions: Make predictions using a model

**Description**
Make predictions using a model
Generic function for testing a model by making predictions

**Usage**
```r
make_predictions(model, data, test, ...)
```

## Default S3 method:
```r
make_predictions(model, data, test, ...)
```

## S3 method for class 'rpart'
```r
make_predictions(model, data, test, ...)
```

## S3 method for class 'boosting'
```r
make_predictions(model, data, test, ...)
```

## S3 method for class 'gbm'
```r
make_predictions(model, data, test, ...)
```
**method_prepare**

**Arguments**

- **model**: A classification or regression model
- **data**: The list of train and holdout data sets
- **test**: The test being conducted
- **...**: Extra arguments to make_predictions

**Methods (by class)**

- **default**: This function is a simple wrapper to `predict`, which it with the trained model and holdout data. Model classes that require extra arguments to predict can do so through a separate implementations or, less desirably, through the extra arguments.
- **rpart**: Calls predict.rpart with appropriate type: "class" for classification problems and "vector" for regression problems. Other problem types are not supported, providing a test with another class throws an error.
- **boosting**: Calls predict.boosting on the created model
- **gbm**: Calls predict.gbm on the created model with `n.trees = 100`

**See Also**

- `predict.rpart`

---

**method_prepare  Method-specific data preparation**

---

**Description**

Generic function for method-specific data preparation, if any is necessary

**Usage**

```r
method_prepare(method, test, ...)
```

**Arguments**

- **method**: The regression or classification method that needs specific data preparation.
- **test**: The test being executed, whose data attribute is a list of the train and holdout data sets, that has already been prepared by `prepare.data`
- **...**: Extra arguments to pass on to class methods
Value

A prepared data.frame

\texttt{identity(data)}

Methods (by class)

- default: Default function for method-specific data preparation. There is no default method-specific preparation, returns \texttt{identity}.
- randomForest: Random Forest specific data preparation. Calls \texttt{group_levels} on data, then relevels the holdout set so it has no levels not found in the training set (using \texttt{prepare_data})

\begin{verbatim}
multisample Make multiple samples of data

Description

Make multiple samples of data
cross_fold: Make 'folds' samples of the data, so all(rbind(folds)==row.names(data))=TRUE
random: Makes iterations random samples of size holdout * nrow(data)

Usage

multisample.cross_fold(data, folds = 10, dependent,
                        preserve_distribution = FALSE)

multisample.random(data, holdout = 0.2, iterations = 10, dependent,
                    preserve_distribution = FALSE)

Arguments

\begin{verbatim}
data          Data to sample
folds        Number of folds to create
dependent     The dependent variable in the data. Used only if preserve_distribution=TRUE
preserve_distribution
              Logical, only applicable if the dependent variable is a factor
holdout      The fraction of data to be used as holdout set
iterations   Number of iterations to make
\end{verbatim}

Value

A list of numeric vectors of length 'folds'
multitest
Create and run multiple instances of a test

Description

Wrapper for creating multiple copies of a test and running them. This function supports cross validation and regular sampling. Cross validation splits the data into 'iterations' number of folds, and uses one fold as holdout, using every other fold as training set. This is repeated 'iteration's times, using every fold as holdout exactly once. Non-cross validation takes a random sample of size holdout * nrow(data) and uses it as holdout, the rest is used for training. This is repeated 'iteration's times. Test creation and execution is handled by create_and_run_test

Usage

multitest(data, dependent, problem = c("classification", "regression"),
method, name, description = "", data_transform = identity,
itерations = 10, holdout = 0.2, cross_validation = FALSE,
preserve_distribution = FALSE)

Arguments

data A data frame
dependent The dependent variable: the name of the column containing the prediction goal
problem Either classification or regression. This influences how the algorithms are trained and what method is used to determine performance
method The regression or classification method
name The name of the test. Printed in the test results
description Optional. A more elaborate description of the test
data_transform A quoted function name that transforms the data. It should maintain it in data frame form and maintain the dependent variable.
itерations The number of times the test is to be performed. If cross-validation is used, this is the number of folds
holdout Sample testing only. The fraction of data to be used as holdout set
cross_validation Logical. Should cross validation be used?
preserve_distribution Logical, classification problems only. Should the distribution of factors in the dependent variable be as similar as possible between holdout and training sets?

Value

A list of class 'multitest_results_' + problem, containing the test results of each iteration
Examples

## Not run:
library(crtests)
library(randomForest)
library(rpart)
library(caret)
library(stringr)

# A classification multitest
multitest(data = iris,
          dependent = "Species",
          problem = "classification",
          method = "randomForest",
          name = "An example classification multitest",
          iterations = 10,
          cross_validation = TRUE,
          preserve_distribution = TRUE
)

# A regression multitest
multitest(data = iris,
          dependent = "Sepal.Width",
          problem = "regression",
          method = "rpart",
          name = "An example regression multitest",
          iterations = 15,
          cross_validation = FALSE,
)

## End(Not run)

### Description

Create an evaluation of multiple tests

### Usage

`multitest_evaluation(evaluations, iterations, cross_validation, preserve_distribution, name, method, problem)`

### Arguments

- `evaluations`: List of evaluation objects
- `iterations`: Numeric. Number of times the test was conducted
**na_count**

Count the number of NAs in an object

**Description**

Count the number of NAs in an object

**Usage**

```r
na_count(x, ...)
```

```r
## S3 method for class 'data.frame'
na_count(x, columns = c(), ...)
```

```r
## Default S3 method:
na_count(x, ...)
```

**Arguments**

- `x` An object, either a vector or a data.frame
- `...` Extra arguments to `na_count`
- `columns` Vector of column names

### Value

An object of type `multitest_evaluation`. Attributes are:

- `evaluations` List of evaluations
- `iterations` Number of times the test was conducted
- `cross_validation` Was cross-validation used as a sampling strategy?
- `preserve_distribution` Was preservation of class distribution between training and holdout set attempted?
- `name` Name of the test
- `method` Name of the method used in the test
- `problem` Name of the machine learning problem
**Methods (by class)**

- **data.frame**: If columns are specified, returns the maximum of the count of NAs for those columns. Otherwise, it returns the number of rows that have a NA in any column.
- **default**: Calls `na.omit` on `x`, and returns the length of the result. This is only meaningful for one-dimensional objects (vectors).

---

**prepare**

*Prepare the data for the specified test.*

**Description**

This allows for different implementations for regression or classification

**Usage**

```r
prepare(test, ...)  
## Default S3 method:  
prepare(test, ...)  
```

**Arguments**

- `test` The test for which data is prepared
- `...` Extra arguments to prepare

**Value**

`data` A list containing prepared train (`data$train`) and holdout (`data$holdout`) data frames. Extra method specific preparation is executed through a call to `method_prepare`

**Methods (by class)**

- **default**: The default method releves the holdout set, so the holdout and train set are completely independent, and to prevent problems with certain algorithms that can’t deal with different factor levels across train and holdout set
**prepare_data**

*Prepare data for training or testing.*

**Description**

This function removes all missing values, including those introduced after (optional) releveling.

**Usage**

```r
prepare_data(df, df_reference, dependent, relevel = TRUE,
             drop.nas = c("dependent", "predictors", "all", "none"))
```

**Arguments**

- `df`  
  The data frame that is to be prepared

- `df_reference`  
  An optional reference data frame, whose factor levels are to be applied to df

- `dependent`  
  The dependent variable of the data

- `relevel`  
  Logical. Should the df be releveled with df_reference’s factor levels?

- `drop.nas`  
  Character vector denoting of which columns the NAs should be removed. See `drop_na` for the available strategies

**Value**

A data frame stripped of missing values

---

**print.evaluation**

*Print an 'evaluation' object*

**Description**

Pretty prints an object of class 'evaluation'.

**Usage**

```r
## S3 method for class 'evaluation'
print(x, digits = max(3,getOption("digits") - 4), ...)
```

**Arguments**

- `x`  
  Object to print

- `digits`  
  Numeric. Number of digits to print. Defaults to `max(3,getOption("digits")-4)`

- `...`  
  Further arguments to `print.evaluation`
Details

Prints the object to look like a table

Examples

data(iris)
# A classification test
test <- createtest(data = iris,
                  dependent = "Species",
                  problem = "classification",
                  method = "randomForest",
                  name = "An example classification test",
                  train_index = sample(150, 100)
)
## Not run:
# Run the test. The result is an object of class "evaluation"
evaluation <- runtest(test)
print(evaluation)
## End(Not run)
print.multitest_evaluation.summary

Print a multitest_evaluation.summary object

Description

Print a multitest_evaluation.summary object

Usage

## S3 method for class 'multitest_evaluation.summary'
print(x, digits = max(3, 
    getOption("digits") - 4), ...)

Arguments

x
Object to print

digits
Numeric. Number of digits to print. Defaults to max(3, getOption("digits")-4)

... Further arguments to print.multitest_evaluation.summary

Examples

## Not run:
library(crcrte)
library(randomForest)
library(rpart)
library(caret)
library(stringr)

# A classification multitest. The result is an object of class multitest_evaluation
multitest_evaluation <- multitest(data = iris,
    dependent = "Species",
    problem = "classification",
    method = "randomForest",
    name = "An example classification multitest",
    iterations = 10,
    cross_validation = TRUE,
    preserve_distribution = TRUE
)
print(summary(multitest_evaluation))

## End(Not run)
random_string Generate a random string

Description
Generates a random string of length length

Usage
random_string(length)

Arguments
length Length of string to generate

Value
If length>0: A random sequence of characters of length length, otherwise an empty string

regression_model Fit a regression model Generic function for fitting a regression model

Description
Fit a regression model
Generic function for fitting a regression model

Usage
regression_model(method, formula, training_data, ...)

## Default S3 method:
regression_model(method, formula, training_data, ...)

Arguments
method The regression method to use
formula An object of class 'formula', used to fit a model to the data
training_data Train data used to fit the model
... Further arguments

Value
model The fitted model
Methods (by class)

- **default**: Default function for fitting a regression model
  
  This gets the method and calls it using formula and data

---

### remove_names

**Set any names of x to ""**

**Description**

Set any names of x to ""

Set row and column names to "" for pretty printing

**Usage**

```r
remove_names(x)
```

```r
## S3 method for class 'matrix'
remove_names(x)
```

**Arguments**

- **x**: An object that has a 'names' property, typically a matrix, list or data.frame
- **matrix**: Matrix to 'remove' colnames and rownames from

**Value**

Matrix where colnames and rownames consist of only ""

---

Methods (by class)

- **matrix**: Remove names from a matrix

---

### replace_names

**Replace strings in the names of an object**

**Description**

Replaces strings matching the pattern in the names of the object by the replacement. If applicable, both row and column names could be replaced. This function is a simple wrapper to `str_replace_all`

Default method that replaces names(object)

Replaces row.names in the object, then dispatches to the default

Replace row.names and col.names in the object
Usage

replace_names(object, pattern, replacement, ...)

## Default S3 method:
replace_names(object, pattern = "\./",
             replacement = " ", ...)

## S3 method for class 'data.frame'
replace_names(object, pattern = "\./",
             replacement = " ", replace_rownames = TRUE, replace_colnames = TRUE,
             ...)

## S3 method for class 'matrix'
replace_names(object, pattern = "\./", replacement = " ",
             replace_rownames = TRUE, replace_colnames = TRUE, ...)

Arguments

object Object of which the names are to be changed
pattern Pattern to look for, as defined by a POSIX regular expression
replacement Replacement string
...
... extra arguments to replace_names
replace_rownames Logical. Should row names be replaced?
replace_colnames Logical. Should column names be replaced?

Methods (by class)

- default: Replace names of an object
- data.frame: Replace names of a data.frame
- matrix: Replace names in a matrix

See Also

str_replace_all

runtest  
Run a classification or regression test

Description

Run a classification or regression test
Usage

runtest(test, ...)

## Default S3 method:
runtest(test, ...)

Arguments

test An object of class 'classification' or 'regression'

... Extra arguments to runtest

Methods (by class)

• default: The default test run subsequently calls prepare, train_model, make_predictions, evaluate

Examples

data(iris)
# A classification test
test <- createtest(data = iris,
  dependent = "Species",
  problem = "classification",
  method = "randomForest",
  name = "An example classification test",
  train_index = sample(150, 100)
)
## Not run:
# Run the test
runtest(test)
## End(Not run)

summary.evaluation Summary of an evaluation

Description

Produces a summary of an evaluation, consisting of the test attributes and the performance measures

Usage

## S3 method for class 'evaluation'
summary(object, include_test_attributes = TRUE, ...)

summary.evaluation Summary of an evaluation

Arguments

test An object of class 'classification' or 'regression'

... Extra arguments to runtest

Methods (by class)

• default: The default test run subsequently calls prepare, train_model, make_predictions, evaluate

Examples

data(iris)
# A classification test
test <- createtest(data = iris,
  dependent = "Species",
  problem = "classification",
  method = "randomForest",
  name = "An example classification test",
  train_index = sample(150, 100)
)
## Not run:
# Run the test
runtest(test)
## End(Not run)
Arguments

object Evaluation object to make summary of
include_test_attributes Logical. Should all attributes of the test be included in the output?
...
Extra arguments to summary.evaluation

Examples

data(iris)
# A classification test
test <- createtest(data = iris,
  dependent = "Species",
  problem = "classification",
  method = "randomForest",
  name = "An example classification test",
  train_index = sample(150, 100)
)
## Not run:
# Run the test. The result is an object of class "evaluation"
evaluation <- runtest(test)
summary(results)
## End(Not run)
Examples

```r
## Not run:
library(crtests)
library(randomForest)
library(rpart)
library(caret)
library(stringr)

# A classification multitest. The result is an object of class multitest_evaluation
multitest_evaluation <- multitest(data = iris,
                                   dependent = "Species",
                                   problem = "classification",
                                   method = "randomForest",
                                   name = "An example classification multitest",
                                   iterations = 10,
                                   cross_validation = TRUE,
                                   preserve_distribution = TRUE)

summary(multitest_evaluation)
```

## End(Not run)

---

**train_model**

*Train a classification or regression model*

**Description**

Generic function for training a model.

**Usage**

```r
train_model(test, data, ...)
```

```r
## S3 method for class 'classification'
train_model(test, data, ...)
```

```r
## S3 method for class 'regression'
train_model(test, data, ...)
```

**Arguments**

- `test` The test object. This is passed so the method can be extracted.
- `data` An object of class "regression" or "classification" with at least x, y, train and holdout
- `...` Extra arguments to pass to the classification or regression method
Methods (by class)

- **classification**: Train a model for classification using a classifier algorithm. This function wraps the actual classifier.
- **regression**: Train (fit) a regression model. This function wraps a regression algorithm.

---

**Utility functions**

**Description**

Utility functions

Utility for testing how a function deals with missing required arguments. It calls the function `length(args)` times, each time omitting one argument.

**Usage**

`missing_argument_test(fun, args, outcomes)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fun</code></td>
<td>Function to test</td>
</tr>
<tr>
<td><code>args</code></td>
<td>Complete list of required arguments to <code>fun</code></td>
</tr>
<tr>
<td><code>outcomes</code></td>
<td>List of length <code>args</code> with expected outcomes for each test. Names should match those of <code>args</code>. Values should be either &quot;FAIL&quot;, if the test is expected to throw an error, or anything else if it is expected to pass.</td>
</tr>
</tbody>
</table>
Index

apply_levels, 2
argument_match_test, 3

capitalize_first, 3
classification_model, 4
confusionMatrix, 9
create_and_run_test, 6, 15
createtest, 5, 6
crtests, 7
crtests-package (crtests), 7
drop_na, 7, 19
evaluate, 8, 25
evaluate_problem, 8, 8
evaluation, 9
extract_formula, 9

factor_length, 10
group_levels, 10, 14
group_levels.default, 11
identity, 14
is_complete_row, 12

make_predictions, 12, 25
method_prepare, 13
missing_argument_test (util), 28
multisample, 14
multitest, 15
multitest_evaluation, 16

NA, 12
na.omit, 18
na_count, 17

predict, 13
prepare, 18, 25
prepare_data, 13, 14, 19

print.evaluation, 19
print.multitest_evaluation, 20
print.multitest_evaluation.summary, 21
random_string, 22
regression_model, 22
remove_names, 23
replace_names, 23
runtest, 6, 24
str_replace_all, 23, 24
summary.evaluation, 25
summary.multitest_evaluation, 26

train_model, 25, 27
util, 28