Package: aslib (via r-universe)

October 27, 2026

Title Interface to the Algorithm Selection Benchmark Library

Description Provides an interface to the algorithm selection benchmark library at <https://www.coseal.net/aslib/> and the 'LLAMA' package (<https://cran.r-project.org/package=llama>) for building algorithm selection models; see Bischl et al. (2016) <doi:10.1016/j.artint.2016.04.003>.

Maintainer Lars Kotthoff <larsko@uwyo.edu>

URL https://github.com/coseal/aslib-r/

BugReports https://github.com/coseal/aslib-r/issues

License GPL-3

Imports batchtools, data.table, BBmisc, checkmate, corrplot, ggplot2, llama, mlr, parallelMap, ParamHelpers, plyr, reshape2, RWeka, stringr, yaml

Suggests testthat, rpart

ByteCompile yes

Encoding UTF-8

Version 0.1.3

RoxygenNote 7.3.2

NeedsCompilation no

Author Bernd Bischl [aut], Lars Kotthoff [aut, cre], Pascal Kerschke [ctb, aut], Damir Pulatov [ctb, aut]

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

Date/Publication 2025-08-19 17:40:02 UTC

RemoteUrl https://github.com/cran/aslib

RemoteRef HEAD

RemoteSha fc18e5955a60a8d6b3b25c259d7a1759e3efa171

Contents

ASScenarioDesc . . . 2
checkDuplicatedInstances . . . 3
convertAlgoPerfToWideFormat . . . 4
convertToLlama . . . 4
convertToLlamaCVFolds . . . 5
createCVSplits . . . 6
findDominatedAlgos . . . 7
fixFeckingPresolve . . . 7
getAlgorithmNames . . . 8
getCosealASScenario . . . 9
getCostsAndPresolvedStatus . . . 9
getDefaultFeatureStepNames . . . 10
getFeatureNames . . . 11
getFeatureStepNames . . . 11
getInstanceNames . . . 12
getNumberOfCVFolds . . . 12
getNumberOfCVReps . . . 13
getProvidedFeatures . . . 13
getSummedFeatureCosts . . . 14
imputeAlgoPerf . . . 14
parseASScenario . . . 15
plotAlgoCorMatrix . . . 17
plotAlgoPerf . . . 18
runLlamaModels . . . 19
summarizeAlgoPerf . . . 20
summarizeAlgoRunstatus . . . 21
summarizeFeatureSteps . . . 22
summarizeFeatureValues . . . 22
summarizeLlamaExps . . . 23
writeASScenario . . . 23

Index **25**

`ASScenarioDesc` *S3 class for ASScenarioDesc.*

Description

Object members

Details

scenario_id [`character(1)`] Name of scenario.

performance_measures [`character`] Names of measures.

maximize [**named** `character`] Maximize measure?

performance_type [**named** `character`] Either "runtime" or "solution_quality".

algorithm_cutoff_time [`numeric(1)`] Cutoff time for an algorithm run.

algorithm_cutoff_memory [`numeric(1)`] Cutoff memory for an algorithm run.

features_cutoff_time [`numeric(1)`] Cutoff time for an instance feature run.

features_cutoff_memory [`numeric(1)`] Cutoff memory for an instance feature run.

algorithm_features_cutoff_time [`numeric(1)`] Cutoff time for an algorithm feature run.

algorithm_features_cutoff_memory [`numeric(1)`] Cutoff memory for an algorithm feature run.

feature_steps [**named** `list` **of** `character`] Names of feature processing steps, the other feature steps they require, and the features they provide.

metainfo_algorithms [**named** `list` **of lists of** `character`] Names of algorithms and meta-information about them.

`checkDuplicatedInstances` *Checks the feature data set for duplicated instances.*

Description

Potentially duplicated instances are detected by grouping all instances with equal feature vectors.

Usage

```
checkDuplicatedInstances(asscenario)
```

Arguments

`asscenario`	[`ASScenario`] Algorithm selection scenario.

Value

[`list` of `character`]. List of instance id vectors where corresponding feature vectors are the same. Only groups of at least 2 elements are returned.

convertAlgoPerfToWideFormat

Converts `algo.runs` *object of a scenario to wide format.*

Description

The first 2 columns are "instance_id" and "repetition". The remaining ones are the measured performance values. The feature columns are in the same order as "features_deterministic", "features_stochastic" in the description object. `NA` means the performance value is not available, possibly because the algorithm run was aborted. The data.frame is sorted by "instance_id", then "repetition".

Usage

```
convertAlgoPerfToWideFormat(desc, algo.runs, measure)
```

Arguments

`desc`	[ASScenarioDesc] Description object of scenario.
`algo.runs`	[data.frame] Algo runs data.frame from scenario.
`measure`	[character(1)] Selected performance measure. Default is first measure in scenario.

Value

[data.frame].

convertToLlama

Convert an ASScenario scenario object to a llama data object.

Description

For features, mean values are computed across repetitions. For algorithms, repetitions are not supported at the moment and will result in an error.

Usage

```
convertToLlama(asscenario, measure, feature.steps)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
measure	[character(1)] Measure to use for modeling. Default is first measure in scenario.
feature.steps	[character] Which feature steps are allowed? Default are the default feature steps or all steps in case no defaults were defined.

Details

Note that feature step dependencies are currently not supported explicitly by LLAMA. The conversion checks that all dependencies are satisfied, but subsequent feature selection on the LLAMA data frame may not work as expected.

Value

Result of calling input.

---

convertToLlamaCVFolds *Convert an ASScenario scenario object to a llama data object with cross-validation folds.*

Description

For features, mean values are computed across repetitions. For algorithms, repetitions are not supported at the moment and will result in an error.

Usage

```
convertToLlamaCVFolds(
  asscenario,
  measure,
  feature.steps,
  algorithm.feature.steps,
  cv.splits
)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
measure	[character(1)] Measure to use for modelling. Default is first measure in scenario.

feature.steps [character]
Which instance feature steps are allowed? Default are the default instance feature steps or all steps in case no defaults were defined.

algorithm.feature.steps [character]
Which algorithm feature steps are allowed? Default are the default algorithm feature steps or all steps in case no defaults were defined.

cv.splits [data.frame]
Data frame defining the split of the data into cross-validation folds, as returned by createCVSplits. Default are the splits asscenario$cv.splits

Value

Result of calling input with data partitioned into folds.

---

createCVSplits *Create cross-validation splits for a scenario.*

Description

Create a data.frame that defines cross-validation splits for a scenario,

and potentially store it in an ARFF file.

The mlr package is used to generate the splits, see makeResampleDesc and makeResampleInstance.

Usage

```
createCVSplits(asscenario, reps = 1L, folds = 10L, file = NULL)
```

Arguments

asscenario [ASScenario]
Algorithm selection scenario.

reps [integer]
CV repetitions. Default is 1.

folds [integer]
CV folds. Default is 10.

file [character]
If not missing, where to save the returned splits as an ARFF file via write.arff. Default is no saving.

Value

[data.frame]. Splits as defined in the algorithm benchmark repository specification text. Has columns: "instance_id", "fold", "rep". Defines which instances go into the test set for each replication / fold during CV. The training set are the remaining instances, in exactly the order as given by the data.frame for the current repetition.

`findDominatedAlgos` *Creates a table that shows the dominance of one algorithm over another one.*

Description

If NAs occur, they are imputed (before aggregation) by `base + 0.3 * range`. `base` is the cutoff value for runtimes scenarios with cutoff or the worst performance for all others.

Stochastic replications are aggregated by the mean value.

Usage

```
findDominatedAlgos(asscenario, measure, reduce = FALSE, type = "logical")
```

Arguments

`asscenario`	[ASScenario] Algorithm selection scenario.
`measure`	[character(1)] Measure for algorithm performance. Default is first measure in scenario.
`reduce`	[logical(1)] Should the resulting matrix be reduced to algorithms that a are either dominated by or dominate another algorithm? Default is `FALSE`.
`type`	[character(1)] Data type of the result object. "logical": Logical matrix, TRUE means row algorithm dominates column algorithm. "character": Same information but more human-readable. States how the row relates to the column.

Value

[matrix]. See above.

`fixFeckingPresolve` *Bakes presolving stuff into a LLAMA data frame.*

Description

Determines whether any of the feature groups in the LLAMA data frame presolve any of the instances. If so, the performances of all algorithms in the portfolio are set to the runtime of the first used feature group that presolves the respective instance. Furthermore, the success of all algorithms on those instances is set to true.

Usage

```
fixFeckingPresolve(asscenario, ldf)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
ldf	[LLAMA data frame] LLAMA data frame to modify.

Details

These modifications are done on the main LLAMA data and on any test splits. They are *not* done on the training data. This function should only ever be used to evaluate the performance of an actual selector that uses features (i.e. not VBS or single best). Using it in polite company is to be avoided.

Value

The LLAMA data frame with presolving baked into the algorithm performances.

---

getAlgorithmNames *Returns algorithm names of scenario.*

Description

Returns algorithm names of scenario.

Usage

```
getAlgorithmNames(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[character].

getCosealASScenario *Retrieves a scenario from the Coseal Github repository and parses into an S3 object.*

Description

Uses subversion export to retrieve a specific scenario from the official Coseal Github repository. The scenario is checked out into a temporary directory and parsed with `parseASScenario`.

Usage

```
getCosealASScenario(name)
```

Arguments

name	[character(1)] Name of benchmark data set.

Value

[ASScenario]. Description object.

Examples

```
## Not run:
  sc = getCosealASScenario("CSP-2010")

## End(Not run)
```

getCostsAndPresolvedStatus *Return whether an instance was presolved and which step did it.*

Description

Return whether an instance was presolved and which step did it.

Usage

```
getCostsAndPresolvedStatus(asscenario, feature.steps, type)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
feature.steps	[character] Which feature steps are allowed? Default is all steps.
type	[character(1)] Feature type (instance or algorithmic).

Value

[list]. Below, n is the number of instances. All following object are ordered by "instance_id".

is.presolved [logical(n)]
: Was instance presolved? Named by instance ids.

solve.steps [character(n)]
: Which step solved it? NA if no step did it. Named by instance ids.

costs [numeric(n)]
: Feature costs for using the steps. Named by instance ids. NULL if no costs are present.

---

getDefaultFeatureStepNames

Returns the default feature step names of scenario.

Description

Returns the default feature step names of scenario.

Usage

```
getDefaultFeatureStepNames(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[character].

getFeatureNames *Returns feature names of scenario.*

Description

Returns feature names of scenario.

Usage

```
getFeatureNames(asscenario, type)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
type	[character(1)] Feature type (instance or algorithmic).

Value

[character].

getFeatureStepNames *Returns feature step names of scenario.*

Description

Returns feature step names of scenario.

Usage

```
getFeatureStepNames(asscenario, type)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
type	[character(1)] Feature type (instance or algorithmic).

Value

[character].

getInstanceNames *Returns instance names of scenario.*

Description

Returns instance names of scenario.

Usage

```
getInstanceNames(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[character].

getNumberOfCVFolds *Returns number of CV folds.*

Description

Returns number of CV folds.

Usage

```
getNumberOfCVFolds(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[integer(1)].

getNumberOfCVReps *Returns number of CV repetitions.*

Description

Returns number of CV repetitions.

Usage

```
getNumberOfCVReps(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[integer(1)].

getProvidedFeatures *Return features that are useable for a given set of feature steps.*

Description

Return features that are useable for a given set of feature steps.

Usage

```
getProvidedFeatures(asscenario, steps, type)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
steps	[character] Feature steps. Default are all feature steps.
type	[character(1)] Feature type (instance or algorithmic).

Value

[character].

getSummedFeatureCosts *Returns feature costs of scenario, summed over all instances.*

Description

Returns feature costs of scenario, summed over all instances.

Usage

```
getSummedFeatureCosts(asscenario, feature.steps)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
feature.steps	[character] Sum costs only for these selected steps. Default are all feature steps.

Value

[character].

imputeAlgoPerf *Imputes algorithm performance for runs which have NA performance values.*

Description

The following formula is used for imputation: `base +- range.scalar * range.span + N(0, sd = jitter * range.span)`
With `range.span = max - min`.

Returns an object like `algo.runs` of `asscenario`, but drops the runstatus and all other measures.

Usage

```
imputeAlgoPerf(
  asscenario,
  measure,
  base = NULL,
  range.scalar = 0.3,
  jitter = 0,
  impute.zero.vals = FALSE
)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
measure	[character(1)] Measure to impute. Default is first measure in scenario.
base	[numeric(1)] See formula. Default is NULL, which means maximum of performance values if measure should be minimized, or minimum for maximization case.
range.scalar	[numeric(1)] See formula. Default is 0.3.
jitter	[numeric(1)] See formula. Default is 0.
impute.zero.vals	[logical(1)] Should values which are exactly 0 be imputed to 1e-6? This allows to take the logarithm later on, handy for subsequent visualizations. Note that this really only makes sense for non-negative measures! Default is FALSE.

Value

[data.frame].

---

parseASScenario *Parses the data files of an algorithm selection scenario into an S3 object.*

Description

Object members

Let n be the number of (replicated) instances, m the number of unique instances, p the number of features, s the number of feature steps and k the number of algorithms.

desc [ASScenarioDesc] Description object, containing further info.

feature.runstatus [data.frame(n, s + 2)] Runstatus of instance feature computation steps. The first 2 columns are "instance_id" and "repetition", the remaining are the status factors. The step columns are in the same order as the feature steps in the description object. The factor levels are always: ok, presolved, crash, timeout, memout, other. No entry can be NA. The data.frame is sorted by "instance_id", then "repetition".

algorithm.feature.runstatus [data.frame(k, s + 1)] Runstatus of algorithm feature computation steps. The first column is "algorithm", the remaining are the status factors. The step columns are in the same order as the feature steps in the description object. The factor levels are always: ok, crash, timeout, memout, other. No entry can be NA. The data.frame is sorted by "algorithm".

feature.costs [data.frame(n, s + 2)] Costs of instance feature computation steps. The first 2 columns are "instance_id" and "repetition", the remaining are numeric costs of the instance feature steps. The step columns are in the same order as the feature steps in the description object. NA means the cost is not available, possibly because the feature computation was aborted. The data.frame is sorted by "instance_id", then "repetition". If no cost file is available at all, NULL is stored.

algorithm.feature.costs [data.frame(n, s + 1)] Costs of algorithm feature computation steps. The first column is "algorithm", the remaining are numeric costs of the algorithmic feature steps. The step columns are in the same order as the feature steps in the description object. NA means the cost is not available, possibly because the feature computation was aborted. The data.frame is sorted by "algorithm". If no cost file is available at all, NULL is stored.

feature.values [data.frame(n, p + 2)] Measured feature values of instances. The first 2 columns are "instance_id" and "repetition". The remaining ones are the measured instance features. The feature columns are in the same order as "instance_features_deterministic", "features_stochastic" in the description object. NA means the feature is not available, possibly because the feature computation was aborted. The data.frame is sorted by "instance_id", then "repetition".

algorithm.feature.values [data.frame(k, p + 1)] Measured feature values of algorithms The first column is "algorithm". The remaining ones are the measured algorithmic features. The feature columns are in the same order as "algorithm_features_deterministic", "algorithm_features_stochastic" in the description object. NA means the feature is not available, possibly because the feature computation was aborted. The data.frame is sorted by "algorithm".

algo.runs [data.frame] Runstatus and performance information of the algorithms. Simply the parsed ARFF file. See convertAlgoPerfToWideFormat for a more convenient format.

algo.runstatus [data.frame(n, k + 2)] Runstatus of algorithm runs. The first 2 columns are "instance_id" and "repetition", the remaining are the status factors. The step columns are in the same order as the feature steps in the description object. The factor levels are always: ok, presolved, crash, timeout, memout, other. No entry can be NA. The data.frame is sorted by "instance_id", then "repetition".

cv.splits[data.frame(m, 3)] Definition of cross-validation splits for each replication of a repeated CV with folds. Has columns "instance_id", "repetition" and "fold". The instances with fold = i for a replication r constitute the i-th test set for the r-th CV. The training set is the "instance_id" column with repetition = r, in the same order, when the test set is removed. The data.frame is sorted by "repetition", then "fold", then "instance_id". If no CV file is available at all, NULL is stored, and a warning is issued, although this should not happen.

Usage

```
parseASScenario(path)
```

Arguments

path	[character(1)] Path to directory of benchmark data set.

Value

[ASScenario]. Description object.

See Also

writeASScenario

Examples

```
## Not run:
  sc = parseASScenario("/path/to/scenario")

## End(Not run)
```

---

plotAlgoCorMatrix *Plots the correlation matrix of the algorithms.*

Description

If NAs occur, they are imputed (before aggregation) by `base + 0.3 * range`. `base` is the cutoff value for runtimes scenarios with cutoff or the worst performance for all others.

Stochastic replications are aggregated by the mean value.

Usage

```
plotAlgoCorMatrix(
  asscenario,
  measure,
  order.method = "hclust",
  hclust.method = "ward.D2",
  cor.method = "spearman"
)
```

Arguments

`asscenario`	[ASScenario] Algorithm selection scenario.
`measure`	[`character(1)`] Measure to plot. Default is first measure in scenario.
`order.method`	[`character(1)`] Method for ordering the algorithms within the plot. Possible values are "hclust" (for hierarchical clustering order), "FPC" (first principal component order), "AOE" (angular order of eigenvectors), "original" (original order) and "alphabet" (alphabetical order). See `corrMatOrder`. Default is "hclust".
`hclust.method`	[`character(1)`] Method for hierarchical clustering. Only useful, when `order.method` is set to "hclust", otherwise ignored. Possible values are: "ward.D2", "single", "complete", "average", "mcquitty", "median" and "centroid". See `corrMatOrder`. Default is "ward.D2".

cor.method [character(1)]
Method to be used for calculating the correlation between the algorithms. Possible values are "pearson", "kendall" and "spearman". See cor. Default is "spearman".

Value

See corrplot.

---

plotAlgoPerf *EDA plots for performance values of algorithms across all instances.*

Description

If NAs occur, they are imputed (before aggregation) by base + 0.3 range + jitter . base is is the cutoff value for runtimes scenarios with cutoff or the worst performance for all others.

For the CDFs we only show the visible area where successful runs occurred.

Stochastic replications are aggregated by the mean value.

Usage

```
plotAlgoPerfBoxplots(
  asscenario,
  measure,
  impute.zero.vals = FALSE,
  log = FALSE,
  impute.failed.runs = TRUE,
  rm.censored.runs = TRUE
)

plotAlgoPerfCDFs(
  asscenario,
  measure,
  impute.zero.vals = FALSE,
  log = FALSE,
  rm.censored.runs = TRUE
)

plotAlgoPerfDensities(
  asscenario,
  measure,
  impute.failed.runs = TRUE,
  impute.zero.vals = FALSE,
  log = FALSE,
  rm.censored.runs = TRUE
)
```

```
plotAlgoPerfScatterMatrix(
  asscenario,
  measure,
  impute.zero.vals = FALSE,
  log = FALSE,
  rm.censored.runs = TRUE
)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
measure	[character(1)] Measure to plot. Default is first measure in scenario.
impute.zero.vals	[logical(1)] Should values which are exactly 0 be imputed to 1e-6? This allows to take the logarithm later on, handy for subsequent visualizations. Note that this really only makes sense for non-negative measures! Default is FALSE.
log	[logical(1)] Should the performance values be log10-transformed in the plot? Default is FALSE.
impute.failed.runs	[logical(1)] Should runtimes for failed runs be imputed? Default is TRUE.
rm.censored.runs	[logical(1)] Should runtimes for censored runs (i.e. runs that have hitted the walltime) be removed (and eventually be imputed along with the remaining NAs)? Default is TRUE.

Value

ggplot2 plot object.

---

runLlamaModels *Creates a registry which can be used for running several Llama models on a cluster.*

Description

It is likely that you need to install some additional R packages for this from CRAN or extra Weka learner. The latter can be done via e.g. `WPM("install-package", "XMeans")`.

Feature costs are added for real prognostic models but not for baseline models.

Usage

```
runLlamaModels(
  asscenarios,
  feature.steps.list = NULL,
  baselines = NULL,
  learners = list(),
  par.sets = list(),
  rs.iters = 100L,
  n.inner.folds = 2L
)
```

Arguments

asscenarios	[(list of) ASScenario] Algorithm selection scenarios.
feature.steps.list	[list of character] Named list of feature steps we want to use. Must be named with scenario ids. Default is to take the default feature steps from the scenario.
baselines	[character] Vector of characters, defining the baseline models. Default is c("vbs", "singleBest", "singleBestByPar", "singleBestBySuccesses").
learners	[list of Learner] mlr learners to use for modeling. Default is none.
par.sets	[list of ParamSet] Param sets for learners to tune via random search. Pass an empty param set, if you want no tuning. Must be in of same length as learners and in the same order. Default is none.
rs.iters	[integer(1)] Number of iterations for random search hyperparameter tuning. Default is 100.
n.inner.folds	[integer(1)] Number of cross-validation folds for inner CV in hyperparameter tuning. Default is 2L.

Value

batchtools registry.

---

summarizeAlgoPerf *Creates summary data.frame for algorithm performance values across all instances.*

Description

Creates summary data.frame for algorithm performance values across all instances.

Usage

```
summarizeAlgoPerf(asscenario, measure)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
measure	[character(1)] Selected measure. Default is first measure in scenario.

Value

[data.frame].

---

summarizeAlgoRunstatus

Creates summary data.frame for algorithm runstatus across all instances.

Description

Creates summary data.frame for algorithm runstatus across all instances.

Usage

```
summarizeAlgoRunstatus(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[data.frame].

summarizeFeatureSteps *Creates a data.frame that summarizes the feature steps.*

Description

Creates a data.frame that summarizes the feature steps.

Usage

```
summarizeFeatureSteps(asscenario)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.

Value

[data.frame].

summarizeFeatureValues *Creates summary data.frame for feature values across all instances.*

Description

Creates summary data.frame for feature values across all instances.

Usage

```
summarizeFeatureValues(asscenario, type)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
type	[character(1)] Feature type (instance or algorithmic).

Value

[data.frame].

summarizeLlamaExps *Creates summary data.table for runLlamaModel experiments.*

Description

Creates summary data.table for runLlamaModel experiments.

Usage

```
summarizeLlamaExps(
  reg,
  ids = findSubmitted(),
  fun = function(job, res) {
     return(list(succ = res$succ, par10 = res$par10, mcp =
    res$mcp))
 },
  missing.val = list(succ = 0, par10 = Inf, mcp = Inf)
)
```

Arguments

reg	[Registry] batchtools registry.
ids	[data.table] Selected job ids. Default is all submitted jobs.
fun	[function()] Function to aggregate results with. Default is a function that returns succ, par10 and mcp values. For a detailed description, see [reduceResultsList].
missing.val	[list(1)] List with defaults for missing values that are needed for aggregation. For a detailed description, see [reduceResultsList].

Value

[data.table].

writeASScenario *Writes an algorithm selection scenario to a directory.*

Description

Splits an algorithm selection scenario into description, feature values / runstatus / costs, algorithm performance and cv splits and saves those data sets as single ARFF files in the given directory.

Usage

```
writeASScenario(asscenario, path = asscenario$desc$scenario_id)
```

Arguments

asscenario	[ASScenario] Algorithm selection scenario.
path	[character(1)] Path to write scenario to. Default is the name of the scenario.

See Also

parseASScenario

Index

ASScenario, *3*, *5–17*, *19–22*, *24*
ASScenario *(*parseASScenario*)*, 15
ASScenarioDesc, 2, *4*, *15*

checkDuplicatedInstances, 3
convertAlgoPerfToWideFormat, 4, *16*
convertToLlama, 4
convertToLlamaCVFolds, 5
cor, *18*
corrMatOrder, *17*
corrplot, *18*
createCVSplits, 6, *6*

findDominatedAlgos, 7
fixFeckingPresolve, 7

getAlgorithmNames, 8
getCosealASScenario, 9
getCostsAndPresolvedStatus, 9
getDefaultFeatureStepNames, 10
getFeatureNames, 11
getFeatureStepNames, 11
getInstanceNames, 12
getNumberOfCVFolds, 12
getNumberOfCVReps, 13
getProvidedFeatures, 13
getSummedFeatureCosts, 14

imputeAlgoPerf, 14
input, *5*, *6*

Learner, *20*

makeResampleDesc, *6*
makeResampleInstance, *6*

ParamSet, *20*
parseASScenario, 15, *24*
plotAlgoCorMatrix, 17
plotAlgoPerf, 18
plotAlgoPerfBoxplots *(*plotAlgoPerf*)*, 18
plotAlgoPerfCDFs *(*plotAlgoPerf*)*, 18
plotAlgoPerfDensities *(*plotAlgoPerf*)*, 18
plotAlgoPerfScatterMatrix *(*plotAlgoPerf*)*, 18

reduceResultsList, *23*
Registry, *23*
runLlamaModels, 19

summarizeAlgoPerf, 20
summarizeAlgoRunstatus, 21
summarizeFeatureSteps, 22
summarizeFeatureValues, 22
summarizeLlamaExps, 23

write.arff, *6*
writeASScenario, *17*, 23