Package ‘rmcfs’

May 12, 2019

Title The MCFS-ID Algorithm for Feature Selection and Interdependency Discovery

Version 1.3.0

Date 2019-05-11

Depends rJava (>= 0.5-0), R (>= 2.70)

Suggests testthat, R.rsp

VignetteBuilder R.rsp

Imports yaml, ggplot2, gridExtra, reshape2, dplyr, stringi, igraph,
data.table (>= 1.0.1)

SystemRequirements Java (>= 7)

Description MCFS-ID (Monte Carlo Feature Selection and Interdependency Discovery) is a Monte Carlo method-based tool for feature selection. It also allows for the discovery of interdependencies between the relevant features. MCFS-ID is particularly suitable for the analysis of high-dimensional, 'small n large p' transactional and biological data. M. Draminski, J. Koronacki (2018) <doi:10.18637/jss.v085.i12>.

License GPL-3

URL www.ipipan.eu/staff/m.draminski/mcfs.html

LazyData yes

NeedsCompilation no

Author Michal Draminski [aut, cre],
Jacek Koronacki [aut],
Julian Zubek [ctb]

Maintainer Michal Draminski <michal.draminski@ipipan.waw.pl>

Repository CRAN

Date/Publication 2019-05-12 05:00:13 UTC

R topics documented:

  artificial.data ................................................................. 2
  build.idgraph ................................................................. 3
artificial.data

Description

Creates data.frame with artificial data. The last six columns are nominal and highly correlated to feature ‘class’. This data set consists of objects from 3 classes, A, B and C, that contain 40, 20, 10 objects, respectively (70 objects altogether). For each object, 6 binary features (A1, A2, B1, B2, C1 and C2) are created and they are ‘ideally’ or ‘almost ideally’ correlated with class feature. If an object’s ‘class’ equals ‘A’, then its features A1 and A2 are set to class value ‘A’; otherwise A1 = A2 = 0. If an object’s ‘class’ is ‘B’ or ‘C’, the processing is analogous, but some random corruption is introduced. For 2 observations from class ‘B’ and both attributes B1/B2, their values ‘B’ are replaced by ‘0’. For 4 observations from class ‘C’ and both attributes C1/C2, their values ‘C’ are replaced by ‘0’. The number of corrupted values for each class is defined by corruption parameter. The data also contains additional rnd_features = 500 random numerical features with uniformly [0,1] distributed values.

Usage

artificial.data(rnd_features = 500, size = c(40, 20, 10),
               corruption = c(0, 2, 4), seed = NA)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rnd_features</td>
<td>number of numerical random features.</td>
</tr>
<tr>
<td>size</td>
<td>size of classes A, B, and C.</td>
</tr>
<tr>
<td>corruption</td>
<td>defines the number of corrupted values for a pairs of columns A1/A2, B1/B2, C1/C2,</td>
</tr>
<tr>
<td>seed</td>
<td>seed for random number generator.</td>
</tr>
</tbody>
</table>
build.idgraph

Value
data.frame with six important features.

Examples
d <- artificial.data(rnd_features = 500)
showme(d)

---

build.idgraph  Constructs interdependencies graph

Description
Constructs the ID-Graph (igraph/idgraph object) from mcfs_result object returned by mcfs function. The number of top features included and the number of ID-Graph edges can be customized.

Usage
build.idgraph(mcfs_result,
              size = NA,
              size_ID = NA,
              self_ID = FALSE,
              outer_ID = FALSE,
              orphan_nodes = FALSE,
              size_ID_mult = 3,
              size_ID_max = 100)

Arguments
mcfs_result  results returned by mcfs function.
size  number of top features to select. If size = NA, then size is defined by mcfs_result$cutoff_value parameter.
size_ID  number of interdependencies (edges in ID-Graph) to be included. If size_ID = NA, then parameter size_ID is defined by multiplication size_ID_mult*size.
self_ID  if self_ID = TRUE, then include self-loops from ID-Graph.
outer_ID  if outer_ID = TRUE, then include all interactions between a feature from the top set features (defined by size parameter) with any other feature.
orphan_nodes  if plot_all_nodes = TRUE, then include all nodes, even if they are not connected to any other node (isolated nodes).
size_ID_mult  If size_ID_mult = 3 there will be 3 times more edges than features (nodes) presented on the ID-Graph. It works only if size = NA and size_ID = NA
size_ID_max  maximum number of interactions to be included from ID-Graph (the upper limit).
Value

`igraph/idgraph` S3 object that can be: plotted in R, exported to graphML (XML format) or saved as csv or rds files.

Examples

```r
## Not run: ###dontrunbegin

# create input data
data <- artificial.data(rnd_features = 10)
showme(data)

# Parametrize and run MCFS-ID procedure
result <- mcfs(class = 1, data, cutoffPermutations = 0, featureFreq = 50,
buildID = TRUE, finalCV = FALSE, finalRuleset = FALSE,
threadsNumber = 2)

# build interdependencies graph for top 6 features
# and top 12 interdependencies and plot all nodes
gid <- build.idgraph(result, size = 6, size_ID = 12, orphan_nodes = TRUE)
plot(gid, label_dist = 1)

# Export graph to graphML (XML structure)
path <- tempdir()
igraph::write.graph(gid, file = file.path(path, "artificial.graphml"),
format = "graphml", prefixAttr = FALSE)

## End(Not run)###dontrunend
```

---

**export.plots**

Exports MCFS-ID result plots

**Description**

Saves all MCFS-ID result plots in the specified directory.

**Usage**

```r
export.plots(mcfs_result, data = NULL, idgraph = NULL,
path, label = "mcfs", color = "darkred",
size = NA, image_width = 8, image_height = 6,
plot_format = c("pdf","svg","png"), cex = 1)
```

**Arguments**

- `mcfs_result`: result from `mcfs` function.
- `data`: input data frame used to produce `mcfs_result`. 
idgraph  
  idgraph/igraph S3 object representing feature interdependencies. This object is
  produced by build.idgraph function.

path  
  path to the where plot files should be saved.

label  
  a common prefix label of all plot files.

color  
  it defines main color of all plots.

size  
  number of features to plot.

image_width  
  width of plots (in inches).

image_height  
  height of plots (in inches).

plot_format  
  image format of plot files - one of the following: "pdf","svg","png".

cex  
  size of fonts.

Examples

```r
# Not run: #dontrunbegin

# create input data
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
result <- mcfs(class~, adata, cutoffPermutations = 0, featureFreq = 50,
               finalCV = FALSE, finalRuleset = FALSE, threadsNumber = 2)

# build interdependencies graph for top 6 features
# and top 12 interdependencies and plot all nodes
gid <- build.idgraph(result, size = 6, size_ID = 12, orphan_nodes = TRUE)

#export plot files
export.plots(result, adata, idgraph = gid, path = tempdir(), label = "mcfs", color = "darkgreen")

# End(Not run)#dontrunend
```

Description

Saves MCFS-ID result into set csv files.

Usage

```r
export.result(mcfs_result, path = "./", label = "rmcfs", zip = TRUE)
```
Arguments

mcfs_result  result of the MCFS-ID experiment returned by mcfs function.
path        path to the MCFS-ID results files. This parameter can also point to the zip result file.
label       label of the experiment and common name for output files.
zip         if = TRUE, saves all results data as one zip file.

Examples

```r
## Not run: ##dontrunbegin

# create input data
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
result <- mcfs(class ~ ., adata, cutoffPermutations = 0, featureFreq = 10,
               finalCV = FALSE, finalRuleset = FALSE, threadsNumber = 2)

# Export and import R result to/from files
path <- file.path(tempdir(), "artificial.zip")
export.result(result, path = path)
result <- import.result(path = path)

## End(Not run)##dontrunend
```

fix.data

Fixes input data values, column names and attributes types

Description

Fixes any input data to prepare them to export to ARFF/ADX formats. If after exporting data to ARFF/ADX formats there are some problems in running Java MCFS or WEKA, try to use this function before. This function fixes data values (e.g. space " " is replaced by "]") and data types (e.g. all Date columns converted to character in R).

Usage

```r
fix.data(x,
     type = c("all", "names", "values", "types"),
     source_chars = c(" ", "\", ",", "/", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "]", "])",
     destination_char = " ",
     numeric_class = c("difftime"),
     nominal_class = c("factor", "logical", "Date", "POSIXct", "POSIXt"))
```
Arguments

- **x**: input data frame to be fixed.
- **type**: 
  - `all`: fixes: column names, data values, data types.
  - `names`: fixes only column names. All characters determined by `source_chars` parameter are replaced by `destination_char` (e.g. space " " is replaced by ".").
  - `values`: fixes only data values. All characters determined by `source_chars` parameter are replaced by `destination_char` (e.g. space " " is replaced by ".").
  - `types`: fixes only data types (e.g. all possible nominal columns as (Date or logical) converted to character).
- **source_chars**: characters that will be replaced in column names and data values.
- **destination_char**: character that will be inserted in column names and data values.
- **numeric_class**: vector of class labels to be casted as numeric.
- **nominal_class**: vector of class labels to be casted as character.

Value

data.frame with fixed values and types (depends on type parameter).

Examples

```r
## Not run: #dontrunbegin

# create artificial data
adata <- artificial.data(rnd_features = 1000)

# Fix data types and data values - remove "," "/" from values and fix data types
# This function may help if mcfs has any problems with input data
adata.fixed <- fix.data(adata)

## End(Not run)#dontrunend
```

---

**import.result**

*Reads csv result files produced by the MCFS-ID Java module*

**Description**

Reads csv result files produced by the MCFS-ID Java module.

**Usage**

```r
import.result(path = "./", label = NA)
```
Arguments

- **path**: path to the MCFS-ID results files. This parameter can also point to the zip result file.
- **label**: experiment label for results files (name of the data).

Value

the result of the MCFS-ID experiment returned by `mcfs` function.

Examples

```r
## Not run: ##dontrunbegin

# create input data
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
result <- mcfs(class, adata, cutoffPermutations = 0, featureFreq = 10,
               finalCV = FALSE, finalRuleset = FALSE, threadsNumber = 2)

data.frame(result, cutoff = result$cutoff)

# Export and import R result to/from files
path <- file.path(tempdir(), "artificial.zip")
export.result(result, path = path)
result <- import.result(path = path)

## End(Not run)##dontrunend
```

mcfs

---

**MCFS-ID** *(Monte Carlo Feature Selection and Interdependency Discovery)*

Description

Performs Monte Carlo Feature Selection (MCFS-ID) on a given data set. The data set should define a classification problem with discrete/nominal class labels. This function returns features sorted by RI as well as cutoff value, ID-Graph edges that denote interdependencies (ID), evaluation of top features and other statistics.

Usage

```r
mcfs(formula, data,
     attrWeights = NULL,
     projections = 'auto',
     projectionSize = 'auto',
     featureFreq = 100,
```
mcfs

splits = 5,
splitSetSize = 500,
balance = 'auto',
cutoffMethod = c("permutations", "criticalAngle", "kmeans", "mean", "contrast"),
cutoffPermutations = 20,
mode = 1,
buildID = TRUE,
finalRuleset = TRUE,
finalCV = TRUE,
finalCVsetSize = 1000,
seed = NA,
threadsNumber = 4)

Arguments

formula specifies decision attribute and relation between class and other attributes (e.g. class~.). The target attribute can be nominal (then MCFS-ID uses decision tree) or numerical (then MCFS-ID uses regression tree).
data defines input data.frame containing all features with decision attribute included. This data.frame must contain proper types of columns. Columns character, factor, Date, POSIXct, POSIXt are treated as nominal/categorical and remaining columns as numerical/continuous. Decision attribute defined by formula can be nominal or numerical.
attrWeights defines vector of length = ncol(data) of attributes weights - weight 10 denotes 10 times larger chance for the attribute to be selected to the random subset than if weight equals to 1.
projections defines the number of subsets (projections) with randomly selected features. This parameter is usually set to a few thousands and is denoted in the paper as s. By default it is set to 'auto' and this value is based on size of input data set and featureFreq parameter.
projectionSize defines the number of features in one subset. It can be defined by an absolute value (e.g. 100 denotes 100 randomly selected features) or by a fraction of input attributes (e.g. 0.05 denotes 5% of input features). This parameter is denoted in the paper as m. If is set to 'auto' then projectionSize equals to \( \sqrt{d} \), where d is the number of input features. Minimum number of features in one subset is 1.
featureFreq determines how many times each input feature should be randomly selected when projections = 'auto'.
splits defines the number of splits of each subset. This parameter is denoted in the paper as t. The size of the training set in the input subset is always set on 66%.
splitSetSize determines whether to limit input dataset size. It helps to speedup computation for data sets with a large number of objects. If the parameter is larger than 1, it determines the number of objects that are drawn at random for each of the \( s \cdot t \) decision trees. If splitSetSize = 0 then the MCFS uses all objects in each iteration.
balance determines the way to balance classes. It should be set to 2 or higher if input dataset contains heavily unbalanced classes. Each subset s will contain all the
objects from the least frequent class and randomly selected set of objects from each of the remaining classes. This option helps to select features that are important for discovering a relatively rare class. The parameter defines the maximal imbalance ratio. If the ratio is set to 2, then subset $s$ will contain the number of objects from each class (but the least frequent one) proportional to the square root of the class size $size(c)^{1/2}$. If $balance = 0$ then balancing is turned off. If $balance = 1$ it is on but does not change the size of classes. Default value is ‘auto’.

cutoffMethod determines the final cutoff method. Default value is ‘permutations’. The methods of finding cutoff value between important and unimportant attributes are the following:

- **permutations** - the method consists in permuting the decision attribute at least 20 times and running the MCFS-ID algorithm for each permutation. The set of the maximal RIs from all these experiments is assumed approximately normally distributed and a critical value based on the the one-sided (upper-tailed) Student’s t-test (at 95% significance level) is provided. A feature is declared informative if its RI in the original ranking (without any permutation) exceeds the obtained critical value. A more detailed description of this method is included in the paper.

- **criticalAngle** - critical angle method is based on the plot of the features’ RIs in decreasing order of size, with the corresponding features equally spaced along the abscissa. The plot can be seen as piecewise linear function, where each linear segment joins two neighboring RIs. Roughly speaking, the cutoff (placed on the abscissa) corresponds to this point on the plot where the slope of consecutive segments changes significantly.

- **kmeans** - the method is based on clustering the RI values into two clusters by the k-means algorithm. It sets the cutoff where the two clusters are separated. This method is quite valuable when data contains a subset of very informative features.

- **mean** - cutoff value is set on mean values obtained from all the implemented methods.

- **contrast** - This method adds 10% contrast (pure numerical random) attributes to the data then MCFS-ID is executed. Position of top 5% of them determines cutoff value. Usually it gives the largest cutoff because it select all attributes that are more informative than pure noise.

cutoffPermutations determines the number of permutation runs. It needs at least 20 permutations (cutoffPermutations = 20) for a statistically significant result. Minimum value of this parameter is 3, however if it is 0 then permutations method is turned off.

mode determines number of stages in MCFS filtering. If mode = 2 then MCFS is running new method that is based on two stage filtering. This method is much faster for BIG DATA - 1st stage filtering is performed based on contrast attributes (same as cutoffMethod = ‘contrast’) and 2nd stage is performed based on permutations experiments. If mode = 1 then it always runs one stage filtering the same as in rmcfs 1.2.x.
buildID if = TRUE, Interdependencies Discovery is on and all ID-Graph edges are collected.

finalRuleset if = TRUE, classification rules (by ripper algorithm) are created on the basis of the final set of features.

finalCV if = TRUE, it runs 10 folds cross validation (cv) experiments on the final set of features. The following set of classifiers is used: C4.5, NB, SVM, kNN, logistic regression and Ripper.

finalCVsetSize limits the number of objects used in the final cv experiment. For each out of 3 cv repetitions, the objects are selected randomly from the uniform distribution.

seed seed for random number generator in Java. By default the seed is random. Replication of the result is possible only if threadsNumber = 1.

threadsNumber number of threads to use in computation. More threads needs more CPU cores as well as memory usage is a bit higher. It is recommended to set this value equal to or less than CPU available cores.

Value

data input data.frame limited to the top important features set.
target decision attribute name.
RI data.frame that contains all features with relevance scores sorted from the most relevant to the least relevant. This is the ranking of features.
ID data.frame that contains features interdependencies as graph edges. It can be converted into a graph object by buildNidgraph function.
distances data.frame that contains convergence statistics of subsequent projections.
cmatrix confusion matrix obtained from all s · t decision trees.
cutoff data.frame that contains cutoff values obtained by the following methods: mean, kmeans, criticalAngle, permutations (max RI).
cutoff_value the number of features chosen as informative by the method defined by parameter cutoffMethod.
cv_accuracy data.frame that contains classification results obtained by cross validation performed on cutoff_value features. This data.frame exists if finalCV = T.
permutations this data.frame contains the following results of permutation experiments:
  • perm_x all RI values obtained from all permutation experiments;
  • RI RI obtained for reference MCFS experiment (i.e., the experiment on the original data); p-values from Anderson-Darling normality test applied separately for each feature to the cutoffPermutations RI set;
  • t_test_p p-values from Student-t test applied separately for each feature to the cutoffPermutations RI vs. reference RI. This data.frame exists if parameter cutoffPermutations > P.
jrip classification rules produced by ripper algorithm and related cross validation result obtained for top features.
params all settings used by MCFS-ID.
exec_time execution time of MCFS-ID.
References


Examples

```r
## Not run: ##dontrunbegin

# # Artificial data # #
# # create input data and review it
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
result <- mcf(class ~ ., adata, cutoffPermutations = 3, featureFreq = 50,
             buildID = TRUE, finalCV = FALSE, finalRuleset = FALSE,
             threadsNumber = 2)

# Print basic information about mcfs result
print(result)

# Review cutoff values for all methods
print(result$cutoff)

# Review cutoff value used in plots
print(result$cutoff_value)

# Plot & print out distances between subsequent projections.
# These are convergence MCFS-ID statistics.
plot(result, type = "distances")
print(result$distances)

# Plot & print out 50 most important features and show max RI values from
# permutation experiment.
plot(result, type = "ri", size = 50)
print(head(result$RI, 50))

# Plot & print out 50 strongest feature interdependencies.
plot(result, type = "id", size = 50)
print(head(result$ID, 50))

# Plot features ordered by RI. Parameter 'size' is the number of
# top features in the chart. By default it is set on cutoff_value + 10
plot(result, type = "features", cex = 1)

# Here we set 'size' at fixed value 10.
plot(result, type = "features", size = 10)
```
# Plot cv classification result obtained on top features.
# In the middle of x axis red label denotes cutoff_value.
plot(result, type = "cv", cv_measure = "wacc", cex = 0.8)

# Plot & print out confusion matrix. This matrix is the result of
# all classifications performed by all decision trees on all s*t datasets.
plot(result, type = "cmatrix")

# build interdependencies graph (all default parameters).
gid <- build.idgraph(result)
plot(gid, label_dist = 1)

# build interdependencies graph for top 6 features
# and top 12 interdependencies and plot all nodes
gid <- build.idgraph(result, size = 6, size_ID = 12, orphan_nodes = TRUE)
plot(gid, label_dist = 1)

# Export graph to graphML (XML structure)
path <- tempdir()
igraph::write.graph(gid, file = file.path(path, "artificial.graphml"),
                     format = "graphml", prefixAttr = FALSE)

# Export and import results to/from csv files
export(result, path = path, label = "artificial")
result <- import.result(path = path, label = "artificial")

# Load Alizadeh dataset.
# A 4026 x 62 gene expression data matrix of log-ratio values. The last column contains
# the annotations of the 62 samples with respect to the cancer types C, D, F.
# The data are from the lymphoma/leukemia study of A. Alizadeh et al., Nature 403:503-511 (2000),
alizadeh <- read.csv(file="http://www.ipipan.eu/staff/m.draminski/files/data/alizadeh.csv",
                      stringsAsFactors = FALSE)
showme(alizadeh)

# Fix data types and data values - replace characters such as "," = "" "/" etc.
# from values and column names and fix data types
# This function may help if mcfs has any problems with input data
alizadeh <- fix.data(alizadeh)

# Run MCFS-ID procedure on default parameters.
# For larger real data (thousands of features) default 'auto' settings are the best.
# This example may take 10-20 minutes but this one is a real dataset with 4026 features.
# Set up more threads according to your CPU cores number.
result <- mcfs(class~., alizadeh, featureFreq = 100, cutoffPermutations = 10, threadsNumber = 8)

# Print basic information about mcfs result.
print(result)
# Plot & print out distances between subsequent projections.
plot(result, type="distances")

# Show RI values for top 500 features and max RI values from permutation experiment.
plot(result, type = "ri", size = 500)

# Plot heatmap on top features, only numeric features are presented
plot(result, type = "heatmap", size = 20, heatmap_norm = 'norm', heatmap_fun = 'median')

# Plot cv classification result obtained on top features.
# In the middle of x axis red label denotes cutoff_value.
plot(result, type = "cv", cv_measure = "wacc", cex = 0.8)

# build interdependencies graph.
gid <- build.idgraph(result, size = 20)
plot.idgraph(gid, label_dist = 0.3)

## End(Not run)###dontrunend

---

**plot.idgraph**

*Plots interdependencies graph*

**Description**

Invokes `plot.igraph` with predefined parameters to visualize interdependencies graph (ID-Graph). Standard `plot` function with custom parameters may be used instead of this one.

**Usage**

```r
## S3 method for class 'idgraph'
plot(x,
     label_dist = 0.5,
     color = 'darkred',
     cex = 1, ...)
```

**Arguments**

- **x** `idgraph/igraph` S3 object representing feature interdependencies. This object is produced by `build.idgraph` function.
- **label_dist** space between the node’s label and the corresponding node in the plot.
- **color** it defines color of the graph nodes.
- **cex** size of fonts.
- **...** additional plotting parameters.
Examples

```r
## Not run: ###dontrunbegin

# create input data
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
result <- mcfs(class~, adata, cutoffPermutations = 0, featureFreq = 50,
               finalCV = FALSE, finalRuleset = FALSE, threadsNumber = 2)

# build interdependencies graph for top 6 features
# and top 12 interdependencies and plot all nodes
gid <- build.idgraph(result, size = 6, size_ID = 12, orphan_nodes = TRUE)
plot(gid, label_dist = 1)

## End(Not run)###dontrunend
```

---

**plot.mcfs**

Plots various MCFS result components

---

**Description**

Plots various aspects of the MCFS-ID result.

**Usage**

```r
## S3 method for class 'mcfs'
plot(x, type = c("features", "ri", "id", "distances", "cv", "cmatrix", "heatmap"),
     size = NA,
     ri_permutations = c("max", "all", "sorted", "none"),
     diff_bars = TRUE,
     features_margin = 10,
     cv_measure = c("wacc", "acc", "pearson", "MAE", "RMSE", "SMAPE"),
     heatmap_norm = c('none', 'norm', 'scale'),
     heatmap_fun = c('median', 'mean'),
     color = c('darkred'),
     gg = TRUE,
     cex = 1, ...)
```

**Arguments**

- `x` 'mcfs' S3 object - result of the MCFS-ID experiment returned by `mcfs` function.
- `type` -
  - `features` plots top features set along with their RI. It is a horizontal barplot that shows important features in red color and unimportant in grey.
  - `ri` plots top features set with their RIs as well as max RI obtained from permutation experiments. Red color denotes important features.
• id plots top ID values obtained from the MCFS-ID.
• distances plots distances (convergence diagnostics of the algorithm) between subsequent feature rankings obtained during the MCFS-ID experiment.
• cv plots cross validation results based on top features.
• cmatrix plots the confusion matrix obtained on all $s \cdot t$ trees.
• heatmap plots heatmap results based on top features. Only numeric features can be presented on the heatmap.

size number of features to plot.
ri_permutations
  if type = "ri" and ri_permutations = "max", then it additionally shows horizontal lines that correspond to max RI values obtained from each single permutation experiment.
diff_bars
  if type = "ri" or type = "id" and diff_bars = T, then it shows difference values for RI or ID values.
features_margin
  if type = "features", then it determines the size of the left margin of the plot.
cv_measure
  if type = "cv", then it determines the type of accuracy shown in the plot: weighted or unweighted accuracy ("wacc" or "acc"). If target attribute is numeric it is possible to review one of the following prediction quality measures: ("pearson", "MAE", "RMSE", "SMAPE")
heatmap_norm
  if type = "heatmap", then it defines type of input data normalization 'none' - without any normalization, 'norm' - normalization within range [-1,1], 'scale' - standardization/centering by mean and stdev.
heatmap_fun
  if type = "heatmap", then it determines calculation 'mean' or 'median' within the class to be shown as heatmap color intensity.
color
  it defines main color of the following type of plots: 'ri', 'id', 'heatmap', 'features' and 'cmatrix'.

... additional plotting parameters.

Examples

```r
## Not run: ##dontrun

# Create input data.
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure.
result <- mcfs(class~., adata, cutoffPermutations = 0, featureFreq = 10,
               finalCV = FALSE, finalRuleset = TRUE, threadsNumber = 2)

# Plot & print out distances between subsequent projections.
# These are convergence MCFS-ID statistics.
```
print.mcfs

plot(result, type = "distances")
print(result$distances)

# Plot & print out 50 most important features and show max RI values from # permutation experiment.
plot(result, type = "ri", size = 50)
print(head(result$RI, 50))

# Plot & print out 50 strongest feature interdependencies.
plot(result, type = "id", size = 50)
print(head(result$ID, 50))

# Plot features ordered by RI. Parameter 'size' is the number of # top features in the chart. By default it is set on cutoff_value + 10
plot(result, type = "features", cex = 1)

# Here we set 'size' at fixed value 10.
plot(result, type = "features", size = 10)

# Plot cv classification result obtained on top features. # In the middle of x axis red label denotes cutoff_value. # plot(result, type = "cv", measure = "wacc", cex = 0.8)

# Plot & print out confusion matrix. This matrix is the result of # all classifications performed by all decision trees on all s*t datasets. plot(result, type = "cmatrix")

## End(Not run)###dontrunend

---

**print.mcfs**  
*Prints mcfs result*

### Description

Prints basic information about the MCFS-ID result: top features, cutoff values, confusion matrix obtained for s·t trees and classification rules obtained by *Ripper* (*jrip*) algorithm.

### Usage

```
## S3 method for class 'mcfs'
print(x, ...)
```

### Arguments

- `x`  
  'mcfs' object - result of the MCFS-ID experiment returned by *mcfs* function.
- `...`  
  additional printing parameters.
prune.data

Examples

```r
## Not run: ##dontrunbegin

# create input data
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
result <- mcfs(class=. , adata, cutoffPermutations = 0, featureFreq = 10,
               finalCV = FALSE, finalRuleset = TRUE, threadsNumber = 2)

# Print basic information about mcfs result.
print(result)

## End(Not run)##dontrunend
```

---

**prune.data**

*Filters input data*

**Description**

Selects columns from input data based on the highest RIs of attributes.

**Usage**

```r
prune.data(x, mcfs_result, size = NA)
```

**Arguments**

- `x` input data.frame.
- `mcfs_result` result from `mcfs` function.
- `size` number of top features to select from input data. If `size = NA`, then it is defined
  by `mcfs_result$cut off_value` parameter.

**Value**

data.frame with selected columns.

**Examples**

```r
## Not run: ##dontrunbegin

# create input data
adata <- artificial.data(rnd_features = 10)
showme(adata)

# Parametrize and run MCFS-ID procedure
```
result <- mcfs(class~., adata, cutoffPermutations = 0, featureFreq = 10, finalCV = FALSE, finalRuleset = FALSE, threadsNumber = 2)

head(prune.data(adata, result, size = result$cutoff_value))

## End(Not run)###dontrunend

---

### read.adh

**Reads data from ADH**

**Description**

Imports data from ADH format. This format is based on two files: `adhl` that contains ADX header and `csv` that contains the data.

**Usage**

```r
read.adh(file = "")
```

**Arguments**

- `file` exported filename

**Examples**

```r
## Not run: ###dontrunbegin

# create artificial data
adata <- artificial.data(rnd_features = 10000)
write.adx(adata, file = file.path(tempdir(), "adata.adx"), target = "class")
adata <- read.adx(file = file.path(tempdir(), "adata.adx"))

## End(Not run)###dontrunend
```

---

### read.adx

**Reads data from ADX**

**Description**

Imports data from ADX format.

**Usage**

```r
read.adx(file = "")
```
Arguments

file exported filename

Examples

```r
## Not run: ###dontrunbegin

# create artificial data
adata <- artificial.data(rnd_features = 1000)

writeadx(adata, file = file.path(tempdir(), "adata.adx"), target = "class")
adata <- readadx(file = file.path(tempdir(), "adata.adx"))

## End(Not run)###dontrunend
```

---

**Basic data information**

Description

Prints basic information about the data.frame.

Usage

```r
showme(x, size = 10, show = c("tiles", "head", "tail", "none"))
```

Arguments

- **x**: input data frame.
- **size**: number of rows/columns to be printed.
- **show**: parameters that controls print content.
  - **tiles**: shows top left and bottom right cells (size of both subsets is controlled by size parameter)
  - **head**: shows top size rows
  - **tail**: shows bottom size rows
  - **none**: does not show the content

Examples

```r
# create artificial data
adata <- artificial.data(rnd_features = 1000)
showme(adata)
```
write.adh

**writes data to ADH**

---

**Description**

Exports data into ADH format. This format is based on two files: 'adh' that contains ADX header and 'csv' that contains the data.

**Usage**

```r
write.adh(x, file = "", target = NA, chunk_size = 100000, zip = FALSE)
```

**Arguments**

- `x`: data frame with data
- `file`: exported filename
- `target`: sets target attribute in ADH format. Default value is NA what refers to the last column.
- `chunk_size`: defines size of chunk (number of cells) that are processed and exported. The bigger the value, the function is faster for small data and slower for big data.
- `zip`: whether to create zip archive.

**Examples**

```r
## Not run: ###dontrunbegin

# create artificial data
adata <- artificial.data(rnd_features = 1000)

# Fix input data to be consistent with ARFF and ADX formats.
# It is not necessary but for some data can help to export in proper format.
adata <- fix.data(adata)
write.adh(adata, file = file.path(tempdir(), "adata.adh"), target = "class")

## End(Not run)###dontrunend
```

---

write.adx

**writes data to ADX**

---

**Description**

Exports data into ADX format.
Usage

\texttt{write.adx(x, file = "", target = NA, chunk\_size = 100000, zip = FALSE)}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} data frame with data
  \item \texttt{file} \hspace{1cm} exported filename
  \item \texttt{target} \hspace{1cm} sets target attribute in ADX format. Default value is NA what refers to the last column.
  \item \texttt{chunk\_size} \hspace{1cm} defines size of chunk (number of cells) that are processed and exported. The bigger the value, the function is faster for small data and slower for big data.
  \item \texttt{zip} \hspace{1cm} whether to create zip archive.
\end{itemize}

Examples

```
## Not run: ###dontrunbegin

# create artificial data
adata <- artificial.data(rnd\_features = 1000)

# Fix input data to be consistent with ARFF and ADX formats.
# It is not necessary but for some data can help to export in proper format.
adata <- fix.data(adata)
write.adx(adata, file = file.path(tempdir(), "adata.adx"), target = "class")

## End(Not run)###dontrunend
```

\textbf{write.arff} \hspace{1cm} \textit{Writes data to ARFF}

Description

Exports data into ARFF format. This format is used by Weka Data Mining software \url{http://www.cs.waikato.ac.nz/ml/weka/}.

Usage

\texttt{write.arff(x, file = "", target = NA, chunk\_size=100000, zip = FALSE)}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} data frame with data
  \item \texttt{file} \hspace{1cm} exported filename
  \item \texttt{target} \hspace{1cm} sets target attribute in ARFF format. Default value is NA what refers to the last column.
\end{itemize}
chunk_size  it defines size of chunk (number of cells) that are processed and exported. The bigger the value, the function is faster for small data and slower for big data.

zip whether to create zip archive.

Examples

```r
## Not run: ###dontrunbegin

# create artificial data
data <- artificial.data(rnd_features = 1000)

# Fix input data to be consistent with ARFF and ADX formats.
# It is not necessary but for some data can help to export in proper format.
data <- fix.data(data)
write.arff(data, file = file.path(tempdir(), "data.arff"), target = "class")

## End(Not run)###dontrunend
```
Index

artificial.data, 2

build.idgraph, 3, 5, 11, 14

export.plots, 4
export.result, 5

fix.data, 6

import.result, 7

mcfs, 3, 4, 6, 8, 8, 15, 17, 18

plot.idgraph, 14
plot.mcfs, 15
print.mcfs, 17
prune.data, 18

read.adh, 19
read.adx, 19

showme, 20

write.adh, 21
write.adx, 21
write.arff, 22