Package 'rmcfs'

July 3, 2017

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

Version 1.2.6

Date 2017-07-03

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

Suggests testthat

Imports yaml, ggplot2, reshape2, dplyr, igraph (>= 1.0.1)

SystemRequirements Java (>= 6.0)

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.

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 2017-07-03 16:37:50 UTC

R topics documented:

`alizadeh` *Loads sample data from a lymphoma/leukemia gene expression study*

Description

The sample data for the MCFS-ID algorithm

Usage

```
data(alizadeh)
```

Format

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.

Source

The data are from the lymphoma/leukemia study of A. Alizadeh et al., Nature 403:503-511 (2000), http://llmpp.nih.gov/lymphoma/index.shtml

---

`artificial.data` *Creates artificial dataset*

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

`rnd_features`	number of numerical random features.
`size`	size of classes *A*, *B*, and *C*.
`corruption`	defines the number of corrupted values for a pairs of columns *A1/A2*, *B1/B2*, *C1/C2*,
`seed`	seed for random number generator.

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,
                      plot_all_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.
plot_all_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

```
## Not run:

# 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,
               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, plot_all_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)
```

`export.result` *Saves MCFS-ID result into set csv files*

Description

Saves csv files with result obtained by the MCFS-ID.

Usage

```
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 result *.csv files.
`label`	label of the experiment and common name for output files.
`zip`	if = `TRUE`, saves all results data as one zip file.

Examples

```
# 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 <- tempdir()
export.result(result, path = path, label = "artificial")
result <- import.result(path = path, label = "artificial")
```

`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

```
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

```
# Load alizadeh dataset.
data(alizadeh)
alizadeh <- alizadeh[1:500]

# 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
alizadeh.fixed <- fix.data(alizadeh)
```

`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

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

Arguments

`path`	path to the MCFS-ID results *.csv files.
`label`	experiment label for results files (name of the data).

Value

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

Examples

```
# 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 <- tempdir()
export.result(result, path = path, label = "artificial")
result <- import.result(path = path, label = "artificial")
```

`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. For a detailed description of the MCFS-ID algorithm see citation below. If you want to use dmLab or MCFS-ID in your publication, please cite the paper:

M.Draminski, A.Rada-Iglesias, S.Enroth, C.Wadelius, J. Koronacki, J.Komorowski, 'Monte Carlo feature selection for supervised classification', BIOINFORMATICS 24(1): 110-117 (2008).

Usage

```
mcfs(formula, data,
    projections = 'auto',
    projectionSize = 'auto',
    featureFreq = 150,
    splits = 5,
    splitSetSize = 1000,
    balance = 'auto',
    cutoffMethod = c("permutations", "criticalAngle", "kmeans", "mean"),
    cutoffPermutations = 20,
    buildID = TRUE,
    finalRuleset = TRUE,
    finalCV = TRUE,
    finalCVSetSize = 1000,
    finalCVRepetitions = 3,
    seed = NA,
    threadsNumber = 2)
```

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.
`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 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.

`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.

`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 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 cv repetition, the objects are selected randomly from the uniform distribution.

`finalCVRepetitions`
: defines the number of repetitions of the cv experiment. The more repetitions, the more stable result.

`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 `build.idgraph` function.

`distances`
: *data.frame* that contains convergence statistics of subsequent projections.

`cmatrix`
: confusion matrix obtained from all $s \cdot t$ 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_norm` 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 > 0.

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.

Examples

```
## Not run:
####################################
######### Artificial data ##########
####################################

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

# Parametrize and run MCFS-ID procedure
result <- mcfs(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_norm. 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, plot_all_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(result, path = path, label = "artificial")
result <- import.result(path = path, label = "artificial")

####################################
########## Alizadeh data ###########
####################################

# Load Alizadeh dataset.
data(alizadeh)
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.
result <- mcfs(class~., alizadeh, threadsNumber = 4)

# 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)
```

`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

```
  ## S3 method for class 'idgraph'
plot(x,
          label_dist = 0.5,
          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.
`cex`	size of fonts.
`...`	additional plotting parameters.

Examples

```
  ## Not run:
  # 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, plot_all_nodes = TRUE)
plot(gid, label_dist = 1)


## End(Not run)
```

`plot.mcfs` *Plots various MCFS result components*

Description

Plots various aspects of the MCFS-ID result.

Usage

```
  ## S3 method for class 'mcfs'
plot(x, type = c("ri", "id", "distances", "features", "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'),
        heatmap_colors = c('white', 'red'),
        cex = 1, ...)
```

Arguments

`x`	'mcfs' S3 object - result of the MCFS-ID experiment returned by `mcfs` function.
`type`	• `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. • `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. • `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.
heatmap_colors	if type = "heatmap", then it defines low and hi colors on the heatmap.
cex	size of fonts.
...	additional plotting parameters.

Examples

```
# 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.
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_norm. 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")
```

`print.mcfs` *Prints mcfs result*

Description

Prints basic information about the MCFS-ID result: top features, cutoff values, confusion matrix obtained for $s \cdot t$ 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.

Examples

```
# 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)
```

read.adh *Reads data from ADH*

Description

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

Usage

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

Arguments

file	exported filename

Examples

```
# Load alizadeh dataset.
data(alizadeh)
d <- alizadeh

write.adh(d, file = file.path(tempdir(), "alizadeh.adh"), target = "class")
d <- read.adh(file = file.path(tempdir(), "alizadeh.adh"))
```

read.adx *Reads data from ADX*

Description

Imports data from ADX format.

Usage

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

Arguments

file	exported filename

Examples

```
# Load alizadeh dataset.
data(alizadeh)
d <- alizadeh

write.adx(d, file = file.path(tempdir(), "alizadeh.adx"), target = "class")
d <- read.adx(file = file.path(tempdir(), "alizadeh.adx"))
```

`refine.data` *Filters input data*

Description

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

Usage

```
refine.data(data, mcfs_result, size = NA)
```

Arguments

`data`	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$cutoff_value` parameter.

Value

data.frame with selected columns.

Examples

```
# 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(refine.data(adata, result, size = result$cutoff_value))
```

showme *Basic data information*

Description

Prints basic information about the data.frame.

Usage

```
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

```
# Load alizadeh dataset.
data(alizadeh)
showme(alizadeh)
```

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

```
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

```
# Load alizadeh dataset.
data(alizadeh)
d <- alizadeh

#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.
d <- fix.data(d)
write.adh(d, file = file.path(tempdir(), "alizadeh.adh"), target = "class")
```

---

write.adx *Writes data to ADX*

Description

Exports data into ADX format.

Usage

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

Arguments

x	data frame with data
file	exported filename
target	sets target attribute in ADX 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

```
# Load alizadeh dataset.
data(alizadeh)
d <- alizadeh

#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.
d <- fix.data(d)
write.adx(d, file = file.path(tempdir(), "alizadeh.adx"), target = "class")
```

`write.arff` *Writes data to ARFF*

Description

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

Usage

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

Arguments

`x`	data frame with data
`file`	exported filename
`target`	sets target attribute in ARFF format. Default value is NA what refers to the last column.
`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

```
# Load alizadeh dataset.
data(alizadeh)
d <- alizadeh

#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.
d <- fix.data(d)
write.arff(d, file = file.path(tempdir(), "alizadeh.arff"), target = "class")
```

Index