

# Package ‘MDFS’

May 10, 2023

**Title** MultiDimensional Feature Selection

**Version** 1.5.3

**Date** 2023-05-09

**URL** <https://www.mdfs.it/>

**Description** Functions for MultiDimensional Feature Selection (MDFS):  
calculating multidimensional information gains, scoring variables,  
finding important variables, plotting selection results.  
This package includes an optional CUDA implementation that speeds up  
information gain calculation using NVIDIA GPGPUs.  
R. Piliszek et al. (2019) <[doi:10.32614/RJ-2019-019](https://doi.org/10.32614/RJ-2019-019)>.

**Depends** R (>= 3.4.0)

**License** GPL-3

**SystemRequirements** C++17

**NeedsCompilation** yes

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**Author** Radosław Piliszek [aut, cre],  
Krzysztof Mnich [aut],  
Paweł Tabaszewski [aut],  
Szymon Migacz [aut],  
Andrzej Sulecki [aut],  
Witold Remigiusz Rudnicki [aut]

**Maintainer** Radosław Piliszek <[r.piliszek@uwb.edu.pl](mailto:r.piliszek@uwb.edu.pl)>

**Repository** CRAN

**Date/Publication** 2023-05-09 23:40:06 UTC

## R topics documented:

AddContrastVariables . . . . . 2

as.data.frame.MDFS . . . . .	3
ComputeInterestingTuples . . . . .	3
ComputeInterestingTuplesDiscrete . . . . .	5
ComputeMaxInfoGains . . . . .	7
ComputeMaxInfoGainsDiscrete . . . . .	8
ComputePValue . . . . .	9
Discretize . . . . .	11
GenContrastVariables . . . . .	12
GetRange . . . . .	12
madelon . . . . .	13
MDFS . . . . .	14
mdfs_omp_set_num_threads . . . . .	15
plot.MDFS . . . . .	16
RelevantVariables . . . . .	16
RelevantVariables.MDFS . . . . .	17

## Index 18

---

AddContrastVariables *Add contrast variables to data*

---

### Description

This function is deprecated. Please use GenContrastVariables instead.

### Usage

```
AddContrastVariables(data, n.contrast = max(ncol(data)/10, 30))
```

### Arguments

data	data organized in matrix with separate variables in columns
n.contrast	number of contrast variables (defaults to max of 1/10 of variables number and 30)

### Value

A list with the following key names:

- indices – vector of indices of input variables used to construct contrast variables
- x – data with contrast variables appended to it
- mask – vector of booleans making it easy to select just contrast variables

---

as.data.frame.MDFS     *as.data.frame S3 method implementation for MDFS*

---

**Description**

as.data.frame S3 method implementation for MDFS

**Usage**

```
## S3 method for class 'MDFS'  
as.data.frame(x, ...)
```

**Arguments**

x	an MDFS object
...	ignored

**Value**

data.frame

---

ComputeInterestingTuples  
*Interesting tuples*

---

**Description**

Interesting tuples

**Usage**

```
ComputeInterestingTuples(  
  data,  
  decision = NULL,  
  dimensions = 2,  
  divisions = 1,  
  discretizations = 1,  
  seed = NULL,  
  range = NULL,  
  pc.xi = 0.25,  
  ig.thr = 0,  
  I.lower = NULL,  
  interesting.vars = vector(mode = "integer"),  
  require.all.vars = FALSE,  
  return.matrix = FALSE,
```

```

    stat_mode = "MI",
    average = FALSE
)

```

### Arguments

<code>data</code>	input data where columns are variables and rows are observations (all numeric)
<code>decision</code>	decision variable as a binary sequence of length equal to number of observations
<code>dimensions</code>	number of dimensions (a positive integer; 5 max)
<code>divisions</code>	number of divisions (from 1 to 15)
<code>discretizations</code>	number of discretizations
<code>seed</code>	seed for PRNG used during discretizations (NULL for random)
<code>range</code>	discretization range (from 0.0 to 1.0; NULL selects probable optimal number)
<code>pc.xi</code>	parameter xi used to compute pseudocounts (the default is recommended not to be changed)
<code>ig.thr</code>	IG threshold above which the tuple is interesting (0 and negative mean no filtering)
<code>I.lower</code>	IG values computed for lower dimension (1D for 2D, etc.)
<code>interesting.vars</code>	variables for which to check the IGs (none = all)
<code>require.all.vars</code>	boolean whether to require tuple to consist of only interesting.vars
<code>return.matrix</code>	boolean whether to return a matrix instead of a list (ignored if not using the optimised method variant)
<code>stat_mode</code>	character, one of: "MI" (mutual information, the default; becomes information gain when decision is given), "H" (entropy; becomes conditional entropy when decision is given), "VI" (variation of information; becomes target information difference when decision is given); decides on the value computed
<code>average</code>	boolean whether to average over discretisations instead of maximising (the default)

### Details

If running in 2D and no filtering is applied, this function is able to run in an optimised fashion. It is recommended to avoid filtering in 2D if only it is feasible.

This function calculates what `stat_mode` dictates. When `decision` is omitted, the `stat_mode` is calculated on the descriptive variables. When `decision` is given, the `stat_mode` is calculated on the decision variable, conditional on the other variables. Translate "IG" to that value in the rest of this function's description.

**Value**

A `data.frame` or `NULL` (following a warning) if no tuples are found.

The following columns are present in the `data.frame`:

- `Var` – interesting variable index
- `Tuple.1`, `Tuple.2`, ... – corresponding tuple (up to dimensions columns)
- `IG` – information gain achieved by `var` in `Tuple.*`

Additionally attribute named `run.params` with run parameters is set on the result.

**Examples**

```
ig.1d <- ComputeMaxInfoGains(madelon$data, madelon$decision, dimensions = 1, divisions = 1,
                             range = 0, seed = 0)
ComputeInterestingTuples(madelon$data, madelon$decision, dimensions = 2, divisions = 1,
                         range = 0, seed = 0, ig.thr = 100, I.lower = ig.1d$IG)
```

---

ComputeInterestingTuplesDiscrete  
*Interesting tuples (discrete)*

---

**Description**

Interesting tuples (discrete)

**Usage**

```
ComputeInterestingTuplesDiscrete(
  data,
  decision = NULL,
  dimensions = 2,
  pc.xi = 0.25,
  ig.thr = 0,
  I.lower = NULL,
  interesting.vars = vector(mode = "integer"),
  require.all.vars = FALSE,
  return.matrix = FALSE,
  stat_mode = "MI"
)
```



---

ComputeMaxInfoGains    *Max information gains*

---

### Description

Max information gains

### Usage

```

ComputeMaxInfoGains(
  data,
  decision,
  contrast_data = NULL,
  dimensions = 1,
  divisions = 1,
  discretizations = 1,
  seed = NULL,
  range = NULL,
  pc.xi = 0.25,
  return.tuples = FALSE,
  interesting.vars = vector(mode = "integer"),
  require.all.vars = FALSE,
  use.CUDA = FALSE
)

```

### Arguments

<code>data</code>	input data where columns are variables and rows are observations (all numeric)
<code>decision</code>	decision variable as a binary sequence of length equal to number of observations
<code>contrast_data</code>	the contrast counterpart of data, has to have the same number of observations - not supported with CUDA
<code>dimensions</code>	number of dimensions (a positive integer; 5 max)
<code>divisions</code>	number of divisions (from 1 to 15; additionally limited by dimensions if using CUDA)
<code>discretizations</code>	number of discretizations
<code>seed</code>	seed for PRNG used during discretizations (NULL for random)
<code>range</code>	discretization range (from 0.0 to 1.0; NULL selects probable optimal number)
<code>pc.xi</code>	parameter xi used to compute pseudocounts (the default is recommended not to be changed)
<code>return.tuples</code>	whether to return tuples (and relevant discretization number) where max IG was observed (one tuple and relevant discretization number per variable) - not supported with CUDA nor in 1D

interesting.vars variables for which to check the IGs (none = all) - not supported with CUDA  
 require.all.vars boolean whether to require tuple to consist of only interesting.vars  
 use.CUDA whether to use CUDA acceleration (must be compiled with CUDA)

### Value

A `data.frame` with the following columns:

- IG – max information gain (of each variable)
- Tuple.1, Tuple.2, ... – corresponding tuple (up to dimensions columns, available only when `return.tuples == T`)
- Discretization.nr – corresponding discretization number (available only when `return.tuples == T`)

Additionally attribute named `run.params` with run parameters is set on the result.

### Examples

```
ComputeMaxInfoGains(madelon$data, madelon$decision, dimensions = 2, divisions = 1,
                    range = 0, seed = 0)
```

---

ComputeMaxInfoGainsDiscrete  
*Max information gains (discrete)*

---

### Description

Max information gains (discrete)

### Usage

```
ComputeMaxInfoGainsDiscrete(
  data,
  decision,
  contrast_data = NULL,
  dimensions = 1,
  pc.xi = 0.25,
  return.tuples = FALSE,
  interesting.vars = vector(mode = "integer"),
  require.all.vars = FALSE
)
```



**Arguments**

<code>data</code>	input data where columns are variables and rows are observations (all discrete with the same number of categories)
<code>decision</code>	decision variable as a binary sequence of length equal to number of observations
<code>contrast_data</code>	the contrast counterpart of data, has to have the same number of observations
<code>dimensions</code>	number of dimensions (a positive integer; 5 max)
<code>pc.xi</code>	parameter xi used to compute pseudocounts (the default is recommended not to be changed)
<code>return.tuples</code>	whether to return tuples where max IG was observed (one tuple per variable) - not supported with CUDA nor in 1D
<code>interesting.vars</code>	variables for which to check the IGs (none = all) - not supported with CUDA
<code>require.all.vars</code>	boolean whether to require tuple to consist of only interesting.vars

**Value**

A `data.frame` with the following columns:

- `IG` – max information gain (of each variable)
- `Tuple.1, Tuple.2, ...` – corresponding tuple (up to `dimensions` columns, available only when `return.tuples == T`)
- `Discretization.nr` – always 1 (for compatibility with the non-discrete function; available only when `return.tuples == T`)

Additionally attribute named `run.params` with run parameters is set on the result.

**Examples**

```
ComputeMaxInfoGainsDiscrete(madelon$data > 500, madelon$decision, dimensions = 2)
```

---

ComputePValue

*Compute p-values from information gains and return MDFS*

---

**Description**

Compute p-values from information gains and return MDFS

**Usage**

```

ComputePValue(
  IG,
  dimensions,
  divisions,
  response.divisions = 1,
  df = NULL,
  contrast.mask = NULL,
  ig.in.bits = TRUE,
  ig.doubled = FALSE,
  one.dim.mode = "exp",
  irr.vars.num = NULL,
  ign.low.ig.vars.num = NULL,
  min.irr.vars.num = NULL,
  max.ign.low.ig.vars.num = NULL,
  search.points = 8,
  level = 0.05
)

```

**Arguments**

IG	max conditional information gains
dimensions	number of dimensions
divisions	number of divisions
response.divisions	number of response divisions (i.e. categories-1)
df	vector of degrees of freedom for each variable (optional)
contrast.mask	boolean mask on IG specifying which variables are contrast variables (or NULL if none, otherwise at least 3 variables must be marked)
ig.in.bits	TRUE if input is in binary log (as opposed to natural log)
ig.doubled	TRUE if input is doubled (to follow the chi-squared distribution)
one.dim.mode	'exp' for exponential distribution, 'lin' for linear function of chi-squared or 'raw' for raw chi-squared
irr.vars.num	if not NULL, number of irrelevant variables, specified by the user
ign.low.ig.vars.num	if not NULL, number of ignored low IG variables, specified by the user
min.irr.vars.num	minimum number of irrelevant variables (NULL selects probable optimal number)
max.ign.low.ig.vars.num	maximum number of ignored low IG variables (NULL selects probable optimal number)
search.points	number of points in search procedure for the optimal number of ignored variables
level	acceptable error level of goodness-of-fit one-sample Kolmogorov-Smirnov test (used only for warning)

**Value**

A `data.frame` with class set to `MDFS`. Can be coerced back to `data.frame` using `as.data.frame`.

The following columns are present:

- `IG` – information gains (input copy)
- `chi.squared.p.value` – chi-squared p-values
- `p.value` – theoretical p-values

Additionally the following `attributes` are set:

- `run.params` – run parameters
- `sq.dev` – vector of square deviations used to estimate the number of irrelevant variables
- `dist.param` – distribution parameter
- `err.param` – squared error of the distribution parameter
- `fit.p.value` – p-value of fit

**Examples**

```
ComputePValue(madelon$IG.2D, dimensions = 2, divisions = 1)
```

---

Discretize

*Discretize variable on demand*

---

**Description**

Discretize variable on demand

**Usage**

```
Discretize(data, variable.idx, divisions, discretization.nr, seed, range)
```

**Arguments**

<code>data</code>	input data where columns are variables and rows are observations (all numeric)
<code>variable.idx</code>	variable index (as it appears in data)
<code>divisions</code>	number of divisions
<code>discretization.nr</code>	discretization number (positive integer)
<code>seed</code>	seed for PRNG
<code>range</code>	discretization range

**Value**

Discretized variable.

**Examples**

```
Discretize(madelon$data, 3, 1, 1, 0, 0.5)
```

---

GenContrastVariables *Generate contrast variables from data*

---

### Description

Generate contrast variables from data

### Usage

```
GenContrastVariables(data, n.contrast = max(ncol(data), 30))
```

### Arguments

data	data organized in matrix with separate variables in columns
n.contrast	number of contrast variables (defaults to max of 1/10 of variables number and 30)

### Value

A list with the following key names:

- indices – vector of indices of input variables used to construct contrast variables
- x – data with contrast variables appended to it
- mask – vector of booleans making it easy to select just contrast variables

### Examples

```
GenContrastVariables(madelon$data)
```

---

GetRange *Get the recommended range for multiple discretisations*

---

### Description

Get the recommended range for multiple discretisations

### Usage

```
GetRange(k = 3, n, dimensions, divisions = 1)
```

### Arguments

k	the assumed minimum number of objects in a bucket (the default is the recommended value)
n	the total number of objects considered
dimensions	the number of dimensions of analysis
divisions	the number of divisions of discretisations

**Value**

The recommended range value (a floating point number).

**Examples**

```
GetRange(n = 250, dimensions = 2)
```

---

madelon	<i>An artificial dataset called MADELON</i>
---------	---

---

**Description**

An artificial dataset containing data points grouped in 32 clusters placed on the vertices of a five dimensional hypercube and randomly labeled 0/1.

**Usage**

```
madelon
```

**Format**

A list of two elements:

**data** 2000 by 500 matrix of 2000 objects with 500 features

**decision** vector of 2000 decisions (labels 0/1)

**IG.2D** example 2D IG computed using ComputeMaxInfoGains

**Details**

The five dimensions constitute 5 informative features. 15 linear combinations of those features are added to form a set of 20 (redundant) informative features. There are 480 distractor features called 'probes' having no predictive power.

Included is the original training set with label -1 changed to 0.

**Source**

<https://archive.ics.uci.edu/ml/datasets/Madelon>

MDFS

*Run end-to-end MDFS***Description**

Run end-to-end MDFS

**Usage**

```
MDFS(
  data,
  decision,
  n.contrast = max(ncol(data), 30),
  dimensions = 1,
  divisions = 1,
  discretizations = 1,
  range = NULL,
  pc.xi = 0.25,
  p.adjust.method = "holm",
  level = 0.05,
  seed = NULL,
  use.CUDA = FALSE
)
```

**Arguments**

<code>data</code>	input data where columns are variables and rows are observations (all numeric)
<code>decision</code>	decision variable as a boolean vector of length equal to number of observations
<code>n.contrast</code>	number of contrast variables (defaults to max of 1/10 of variables number and 30)
<code>dimensions</code>	number of dimensions (a positive integer; on CUDA limited to 2–5 range)
<code>divisions</code>	number of divisions (from 1 to 15)
<code>discretizations</code>	number of discretizations
<code>range</code>	discretization range (from 0.0 to 1.0; NULL selects probable optimal number)
<code>pc.xi</code>	parameter xi used to compute pseudocounts (the default is recommended not to be changed)
<code>p.adjust.method</code>	method as accepted by <code>p.adjust</code> ("BY" is recommended for FDR, see Details)
<code>level</code>	statistical significance level
<code>seed</code>	seed for PRNG used during discretizations (NULL for random)
<code>use.CUDA</code>	whether to use CUDA acceleration (must be compiled with CUDA; NOTE: the CUDA version might provide a slightly lower sensitivity due to a lack of native support for <code>contrast_data</code> )

**Details**

In case of FDR control it is recommended to use Benjamini-Hochberg-Yekutieli p-value adjustment method ("BY" in `p.adjust`) due to unknown dependencies between tests.

**Value**

A `list` with the following fields:

- `contrast.indices` – indices of variables chosen to build contrast variables
- `contrast.variables` – built contrast variables
- `MIG.Result` – result of `ComputeMaxInfoGains`
- `MDFS` – result of `ComputePValue` (the MDFS object)
- `statistic` – vector of statistic's values (IGs) for corresponding variables
- `p.value` – vector of p-values for corresponding variables
- `adjusted.p.value` – vector of adjusted p-values for corresponding variables
- `relevant.variables` – vector of relevant variables indices

**Examples**

```
MDFS(madelon$data, madelon$decision, dimensions = 2, divisions = 1,  
      range = 0, seed = 0)
```

---

```
mdfs_omp_set_num_threads  
      Call omp_set_num_threads
```

---

**Description**

Call `omp_set_num_threads`

**Usage**

```
mdfs_omp_set_num_threads(num_threads)
```

**Arguments**

`num_threads`     input data where columns are variables and rows are observations (all numeric)

plot.MDFS

*Plot MDFS details*

---

**Description**

Plot MDFS details

**Usage**

```
## S3 method for class 'MDFS'  
plot(x, plots = c("ig", "c", "p"), ...)
```

**Arguments**

x	an MDFS object
plots	plots to plot (ig for max IG, c for chi-squared p-values, p for p-values)
...	passed on to <a href="#">plot</a>

---

RelevantVariables

*Find indices of relevant variables*

---

**Description**

Find indices of relevant variables

**Usage**

```
RelevantVariables(fs, ...)
```

**Arguments**

fs	feature selector
...	arguments passed to methods

**Value**

indices of important variables



---

RelevantVariables.MDFS

*Find indices of relevant variables from MDFS*

---

### Description

Find indices of relevant variables from MDFS

### Usage

```
## S3 method for class 'MDFS'  
RelevantVariables(fs, level = 0.05, p.adjust.method = "holm", ...)
```

### Arguments

fs	an MDFS object
level	statistical significance level
p.adjust.method	method as accepted by <a href="#">p.adjust</a> ("BY" is recommended for FDR, see <a href="#">Details</a> )
...	ignored

### Details

In case of FDR control it is recommended to use Benjamini-Hochberg-Yekutieli p-value adjustment method ("BY" in [p.adjust](#)) due to unknown dependencies between tests.

### Value

indices of relevant variables

# Index

## \* datasets

madelon, [13](#)

AddContrastVariables, [2](#)

as.data.frame, [11](#)

as.data.frame.MDFS, [3](#)

attributes, [11](#)

ComputeInterestingTuples, [3](#)

ComputeInterestingTuplesDiscrete, [5](#)

ComputeMaxInfoGains, [7](#)

ComputeMaxInfoGainsDiscrete, [8](#)

ComputePValue, [9](#)

data.frame, [5](#), [6](#), [8](#), [9](#), [11](#)

Discretize, [11](#)

GenContrastVariables, [12](#)

GetRange, [12](#)

list, [15](#)

madelon, [13](#)

MDFS, [14](#)

mdfs\_omp\_set\_num\_threads, [15](#)

NULL, [5](#), [6](#)

p.adjust, [14](#), [15](#), [17](#)

plot, [16](#)

plot.MDFS, [16](#)

RelevantVariables, [16](#)

RelevantVariables.MDFS, [17](#)