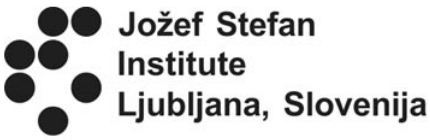
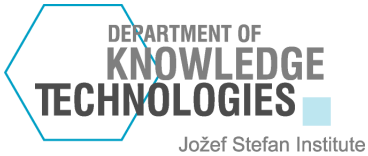
IJS delovno poročilo
IJS DP-13728

2022

Marko Bohanec

DEXiR: A Package for Using DEXi Models in R





Package 'DEXiR'

January 25, 2022

Title DEXi Library for R

Version 0.0.0.9000

Description DEXiR is a software package for using DEXi models in R. DEXi models are hierarchical qualitative multi-criteria decision models developed according to the method DEX (Decision EXpert, see https://en.wikipedia.org/wiki/Decision_EXpert), using the program DEXi (http://kt.ijs.si/MarkoBohanec/dexi.html).
A typical workflow with DEXiR consists of:
(1) reading a .dxi file, previously made using the DEXi software (function read_dexi()),
(2) making a data frame containing input values of one or more decision alternatives, and
(3) evaluating those alternatives (function evaluate()).
DEXiR is restricted to using models produced externally by the DEXi software and does not provide functionality for creating and/or editing DEXi models directly in R.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Imports xml2,
methods,
stringr

Collate 'DEXiR.R'
'DexiClasses.R'
'DexiAlternatives.R'
'DexiFunctions.R'
'DexiScales.R'
'DexiAttributes.R'
'DexiData.R'
'DexiUtils.R'
'DexiEvaluate.R'
'DexiModels.R'
'DexiValues.R'

R topics documented:

`DEXiR-package` *DEXiR: A package for using DEXi models in R*

Description

DEXiR is a software package for using DEXi models in R. The main function is evaluating decision alternatives using a model previously developed by DEXi software.

DEXi Models

DEXi models are hierarchical qualitative rule-based multi-criteria decision models developed using the method DEX (Decision EXpert, https://en.wikipedia.org/wiki/Decision_EXpert), using the program DEXi (http://kt.ijs.si/MarkoBohanec/dexi.html).

In general, a DEXi model consists of a hierarchy of qualitative (symbolic linguistic, discrete) variables, called *attributes*. Each attribute represents some observable property (such as Price or Performance) of decision alternatives under study. An attribute can take values from a set of words (such

as "low; medium; high" or "unacc; acc; good; exc"), which is usually small (up to five elements) and preferentially ordered from "bad" to "good" values.

The *hierarchy* of attributes represents a decomposition of a decision problem into sub-problems, so that higher-level attributes depend on the lower-level ones. Consequently, the terminal nodes represent inputs, and non-terminal attributes represent the outputs of the model. Among these, the most important are one or more root attributes, which represent the final evaluation(s) of the alternatives.

The *evaluation* of decision alternatives (i.e., hierarchical aggregation of values from model inputs to outputs) is governed by *decision rules*, defined for each non-terminal attribute by the creator of the model (usually referred to as a "decision maker").

Terminological remarks

DEX DEX (Decision EXpert) refers to a general multi-attribute decision modeling method, characterized by using qualitative attribute hierarchies and decision tables. For further information, see (Trdin, Bohanec, 2018)).

DEXi DEXi ("DEX for instruction") refers to DEXi software. DEXi implements a subset of DEX, for instance, it is restricted to set-based evaluation methods. DEXi supports the creation and editing of *DEXi models*, which are saved on `.dxi` files and subsequently read by DEXiR for processing in R. For further information on DEXi, see http://kt.ijs.si/MarkoBohanec/dexi.html and (Bohanec, 2020).

DEXiR DEXiR is this R package. It is capable of reading and processing DEXi models with some extensions towards the full DEX (for example, using value distributions).

DEXiR Functionality

Models developed using the DEXi software are stored in XML-formatted `.dxi` files. In order to use DEXi models in R, DEXiR supports the following tasks:

1. Reading DEXi models from `.dxi` files into the R environment, using `read_dexi`.
2. Making data frames containing data (both input and output) about considered decision alternatives, using `set_alternative`.
3. Evaluating decision alternatives, using `evaluate`.

By default, evaluation is based on sets, which is a standard evaluation procedure of DEXi. DEXiR extends this by supporting:

- evaluations using probabilistic and fuzzy value distributions (see `evaluate`);
- "pruned" evaluation, when the evaluation starts from selected non-terminal attribute(s) upwards.

Limitations

DEXiR has been designed to facilitate *using* DEXi models in R produced externally by the DEXi software. DEXiR does not provide any explicit means for creating and/or editing DEXi models in R.

A typical DEXiR workflow

This example uses a simple DEXi model for evaluating cars, which is distributed together with the DEXi software (including DEXiR) and is used throughout DEX literature to illustrate the methodological approach (https://en.wikipedia.org/wiki/Decision_EXpert).

First, this model is loaded into R and printed as follows:

```
> Car <- read_dexi("data/Car.dxi")
> Car
DEXi Model:  CAR_MODEL
Description: Car demo
index id          structure          scale                     funct
  [1] CAR_MODEL   CAR_MODEL
  [2] CAR         +- CAR             unacc; acc; good; exc (+) 12 3x4
  [3] PRICE         |- PRICE         high; medium; low (+)     9 3x3
  [4] BUY.PRICE     | |- BUY.PRICE   high; medium; low (+)
  [5] MAINT.PRICE   | +- MAINT.PRICE high; medium; low (+)
  [6] TECH.CHAR.    +- TECH.CHAR.    bad; acc; good; exc (+)   9 3x3
  [7] COMFORT         |- COMFORT     small; medium; high (+)   36 3x4x3
  [8] X.PERS          | |- #PERS     to_2; 3-4; more (+)
  [9] X.DOORS         | |- #DOORS    2; 3; 4; more (+)
 [10] LUGGAGE         | +- LUGGAGE   small; medium; big (+)
 [11] SAFETY          +- SAFETY      small; medium; high (+)
```

Rows in the table correspond to individual attributes. The columns represent the following:

`index` Indices of attributes.

`id` Unique attribute names, generated by DEXiR from original DEXi names, in order to provide syntactically correct variable names in R and allow unambiguous referencing of attributes.

codestructure The hierarchical structure of attributes, named as in the original DEXi model.

codescale Value scales associated with each attribute. The symbol "(+)" indicates that the corresponding scale is ordered preferentially in increasing order.

codefunct Information about the size (number of rules) and dimensions of the corresponding decision tables.

Looking at the structure of attributes, please notice that the attribute at index `[1]` is virtual and does not actually appear in the original DEXi model. It is necessary in DEXiR to facilitate models that have multiple root attributes. The "real" root of the Car model is actually `[2]` CAR. It depends on two lower-level attributes, PRICE and TECH.CHAR. These are decomposed further. Overall, the model consists of

- six input (*basic*) attributes: BUY.PRICE, MAINT.PRICE, X.PERS, X.DOORS, LUGGAGE and SAFETY, and
- four output (*aggregate*) attributes: CAR, PRICE, TECH.CHAR. and COMFORT.

Among the latter, CAR is the most important and represents the overall evaluation of cars.

The next step usually consists of defining a data frame representing decision alternatives (i.e., cars in this case). The Car model already comes with a data table about two cars:

```
> Car$alternatives
  name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 Car1   4     3         2           3          4       3      3       3       3      3
2 Car2   3     2         2           2          3       3      3       3       3      2
```

In this data frame, attribute values are represented by ordinal numbers w.r.t. the corresponding scales. A more readable output can be made using `DexiModel$as_character`:

```
> Car$as_character(Car$alternatives)
  name  CAR  PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 Car1  exc    low    medium         low        exc    high   more       4     big   high
2 Car2 good medium    medium      medium       good    high   more       4     big medium
```

This data can be edited using common R data.frame functions. Also, DEXiR provides the method `DexiModel$alternative` for defining a single decision alternative, for example:

```
> alt <- Car$alternative("MyCar1",
         BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4",
         LUGGAGE=2, SAFETY="medium")
> alt
    name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1  NA    NA         3           2         NA      NA      3       3       2      2
```

Finally, such data tables can be evaluated using `DexiModel$evaluate`:

```
> eval <- Car$evaluate(alt)
> eval
    name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1   4     3         3           2          3       3      3       3       2      2
> Car$as_character(eval)
    name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1 exc   low       low      medium       good    high   more       4  medium medium
```

On the use of values in DEXi models

DEXi values are used throughout DEXi models. They provide input values and carry results of evaluations in data frames that contain data about decision alternatives. Values are also used in definitions of `DexiFunction`s and are returned by `DexiFunction$evaluate` when evaluating some function for a given set of arguments.

In DEXi, values are always bound to the context provided by a `DexiScale`. Since each fully defined `DexiAttribute` is associated with some scale, we can generalize the scale context to attributes and speak about "assigning some value to an attribute".

The scale type determines the type and possible range of values that can be assigned to an attribute. DEXiR implements two scale types: `DexiContinuousScale` and `DexiDiscreteScale`. Regarding the values, the former is really simple: it allows assigning any single real number to the corresponding attribute. In other words, continuous DEXi values are of type numeric(1).

`DexiDiscreteScale` is the main scale type used throughout DEXi models and supports a wider range of value types.

The "normal" and most common discrete value is a "single qualitative value". For illustration, let us use the scale composed of four qualitative values: `"unacc"`, `"acc"`, `"good"`, `"exc"`. Then, "a single qualitative value" denotes one of these words. Internally in DEXiR, such values are not represented by character strings, but rather by ordinal numbers, so that ord(`"unacc"`) = 1, ord(`"acc"`) = 2, etc. Some DEXiR functions can convert between the two representations, see `DexiModel$as_character` and `set_alternative`.

In order to cope with missing, incomplete or uncertain data, DEX extends the concept of single values to value *sets* and *distributions*. In DEXiR, wherever it is possible to use a single qualitative value, it is also possible to use a value set or distribution. This is the main reason that all DEXiR data

structures related to DEXi values are represented by lists rather than plain vectors. This includes all data frames that represent decision alternatives and all functions that return qualitative values. Also note that while sets are fully implemented in the current DEXi software, distributions are not and are thus considered extensions towards the full DEX method.

A *DEXi value set* is a subset of the full range of a `DexiDiscreteScale` values. For the above example, the full range of ordinal values is `1:4`, and some possible subsets are `c(2)`, `c(2,4)`, `c(1,2,3)` and `1:4`. Internally, sets are represented by plain integer vectors or plain numeric vectors containing integer numbers.

A *DEXi value distribution* associates each `DexiDiscreteScale` value with some number, generally denoted p and normally expected to be in the [0,1] interval. Depending on the context and used evaluation method (see `evaluate`), p can be interpreted as *probability* or *fuzzy set membership*. In DEXiR, value distributions are represented using the S3 class "distribution" (see `distribution`). For example, `distribution(0.5,0,0.2,0.3)` represents a value distribution over the above scale example, assigning $p = 0.5$ to `"unacc"`, $p = 0.0$ to `"acc"`, $p = 0.2$ to `"good"` and $p = 0.3$ to `"exc"`.

Remarks:

- The value `distribution(0.5,0,0.2,0.3)` is internally represented as `c(0.5,0,0.2,0.3)`, whose `class()` is `"distribution"`.
- Using a special class for distributions is necessary to distinguish them from sets. For instance, the notation `c(1,1)` is ambiguous and would be interpreted differently as a set or distribution.
- Some DEXiR functions (see `DexiModel$as_character` and `set_alternative`) support the formulation of distributions in the form of named vectors or lists, for instance ‘`list(unacc=0.5,good=0.2,exc=0.3)`.
- In data frames that contain data about decision alternatives, numeric vectors that contain non-integer values are implicitly interpreted as distributions rather than sets.

Examples of using value sets and distributions

First, let us consider a car for which we have no evidence about its possible maintenance costs. For the value pf `MAINT.PRICE`, we may use `"*"`, which denotes the full range of the corresponding attribute values (equivalent to `1:3` or `c(1,2,3)` in this case). Notice how the evaluation method considers all the possible values of `MAINT.PRICE` and propagates them upwards.

```
alt <- Car$alternative("MyCar1a",
      BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt)
    name  CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1a 1, 4  1, 3         3     1, 2, 3          3       3      3       3       2      2
```

The above evaluation result is not really useful, as the car turns out to be `c(1,4)`, that is, either `"unacc"` or `"exc"`, depending on maintenance costs. Thus, let us try using value distribution for `MAINT.PRICE`, telling DEXiR that low maintenance costs are somewhat unexpected ($p = 0.1$) and that medium costs ($p = 0.6$) are more likely than high ($p = 0.3$). Using the evaluation method `"prob"` (where p's are interpreted as probabilities) gives the following results:

```
alt <- Car$alternative("MyCar1b",
       BUY.PRICE="low", MAINT.PRICE=distribution(0.1, 0.6, 0.3),
       X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt, method = "prob")
   name           CAR      PRICE BUY.PRICE  MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1b 0.1, 0.0, 0.0, 0.9 0.1, 0.0, 0.9       3 0.1, 0.6, 0.3 0, 0, 1, 0 0, 0, 1     3      3      2      2
```

In this case, the final evaluation of `CAR` is `distribution(0.1,0.0,0.0,0.9)`, that is, `list(unacc=0.1,exc=0.9)`. It is much more likely that `MyCar1b` is `"exc"` than `"unacc"`.

References

- *Decision EXpert*. Wikipedia, https://en.wikipedia.org/wiki/Decision_EXpert.
- *DEXi: A Program for Multi-Attribute Decision Making*. http://kt.ijs.si/MarkoBohanec/dexi.html.
- Bohanec, M.: *DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 5.04*. IJS Report DP-13100, Jožef Stefan Institute, Ljubljana, 2020. http://kt.ijs.si/MarkoBohanec/pub/DEXiManual504.pdf.
- Trdin, N., Bohanec, M.: Extending the multi-criteria decision making method DEX with numeric attributes, value distributions and relational models. *Central European Journal of Operations Research*, 1-24, 2018. https://doi.org/10.1007/s10100-017-0468-9.

`and_function` *and_function*

Description

Determine the function to be used in the conjunctive aggregation step of `evaluate`.

Usage

```
and_function(method, and = NULL)
```

Arguments

`method`	One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`.
`and`	Some conjunctive aggregation function of the form `function(num_vector)`, or `NULL`.

Value

Returns `and` if not `NULL`. Otherwise, it determines the result depending on `method`:

`"set": function(x) 0`

`"prob": prod`

`"fuzzy": min`

`"fuzzynorm": min`

Fails with an error if the result is not an R function.

See Also

`evaluate`, `or_function`.

att_names *att_names*

Description

att_names

Usage

```
att_names(atts, use_id = TRUE)
```

Arguments

atts	A vector of DexiAttributes.
use_id	Determines whether to return attribute IDs or original DEXi names.

Value

A character vector of attribute IDs or names.

bounded_scale_value *bounded_scale_value*

Description

bounded_scale_value is a wrapper around scale_value that makes sure that the resulting values lie within the bounds set up by the scale.

Usage

```
bounded_scale_value(value, scale)
```

Arguments

value	Any DEXi value, including value sets and distributions.
scale	A DexiScale or derived object.

Value

For continuous scales, value is returned "as is". For discrete scales, all elements of value that lie outside of scale$full_range() are removed. If this results in an empty value set or distribution, NULL is returned.

See Also

scale_value

Examples

```
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
bounded_scale_value(NA, scl)                              # NA
bounded_scale_value(1, scl)                               # 1
bounded_scale_value(4, scl)                               # NULL
bounded_scale_value(c(0, 1, 3,  4, 5), scl)               # c(1, 3)
bounded_scale_value(distribution(0.1, 0.2, 0.3, 0.4), scl) # distribution(0.1, 0.2, 0.3)
```

`default_quality` *default_quality*

Description

Make a default discrete scale quality vector depending on the scale's `order` and `nvals`.

Usage

```
default_quality(order, nvals)
```

Arguments

`order`	character(1), one of "ascending", "descending" or "none".
`nvals`	integer, the number of qualitative values of considered `DexiDiscreteScale`.

Value

character vector of length `nvals`, containing `"bad"`, `"none"` or `"good"`.

Examples

```
default_quality("ascending", 5)
default_quality("descending", 5)
default_quality("none", 5)
default_quality("ascending", 2)
default_quality("ascending", 1)
```

`DexiAttribute-class` *DexiAttribute*

Description

`DexiAttribute` is a RC class representing a DEXi attribute in R.

Details

In a DEXi model, attributes are variables that represent observed properties of decision alternatives. Attributes are structured in a tree, so each attribute may, but need not, have one or more direct descendants (lower-level attributes) in the tree. Attributes without descendants are called *basic* and serve as model inputs. Attributes with one or more descendants are called *aggregate* and represent model outputs. In order to represent attribute hierarchies rather than plain trees, some attributes may be *linked*: two attributes of which one links to another one collectively represent, in a conceptual sense, a single attribute in the hierarchy.

When completely defined, each attribute is associated with a value scale represented by a `DexiScale` object. An object `DexiFunction` is also defined for each aggregate attribute, aimed at defining the aggregation of the attribute's inputs to values of that attribute.

Fields

`name` character. Name of the attribute as defined in the original DEXi model. Notice that such names may not be unique and may contain characters that cannot be used for variable names in R.

`id` character. A unique identification of the attribute in the model. Derived from `name` so that it can be used as a variable name in R.

`description` character. An optional textual description of the attribute.

`inputs` list of `DexiAttribute`s. A list of immediate descendants of this attribute in the tree/hierarchy. `NULL` for basic attributes.

`link` `DexiAttribute`. `NULL` or a link to another `DexiAttribute`

`scale` `DexiScale`. Value scale associated with this attribute, or `NULL`.

`funct` `DexiFunction`. Aggregation function associated with this attribute, or `NULL`.

`parent` `DexiAttribute` or `DexiModel` (only for `DexiModel$root`). Parent attribute of this attribute in the tree/hierarchy. The `DexiModel$root`'s parent is the `DexiModel`, which contains all those attributes.

`.alternatives` list. An internal field providing temporary storage for names or values of alternatives while reading them from a `.dxi` file.

Methods

`affects(ant)` `ant` (as "antecedent") is some `DexiAttribute`. The function returns `TRUE` if `ant` lies on the path leading from this attribute towards the root, and is therefore affected by this attribute.

`count()` Return the number of `inputs` of this attribute.

`dim()` Dimensions of the value space determined by this attribute's `inputs`. Result: a numeric vector of length equal to `ninp()`, containing `DexiScale$count()` of all descendant attributes, or `NA` for attributes without associated scales. For basic attributes, `dim()` returns `NULL`.

`initialize(name = "", description = "", inputs = list(), id = "", link = NULL, scale = NULL, funct = NULL, parent = NULL, ...)` Initialize a `DexiAttribute` object.

`inp_index(inp)` Return the index of attribute `inp` in `inputs` of this attribute.

`is_aggregate()` Logical: `TRUE` for aggregate attributes (attributes whose `ninp() > 0`).

`is_basic(include_linked = TRUE)` Logical: `TRUE` for basic attributes (attributes whose `ninp() == 0`. `include_linked` determines whether linked attributes are counted as basic (`TRUE`) or not (`FALSE`).

`is_link()` Logical: Indicates whether or not this is a linked attribute.

`level()` Return the level of this attribute in the hierarchy. The level of `DexiModel$root` is 0.

`model()` Return the `DexiModel` that contains this attribute.

`ninp()` Return the number of `inputs` of this attribute.

`structure()` Make an indentation string for this attribute, used for printing it in `show()`.

`tree_indent(none = " ", thru = "|", link = "*", last = "+", line = "-")` Construct a string for representing the indentation of this attribute in the model structure. The arguments `none`, `thru`, `link`, `last` and `line` are character strings to be used in the construction.

`verify()` Check the correctnes of a `DexiAttribute` object and its fields. Result: `error()` or `TRUE`.

Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# For example, consider attribute PRICE
att <- Car$attrib("PRICE")

# Print fields and basic properties of att
att$verify()
att$name
att$id
att$description
att_names(att$inputs)
att$link
att$scale
att$funct
att_names(att$parent)
att$is_aggregate()
att$is_basic()
att$is_link()
att$level()
att$count()
att$ninp()
att$dim()
att$model()
att$structure()

# Check if att affects attribute CAR
att$affects(Car$attrib("CAR"))

# Find the index of other attributes in att's inputs
att$inp_index(Car$attrib("MAINT.PRICE"))
att$inp_index(Car$attrib("CAR"))
```

DexiContinuousScale-class

DexiContinuousScale

Description

`DexiContinuousScale` is a RC class, derived from `DexiScale`, representing continuous value scales in R.

Details

An attribute associated with a continuous scale can take any single numeric value from `[-Inf,+Inf]`.

`DexiContinuousScale` defines two numeric bounds, called `low_point` and `high_point`, such that `low_point <= high_point`. These values partition preferentially ordered scales in three preferential classes ("qualities"): `"bad"`, `"none"` (in the sense of `"neutral"`), and `"good"`. For a scale with `order = "ascending"`, the three corresponding intervals are `[-Inf,low_point]`, `(low_point,high_point)` and `[high_point,+Inf]`. For `order = "descending"`, the order of qualities is reversed. Scales with `order = "none"` have only one associated quality, `"none"`, for the whole range of values.

Currently, the DEXi software does not support continuous scales. `DexiContinuousScale` has been implemented in DEXiR for future compatibility with extended DEXi software that is under development.

Fields

`low_point` numeric. A bound for the quality interval `[-Inf,low_point]`.

`high_point` numeric. A bound for the quality interval `[high_point,+Inf]`.

Methods

`count()` Return the number of scale elements. Equal to `NA` for `DexiScale`, `0` for `DexiContinuousScale`, and equal to `nvals >= 0` for `DexiDiscreteScale`.

`equal(scl)` Check if this scale is equal to scale `scl`. Needed for attribute linking.

`initialize(order = EnumOrder, ...)` Initialize a `DexiScale` object.

`to_string()` Return a string representation of this scale for printing.

`value_quality(value)` Return the quality (preferential class) of `value` on this scale: one of the strings `"bad"`, `"none"` or `"good"`. Always `"none"` for `DexiScale` and scales with `order = "none"`.

`verify()` Check the correctnes of this scale object and its fields. Result: `error()` or `TRUE`.

`DexiDiscreteScale-class`

DexiDiscreteScale

Description

`DexiDiscreteScale` is a RC class, derived from `DexiScale`, representing qualitative (symbolic, discrete, verbal) value scales in R. Such scales are typical for DEXi models and are the only scale type currently supported by the DEXi software.

Details

An attribute associated with a discrete scale can take values from a finite (and usually small) set of string values contained in the character vector `values`. Additionally, each of these values is associated with one of the qualities `"bad"`, `"none"` or `"good"`. The latter are contained in the character vector `quality`, which is of the same length as `values`.

Fields

`values` character. Vector of qualitative scale values. Example: `scale$values <-c("low","medium","high")`.

`nvals` integer. Equal to `length(values)`.

`quality` character. Vector of qualities, corresponding to `values`. Should be the of the same length as `values`. Example: `scale$quality <-c("bad","none","good")`.

`descriptions` character. A vector of textual descriptions of the corresponding `values`. Should be the of the same length as `values`.

Methods

`count()` Return the number of scale elements. Equal to `NA` for `DexiScale`, `0` for `DexiContinuousScale`, and equal to `nvals >= 0` for `DexiDiscreteScale`.

`equal(scl)` Check if this scale is equal to scale `scl`. Needed for attribute linking.

`full_range()` Return the vector that represents the full range of values on this scale. Equal to `NA` for `DexiScale` and `DexiContinuousScale`, and `1 : scale$nvals` for `DexiDiscreteScale`.

`initialize(order = EnumOrder, ...)` Initialize a `DexiScale` object.

`is_discrete()` Logical: Is this scale discrete?

`to_string()` Return a string representation of this scale for printing.

`value_index(value)` Find the index of `value` (character(1)) on this scale. Equal to `NA` for `DexiScale` and `DexiContinuousScale`. With `DexiDiscreteScale` objects, it returns a numeric index or `NA` of `value` in `scale$values`.

`value_quality(value)` Return the quality (preferential class) of `value` on this scale: one of the strings `"bad"`, `"none"` or `"good"`. Always `"none"` for `DexiScale` and scales with `order = "none"`.

`verify()` Check the correctnes of this scale object and its fields. Result: `error()` or `TRUE`.

Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# For example, consider the scale of attribute PRICE
scl <- Car$attrib("PRICE")$scale

# Print fields and basic properties of scl
scl$verify()
scl$values
scl$quality
scl$descriptions
scl$nvals
scl$count()
scl$is_discrete()
scl$is_continuous()
scl$to_string()
scl$full_range()

# Find value indices
scl$value_index("medium")
scl$value_index("med")
```

```
# Is scl equal to the scale of BUY.PRICE?
scl$equal(Car$attrib("PRICE")$scale)
```

DexiDiscretizeFunction-class
DexiDiscretizeFunction

Description

DexiDiscretizeFunction is a RC class, derived from DexiFunction. Functions of this type discretize numerical values of continuous attributes to qualitative values of discrete attributes. More precisely, a DexiDiscretizeFunction can be defined only for a discrete attribute that has exactly one continuous input. Then, the function discretizes numeric values of the input attribute and maps them to discrete values of the parent attribute.

Details

Objects of class DexiDiscretizeFunction define discretization rules in terms of three lists: values, bounds and assoc. Using n <-nvals() to denote the length of values, the required lengths of bounds and assoc are n -1,

The list bounds refers to values of the input attribute and partitions its scale in n intervals [-Inf,bound[[1]]], [bound[[1]],bound[[2]]], ..., [bound[[n -1]]],+Inf]. The list values then defines the output values for each interval. The list assoc contains strings "up" or "down" that indicate to which interval, lower or higher, belong the corresponding bounds.

Fields

attribute DexiAttribute. The attribute this function is associated with. Requirements: attribute must be discrete (i.e., associated with a DexiDiscreteScale) and must have exactly one continuous input attribute (i.e., associated with a DexiContinuousScale).

values A list of output values corresponding to each interval defined by bounds. List elements are in general value sets, i.e., integer vectors of value indices w.r.t. attribute$scale.

bounds A vector of numeric values that partitions the input scale in intervals.

assoc A vector of strings "up" or "down". For each i in 1:n-1, assoc[[i]] indicates how to map the value of bounds[[i]]: to value[[i]] ("down") or value[[i + 1]] ("up").

Methods

bound_assoc(idx, default = "down") Given idx, a bounds index, return the corresponing association ("down" or "up").

evaluate(x) A silent wrapper around value(x); it returns NULL when value(x) fails with an error.

nargs() Return the number of function arguments.

nvals() Return the length of values.

to_string() Return an informative string about this function's values and bounds.

`value(x)` Return the function value for arguments `x`, where arguments are a numeric vector of length equal to `att$inputs`. Additionally, arguments of a `DexiTabularFunctions$value()` must be integer numbers, and the argument of `DexiDiscretizeFunctions$value()` must be a single number.

`verify()` Check the correctnes of this function object and its fields. Result: `error()` or `TRUE`.

Examples

```
# Create a DexiDiscretizeFunction (without association to any attributes or scales)
fnc <- DexiDiscretizeFunction(bounds = c(-1, 2), values = list(1, 3, 5),
                              assoc = c("up", "down"))

# Print fields and basic properties of fnc

fnc$verify()
fnc$nargs()
fnc$nvals()
fnc$to_string()

fnc$bound_assoc(1)
fnc$bound_assoc(2)

# Try some discretizations
sapply(c(-1.1, -1, 0, 1, 2, 3), fnc$evaluate)
```

`DexiFunction-class` *DexiFunction*

Description

`DexiFunction` is a base RC class for representing DEXi aggregation and discretization functions in R.

Details

DEXi functions are generally associated with aggregate attributes. For some aggregate attribute `att`, `att$funct` defines the mapping from values of `att$inputs` to values of `att`.

`DexiFunction` is a base class that defines fields and methods common to all functions:

- method `value(x)`: returns the function value for arguments `x`. Arguments are assumed to be a numeric vector of length equal to `att$inputs`.
- method `evaluate(x)` is a silent wrapper around `value(x)`; it returns `NULL` when `value(x)` fails with an error.

DEXiR implements two other function classes derived from `DexiFunction`: `DexiTabularFunction` and `DexiDiscretizeFunction`.

Methods

`evaluate(x)` A silent wrapper around `value(x)`; it returns `NULL` when `value(x)` fails with an error.

`value(x)` Return the function value for arguments `x`, where arguments are a numeric vector of length equal to `att$inputs`. Additionally, arguments of a `DexiTabularFunctions$value()` must be integer numbers, and the argument of `DexiDiscretizeFunctions$value()` must be a single number.

`verify()` Check the correctnes of this function object and its fields. Result: `error()` or `TRUE`.

`DexiModel-class` *DexiModel*

Description

`DexiModel` is a RC class representing a DEXi model in R.

Details

Normally, `DexiModel` objects are created by reading from a `.dxi` file, previously developed by the DEXi software. In principle, all fields of a `DexiModel` should be considered read-only. DEXiR does not provide any explicit functionality for creating and changing DEXi models in R. Of course, models can still be created and modified in R, but without integrity and consistency guarantees.

Fields

`name` character. Name of the model.

`description` character. An optional textual description of the model.

`linking` logical. Indicates whether or not the model uses linked attributes, which are used in DEXi to represent hierarchies of attributes (i.e., directed acyclic graphs) rather than trees.

`root` `DexiAttribute`. The virtual root of all subtrees/hierarchies of attributes in the model.

`attributes` list. A list of all `DexiAttribute`s that constitute the model.

`att_names` character. A list of all attribute names, as defined in the original DEXi model. Notice that these names may contain whitespace and other "strange" characters, and may not be unique.

`att_ids` character. A list of unique attribute IDs generated by DEXiR from `att_names` using `make.unique`. When using the DEXiR package, it is strongly advised to refer to attributes with their IDs rather than DEXi names.

`basic` list. A list of all basic (input) `DexiAttribute`s in the model.

`aggregate` list. A list of all aggregate (output) `DexiAttribute`s in the model.

`links` list. A list of all linked `DexiAttribute`s in the model.

`basic_ids` character. A vector of all basic attributes' unique names.

`aggregate_ids` character. A vector of all aggregate attributes' unique names.

`link_ids` character. A vector of all linked attributes' unique names.

`alternatives` data.frame. A data frame representing decision alternatives contained in the `.dxi` file.

Methods

`alternative(name = "NewAlternative", ...)` Create a data frame containing data of one decision alternative. `name`, character(1), represents the alternative's name. The arguments `...` define the alternative's values to be put in the data frame. Please see `set_alternative` for the syntax of `...`.

`as_character(alt, transpose = FALSE, structure = FALSE, round = NULL)` The argument `alt` is assumed to be a data frame containing data of one or more decision alternatives with values represented by numeric vectors. `as_character(alt)` transforms the values of `alt` into a more human-readable form using character strings. Additionally, `transpose = TRUE` transposes the data frame, so that rows correspod to attributes and columns to alternatives. `structure = TRUE` additionally displays the tree structure of attributes; the latter works only with `transpose = TRUE`. `round` denotes the number of decimal digits for printing numeric values.

`att_index(atts, use_id = TRUE)` Find the indices of attributes. `atts` is a character vector of attribute IDs (when `use_id = TRUE`) or original DEXi attribute names (when `use_id = FALSE`). Result: a numeric vector containing the set of indices. Example: `Car$att_index(c("PRICE","TECH.CHAR."))`

`att_stat()` Count the number of all attributes (including the virtual root), as well as the number of basic, aggregate and linked attributes in the model. Result: a numeric vector containing the four counts.

`attrib(atts)` A general function for finding attributes in the model. `atts` is a vector or list of `DexiAttributes`, attribute indices (integer) or attribute IDs (character). Result: a list of found `DexiAttributes` (or `NA`s if not found). Example: `Car$attrib(list(5,"PRICE","TECH.CHAR."))`

`evaluate(...)` Calls `evaluate(.self,...)` to evaluate decision alternatives. Please see `evaluate` for the description of `...` arguments.

`initialize(name = "", description = "", root = NULL, linking = FALSE, ...)` Initialize a `DexiModel` object.

`link_attributes()` Carries out the linking of attributes. DEXi attributes that have the same names and value scales, and satisfy some other constraints to prevent making cycles in the model, are linked together so that they logically represent a single attribute. In this way, a tree of attributes is conceptually turned in a hierarchy (directed acyclic graph). If `linking = TRUE`, `link_attributes` is called by `setup()` after reading the model.

`scale(atts)` Find attribute scales. `atts` is a vector of `DexiAttributes`. Result: a vector of the corresponding `DexiScale`s (or `NA`s).

`setup()` Called by `initialize()` as the last step that establishes consistent internal data structures by making unique attribute IDs, linking attributes (if required), making lists of attributes and their IDs, and creating a data frame of alternatives.

`verify()` Check the correctnes of a `DexiModel` object and its fields. Result: `error()` or `TRUE`.

See Also

`evaluate`, `set_alternative`, `read_dexi`

Examples

```
# Get ".dxi" file name
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")

# Read DEXi model
```

```
Car <- read_dexi(CarDxi)

# Print fields of Car
Car
Car$verify()
Car$name
Car$description
Car$linking
att_names(Car$attributes)
Car$att_names
Car$att_ids
Car$basic_ids
Car$aggregate_ids
Car$att_stat()
Car$scale(Car$aggregate)

# Find some attributes in the model
Car$attributes[[3]]
Car$attrib("PRICE")
Car$att_index("PRICE")

# Display alternatives loaded from "Car.dxi"
Car$alternatives
Car$as_character(Car$alternatives)
Car$as_character(Car$alternatives, transpose = TRUE)
Car$as_character(Car$alternatives, transpose = TRUE, structure = TRUE)

# Define and evaluate a decision alternative (some car)
alt <- Car$alternative("MyCar",
        BUY.PRICE="low", MAINT.PRICE=2, X.PERS=3, X.DOORS=3, LUGGAGE="medium", SAFETY=2)
Car$evaluate(alt)
Car$as_character(Car$evaluate(alt))

# Employ the set-based evaluation
#(notice how the value of SAFETY propagates upwards to TECH.CHAR.)
alt <- Car$alternative("MyCar",
      BUY.PRICE="low", MAINT.PRICE=2, X.PERS=3, X.DOORS=3, LUGGAGE="medium", SAFETY=c(2,3))
Car$evaluate(alt)
Car$as_character(Car$evaluate(alt))
```

`DexiScale-class` *DexiScale*

Description

`DexiScale` is a base RC class representing value scales in R.

Details

A value scale defines the type and set of values that can be assigned to some `DexiAttribute`. `DexiScale` is a base scale class that defines fields and methods common to all scales:

- whether or not the scale is preferentially ordered (and in which direction),
- scale type (discrete or continuous),

- the number of scale elements, if countable,
- partition of scale elements in three preferential classes: "bad", "good" and "none",
- helper methods value_index and full_range.

DEXiR implements two other scale classes derived from DexiScale: DexiContinuousScale and DexiDiscreteScale.

Fields

order character. Preferential order of the scale. Possible values: "ascending", "descending" or "none".

Methods

count() Return the number of scale elements. Equal to NA for DexiScale, 0 for DexiContinuousScale, and equal to nvals >= 0 for DexiDiscreteScale.

equal(scl) Check if this scale is equal to scale scl. Needed for attribute linking.

full_range() Return the vector that represents the full range of values on this scale. Equal to NA for DexiScale and DexiContinuousScale, and 1 : scale$nvals for DexiDiscreteScale.

initialize(order = EnumOrder, ...) Initialize a DexiScale object.

is_continuous() Logical: Is this scale continuos?

is_discrete() Logical: Is this scale discrete?

to_string() Return a string representation of this scale for printing.

value_index(value) Find the index of value (character(1)) on this scale. Equal to NA for DexiScale and DexiContinuousScale. With DexiDiscreteScale objects, it returns a numeric index or NA of value in scale$values.

value_quality(value) Return the quality (preferential class) of value on this scale: one of the strings "bad", "none" or "good". Always "none" for DexiScale and scales with order = "none".

verify() Check the correctnes of this scale object and its fields. Result: error() or TRUE.

DexiTabularFunction-class

DexiTabularFunction

Description

DexiTabularFunction is a RC class, derived from DexiFunction. Functions of this type aggregate attribute values according to *decision rules*, defined in terms of a *decision table*.

Details

A decision table contains as many decision rules as there are possible combinations of input attributes' values. For instance, if some attribute has two inputs whose discrete scales have three and four values, respectively (i.e., attribute$dim() == c(3,4)), then the number of rules is equal to prod(attribute$dim()) == 12. Each rule defines the value of attribute for one of the possible combinations of values of attribute$inputs. Thus, a decision table can be interpreted as a lookup table that, given a vector of values of attribute$inputs (i.e., function arguments) returns the corresponding attribute value.

Objects of class DexiTabularFunction store decision rules in values, a multi-dimensional list that contains rule values. In most cases, a rule value is a single integer, representing an ordinal number of some value from attribute$scale. In a general case, however, a rule value can be an integer vector, representing a (sub)set of values from attribute$scale.

Fields

attribute DexiAttribute. The attribute this function is associated with. Both the attribute and its inputs are required to be discrete (i.e., associated with DexiDiscreteScales).

values A multi-dimensional list of rule values. The dimensions of the list are equal to attribute$dim(), and the length of the list is nvals() == prod(dim). The list contains rule values that are in general value sets, i.e., integer vectors of value indices w.r.t. attribute$scale.

args A list of integer vectors, containing all possible combinations of values of attribute$inputs. args and values are of the same length and ordered so that, for each i, args[[i]]) defines function arguments that map to values[[i]]).

Methods

evaluate(x) A silent wrapper around value(x); it returns NULL when value(x) fails with an error.

nargs() Return the number of function arguments.

nvals() Return the function size (number of rules).

to_string() Return a short informative string about the size and dimensions of values.

value(x) Return the function value for arguments x, where arguments are a numeric vector of length equal to att$inputs. Additionally, arguments of a DexiTabularFunctions$value() must be integer numbers, and the argument of DexiDiscretizeFunctions$value() must be a single number.

verify() Check the correctnes of this function object and its fields. Result: error() or TRUE.

See Also

dexi_index, dexi_table, make_args

Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# For example, consider the function of attribute CAR
fnc <- Car$attrib("CAR")$funct

# Print fields and basic properties of fnc
fnc$verify()
att_names(fnc$attribute)
fnc$values
fnc$args
fnc$nargs()
fnc$nvals()
fnc$to_string()

# Try some args to value mappings
```

```
fnc$evaluate(c(1, 1))
fnc$evaluate(c(2, 2))
fnc$evaluate(c(3, 4))
fnc$evaluate(c(4, 4)) # the first argument is out of bounds, returns NULL
```

`dexi_bool` *dexi-bool*

Description

Convert a DEXi string to logical. `"TRUE"`, `"T"` and `"1"` are interpreted as `TRUE`, all other strings as `FALSE`.

Usage

```
dexi_bool(x)
```

Arguments

`x`	character(1).

Value

Logical.

Examples

```
dexi_bool("TRUE")
sapply(c("TRUE", "T", "1", TRUE, 1, "FALSE", "F", "0", NULL, NA, NaN), dexi_bool)
```

`dexi_index` *dexi_index*

Description

Return the index of argument vector `vec` in the decision space `dim`. The index is calculated according to DEXi's sorting rules, which are different to R's.

Usage

```
dexi_index(vec, dim)
```

Arguments

`vec`	Integer vector, representing arguments of some decision rule.
`dim`	Integer vector, representing dimensions of the corresponding decision space. Assumptions: `length(vec) == length(dim)` and, for each `i`, `1 <= vec[[i]] <= dim[[i]]`.

Value

Integer, index of vec.

Examples

```
dexi_index(c(1,1,1), c(2,2,3))
dexi_index(c(1,1,2), c(2,2,3))
dexi_index(c(1,2,3), c(2,2,3))
```

---

dexi_option_value *dexi_option_value*

Description

Conversion of a string to a "DEXi value" (see DEXiR-package) according to "old" DEXi syntax. In .dxi files, the old syntax is used with OPTION XML tags. The reason for replacing the old with the new syntax (see dexi_value) was that the old syntax can not unambiguously represent value distributions.

Usage

```
dexi_option_value(x)
```

Arguments

x	character(1). Contains a sequence of characters, each of which represents an individual ordinal number.

Value

A numeric vector. The conversion uses rule_values(x,add = 1). For special-type parameters, the conversion results are:

x	result
NULL	NULL
a non-character object	NA
"" or "*"	NA
a string starting with "undef"	NA

See Also

DEXiR-package, dexi_value, rule_value

Examples

```
dexi_option_value(NULL)
dexi_option_value(NA)
dexi_option_value("")
dexi_option_value("*")
dexi_option_value("undef")
dexi_option_value("1")
dexi_option_value("012")
```

`dexi_table` *dexi_table*

Description

Create a representation of DEXi's decision table in R.

Usage

```
dexi_table(dim, low, high = NULL)
```

Arguments

`dim`	An integer vector, representing dimensions of the underlying decision space.
`low`	character(1). A string normally read from a `.dxi` file, representing the lower bounds of the corresponding decision rule values (assuming the order according to `dexi_index`). Notice that the string contains zero-based characters, which are converted to one-based integer values used in R.
`high`	character(1) or `NULL`. A string representing the upper bounds of corresponding decision rule values. If `high == NULL`, `high` is assumed to be equal to `low`.

Value

`length(dim)`-dimensional matrix of rule values, which are normally single integer values, but might also be sets of values. Each set is represented by a numeric vector.

Examples

```
# Converting DEXi's value strings to R's numeric vectors.
dexi_table(c(2, 3), "011012")
dexi_table(c(2, 3), "011012", "012112")
```

dexi_value *dexi_value*

Description

Conversion of a string to a "DEXi value" (see DEXiR-package) according to "new" DEXi syntax rules. In `.dxi` files, this syntax is used in `ALTERNATIVE` and `RULE` XML tags. Examples of possible options include:

x	result
NULL or ""	NULL
"*"	NA
a string starting with "undef"	NA
"2"	a single ordinal value, c(2) in this case
"2.1"	a single number, c(2.1) in this case
"1:3"	interval, equivalent to c(1, 2, 3)
"{0;2;3}"	a value set, equivalent to c(0, 2, 3)
"<0;0.3;0.7>"	a value distribution, distribution(0.0, 0.3, 0.7)

Usage

```
dexi_value(x, add = 0)
```

Arguments

x	character(1).
add	A numeric constant to be added to the result. Useful when converting DEXi's zero-based representation to one-based representation used in R, which requires the setting `add = 1`.

Value

A single integer or real number, an integer numeric vector, or a `distribution`.

See Also

DEXiR-package, dexi_option_value, distribution

Examples

```
dexi_value("")
dexi_value(NULL)
dexi_value("*")
dexi_value("UNDEF")
dexi_value("2")
dexi_value("2.1")
dexi_value("1:3")
dexi_value("{0;2;3}")
dexi_value("{0;2;3}", add = 1)
dexi_value("<0;0.3;0.7>")
```

dexi_vector *dexi_vector*

Description

Interpret a string, composed of ";"-separated numbers, as a numeric vector

Usage

```
dexi_vector(x)
```

Arguments

x	character(1).

Value

Numeric vector.

Examples

```
dexi_vector("1;2")
dexi_vector("1.2; 2.3")
```

distribution *distribution*

Description

Create an object as a S3 class `distribution`.

Usage

```
distribution(...)
```

Arguments

...	Expected a comma-separated list of numeric values.

Value

An object, call it `obj`, such that `all(obj == c(...))` and `class(obj) == "distribution"`.

See Also

DEXiR-package, set_to_distr, distr_to_set

Examples

```
distribution(0.1, 0.2, 0.7)
```

distr_to_set *distr_to_set*

Description

Convert a DEXi value distribution to a DEXi value set.

Usage

```
distr_to_set(distr, eps = .Machine$double.eps)
```

Arguments

distr	An S3 object of class distribution.
eps	A numeric value representing the threshold value of p (see DEXiR-package) above which the corresponding elements are considered set members.

Value

A numeric vector determined as which(distr > eps). Notice that distr_to_set is generally a lossy conversion, so that multiple different distrs are converted to the same sets.

See Also

DEXiR-package, distribution, set_to_distr

Examples

```
distr_to_set(distribution(0.2, 0, 0.5, 0.3))
distr_to_set(distribution(0.1, 0, 0.7, 0.2))
distr_to_set(distribution(0.1, 0, 0.7, 0.2), eps = 0.5)
```

equal_scales *equal_scales*

Description

Check if two scales are equal. NULL arguments, indicating undefined scales, are allowed. Two NULL scales are considered equal.

Usage

```
equal_scales(scl1, scl2)
```

Arguments

scl1	A DexiScale (or derived) object, or NULL.
scl2	A DexiScale (or derived) object, or NULL.

Value

Logical.

`evaluate` *evaluate*

Description

Evaluates decision alternatives. Essentially, this is bottom-up aggregation method: starting with basic attributes (or pruned aggregate attributes), values of each alternative are gradually aggregated towards the `root` attribute, according to `evaluation_order`. The aggregation at each individual `DexiAttribute` is governed by the corresponding `DexiAttribute$funct`. When alternative values are sets or distributions (see `DEXiR-package`), then `evaluate` tries all possible combinations of values of the descendant attributes.

Usage

```
evaluate(
  model,
  alternatives = model$alternatives,
  root = model$root,
  method = EnumEvalMethod,
  bounding = FALSE,
  prune = list(),
  norm = NULL,
  and = NULL,
  or = NULL
)
```

Arguments

`model`	`DexiModel`.
`alternatives`	A data frame containing data of one or more decision alternatives.
`root`	`DexiAttribute`
`method`	One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`.
`bounding`	Logical. When `TRUE`, evaluation results are additionally subjected to `bounded_scale_value` to keep them in the bounds set up by the corresponding scale.
`prune`	Character vector, containing IDs of aggregate attributes that should be treated as evaluation inputs (rather than basic attributes).
`norm`	Some normalization function of the form `function(num_vector)`, or `NULL`.
`and`	Some conjunctive aggregation function of the form `function(num_vector)`, or `NULL`.
`or`	Some disjunctive aggregation function of the form `function(num_vector)`, or `NULL`.

Details

evaluate implements four aggregation methods: "set", "prob", "fuzzy" and "fuzzynorm".

The "set" method interprets DEXi values as sets. The output value assigned to some attribute is composed of the union of all attribute$funct evaluations for all possible combinations of values of attribute$inputs.

The remaining three methods interpret DEXi values as value distributions. They follow the same algorithm, but use different functions (see evaluation_parameters) in three algorithm steps: normalization, and conjunctive and disjunctive aggregation. All values distributions involved in calculations are normalized by the function norm. All combinations of attribute$input values are individually evaluated by the corresponding tabular function attribute$funct. The value p of each set of attribute$funct arguments is determined by the conjunctive aggregation function and over p's of individual arguments. Finally, the p of some output value val is determined by the disjunctive aggregation function or, applied on the p's of all partial evaluations that map to val.

For the mathematical background and more details about aggregation in DEX, please see (Trdin, Bohanec, 2018). For default normalization and aggregation functions, see normalize_function, and_function and or_function.

Value

A data frame containing both input and output (evaluated) values of alternatives.

See Also

evaluation_parameters, normalize_function, norm_none, norm_max, norm_sum, and_function, or_function, bounded_scale_value.

Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar_set",
         BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4",
         LUGGAGE=2, SAFETY="medium")
Car$evaluate(alt)

# Try the set-based evaluation using the default "set" method
alt <- Car$alternative("MyCar2",
      BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt)

# Use value distributions and try the methods "prob", "fuzzy" and "fuzzynorm"
alt <- Car$alternative("MyCar_distr",
       BUY.PRICE="low", MAINT.PRICE=distribution(0.1, 0.6, 0.3),
       X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt, method = "prob")
Car$evaluate(alt, method = "fuzzy")
Car$evaluate(alt, method = "fuzzynorm")
```

evaluation_order *evaluation_order*

Description

Determine the evaluation order of attributes. Interpreted as a sequence, the order guarantees that whenever some attribute is reached as a candidate for evaluation, all the previous attributes have been already evaluated.

Usage

```
evaluation_order(att, prune = list())
```

Arguments

att	DexiAttribute. The starting point of evaluation.
prune	A character vector. May contain IDs of aggregate attributes at which the evaluation should stop, treating them as if they were basic attributes.

Value

A character vector of attribute IDs.

See Also

evaluate

Examples

```
#' # Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)


# Full evaluation order, starting with Car$root and without pruning
evaluation_order(Car$root)

# Evaluation order, starting with the TECH.CHAR. attribute
evaluation_order(Car$attrib("TECH.CHAR."))

# evaluation order, starting with Car$root and pruned at "PRICE"
evaluation_order(Car$root, prune = "PRICE")
```

evaluation_parameters *evaluation_parameters*

Description

Make a list containing parameters of DEXi evaluation. The parameters determine which method and normalization/aggregation functions should be used.

Usage

```
evaluation_parameters(
  method = EnumEvalMethod,
  norm = NULL,
  and = NULL,
  or = NULL
)
```

Arguments

method	One of: "set" (default), "prob", "fuzzy" or "fuzzynorm".
norm	Some normalization function of the form function(num_vector), or NULL.
and	Some conjunctive aggregation function of the form function(num_vector), or NULL.
or	Some disjunctive aggregation function of the form function(num_vector), or NULL.

Value

list(method,norm,and,or). For NULL norm, and and or arguments, defaults are taken depending on the method.

See Also

evaluate, normalize_function, norm_none, norm_max, norm_sum, and_function, or_function.

Examples

```
evaluation_parameters("prob", norm = norm_none)
```

flat_text *flat_text*

Description

"Flatten" the function argument using `c(value)`, concatenate the elements and separate them by `" "`.

Usage

```
flat_text(value)
```

Arguments

`value`	Any object that can occur as an argument of `c()` and `as.character()`.

Value

character(1).

is_in_range *is_in_range*

Description

Check whether or not `x` lies the specified range.

Usage

```
is_in_range(x, lb, hb, lassoc = c("up", "down"), hassoc = c("down", "up"))
```

Arguments

`x`	Any object type, but using a non-numeric argument always returns `FALSE`.
`lb`	numeric(1). Lower bound of the interval.
`hb`	numeric(1). Ipper bound of the interval.
`lassoc`	`"up"` or `"down"`, indicating whether `lb` is included in the $lb : hb$ interval (`"up"`) or not (`"down"`). The default is `"up"`.
`hassoc`	`"up"` or `"down"`, indicating whether `hb` is included in the $lb : hb$ interval (`"down"`) or not (`"up"`). The default is `"down"`.

Value

Logical, indicating whether or not `x` lies in the interval $lb : hb$ according to function arguments.

Examples

```
is_in_range(3, 2, 5)
is_in_range(7, 2, 5)
is_in_range(3, 3, 5)
is_in_range(3, 3, 5, lassoc = "down")
```

make_args *make_args*

Description

Make a list of all possible combinations of values in a decision space defined by `dim`.

Usage

```
make_args(dim)
```

Arguments

`dim`	A numeric vector containing upper bounds of the corresponding decision space dimensions. For example, `dim = c(3,4)` defines the space of `3 * 4 == 12` combinations.

Value

A list containing all possible value combinations. List elements are numeric vectors of length equal to `length(dim)`.

Examples

```
make_args(c(3, 4))
```

normalize_function *normalize_function*

Description

Determine the function to be used in the normalization step of `evaluate`.

Usage

```
normalize_function(method, norm = NULL)
```

Arguments

`method`	One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`.
`norm`	Some normalization function of the form `function(num_vector)`, or `NULL`.

Value

Returns `norm` if not `NULL`. Otherwise, it determines the result depending on `method`:

`"set"`: `norm_none`

`"prob"`: `norm_sum`

`"fuzzy"`: `norm_none`

`"fuzzynorm"`: `norm_max`

Fails with an error if the result is not an R function.

See Also

evaluate, norm_none, norm_max, norm_sum,

---

norm_max *norm_max*

Description

Normalize values so that max(values) == max

Usage

```
norm_max(values, max = 1)
```

Arguments

values	A numeric vector.
max	numeric(1).

Value

values normalized so that max(result) == max. Returns unchanged values when max(values) == 0

See Also

norm_none, norm_sum

Examples

```
norm_max(c(0, 0.5, 0.7))
```

---

norm_none *norm_none*

Description

A "do nothing" normalization function.

Usage

```
norm_none(values)
```

Arguments

values	A numeric vector.

Value

Returns unchanged values.

See Also

norm_max, norm_sum

Examples

```
norm_none(c(0, 0.5, 0.7))
```

---

norm_sum *norm_sum*

Description

Normalize values so that sum(values) == sum

Usage

```
norm_sum(values, sum = 1)
```

Arguments

values	A numeric vector.
sum	numeric(1).

Value

values normalized so that sum(result) == sum. Returns unchanged values when sum(values) == 0

See Also

norm_none, norm_max

Examples

```
norm_sum(c(0, 0.5, 0.7))
```

or_function *or_function*

Description

Determine the function to be used in the disjunctive aggregation step of `evaluate`.

Usage

```
or_function(method, or = NULL)
```

Arguments

`method`	One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`.
`or`	Some disjunctive aggregation function of the form `function(num_vector)`, or `NULL`.

Value

Returns `or` if not `NULL`. Otherwise, it determines the result depending on `method`:

`"set": function(x) 1`

`"prob": sum`

`"fuzzy": max`

`"fuzzynorm": max`

Fails with an error if the result is not an R function.

See Also

`evaluate`, `and_function`.

read_dexi *read_dexi*

Description

`read_dexi` reads a definition of a DEXi model from a `.dxi` file or XML string.

Usage

```
read_dexi(dxi)
```

Arguments

`dxi`	character(1). A `.dxi` file name or XML string.

Value

A `DexiModel` RC object.

See Also

DexiModel

Examples

```
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)
```

`rule_value` *rule_value*

Description

Values of decision rules are in `.dxi` files encoded using character strings, where each individual character encodes some function value. The encoding is zero-based, so that `"0"` represents the lowest ordinal number on the corresponding discrete scale. `rule_value(char)` converts a single character to the corresponding ordinal value.

Usage

```
rule_value(ch)
```

Arguments

ch	A single character, such as `"3"` or `"Z"`.

Value

Corresponding integer value

Examples

```
rule_value("1")
rule_value("Z")
```

`rule_values` *rule_values*

Description

Values of decision rules are in `.dxi` files encoded using character strings, where each individual character encodes some function value. The encoding is zero-based, so that `"0"` represents the lowest ordinal number on the corresponding discrete scale. `rule_values(str)` converts the character string to a numeric vector of corresponding ordinal values.

Usage

```
rule_values(str, add = 0)
```

Arguments

str	character(1), a DEXi encoding of a vector of ordinal numbers.
add	An integer constant to be added to the resulting vector. The default is add = 0, however DEXi's ordinal numbers should normally be converted to R's using add = 1.

Value

A numeric vector of the same length as str.

Examples

```
rule_values("01122AC")
rule_values("01122AC", add = 1)
```

---

scale_of *scale_of*

Description

scale_of

Usage

```
scale_of(obj)
```

Arguments

obj	A DexiAttribute or DexiScale.

Value

DexiScale associated with obj, or NA for an undefined scale.

---

scale_value *scale_value*

Description

Check and interpret value on scale.

Usage

```
scale_value(value, scale)
```

Arguments

value	A wide range of possible value types, including integer, double, character and list vectors.
scale	A DexiScale or derived object.

Value

The result is produced depending on `value` and `scale` according to the following tables. For any scale type:

value	result
NULL	NULL
length(value == 0)	NULL
NA	scale$full_range()
other types	ERROR
value contains any NULL or NA	ERROR

For continuous scales:

value	result
length(value != 1)	ERROR
character	ERROR
named object	ERROR
length(value == 1)	unclass(value)

For discrete scales:

value	result
distribution class	value
all-integer numeric vector	value
non all-integer numeric vector	distribution(value)
"*" or "undef"...	scale$full_range()
list of value names	matched value set
list of name=p	distribution(value)

Examples

```
# Examples of successfully checked (witout error) values on a continuous scale
scl <- DexiContinuousScale()
scale_value(NULL, scl)               # NA
scale_value(c(), scl)                # NA
scale_value(list(), scl)             # NA
scale_value(character(), scl)        # NA
scale_value(NA, scl)                 # NA
scale_value(c(NA), scl)              # NA
scale_value(15.5, scl)               # 15.5
scale_value(distribution(15.5), scl) # 15.5

# Examples of successfully checked (without error) values on a discrete scale
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
scale_value(NULL, scl)                        # NA
scale_value(c(), scl)                         # NA
scale_value(list(), scl)                      # NA
scale_value(NA, scl)                          # NA
scale_value("*", scl)                         # 1:3
scale_value("Undefined", scl)                 # 1:3
```

```
scale_value(2, scl)                          # 2
scale_value(c(-1, 2, 4), scl)                # c(-1, 2, 4))
scale_value(distribution(c(-1, 2, 4)), scl)  # distribution(c(-1, 2, 4)))
scale_value(c(-1, 2.2, 4), scl)              # distribution(c(-1, 2.2, 4)))
scale_value("high", scl)                     # 3
scale_value(c("low", "high"), scl)           # c(1,3))
v <- c(0.5, 0.4)
names(v) <- c("low", "high")
scale_value(v, scl)                          # distribution(c(0.5, 0, 0.4)))
scale_value(list(high = 1.1, low = 2.2), scl) # distribution(c(2.2, 0, 1.1)))
```

`scale_values` *scale_values*

Description

A vectorized version of `scale_value`.

Usage

```
scale_values(values, scale)
```

Arguments

`values`	A list of values. For possible value types, see `scale_value`.
`scale`	A `DexiScale` or derived object.

Value

A list determined as `lapply(values,function (v) scale_value(v,scale))`.

See Also

`scale_value`

`set_alternative` *set_alternative*

Description

Set values of a single decision alternative and represent it with a data frame. Usually, only input values are set in this way. The data frame can then be `evaluate`d to set the values of output attributes.

Usage

```
set_alternative(model, alternative, ...)
```

Arguments

`model`	A `DexiModel` object. Required.
`alternative`	`character(1)` or `data.frame`. The first form sets the name of the newly created decision alternative. The second form copies values from `alternative[1,]` to initialize the corresponding columns of the resulting data frame.
`...`	A list of parameters specifying the values of the newly created decision alternative. Each parameter is expected to be in the form `attribute_id=attribute_value`, or is a list of elements of the same form. There are several possible ways to specify `attribute_value`. Taking the scale `CAR = {"unacc"; "acc"; "good"; "exc"}` as an example, the options are: `CAR="unacc"` A single qualitative value. `CAR=2` An ordinal number, indicating `"acc"` in this case. `CAR=c("good", "exc")` A set of qualitative values. `CAR=c(3, 4)` A set of ordinal numbers, equivalent to the above. `CAR=list("good", 4)` A set specified by a mixture of qualitative values and ordinal numbers. `CAR="*"` A full range of ordinal numbers, in this case equivalent to `1:4`. `CAR=distribution(0, 0, 0.7, 0.3)` A value distribution. `CAR=list("good"=0.7, "exc"=0.3)` A value distribution, equivalent to the above. `CAR="undef"` An unknown value, interpreted as `NA`. For attributes associated with continuous scales, only `numeric(1)` `attribute_values` are allowed.

Value

A one-row data frame with columns corresponding to `model`'s attributes, collectively representing a single decision alternative. The columns not copied from `alternative` (as a data frame) nor set by any parameter contain `NA`s.

See Also

`DEXiR-package` notes on values in DEXi models.

`set_to_distr` *set_to_distr*

Description

Convert a DEXi value set to DEXi value distribution.

Usage

```
set_to_distr(set, length = 0)
```

Arguments

set	Normally a numeric vector containing integer numbers.
length	The required length of the resulting distribution vector. The actual length is determined as max(length,max(set)), so the length is extended when too small to hold the whole distribution.

Value

A distribution object of length length. Arguments that are already distributions are returned "as is". Input vectors of length 0 and other types of objects return NA.

See Also

DEXiR-package, distribution, distr_to_set

Examples

```
set_to_distr(c(1, 3, 4))
set_to_distr(c(1, 3, 4), length = 5)
set_to_distr(c(1, 3, 4), length = 0)
```

---

unique_names *unique_names*

Description

Convert names strings to ID strings that are unique and conformant with R's syntactic rules for variable names.

Usage

```
unique_names(names, reserved = c())
```

Arguments

names	character vector. Names to be converted to IDs.
reserved	character vector. Reserved names that should not be used as IDs.

Value

character vector

See Also

make.unique

values_to_str *values_to_str*

Description

Convert numbers to a DEXi string. Implements the reverse operation of `rule_values`.

Usage

```
values_to_str(vals, add = 0)
```

Arguments

`vals`	Numeric vector, containing ordinal values.
`add`	An integer constant to be added to `vals` prior to conversion.

Value

A string representing DEXi's representation of ordinal values. Fails when `vals + add` contains negative numbers.

Examples

```
values_to_str(c(0, 1, 1, 2, 2, 10, 12))
values_to_str(c(1, 2, 2, 3, 3, 11, 13), -1)
```

value_text *value_text*

Description

Converts a DEXi `value` to a human-readable character string that can be printed. Used, for instance, by `DexiModel$as_character()`.

Usage

```
value_text(value, scale, round = NULL)
```

Arguments

`value`	Any DEXi value type (see `DEXiR-package`).
`scale`	A `DexiScale` or derived object.
`round`	An integer number. Indicates the number of decimals for rounding numeric values prior to printing. If `NULL`, no rounding takes place.

Value

character.

Examples

```
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
value_text(NA, scl)
value_text(1, scl)
value_text(c(1, 3), scl)
value_text(distribution(0.1, 0.2, 0.3), scl)
```