Package 'rmdl'

May 3, 2024

Title Language to Manage Many Models

Version 0.1.0

Description A system for describing and manipulating the many models that are generated in causal inference and data analysis projects, as based on the causal theory and criteria of Austin Bradford Hill (1965) <doi:10.1177/003591576505800503>. This system includes the addition of formal attributes that modify base `R` objects, including terms and formulas, with a focus on variable roles in the ``do-calculus'' of modeling, as described in Pearl (2010) <doi:10.2202/1557-4679.1203>. For example, the definition of exposure, outcome, and interaction are implicit in the roles variables take in a formula. These premises allow for a more fluent modeling approach focusing on variable relationships, and assessing effect modification, as described by VanderWeele and Robins (2007) <doi:10.1097/EDE.0b013e318127181b>. The essential goal is to help contextualize formulas and models in causality-oriented workflows.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

Depends R (>= 4.1.0), vctrs (>= 0.5.0), tibble (>= 3.0.0),

Imports stats, utils, generics, methods, dplyr, broom, tidyr, rlang, pillar, purrr, janitor

Suggests testthat (>= 3.0.0), covr, cli, rmarkdown, knitr, ggplot2, gt, survival, cmprsk, tidycmprsk

VignetteBuilder knitr

URL `https://github.com/shah-in-boots/rmdl`

BugReports `https://github.com/shah-in-boots/rmdl/issues`

Config/testthat/edition 3

NeedsCompilation no

Author Anish S. Shah [aut, cre, cph] (<`https://orcid.org/0000-0002-9729-1558`>)

Maintainer Anish S. Shah <`ashah282@uic.edu`>

Repository CRAN

Date/Publication 2024-05-02 22:20:02 UTC

R topics documented:

`data_helpers` *Data summarization and classification methods*

Description

These related functions are intended to analyze a single data vector (e.g. column from a dataset) and help predict its classification, or other relevant attributes. These are simple yet opionated convenience functions.

Usage

```
number_of_missing(x)

is_dichotomous(x)
```

Arguments

x	A vector of any of the atomic types (see [`base::vector()`])

Details

The functions that are currently supported are:

- `number_of_missing()` returns the number of missing values in a vector
- `is_dichotomous()` returns TRUE if the vector is dichotomous, FALSE otherwise

Value

Returns a single value determined by the individual functions

describe *Describe attributes of a* tm *vector*

Description

Describe attributes of a tm vector

Usage

```
describe(x, property)
```

Arguments

x	A vector tm objects
property	A character vector of the following attributes of a tm object: role, side, label, group, description, type, distribution

Value

A list of term = property pairs, where the term is the name of the element (e.g. could be the 'role' of the term).

Examples

```
f <- .o(output) ~ .x(input) + .m(mediator) + random
t <- tm(f)
describe(t, "role")
```

dplyr_extensions *Extending* dplyr *for* tm *class*

Description

The filter() function extension subsets tm that satisfy set conditions. To be retained, the tm object must produce a value of TRUE for all conditions. Note that when a condition evaluates to NA, the row will be dropped, unlike base subsetting with [.

Usage

```
## S3 method for class 'tm'
filter(.data, ...)
```

Arguments

`.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.

`...` <`data-masking`> Expressions that return a logical value, and are defined in terms of the variables in `.data`. If multiple expressions are included, they are combined with the `&` operator. Only rows for which all conditions evaluate to `TRUE` are kept.

Value

An object of the same type as `.data`. The output as the following properties:

- `tm` objects are a subset of the input, but appear in the same order
- Underlying `data.frame` columns are not modified
- Underlying `data.frame` object's attributes are preserved

See Also

`dplyr::filter()` for examples of generic implementation

`estimate_interaction` *Estimating interaction effect estimates*

Description

[Experimental]

When using categorical interaction terms in a `mdl_tbl` object, estimates on interaction terms and their confidence intervals can be evaluated. The effect of interaction on the estimates is based on the levels of interaction term. The estimates and intervals can be derived through the `estimate_interaction()` function. The approach is based on the method described by Figueiras et al. (1998).

Usage

```
estimate_interaction(object, exposure, interaction, conf_level = 0.95, ...)
```

Arguments

`object` A `mdl_tbl` object subset to a single row

`exposure` The exposure variable in the model

`interaction` The interaction variable in the model

`conf_level` The confidence level for the confidence interval

`...` Arguments to be passed to or from other methods

Details

The `estimate_interaction()` requires a `mdl_tbl` object that is a single row in length. Filtering the `mdl_tbl` should occur prior to passing it to this function. Additionally, this function assumes the interaction term is binary. If it is categorical, the current recommendation is to use dummy variables for the corresponding levels prior to modeling.

Value

A `data.frame` with `n = levels(interaction)` rows (for the presence or absence of the interaction term) and `n = 5` columns:

- estimate: beta coefficient for the interaction effect based on level
- conf_low: lower bound of confidence interval for the estimate
- conf_high: higher bound of confidence interval for the estimate
- p_value: p-value for the overall interaction effect *across levels*
- nobs: number of observations within the interaction level
- level: level of the interaction term

References

A. Figueiras, J. M. Domenech-Massons, and Carmen Cadarso, 'Regression models: calculating the confidence intervals of effects in the presence of interactions', Statistics in Medicine, 17, 2099-2105 (1998)

fmls *Vectorized formulas*

Description

This function defines a modified `formula` class that has been vectorized. The `fmls` serves as a set of instructions or a *script* for the formula and its tm. It expands upon the functionality of formulas, allowing for additional descriptions and relationships to exist between the tm.

Usage

```
fmls(
  x = unspecified(),
  pattern = c("direct", "sequential", "parallel", "fundamental"),
  ...
)

is_fmls(x)

key_terms(x)
```

Arguments

`x` Objects of the following types can be used as inputs

- `tm`
- `formula`

`pattern` A `character` from the following choices for pattern expansion. This is how the formula will be expanded, and decides how the covariates will incorporated. See the details for further explanation.

- direct: the covariates will all be included in each formula
- sequential: the covariates will be added sequentially, one by one, or by groups, as indicated
- parallel: the covariates or groups of covariates will be placed in parallel
- fundamental: every formula will be decomposed to a single outcome and predictor in an atomic fashion

`...` Arguments to be passed to or from other methods

Details

This is not meant to supersede a `stats::formula()` object, but provide a series of relationships that can be helpful in causal modeling. All `fmls` can be converted to a traditional `formula` with ease. The base for this object is built on the `tm()` object.

Value

An object of class `fmls`

Patterns

The expansion pattern allows for instructions on how the covariates should be included in different formulas. Below, assuming that *x1*, *x2*, and *x3* are covariates...

$$y = x1 + x2 + x3$$

Direct:

$$y = x1 + x2 + x3$$

Seqential:

$$\begin{gathered} y = x1 \\ y = x1 + x2 \\ y = x1 + x2 + x3 \end{gathered}$$

Parallel:

$$\begin{gathered} y = x1 \\ y = x2 \\ y = x3 \end{gathered}$$

Roles

Specific roles the variable plays within the formula. These are of particular importance, as they serve as special terms that can effect how a formula is interpreted.

Role	Shortcut	Description
outcome	`.o(...)`	**outcome** ~ exposure
exposure	`.x(...)`	outcome ~ **exposure**
predictor	`.p(...)`	outcome ~ exposure + **predictor**
confounder	`.c(...)`	outcome + exposure ~ **confounder**
mediator	`.m(...)`	outcome **mediator** exposure
interaction	`.i(...)`	outcome ~ exposure * **interaction**
strata	`.s(...)`	outcome ~ exposure / **strata**
group	`.g(...)`	outcome ~ exposure + **group**
unknown	`-`	not yet assigned

Formulas can be condensed by applying their specific role to individual runes as a function/wrapper. For example, `y ~ .x(x1) + x2 + x3`. This would signify that `x1` has the specific role of an *exposure*.

Grouped variables are slightly different in that they are placed together in a hierarchy or tier. To indicate the group and the tier, the shortcut can have an `integer` following the `.g`. If no number is given, then it is assumed they are all on the same tier. Ex: `y ~ x1 + .g1(x2) + .g1(x3)`

Warning: Only a single shortcut can be applied to a variable within a formula directly.

Pluralized Labeling Arguments

For a single argument, e.g. for the `tm.formula()` method, such as to identify variable **X** as an exposure, a `formula` should be given with the term of interest on the *LHS*, and the description or instruction on the *RHS*. This would look like `role = "exposure" ~ X`.

For the arguments that would be dispatched for objects that are plural, e.g. containing multiple terms, each `formula()` should be placed within a `list()`. For example, the **role** argument would be written:

`role = list(X ~ "exposure", M ~ "mediator", C ~ "confounder")`

Further implementation details can be seen in the implementation of `labeled_formulas_to_named_list()`.

`formula_helpers` *Tools for working with formula-like objects*

Description

Tools for working with formula-like objects

Usage

```
lhs(x, ...)

rhs(x, ...)

## S3 method for class 'formula'
rhs(x, ...)

## S3 method for class 'formula'
lhs(x, ...)
```

Arguments

`x`	A formula-like object
`...`	Arguments to be passed to or from other methods

Value

A `character` describing part of a `formula` or `fmls` object

`labeled_formulas_to_named_list` *Convert labeling formulas to named lists*

Description

Take list of formulas, or a similar construct, and returns a named list. The convention here is similar to reading from left to right, where the name or position is the term is the on the *LHS* and the output label or target instruction is on the *RHS*.

If no label is desired, then the *LHS* can be left empty, such as `~ x`.

Usage

```
labeled_formulas_to_named_list(x)
```

Arguments

`x`	An argument that may represent a formula to label variables, or can be converted to one. This includes, `list`, `formula`, or `character` objects. Other types will error.

Value

A named list with the index as a `character` representing the term or variable of interest, and the value at that position as a `character` representing the label value.

mdl_tbl *Model tables*

Description

[Experimental]

The model_table() or mdl_tbl() function creates a mdl_tbl object that is composed of either fmls objects or mdl objects, which are thin/informative wrappers for generic formulas and hypothesis-based models. The mdl_tbl is a data frame of model information, such as model fit, parameter estimates, and summary statistics about a model, or a formula if it has not yet been fit.

Usage

```
mdl_tbl(..., data = NULL)

model_table(..., data = NULL)

is_model_table(x)
```

Arguments

...	Named or unnamed mdl or fmls objects
data	A data.frame or tbl_df object, named correspondingly to the underlying data used in the models (to help match)
x	A mdl_tbl object

Details

The table itself allows for ease of organization of model information and has three additional, major components (stored as scalar attributes).

1. A formula matrix that describes the terms used in each model, and how they are combined.
2. A term table that describes the terms and their properties and/or labels.
3. A list of datasets used for the analyses that can help support additional diagnostic testing.

We go into further detail in the sections below.

Value

A mdl_tbl object, which is essentially a data.frame with additional information on the relevant data, terms, and formulas used to generate the models.

Data List

NA

Term Table

NA

Formula Matrix

NA

`models` *Model Prototypes*

Description

[Experimental]

Usage

```
mdl(x = unspecified(), ...)

## S3 method for class 'character'
mdl(
  x,
  formulas,
  parameter_estimates = data.frame(),
  summary_info = list(),
  data_name,
  strata_variable = NA_character_,
  strata_level = NA_character_,
  ...
)

## S3 method for class 'lm'
mdl(
  x = unspecified(),
  formulas = fmls(),
  data_name = character(),
  strata_variable = character(),
  strata_level = character(),
  ...
)

## S3 method for class 'glm'
mdl(
  x = unspecified(),
  formulas = fmls(),
  data_name = character(),
  strata_variable = character(),
  strata_level = character(),
  ...
)

## S3 method for class 'coxph'
mdl(
  x = unspecified(),
  formulas = fmls(),
  data_name = character(),
  strata_variable = character(),
  strata_level = character(),
  ...
)

## Default S3 method:
mdl(x, ...)

model(x = unspecified(), ...)
```

Arguments

`x` Model object or representation

`...` Arguments to be passed to or from other methods

`formulas` Formula(s) given as either an `formula` or as a `fmls` object

`parameter_estimates`

A `data.frame` that contains columns representing terms and individual estimates or coefficients, can be accompanied by additional statistic columns. By default, assumes

- **term** = term name
- **estimate** = estimate or coefficient

`summary_info` A `list` that contains columns representing summary statistic of a model. By default, assumes...

- **nobs** = number of observations
- **degrees_freedom** = degrees of freedom
- **statistic** = test statistic
- **p_value** = p-value for overall model
- **var_cov** = variance-covariance matrix for predicted coefficients

`data_name` String representing name of dataset that was used

`strata_variable`

String of a term that served as a stratifying variable

`strata_level` Value of the level of the term specified by `strata_variable`

Value

An object of the `mdl` class, which is essentially an equal-length list of parameters that describe a single model. It retains the original formula call and the related roles in the formula.

model_table_helpers *Model table helper functions*

Description

[Experimental]

These functions are used to help manage the mdl_tbl object. They allow for specific manipulation of the internal components, and are intended to generally extend the functionality of the object.

- attach_data(): Attaches a dataset to a mdl_tbl object
- flatten_models(): Flattens a mdl_tbl object down to its specific parameters

Usage

```
attach_data(x, data, ...)

flatten_models(x, exponentiate = FALSE, which = NULL, ...)
```

Arguments

x	A mdl_tbl object
data	A data.frame object that has been used by models
...	Arguments to be passed to or from other methods
exponentiate	A logical value that determines whether to exponentiate the estimates of the models. Default is FALSE. If TRUE, the user can specify which models to exponentiate by name using the **which** argument.
which	A character vector of model names to exponentiate. Default is NULL. If **exponentiate** is set to TRUE and **which** is set to NULL, then all estimates will be exponentiated, which is often a *bad idea*.

Value

When using attach_data(), this returns a modified version of the mdl_tbl object however with the dataset attached. When using the flatten_models() function, this returns a simplified data.frame of the original model table that contains the model-level and parameter-level statistics.

Attaching Data

When models are built, oftentimes the included matrix of data is available within the raw model, however when handling many models, this can be expensive in terms of memory and space. By attaching datasets independently that persist regardless of the underlying models, and by knowing which models used which datasets, it can be ease to back-transform information.

Flattening Models

A `mdl_tbl` object can be flattened to its specific parameters, their estimates, and model-level summary statistics. This function additionally helps by allowing for exponentiation of estimates when deemed appropriate. The user can specify which models to exponentiate by name. This heavily relies on the `broom::tidy()` functionality.

`patterns` *Apply patterns to formulas*

Description

The family of `apply_*_pattern()` functions that are used to expand `fmls` by specified patterns. These functions are not intended to be used directly but as internal functions. They have been exposed to allow for potential user-defined use cases.

Usage

```
apply_pattern(x, pattern)

apply_fundamental_pattern(x)

apply_direct_pattern(x)

apply_sequential_pattern(x)

apply_parallel_pattern(x)

apply_rolling_interaction_pattern(x)
```

Arguments

`x`	A `tm` object
`pattern`	A character string that specifies the pattern to use

Details

Currently supported patterns are: fundamental, direct, sequential, parallel.

Value

Returns a `tbl_df` object that has special column names and rows. Each row is essentially a precursor to a new formula.

These columns and rows must be present to be used with the `fmls()` function, and generally are the expected result of the specified pattern. They will undergo further internal modification prior to being turned into a `fmls` object, but this is an developer consideration. If developing a pattern, please use this guide to ensure that the output is compatible with the `fmls()` function.

- outcome: a single term that is the expected outcome variable
- exposure: a single term that is the expected exposure variable, which may not be present in every row
- covariate_*: the covariates expand based on the number that are present (e.g. "covariate_1", "covariate_2", etc)

tm *Create vectorized terms*

Description

[Experimental]

Usage

```
tm(x = unspecified(), ...)

## S3 method for class 'character'
tm(
  x,
  role = character(),
  side = character(),
  label = character(),
  group = integer(),
  type = character(),
  distribution = character(),
  description = character(),
  transformation = character(),
  ...
)

## S3 method for class 'formula'
tm(
  x,
  role = formula(),
  label = formula(),
  group = formula(),
  type = formula(),
  distribution = formula(),
  description = formula(),
  transformation = formula(),
  ...
)

## S3 method for class 'fmls'
tm(x, ...)
```

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

## Default S3 method:
tm(x = unspecified(), ...)

is_tm(x)
```

Arguments

`x` An object that can be coerced to a `tm` object.

`...` Arguments to be passed to or from other methods

`role` Specific roles the variable plays within the formula. These are of particular importance, as they serve as special terms that can effect how a formula is interpreted. Please see the *Roles* section below for further details. The options for roles are as below:

- outcome
- exposure
- predictor
- confounder
- mediator
- interaction
- strata
- group
- unknown

`side` Which side of a formula should the term be on. Options are `c("left", "right", "meta", "unknown")`. The *meta* option refers to a term that may apply globally to other terms.

`label` Display-quality label describing the variable

`group` Grouping variable name for modeling or placing terms together. An integer value is given to identify which group the term will be in. The hierarchy will be `1` to `n` incrementally.

`type` Type of variable, either categorical (qualitative) or continuous (quantitative)

`distribution` How the variable itself is more specifically subcategorized, e.g. ordinal, continuous, dichotomous, etc

`description` Option for further descriptions or definitions needed for the tm, potentially part of a data dictionary

`transformation` Modification of the term to be applied when combining with data

Details

A vectorized term object that allows for additional information to be carried with the variable name.

This is not meant to replace traditional `stats::terms()`, but to supplement it using additional information that is more informative for causal modeling.

Value

A `tm` object, which is a series of individual terms with corresponding attributes, including the role, formula side, label, grouping, and other related features.

Roles

Specific roles the variable plays within the formula. These are of particular importance, as they serve as special terms that can effect how a formula is interpreted.

Role	Shortcut	Description
outcome	`.o(...)`	**outcome** ~ exposure
exposure	`.x(...)`	outcome ~ **exposure**
predictor	`.p(...)`	outcome ~ exposure + **predictor**
confounder	`.c(...)`	outcome + exposure ~ **confounder**
mediator	`.m(...)`	outcome **mediator** exposure
interaction	`.i(...)`	outcome ~ exposure * **interaction**
strata	`.s(...)`	outcome ~ exposure / **strata**
group	`.g(...)`	outcome ~ exposure + **group**
unknown	-	not yet assigned

Formulas can be condensed by applying their specific role to individual runes as a function/wrapper. For example, `y ~ .x(x1) + x2 + x3`. This would signify that `x1` has the specific role of an *exposure*.

Grouped variables are slightly different in that they are placed together in a hierarchy or tier. To indicate the group and the tier, the shortcut can have an `integer` following the `.g`. If no number is given, then it is assumed they are all on the same tier. Ex: `y ~ x1 + .g1(x2) + .g1(x3)`

Warning: Only a single shortcut can be applied to a variable within a formula directly.

Pluralized Labeling Arguments

For a single argument, e.g. for the `tm.formula()` method, such as to identify variable **X** as an exposure, a `formula` should be given with the term of interest on the *LHS*, and the description or instruction on the *RHS*. This would look like `role = "exposure" ~ X`.

For the arguments that would be dispatched for objects that are plural, e.g. containing multiple terms, each `formula()` should be placed within a `list()`. For example, the **role** argument would be written:

`role = list(X ~ "exposure", M ~ "mediator", C ~ "confounder")`

Further implementation details can be seen in the implementation of `labeled_formulas_to_named_list()`.

`update.tm` *Update `tm` objects*

Description

This updates properties or attributes of a `tm` vector. This only updates objects that already exist.

Usage

```
## S3 method for class 'tm'
update(object, ...)
```

Arguments

`object`	A `tm` object
`...`	A series of `field` = `term` ~ `value` pairs that represent the attribute to be updated. Can have a value of `NA` if the goal is to remove an attribute or property.

Value

A `tm` object with updated attributes

Index