Package 'defm'

February 13, 2026

Type Package

Title Estimation and Simulation of Multi-Binary Response Models

Version 0.2.1.0

Description Multi-binary response models are a class of models that allow for the estimation of multiple binary outcomes simultaneously. This package provides functions to estimate and simulate these models using the Discrete Exponential-Family Models [DEFM] framework. In it, we implement the models described in Vega Yon, Valente, and Pugh (2023) <doi:10.48550/arXiv.2211.00627>. DEFMs include Exponential-Family Random Graph Models [ERGMs], which characterize graphs using sufficient statistics, which is also the core of DEFMs. Using sufficient statistics, we can describe the data through meaningful motifs, for example, transitions between different states, joint distribution of the outcomes, etc.

URL https://github.com/UofUEpiBio/defm, https://uofuepibio.github.io/defm/

BugReports https://github.com/UofUEpiBio/defm/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

LinkingTo Rcpp, barry

Imports Rcpp, stats

Depends R (>= 4.1.0), stats4

Suggests texreg, tinytest, barry

NeedsCompilation yes

Author George Vega Yon [aut, cre] (ORCID: <https://orcid.org/0000-0002-3171-0844>),
Department of Veterans Affairs - Rehabilitation, Research, and Development Service [fnd] (Award/W81XWH-18-PH/TBIRP-LIMBIC under Award No. I01 RX003443),
U.S. Army Medical Research Acquisition Activity [fnd] (ORION project, Award #W81XWH1910615)

Maintainer George Vega Yon <g.vegayon@gmail.com>

Repository CRAN

Date/Publication 2026-02-13 07:30:09 UTC

Contents

DEFM *Discrete Exponential Family Model (DEFM)*

Description

Discrete Exponential Family Models (DEFMs) are models from the exponential family that deal with discrete data. Here, we deal with binary arrays which can be used to represent, among other things, networks and multinomial binary Markov processes.

Discrete Exponential Family Models (DEFMs) are models from the exponential family that deal with discrete data. Here, we deal with binary arrays which can be used to represent, among other things, networks and multinomial binary Markov processes.

Usage

```
new_defm_cpp(id, Y, X, order = 1L, copy_data = TRUE)

init_defm(m, force_new = FALSE)

print_stats(m, i = 0L)

nterms_defm(m)

nrow_defm(m)

ncol_defm_y(m)

ncol_defm_x(m)
```

```
nobs_defm(m)

morder_defm(m)

new_defm(id, Y, X, order = 1, copy_data = TRUE)
```

Arguments

id	Integer vector of length n. Observation ids, for example, person id.
Y	0/1 matrix of responses of n_y columns and n rows.
X	Numeric matrix of covariates of size n_x by n.
order	Integer. Order of the markov process, by default, 1.
copy_data	Logical, if TRUE (default) the data is copied to the C++ side. If FALSE, the data is not copied, and the user must ensure that the data is not modified in R while the model exists.
m	An object of class DEFM.
force_new	Logical scalar. When TRUE (default) no cache is used to add new arrays (see details).
i	An integer scalar indicating which set of statistics to print (see details.)

Details

The id vector is used to group the observations. For example, if you have a dataset with multiple individuals, the id vector should contain the individual ids. The Y matrix contains the binary responses, where each column represents a different response variable. The X matrix contains the covariates, which can be used to model the relationship between the responses and the covariates. The order parameter specifies the order of the Markov process, which determines how many previous observations are used to predict the current observation.

The copy_data parameter specifies whether the data should be copied into the model or used as a pointer. If copy_data is TRUE, the data will be copied into the model, which can be useful if you want to avoid duplicating the data in memory. If copy_data is FALSE, the model will use the data as a pointer, which can be more efficient (but dangerous if the data is removed).

The init_defm function initializes the model, which means it computes the sufficient statistics and prepares the model for fitting. The force_new parameter specifies whether to force the model to be consider each array added as completely unique, even if it has the same support set as an existing array. This is an experimental feature and should be used with caution.

The print_stats function prints the supportset of the ith type of array in the model.

Value

An external pointer of class DEFM.

- nterms_defm returns the number of terms in the model.
- nrow_defm returns the number of rows in the model.

- `ncol_defm_y` returns the number of output variables in the model.
- `ncol_defm_x` returns the number of covariates in the model.
- `nobs_defm` returns the number of observations (events) in the model.
- `morder_defm` returns the order of the Markov process.

An external pointer of class `DEFM`.

References

Vega Yon, G. G., Pugh, M. J., & Valente, T. W. (2022). Discrete Exponential-Family Models for Multivariate Binary Outcomes (arXiv:2211.00627). arXiv. https://arxiv.org/abs/2211.00627

See Also

`defm_mle()` for maximum likelihood estimation and `loglike_defm()` for the log-likelihood function.

Examples

```
# Loading Valente's SNS data
data(valentesnsList)

mymodel <- new_defm(
  id = valentesnsList$id,
  Y = valentesnsList$Y,
  X = valentesnsList$X,
  order = 1
)

# Adding the intercept terms and a motif from tobacco to mj
td_logit_intercept(mymodel)
td_formula(mymodel, "{y1, 0y2} > {y1, y2}")

# Initialize the model
init_defm(mymodel)

# Fitting the MLE
defm_mle(mymodel)
```

defm-names *Access to the names of a model's datasets*

Description

Retrieve the column names of the dependent variable (y) and independent variable (x) of an object of class DEFM.

Usage

```
get_Y_names(m)

get_X_names(m)
```

Arguments

m	An object of class DEFM.

Value

A character vector.

A character vector with the names of the dependent or independent variables.

Examples

```
#' Using Valente's SNS data
data(valentesnsList)

# Creating the DEFM object
mymodel <- new_defm(
  id = valentesnsList$id,
  X = valentesnsList$X,
  Y = valentesnsList$Y,
  order = 0
)

# Getting the names
get_X_names(mymodel)
get_Y_names(mymodel)
```

defm_terms *Model specification for DEFM*

Description

Model specification for DEFM

Usage

```
td_ones(m, covar = "")

td_generic(m, mat, covar = "")

td_formula(m, formula, new_name = "")

td_logit_intercept(m, y_indices = as.integer(c()), covar = "")

rule_not_one_to_zero(m, term_indices)

rule_constrain_support(m, term_index, lb, ub)

## S3 method for class 'DEFM'
e1 + e2
```

Arguments

m	An object of class DEFM.
covar	String. Name of a covariate to use as an interaction for the effect. If equal to "", then no interaction effect. is used. used to weight the term.
mat	Integer matrix. The matrix specifies the type of motif to capture (see details.)
formula	Character scalar (see details).
new_name	Character scalar. Name to be assigned for the new term. if empty, then it builds a name based on the formula.
y_indices	Integer vector with the coordinates to include in the term.
term_indices	Non-negative vector of indices. Indicates which outcomes this rule will apply.
term_index	Non-negative scalar. Which term this rule will apply.
lb, ub	Numeric scalars. Lower and upper bounds.
e1, e2	e1 An object of class DEFM (e1) and a character (e2).

Details

In td_generic, users can specify a particular motif to model. Motifs are represented by cells with values equal to 1, for example, the matrix:

```
t0:   1 NA NA
t1:   1  1 NA
```

represents a transition `y0 -> (y1, y2)`. If 0 is a motif of interest, then the matrix should include 0 to mark zero values.

The function `td_formula`, will take the formula and generate the corresponding input for defm::counter_transition(). Formulas can be specified in the following ways:

- Intercept effect: `{...}` No transition, only including the current state.
- Transition effect: `{...}` > `{...}` Includes current and previous states.

The general notation is `[0]y[column id]_[row id]`. A preceeding zero means that the value of the cell is considered to be zero. The column id goes between 0 and the number of columns in the array - 1 (so it is indexed from 0,) and the row id goes from 0 to m_order.

Both Intercepts and Transition can interact with covariates. Using either the `covar` argument or, in the case of formulas, `x [Covar name]`, for example:

- Intercept effect: `{...} x Hispanic` interacts with the Hispanic covar.
- Transition effect: `{...} > {...} x Hispanic` Same.

Intercept effects:

Intercept effects only involve a single set of curly brackets. Using the 'greater-than' symbol (i.e., <) is only for transition effects. When specifying intercept effects, users can skip the `row_id`, e.g., `y0_0` is equivalent to `y0`. If the passed `row id` is different from the Markov order, i.e., `row_id != m_order`, then the function returns with an error.

Examples:

- `"{y0, 0y1}"` is equivalent to set a motif with the first element equal to one and the second to zero.

Transition effects:

Transition effects can be specified using two sets of curly brackets and an greater-than symbol, i.e., `{...}` > `{...}`. The first set of brackets, which we call LHS, can only hold `row id` that are less than `m_order`.

The term `td_logit_intercept` will add what is equivalent to an intercept in a logistic regression. When `y_indices` is specified, then the function will add one intercept per outcome. These can be weighted by a covariate.

The function `rule_not_one_to_zero` will avoid the transition one to zero in a Markov process.

The function `rule_constrain_support` will constrain the support of the model by specifying a lower and upper bound for a given statistic.

The + method is a shortcut for term_formula

Value

Invisible 0.

Examples

```
# Loading Valtente's SNS data
data(valentesnsList)

mymodel <- new_defm(
  id    = valentesnsList$id,
  Y     = valentesnsList$Y,
  X     = valentesnsList$X,
  order = 1
)

# Conventional regression intercept
td_logit_intercept(mymodel)

# Interaction effect with Hispanic
td_logit_intercept(mymodel, covar = "Hispanic")

# Transition effect from only y1 to both equal to 1.
td_formula(mymodel, "{y1, 0y2} > {y1, y2}")

# Same but interaction with Female
td_formula(mymodel, "{y1, 0y2} > {y1, y2} x Female")

# Inspecting the model
mymodel

# Initializing and fitting
init_defm(mymodel)
defm_mle(mymodel)
```

---

get_counters *Extract the counters from a DEFM model*

Description

Counters are functions that are defined in terms of the change statistics. The counters also contain a hasher that is used internally to check whether an array's support is cached or not (see details).

Usage

```
get_counters(model)

## S3 method for class 'DEFM_counters'
counters[i]

set_counter_info(counter, new_name = "", new_desc = "")

## S3 method for class 'DEFM_counters'
length(x)
```

```
## S3 method for class 'DEFM_counters'
as.list(x, ...)

## S3 method for class 'DEFM_counter'
as.list(x, ...)

set_counters_names(x, ...)

## S3 method for class 'DEFM'
set_counters_names(x, ...)

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

Arguments

`model`	A DEFM model object.
`counters`	An object of class `DEFM_counters`.
`i`	Integer from 0 to nterms - 1. Counter to get.
`counter`	An object of class `DEFM_counter`.
`new_name, new_desc`	Strings with the new name and new description, respectively. If empty, no side effect.
`x`	Either a `DEFM_counters` or a `DEFM_counter` object.
`...`	Further arguments passed to the method (not used).

Details

If the hash of an array–which are built using each counters' individual hashing functions–matches an existing array, then, the DEFM models reduce computational burden by recycling computations of the normalizing constant. For example, if a model only includes terms (counters) that do not feature individual-level characteristics like gender or age, then most likely all arrays in that model will use the same normalizing constant.

The function `set_counter_info()` can be used to modify a counter name and description. This is especially useful when a name is particularly long.

Value

- The function `get_counters` returns an external pointer to an object of class `DEFM_counters`.
- The method `[.DEFM_counters` returns an individual counter of class `DEFM_counter`.
- `set_counter_info()` invisibly returns the modified counter.
- The `length` method for `DEFM_counters` returns the number of counters in the vector. This should match the return from `nterms_defm()`.

- The `as.list` methods return a list with the name and description of the counters.
- The function `set_counters_names()` returns the counters invisibly.

`get_stats` *Get sufficient statistics counts*

Description

This function computes the individual counts of the sufficient statistics included in the model.

Usage

```
get_stats(m)
```

Arguments

`m`	An object of class DEFM.

Value

A matrix with the counts of the sufficient statistics.

Examples

```
data(valentesnsList)

mymodel <- new_defm(
  id = valentesnsList$id,
  Y = valentesnsList$Y,
  X = valentesnsList$X,
  order = 1
)

# Adding the intercept terms and a motif from tobacco to mj
td_logit_intercept(mymodel)
td_formula(mymodel, "{y1, 0y2} > {y1, y2}")

# Initialize the model
init_defm(mymodel)

# Get the counts
head(get_stats(mymodel))
```

loglike_defm *Log-Likelihood of DEFM*

Description

Log-Likelihood of DEFM

Usage

```
loglike_defm(m, par, as_log = TRUE)
```

Arguments

m	An object of class DEFM
par	A vector of parameters of length nterms_defm(m).
as_log	Logical scalar. When TRUE (default) returns the log-likelihood, otherwise it returns the likelihood.

Value

Numeric, the computed likelihood or log-likelihood of the model.

Examples

```
# Loading Valtente's SNS data
data(valentesnsList)

mymodel <- new_defm(
  id    = valentesnsList$id,
  Y     = valentesnsList$Y,
  X     = valentesnsList$X,
  order = 1
)

# Adding the intercept terms and a motif from tobacco to mj
td_logit_intercept(mymodel)
td_formula(mymodel, "{y1, 0y2} > {y1, y2}")

# Computing the log-likelihood
loglike_defm(mymodel, par = c(-1, -1, -1, 2), as_log = TRUE)
```

logodds *Maximum Likelihood Estimation of DEFM*

Description

Fits a Discrete Exponential-Family Model using Maximum Likelihood.

Usage

```
logodds(m, par, i, j)

defm_mle(object, start, lower, upper, ...)

summary_table(object, as_texreg = FALSE, ...)

texreg_fancy(fits, fun, skip_intercept = FALSE, ...)
```

Arguments

m	An object of class DEFM.
par	The parameters of the model.
i, j	The row and column of the array to turn on for the log odds.
object	An object of class DEFM.
start	Double vector. Starting point for the MLE.
lower, upper	Lower and upper limits for the optimization (passed to stats4::mle.)
...	Further arguments passed to summary_table (with the exception of as_texreg, which is set to TRUE).
as_texreg	When TRUE, wraps the result in a texreg object
fits	Either a single or a list of defm fit objects.
fun	Function to be called from the texreg package, e.g., texreg::screenreg.
skip_intercept	Whether or not to skip the intercept (logit) terms when printing the table

Details

The likelihood function of the DEFM is closely-related to the Exponential-Family Random Graph Model [ERGM]. Furthermore, the DEFM can be treated as a generalization of the ERGM. The model implemented here can be viewed as an ERGM for a bipartite network, where the actors are individuals and the events are the binary outputs.

If the model features no markov terms, i.e., terms that depend on more than one output, then the model is equivalent to a logistic regression. The example below shows this equivalence.

The function summary_table computes pvalues and returns a table with the estimates, se, and pvalues. If as_texreg = TRUE, then it will return a texreg object.

Value

- `logodds` returns a numeric vector with the log-odds for each observation in the data.

An object of class stats4::mle.

An object of class texreg with additional attributes: `custom.coef.map`, `reorder.coef`, and `groups`.

References

Vega Yon, G. G., Pugh, M. J., & Valente, T. W. (2022). Discrete Exponential-Family Models for Multivariate Binary Outcomes (arXiv:2211.00627). arXiv. https://arxiv.org/abs/2211.00627

See Also

DEFM for objects of class DEFM and `loglike_defm()` for the log-likelihood function of DEFMs.

Examples

```
#' Using Valente's SNS data
data(valentesnsList)

# Creating the DEFM object
logit_0 <- new_defm(
  id = valentesnsList$id,
  X = valentesnsList$X,
  Y = valentesnsList$Y[,1,drop=FALSE],
  order = 0
)

# Building the model
td_logit_intercept(logit_0)
td_logit_intercept(logit_0, covar = "Hispanic")
td_logit_intercept(
  logit_0,
  covar = "exposure_smoke"
)
td_logit_intercept(logit_0, covar = "Grades")
init_defm(logit_0) # Needs to be initialized

# Fitting the model
res_0 <- defm_mle(logit_0)

# Refitting the model using GLM
res_glm <- with(
  valentesnsList,
  glm(Y[,1] ~ X[,1] + X[,3] + X[,7], family = binomial())
  )

# Comparing results
summary_table(res_0)
summary(res_glm)

# Comparing the logodds
```

```
head(logodds(logit_0, par = coef(res_0), i = 0, j = 0))
```

`motif_census` *Motif census*

Description

Calculates the total motif counts for a given model, in terms of the number of times each motif appears in the data.

Usage

```
motif_census(m, y_indices)
```

Arguments

`m`	An object of class DEFM.
`y_indices`	Non-negative integer vector indicating what dependent variables will be included.

Value

A matrix of class defm_motif_census with the motif counts.

References

Vega Yon, G. G., Pugh, M. J., & Valente, T. W. (2022). Discrete Exponential-Family Models for Multivariate Binary Outcomes (arXiv:2211.00627). arXiv. https://arxiv.org/abs/2211.00627

Examples

```
# Loading Valente's SNS data
data(valentesnsList)

mymodel <- new_defm(
  id = valentesnsList$id,
  Y = valentesnsList$Y,
  X = valentesnsList$X,
  order = 1
)

# Adding the intercept terms and a motif from tobacco to mj
td_logit_intercept(mymodel)
td_formula(mymodel, "{y1, 0y2} > {y1, y2}")

# Initialize the model
init_defm(mymodel)
```

```
# Motif counts featuring only the first two variables
motif_census(mymodel, y_indices = 0:1)
```

`sim_defm` *Simulate data using a DEFM*

Description

Simulate data using a DEFM

Usage

```
sim_defm(m, par, fill_t0 = TRUE)
```

Arguments

`m`	An object of class DEFM. The baseline model.
`par`	Numeric vector of model parameters.
`fill_t0`	Logical scalar. When `TRUE` (default) will fill-in the baseline value of each observation (i.e., the starting condition) (see details.)

Details

Each observation in the simulation must have initial condition. In practice, this means we start the markov process with a matrix of size `morder_defm(m) x ncol_defm_y(m)`, i.e., order of the Markov process times the number of output variables. when `fill_t0 = TRUE`, the function return the rows corresponding to baseline states with the original value, otherwise it replaces them with -1. This option is mostly for testing purposes.

Value

An integer vector of size `nrows_defm(m) x ncol_defm_y(m)`.

`valentesns` *Valente's SNS data*

Description

This dataset contains the data used in Valente et al. (2013) to study the influence of peers on adolescent smoking, drinking, and marijuana use. The `valentesnsList` is a transformed version of the data ready to be used to create defm objects.

Usage

```
valentesns
```

Format

The `valentesns` dataset has 1,722 records for 568 individuals, featuring the following 18 columns:

- `id`: Id of the individual.
- `year`: Wave number.
- `Hispanic`: Indicator variable equal to 1 if the individual is Hispanic.
- `Female`: Indicator variable equal to 1 if the individual is female.
- `Grades`: Academic grades ranging from 1 (mostly F) to 5 (mostly As).
- `tobacco`: Indicator variable if the individual ever smoked tobacco.
- `alcohol`: Indicator variable if the individual ever drink alcohol.
- `mj`: : Indicator variable if the individual ever smoked marijuana.
- `sibsmoke` : Indicator variable if the individual's sibling smokes.
- `sibdrink`: Indicator variable if the individual's sibling drinks alcohol.
- `adultdrink`: Indicator variable equal to one if there's an adult who drinks in the household.
- `year_value`: Year of the survey.
- `present`: Indicator variable equal to 1 if the individual was present.
- `school`: School id.
- `has_sib`: Indicator variable equal to 1 if the individual has siblings.
- `exposure_smoke`: Proportion of friends who have smoked tobacco in the past.
- `exposure_drink`: Proportion of friends who have drink alcohol in the past.
- `exposure_mj`: Proportion of friends who have smoked marijuana in the past.

Exposure variables are marked with -1 for each individuals' first wave.

Source

Valente, T. W., Fujimoto, K., Unger, J. B., Soto, D. W., & Meeker, D. (2013). Variations in network boundary and type: A study of adolescent peer influences. Social Networks, 35(3), 309–316. doi:10.1016/j.socnet.2013.02.008.

Index