Package 'tmlenet'

September 28, 2015

Title Targeted Maximum Likelihood Estimation for Network Data

Version 0.1.0

Description Estimation of average causal effects for single time point interventions in network-dependent data (e.g., in the presence of spillover and/or interference). Supports arbitrary interventions (static or stochastic). Implemented estimation algorithms are the targeted maximum likelihood estimation (TMLE), the inverse-probability-of-treatment (IPTW) estimator and the parametric G-computation formula estimator. Asymptotically correct influence-curve-based confidence intervals are constructed for the TMLE and IPTW. The data are assumed to consist of rows of unit-specific observations, each row i represented by variables (F.i,W.i,A.i,Y.i), where F.i is a vector of friend IDs of unit i (i's network), W.i is a vector of i's baseline covariates, A.i is i's exposure (can be binary, categorical or continuous) and Y.i is i's binary outcome. Exposure A.i depends on (multivariate) user-specified baseline summary measure(s) sW.i, where sW.i is any function of i's baseline covariates W.i and the baseline covariates of i's friends in F.i. Outcome Y.i depends on sW.i and (multivariate) user-specified summary measure(s) sA.i, where sA.i is any function of i's baseline covariates and exposure (W.i,A.i) and the baseline covariates and exposures of i's friends. The summary measures are defined with functions def.sW and def.sA. See ?'tmlenet-package' for a general overview.

URL https://github.com/osofr/tmlenet

BugReports https://github.com/osofr/tmlenet/issues

Depends R (>= 3.2.0)

Imports assertthat, data.table, Matrix, methods, R6, Rcpp, simcausal, speedglm, stats, stringr

LinkingTo Rcpp

Suggests doParallel, foreach, igraph, knitr, locfit, matrixStats, RUnit

License GPL-2

LazyData true

NeedsCompilation yes

Author Oleg Sofrygin [aut, cre],
Mark J. van der Laan [aut]

Maintainer Oleg Sofrygin <oleg.sofrygin@gmail.com>

Repository CRAN

Date/Publication 2015-09-28 09:26:59

R topics documented:

`tmlenet-package` *Targeted Maximum Likelihood Estimation for Network Data*

Description

The **tmlenet** R package implements the Targeted Maximum Likelihood Estimation (TMLE) of causal effects under single time point stochastic interventions in network data. The package also implements the Horvitz-Thompson estimator for networks (IPTW) and the parametric g-computation formula estimator. The inference for the TMLE is based on its asymptotic normality and the efficient influence curve for dependent data. The inference for IPTW is based on its corresponding influence curve for dependent data.

Details

The input data structure consists of rows of unit-specific observations, with each row `i` represented by random variables (`F.i`,`W.i`,`A.i`,`Y.i`), where `F.i` is a vector of "*friend IDs*" of unit `i` (also referred to as `i`'s "*network*"), `W.i` is a vector of `i`'s baseline covariates, `A.i` is `i`'s exposure (either binary, categorical or continuous) and `Y.i` is `i`'s binary outcome. Each exposure `A.i` depends on (possibly multivariate) baseline summary measure(s) `sW.i`, where `sW.i` can be any user-specified function of `i`'s baseline covariates `W.i` and the baseline covariates of `i`'s friends in set `F.i` (all `W.j` such that `j` is in `F.i`). Similarly, each outcome `Y.i` depends on `sW.i` and (possibly multivariate) summary measure(s) `sA.i`, where `sA.i` can be any user-specified function of `i`'s baseline covariates and exposure (`W.i`,`A.i`) and the baseline covariates and exposures of `i`'s friends (all `W.j`,`A.j` such that `j` is in `i`'s friend set `F.i`).

The summary measures (`sW.i`,`sA.i`) are defined simultaneously for all `i` with functions `def.sW` and `def.sA`. It is assumed that (`sW.i`,`sA.i`) have the same dimensionality across `i`. The function `eval.summaries` can be used for evaluating these summary measures. All estimation is performed by calling the `tmlenet` function. The vector of friends `F.i` can be specified either as a single column `NETIDnode` in the input data (where each `F.i` is a string of friend IDs or friend row numbers delimited by the character separator `sep`) or as a separate input matrix `NETIDmat` of network IDs (where each row is a vector of friend IDs or friend row numbers). Specifying the network as a matrix generally results in significant improvements to run time.

Routines

The following routines will be generally invoked, in the same order as presented below.

`def.sW` This is the first part of the two part specification of the structural equation model for the outcome `Y`. Defines the (multivariate) baseline-covariate-based summary measure functions that will be later applied to the input data to derive the (multivariate) summary measures `sW`. Each component `sW[j]` of `sW` is defined by an R expression that takes as its input unit's baseline covariates and the baseline covariates of unit's friends. Each argument passed to `def.sW` is considered a separate summary measure, with the `j`th argument defining the `j`the summary measure `sW[j]` and the name of the `j`th argument defining the name of the summary measure `sW[j]`. The arguments of `def.sW` can be either named, unnamed or a mixture of both. When the argument `j` is unnamed, the summary measure name for `sW[j]` is created automatically.

Each summary measure is defined either by an evaluable R expressions or by a string containing an evaluable R expression. These expressions can use a special double-square-bracket subsetting operator `"Var[[index]]"`, which enables referencing the variable `Var` values of unit's friends. For example, `Var[[1]]` will evaluate to a one-dimensional vector of summary measures of length `nrow(data)`, where for each row from the input `data`, this summary measure will contain the `Var` value of the unit's first friend. The ordering of friends is determined by the ordering of friend IDs specified in the network input. In cases when the unit doesn't have any friends, its corresponding value of `Var[[1]]` will evaluate to `NA` by default. However, all such `NA`'s can be replaced by 0's by passing `replaceNAw0 = TRUE` as an additional argument to `def.sW`. One can also use vectors for indexing friend variable `Var` values in `Var[[index]]`. For example, `Var[[1:Kmax]]` will evaluate to a `Kmax`-dimensional summary measure, which will be a matrix with `nrow(data)` rows and `Kmax` columns, where the first column will evaluate to `Var[[1]]`, the second to `Var[[2]]`, and so on, up to the last column

evaluating to `Var[[Kmax]]`. Note that `Kmax` is a special reserved constant that can be used inside the network indexing operators. It is set to the highest number of friends among all units in the input `data` and it is specified by the user input argument `Kmax`. See `def.sW` manual for various examples of summary measures that use the network indexing operators.

`def.sA` Defines treatment summary measures `sA` that can be functions of each unit's exposure & baseline covariates, as well the exposures and baseline covariates of unit's friends. This is the second part of the two part specification of the structural equation model for the outcome `Y`. The syntax is identical to `def.sW` function, except that `def.sA` can consists of functions of baseline covariates as well as the exposure `Anode`.

`eval.summaries` A convenience function that can be used for validating and evaluating the user-specified summary measures. Takes the input dataset and evaluates the summary measures based on objects previously defined with function calls `def.sW` and `def.sA`. Note that this function is called automatically by the `tmlenet` function and does not need to be called by the user prior to calling `tmlenet`.

`tmlenet` Performs estimation of the causal effect of interest using the observed input `data`, the intervention of interest, the network information and the previously defined summary measures `sW`, `sA`.

Datasets

To learn more about the type of data input required by `tmlenet`, see the following example datasets:

- `df_netKmax2`
- `df_netKmax6`
- `NetInd_mat_Kmax6`

Updates

Check for updates and report bugs at `http://github.com/osofr/tmlenet`.

`BinDat` *R6 class for storing the design matrix and binary outcome for a single logistic regression*

Description

This R6 class can request, store and manage the design matrix Xmat, as well as the binary outcome Bin for the logistic regression P(Bin|Xmat). Can also be used for converting data in wide format to long when requested, e.g., when pooling across binary indicators (fitting one pooled logistic regression model for several indicators) The class has methods that perform queries to data storage R6 class DatNet.sWsA to get appropriate data columns & row subsets

Usage

```
BinDat
```

Format

An R6Class generator object

Details

- bin_names - Names of the bins.
- ID - Vector of observation IDs, 1:n, used for pooling.
- pooled_bin_name - Original name of the continuous covariate that was discretized into bins and then pooled.
- nbins - Number of bins.
- outvar - Outcome name.
- predvars - Predictor names.
- pool_cont - Perform pooling of bins?
- outvars_to_pool - Outcome bin indicators to pool?
- subset_expr - Defines the subset which would be used for fitting this model (logical, expression or indices).
- subset_idx - Subset subset_expr converted to logical vector.

Methods

new(reg) Uses reg R6 RegressionClass object to instantiate a new storage container for a design matrix and binary outcome.

show() Print information on outcome and predictor names used in this regression model

newdata() ...

define.subset_idx(...) ...

setdata() ...

logispredict() ...

setdata.long() ...

logispredict.long() ...

Active Bindings

emptydata ...

emptyY ...

emptySubset_idx ...

emptyN ...

getXmat ...

getY ...

`BinOutModel` *R6 class for fitting and making predictions for a single logistic regression with binary outcome B, P(B | PredVars)*

Description

This R6 class can request, store and manage the design matrix Xmat, as well as the binary outcome Bin for the logistic regression P(Bin|Xmat). Can also be used for converting data in wide format to long when requested, e.g., when pooling across binary indicators (fitting one pooled logistic regression model for several indicators) The class has methods that perform queries to data storage R6 class DatNet.sWsA to get appropriate data columns & row subsets

Usage

```
BinOutModel
```

Format

An `R6Class` generator object

Details

- cont.sVar.flag - Is the original outcome variable continuous?
- bw.j - Bin width (interval length) for an outcome that is a bin indicator of a discretized continous outcome.
- glmfitclass - Controls which package will be used for performing model fits (`glm` or `speedglm`).
- bindat - Pointer to an instance of `BinDat` class that contains the data.

Methods

`new(reg)` Uses `reg` R6 `RegressionClass` object to instantiate a new model for a logistic regression with binary outcome.

`show()` Print information on outcome and predictor names used in this regression model

`fit()` ...

`copy.fit()` ...

`predict()` ...

`copy.predict()` ...

`predictAeqa()` ...

Active Bindings

getoutvarnm ...

getoutvarval ...

getsubset ...

getprobA1 ...

getfit ...

wipe.alldat ...

CategorSummaryModel *R6 class for fitting and predicting joint probability for a univariate categorical summary measure sA[j]*

Description

This R6 class defines and fits a conditional probability model P(sA[j]|sW,...) for a univariate categorical summary measure sA[j]. This class inherits from SummariesModel class. Defines the fitting algorithm for a regression model sA[j] ~ sW + ... Reconstructs the likelihood P(sA[j]=sa[j]|sW,...) afterwards. Categorical sA[j] is first redefined into length(levels) bin indicator variables, where levels is a numeric vector of all unique categories in sA[j]. The fitting algorithm estimates the binary regressions for hazard for each bin indicator, Bin_sA[j][i] ~ sW, i.e., the probability that categorical sA[j] falls into bin i, Bin_sA[j]_i, given that sA[j] does not fall in any prior bins Bin_sA[j]_1, ..., Bin_sA[j]_{i-1}. The dataset of bin indicators (BinsA[j]_1,...,BinsA[j]_M) is created inside the passed data or newdata object when defining length(levels) bins for sA[j].

Usage

```
CategorSummaryModel
```

Format

An R6Class generator object

Details

- reg - .
- outvar - .
- levels - .
- nbins - .

Methods

new(reg, DatNet.sWsA.g0, ...) ...

fit(data) ...

predict(newdata) ...

predictAeqa(newdata) ...

Active Bindings

cats ...

ContinSummaryModel *R6 class for fitting and predicting joint probability for a univariate continuous summary measure sA[j]*

Description

This R6 class defines and fits a conditional probability model P(sA[j]|sW,...) for a univariate continuous summary measure sA[j]. This class inherits from SummariesModel class. Defines the fitting algorithm for a regression model sA[j] ~ sW + ... Reconstructs the likelihood P(sA[j]=sa[j]|sW,...) afterwards. Continuous sA[j] is discretized using either of the 3 interval cutoff methods, defined via RegressionClass object reg passed to this class constructor. The fitting algorithm estimates the binary regressions for hazard Bin_sA[j][i] ~ sW, i.e., the probability that continuous sA[j] falls into bin i, Bin_sA[j]_i, given that sA[j] does not belong to any prior bins Bin_sA[j]_1, ..., Bin_sA[j]_{i-1}. The dataset of discretized summary measures (BinsA[j]_1,...,BinsA[j]_M) is created inside the passed data or newdata object while discretizing sA[j] into M bins.

Usage

```
ContinSummaryModel
```

Format

An R6Class generator object

Details

- reg - .
- outvar - .
- intrvls - .
- intrvls.width - .
- bin_weights - .

Methods

new(reg, DatNet.sWsA.g0, DatNet.sWsA.gstar, ...) ...

fit(data) ...

predict(newdata) ...

predictAeqa(newdata) ...

Active Bindings

cats ...

DatNet *R6 class for storing and managing already evaluated summary measures* sW *or* sA *(but not both at the same time).*

Description

Class for evaluating and storing arbitrary summary measures sVar. The summary measures are evaluated based on the user-specified sVar expressions in sVar.object (sW or sA), in the environment of the input data.frame (Odata). The evaluated summary measures from sVar.object are stored as a matrix (self$mat.sVar). Contains methods for replacing missing values with default in gvars$misXreplace. Also contains method for detecting / setting sVar variable type (binary, categor, contin).

Usage

```
DatNet
```

Format

An R6Class generator object

Details

- Kmax - Maximum number of friends for any observation.
- nFnode - Name of the vector that stores the number of friends for each observation (always set to 'nF').
- netind_cl - Pointer to a network instance of class simcausal::NetIndClass.
- Odata - Pointer to the input (observed) data frame.
- mat.sVar - The evaluated matrix of summary measures for sW or sA.
- sVar.object - Instance of the DefineSummariesClass class which contains the summary measure expressions for sW or sA.
- type.sVar - named list of length ncol(mat.sVar) with sVar variable types: "binary"/"categor"/"contin".
- norm.c.sVars - flag = TRUE if continous covariates need to be normalized.
- nOdata - number of observations in the observed data frame.

Methods

```
new(netind_cl, nodes, nFnode, ...) ...
make.sVar(Odata, sVar.object = NULL, type.sVar = NULL, norm.c.sVars = FALSE) ...
def_types_sVar(type.sVar = NULL) ...
norm_c_sVars() ...
fixmiss_sVar() ...
norm.sVar(name.sVar) ...
```

`set.sVar(name.sVar, new.sVar)` ...

`get.sVar(name.sVar)` ...

`set.sVar.type(name.sVar, new.type)` ...

`get.sVar.type(name.sVar)` ...

Active Bindings

`names.sVar` ...

`names.c.sVar` ...

`ncols.sVar` ...

`dat.sVar` ...

`emptydat.sVar` ...

`nodes` ...

`DatNet.sWsA` *R6 class for storing and managing the combined summary measures* `sW` *&* `sA` *from DatNet classes.*

Description

This class inherits from `DatNet` and extends its methods to handle a single matrix dataset of all summary measures `(sA,sW)` The class `DatNet.sWsA` is the only way to access data in the entire package. Contains methods for combining, subsetting, discretizing & binirizing summary measures `(sW,sA)`. For continous sVar this class provides methods for detecting / setting bin intervals, normalization, disretization and construction of bin indicators. The pointers to this class get passed on to `SummariesModel` functions: `$fit()`, `$predict()` and `$predictAeqa()`.

Usage

```
DatNet.sWsA
```

Format

An `R6Class` generator object

Details

- `datnetW` - .
- `datnetA` - .
- `active.bin.sVar` - Currently discretized continous `sVar` column in data matrix `mat.sVar`.
- `mat.bin.sVar` - Matrix of the binary indicators for discretized continuous covariate `active.bin.sVar`.
- `ord.sVar` - Ordinal (categorical) transformation of a continous covariate `sVar`.
- `YnodeVals` - .
- `det.Y` - .
- `p` - .

Methods

new(datnetW, datnetA, YnodeVals, det.Y, ...) ...

addYnode(YnodeVals, det.Y) ...

evalsubst(subsetexpr, subsetvars) ...

get.dat.sWsA(rowsubset = TRUE, covars) ...

get.outvar(rowsubset = TRUE, var) ...

copy.sVar.types() ...

bin.nms.sVar(name.sVar, nbins) ...

pooled.bin.nm.sVar(name.sVar) ...

detect.sVar.intrvls(name.sVar, nbins, bin_bymass, bin_bydhist, max_nperbin) ...

detect.cat.sVar.levels(name.sVar) ...

discretize.sVar(name.sVar, intervals) ...

binirize.sVar(name.sVar, intervals, nbins, bin.nms) ...

binirize.cat.sVar(name.sVar, levels) ...

get.sVar.bw(name.sVar, intervals) ...

get.sVar.bwdiff(name.sVar, intervals) ...

make.dat.sWsA(p = 1, f.g_fun = NULL, sA.object = NULL) ...

Active Bindings

dat.sWsA ...

dat.bin.sVar ...

emptydat.bin.sVar ...

names.sWsA ...

nobs ...

noNA.Ynodevals ...

nodes ...

`def.sW` *Define Summary Measures sA and sW*

Description

Define and store summary measures `sW` and `sA` that can be later processed inside `eval.summaries` or `tmlenet` functions. `def.sW` and `def.sA` return an `R6` object of class `DefineSummariesClass` which stores the user-defined summary measure functions of the baseline covariates `W` and exposure `A`, which can be later evaluated inside the environment of the input `data` data frame. Note that calls to `def.sW` must be used for defining the summary measures that are functions of **only the baseline covariates** `W`, while calls to `def.sA` must be used for defining the summary measures that are functions of both, **the baseline covariates** `W` **and exposure** `A`. Each summary measure is

specified as an evaluable R expression or a string that can be parsed into an evaluable R expression. Any variable name that exists as a named column in the input `data` data frame can be used as part of these expressions. Separate calls to `def.sW/def.sA` functions can be aggregated into a single collection with '+' function, e.g., `def.sW(W1)+def.sW(W2)`. A special syntax is allowed inside these summary expressions:

- `'Var[[index]]'` - will index the friend covariate values of the variable `Var`, e.g., `'Var[[1]]'` will pull the covariate value of `Var` for the first friend, `'Var[[Kmax]]'` of the last friend, and `'Var[[0]]'` is equivalent to writing `'Var'` itself (indexes itself).

A special argument named `replaceNAw0` can be also passed to the `def.sW`, `def.sA` functions:

- `replaceNAw0 = TRUE` - automatically replaces all the missing network covariate values (`NA`) with `0`.

One can then test the evaluation of these summary measures by either passing the returned `DefineSummariesClass` object to function `eval.summaries` or by calling the internal method `eval.nodeforms(data.df, netind_cl)` on the result returned by `def.sW` or `def.sA`. Each separate argument to `def.sW` or `def.sA` represents a new summary measure. The user-specified argument name defines the name of the corresponding summary measure (where the summary measure represents the result of the evaluation of the corresponding R expression specified by the argument). When a particular argument is unnamed, the summary measure name will be generated automatically (see Details, Naming Conventions and Examples below).

Usage

```
def.sW(...)

def.sA(...)

## S3 method for class 'DefineSummariesClass'
sVar1 + sVar2
```

Arguments

`...`	Named R expressions or character strings that specify the formula for creating the summary measures.
`sVar1`	An object returned by a call to `def.sW` or `def.sA` functions.
`sVar2`	An object returned by a call to `def.sW` or `def.sA` functions.

Value

R6 object of class `DefineSummariesClass` which can be passed as an argument to `eval.summaries` and `tmlenet` functions.

Details

The R expressions passed to these functions are evaluated later inside `tmlenet` or `eval.summaries` functions, using the environment of the input data frame, which is enclosed within the user-calling environment.

Note that when observation i has only j-1 friends, the i's value of "W_netFj" is automatically set to NA. This can be an undersirable behavior in some circumstances, in which case one can automatically replace all such NA's with 0's by setting the argument replaceMisVal0 = TRUE when calling functions def.sW or def.sA, i.e., def.sW(W[[1]], replaceMisVal0 = TRUE).

Naming conventions

Naming conventions for summary measures with no user-supplied name (e.g., def.sW(W1)).

.....

- If only one unique variable name is used in the summary expression (only one parent), use the variable name itself to name the summary measure;
- If there is more than 1 unique variable name (e.g., "W1+W2") in the summary expression, throw an exception (user must always supply summary measure names for such expressions).

Naming conventions for the evaluation results of summary measures defined by def.sW & def.sA.

.....

- When summary expression evaluates to a vector result, the vector is first converted to a 1 col matrix, with column name set equal to the summary expression name;
- When the summary measure evaluates to a matrix result and the expression has only one unique variable name (one parent), the matrix column names are generated as follows: for the expressions such as "Var" or "Var[[0]]", the column names "Var" are assigned and for the expressions such as "Var[[j]]", the column names "Var_netFj" are assigned.
- When the summary measure (e.g., named "SummName") evaluates to a matrix and either: 1) there is more than one unique variable name used inside the expression (e.g., "A + 2*W"), or 2) the resulting matrix has empty ("") column names, the column names are assigned according to the convention: "SummName.1", ..., "SummName.ncol", where "SummName" is replaced by the actual summary measure name and ncol is the number of columns in the resulting matrix.

See Also

eval.summaries for evaluation and validation of the summary measures, tmlenet for estimation, DefineSummariesClass for details on how the summary measures are stored and evaluated.

Examples

```
#*****************************************************************************************
# LOAD DATA, LOAD A NETWORK
#*****************************************************************************************
data(df_netKmax6) # load observed data
head(df_netKmax6)
data(NetInd_mat_Kmax6)  # load the network ID matrix
netind_cl <- simcausal:::NetIndClass$new(nobs = nrow(df_netKmax6), Kmax = 6)
netind_cl$NetInd <- NetInd_mat_Kmax6
head(netind_cl$nF)

#*****************************************************************************************
# Example. Equivalent ways of defining the same summary measures.
# Note that 'nF' summary measure is always added to def.sW summary measures.
```

```
# Same rules apply to def.sA function, except that 'nF' is not added.
#*****************************************************************************************
def_sW <- def.sW(W1, W2, W3)
def_sW <- def.sW("W1", "W2", "W3")
def_sW <- def.sW(W1 = W1, W2 = W2, W3 = W3)
def_sW <- def.sW(W1 = W1[[0]], W2 = W2[[0]], W3 = W3[[0]]) # W1[[0]] just means W1
def_sW <- def.sW(W1 = "W1[[0]]", W2 = "W2[[0]]", W3 = "W3[[0]]")

# evaluate the sW summary measures defined last:
resmatW <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
head(resmatW)

# define sA summary measures and evaluate:
def_sA <- def.sA(A, AW1 =A*W1)
resmatA <- def_sA$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
head(resmatA)

#*****************************************************************************************
# Summary measures based on network (friend) values of the variable (matrix result).
#*****************************************************************************************
# W2[[1:Kmax]] means vectors of W2 values of friends (W2_netF_j), j=1, ..., Kmax:
def_sW <- def.sW(netW2 = W2[[0:Kmax]], W3 = W3[[0]])
# evaluation result is a matrix:
resmat <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
# The mapping from the summary measure names to actual evaluation column names:
def_sW$sVar.names.map

# Equivalent way to define the same summary measure is to use syntax '+'
# and omit the names of the two summary measures above
# (the names are assigned automatically as "W2" for the first matrix W2[[0:Kmax]]
# and "W3" for the second summary measure "W3[[0]]")
def_sW <- def.sW(W2[[0:Kmax]]) + def.sW(W3[[0]])
resmat2 <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
head(resmat2)
# The mapping from the summary measure names to actual evaluation column names:
def_sW$sVar.names.map

#*****************************************************************************************
# Define new summary measure as a sum of friend covariate values of W3:
#*****************************************************************************************
# replaceNAw0 = TRUE sets all the missing values to 0
def_sW <- def.sW(sum.netW3 = sum(W3[[1:Kmax]]), replaceNAw0 = TRUE)

# evaluation result:
resmat <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)

#*****************************************************************************************
# More complex summary measures that involve more than one covariate:
#*****************************************************************************************
# replaceNAw0 = TRUE sets all the missing values to 0
def_sW <- def.sW(netW1W3 = W3[[1:Kmax]]*W3[[1:Kmax]])

# evaluation result (matrix):
```

```
resmat <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
# the mapping from the summary measure names to the matrix column names:
def_sW$sVar.names.map

#*****************************************************************************************
# Vector results, complex summary measure (more than one unique variable name):
# NOTE: all complex summary measures must be named, otherwise an error is produced
#*****************************************************************************************
# named expression:
def_sW <- def.sW(sum.netW2W3 = sum(W3[[1:Kmax]]*W2[[1:Kmax]]), replaceNAw0 = TRUE)
mat1a <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)

# the same unnamed expression (trying to run will result in error):
def_sW <- def.sW(sum(W3[[1:Kmax]]*W2[[1:Kmax]]), replaceNAw0 = TRUE)
## Not run:
  mat1b <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)

## End(Not run)

#*****************************************************************************************
# Matrix result, complex summary measure (more than one unique variable name):
# NOTE: all complex summary measures must be named, otherwise an error is produced
#*****************************************************************************************
# When more than one parent is present, the columns are named by convention:
# sVar.name%+%c(1:ncol)

# named expression:
def_sW <- def.sW(sum.netW2W3 = W3[[1:Kmax]]*W2[[1:Kmax]])
mat1a <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)

# the same unnamed expression (trying to run will result in error):
def_sW <- def.sW(W3[[1:Kmax]]*W2[[1:Kmax]])
## Not run:
  mat1b <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)

## End(Not run)

#*****************************************************************************************
# Iteratively building higher dimensional summary measures using '+' function:
#*****************************************************************************************
def_sW <- def.sW(W1) +
          def.sW(netW1 = W2[[1:Kmax]]) +
          def.sW(sum.netW1W3 = sum(W1[[1:Kmax]]*W3[[1:Kmax]]), replaceNAw0 = TRUE)

# resulting matrix of summary measures:
resmat <- def_sW$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
# the mapping from the summary measure names to the matrix column names:
def_sW$sVar.names.map

#*****************************************************************************************
# Examples of summary measures defined by def.sA (functions of baseline and treatment)
#*****************************************************************************************
def_sA <- def.sA(sum.netAW2net = sum((1-A[[1:Kmax]]) * W2[[1:Kmax]]),
```

```
                  replaceNAw0 = TRUE) +
          def.sA(netA = A[[0:Kmax]])

resmat <- def_sA$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
def_sW$sVar.names.map

#*****************************************************************************************
# More summary measures for sA
#*****************************************************************************************
def_sA <- def.sA(netA = "A[[0:Kmax]]") +
          def.sA(sum.AW2 = sum((1-A[[1:Kmax]])*W2[[1:Kmax]]), replaceNAw0 = TRUE)

resmat <- def_sA$eval.nodeforms(data.df = df_netKmax6, netind_cl = netind_cl)
def_sW$sVar.names.map

#*****************************************************************************************
# Using eval.summaries to evaluate summary measures for both, def.sW and def.sA
# based on the (O)bserved data (data.frame) and network
#*****************************************************************************************
def_sW <- def.sW(netW2 = W2[[1:Kmax]]) +
          def.sW(netW3_sum = sum(W3[[1:Kmax]]), replaceNAw0 = TRUE)

def_sA <- def.sA(sum.AW2 = sum((1-A[[1:Kmax]])*W2[[1:Kmax]]), replaceNAw0 = TRUE) +
          def.sA(netA = A[[0:Kmax]])

data(df_netKmax6) # load observed data
data(NetInd_mat_Kmax6)  # load the network ID matrix
res <- eval.summaries(sW = def_sW, sA = def_sA, Kmax = 6, data = df_netKmax6,
                      NETIDmat = NetInd_mat_Kmax6, verbose = TRUE)

# Contents of the list returned by eval.summaries():
names(res)
# matrix of sW summary measures:
head(res$sW.matrix)
# matrix of sA summary measures:
head(res$sA.matrix)
# matrix of network IDs:
head(res$NETIDmat)
# Observed data (sW,sA) stored as "DatNet.sWsA" R6 class object:
res$DatNet.ObsP0
class(res$DatNet.ObsP0)
```

`DefineSummariesClass` *R6 class for parsing and evaluating user-specified summary measures (in* `exprs_list`*)*

Description

This **R6** class that inherits from can parse and evaluate (given the input data frame) the summary measures defined by functions `def.sW` and `def.sA`. The object of this class is generally instantiated

by calling functions def.sA or def.sW. The summary expressions (stored in exprs_list) are evaluated in the environment of the input data.frame. Note that the evaluation results of the summary measures are never stored inside this class, data can be stored only inside DatNet and DatNet.sWsA **R6** classes.

Usage

```
DefineSummariesClass
```

Format

An R6Class generator object

Details

- type - Type of the summary measure, sW or sA, determined by the calling functions def.sW or def.sA.
- exprs_list - Deparsed list of summary expressions (as strings).
- new_expr_names - The summary measure names, if none were provided by the user these will be evaluated on the basis of variable names used in the summary expression itself.
- sVar.names.map - Named list that maps the user specified summary measure names to the corresponding matrix column names from the summary measure evaluation result.

Methods

`new(type)` Instantiate a new object of class `DefineSummariesClass` by providing a type, `"sW"` or `"sA"`.

`set.new.exprs(exprs_list)` Sets the internal summary measure expressions to the list provided in `exprs_list`.

`add.new.exprs(NewSummaries)` Adds new internal summary measure expressions to the existing ones, `NewSummaries` must be an object of class `DefineSummariesClass` (to enable `Object1 + Object2` syntax). item`remove.expr(SummaryName)`Remove expression by name (for removing duplicate 'nF' expressions for repeated calls with def.sW()+def.sW()).

`eval.nodeforms(data.df, netind_cl)` Evaluate the expressions one by one, standardize all names according to one naming convention (described in def.sW), cbinding results together into one output matrix. `data.df` is the input data.frame and `netind_cl` is the input network stored in an object of class NetIndClass.

`df.names(data.df)` List of variables in the input data `data.df` gets assigned to a special variable (`ANCHOR_ALLVARNMS_VECTOR_0`).

Define_sVar *R6 class for parsing and evaluating node R expressions.*

Description

This **R6** class will parse and evaluate (in the environment of the input data) the node formulas defined by function node. The node formula expressions (stored in exprs_list) are evaluated in the environment of the input data.frame.

Usage

```
Define_sVar
```

Format

An R6Class generator object

Details

- exprs_list - Deparsed list of node formula expressions (as strings).
- user.env - Captured user-environment from calls to node that will be used as enclosing environment during evaluation.
- cur.node - Current evaluation node (set by self$eval.nodeforms())
- asis.flags - List of flags, TRUE for "as is" node expression evaluation
- ReplMisVal0 - A logical vector that captures args replaceNAw0=TRUE/FALSE in node function call. If TRUE for a particular node formula in exprs_list then all missing network VarNode values (when nF[i] < Kmax) will get replaced with with corresponding value in codesVar.misXreplace (default is 0).
- sVar.misXreplace - Replacement values for missing sVar, vector of length(exprs_list).
- netind_cl - Pointer to a network instance of class simcausal::NetIndClass.
- Kmax - Maximum number of friends for any observation.
- Nsamp - Sample size (nrows) of the simulation dataset.
- node_fun - List that contains special subsetting functions '[' and '[[', where '[' is used for subsetting time-varyng nodes and '[[' is used for subsetting network covariate values.

Methods

new(netind_cl Instantiates new object of class Define_sVar. netind_cl is the input network stored in an object of class NetIndClass.

set.new.exprs(exprs_list) Sets the internal node formula expressions to the list provided in exprs_list.

eval.nodeforms(cur.node, data.df) Evaluate the expressions one by one, returning a list with evaluated expressions. cur.node is the current node object defined with function node and data.df is the input data.frame.

df.names(data.df) List of variables in the input data data.df gets assigned to a special variable (ANCHOR_ALLVARNMS_VECTOR_0).

df_netKmax2 *An example of a row-dependent dataset with known network of at most 2 friends.*

Description

Simulated dataset containing measured i.i.d. baseline covariate (W1), dependent binary exposure (A) and binary binary outcome (Y), along with a known network of friends encoded by strings on space separated friend IDs in Net_str. The 1,000 baseline covariates W1 were sampled as i.i.d., while the exposure value of A for each observation i was sampled conditionally on the value of i's baseline covariate W1[i], as well as the baseline covariate values of i's friends in Net_str. Similarly, the binary outcome Y for each observation was generated conditionally on i's exposure and baseline covariates values in (W1[i],A[i]), as well as the values of exposures and baseline covariates of i's friends in Net_str. Individual variables are described below.

Usage

```
data(df_netKmax2)
```

Format

A data frame with 1,000 dependent observations (rows) and 6 variables:

IDs unique observation identifier

Y binary outcome that depends on unit's baseline covariate value and exposure in W1, A, as well as the baseline covariate values and exposures W1, A of observations in the friend network Net_str

nFriends number of friends for each observation (row), range 0-2

W1 binary baseline covariate (independent)

A binary exposure status that depends on unit's baseline covariate value in W1, as well as the baseline covariate values W1 of observations in the friend network Net_str

Net_str each observation is a string of space separated friend IDs (this can be either observation IDs or just space separated friend row numbers)

df_netKmax6 *An example of a row-dependent dataset with known network of at most 6 friends.*

Description

Simulated dataset containing 3 measured i.i.d. baseline covariates (W1, W2, W3), dependent binary exposure (A) and dependent binary binary outcome (Y), along with a known network of friends encoded by strings on space separated friend IDs in Net_str. The baseline covariates (W1,W2,W3) were sampled as i.i.d., while the exposure value of A for each observation i was sampled conditionally on the values of i's baseline covariates (W1[i] W2[i], W3[i]), as well as the baseline covariate values of i's friends in Net_str. Similarly, the binary outcome Y for each observation was generated conditionally on i's exposure and baseline covariates values in (W1[i],W2[i],W3[i],A[i]), as well as the values of exposures and baseline covariates of i's friends in Net_str. Individual variables are described below.

Usage

```
data(df_netKmax6)
```

Format

A data frame with 1,000 dependent observations (rows) and 6 variables:

IDs unique observation identifier

W1 categorical baseline covariate (independent), range 0-5

W2 binary baseline covariate (independent)

W3 binary baseline covariate (independent)

A binary exposure that depends on unit's baseline covariate values, as well as the baseline covariate values of observations in the friend network Net_str

Y binary outcome that depends on unit's baseline covariate value and exposure, as well as the baseline covariate values and exposures of observations in the friend network Net_str

nFriends number of friends for each observation (row), range 0-6

Net_str a vector of strings, where for each observation its a string of space separated friend IDs (this can be either observation IDs or just space separated friend row numbers)

---

eval.summaries *Evaluate Summary Measures sA and sW*

Description

Take input data, create a network matrix (when input network matrix not provided) and evaluate the summary measures previously defined with functions def.sW and def.sA. This function is called internally by tmlenet for the evaluation of the summary measures. The R6 class object named DatNet.ObsP0 that is returned by this function can be supplied as an input to the tmlenet function. When DatNet.ObsP0 is used as an input to tmlenet, the rest of the input arguments already provided to this function can be omitted from the tmlenet function call.

Usage

```
eval.summaries(data, Kmax, sW, sA, IDnode = NULL, NETIDnode = NULL,
  sep = " ", NETIDmat = NULL, verbose = getOption("tmlenet.verbose"))
```

Arguments

data	Same as tmlenet input argument.
Kmax	Same as tmlenet input argument.
sW	Same as tmlenet input argument.
sA	Same as tmlenet input argument.
IDnode	(Optional) Same as tmlenet input argument.
NETIDnode	(Optional) Same as tmlenet input argument.
sep	Optional friend ID character separator for friends listed in NETIDnode column of data, default is ' '; same as tmlenet input argument optPars$sep.
NETIDmat	(Optional) Same as tmlenet input argument.
verbose	Set to TRUE to print messages on status and information to the console. Turn this on by default using options(tmlenet.verbose=TRUE).

Value

A named list that contains:

- sW.matrix - Matrix of evaluated summary measures for sW.
- sA.matrix - Matrix of evaluated summary measures for sA.
- NETIDmat - Network ID matrix that can be used for NETIDmat input argument to tmlenet.
- DatNet.ObsP0 - R6 object of class DatNet.sWsA that stores all the summary measures and the network information. This object be passed to tmlenet as an argument, in which case the arguments already provided to eval.summaries no longer need to be specified to tmlenet.

See Also

tmlenet for estimation of network effects and def.sW for defining the summary measures.

Examples

```
#*****************************************************************************
# Define some summary measures for sW
#*****************************************************************************
def_sW <- def.sW(W1, W2, W3) +
          def.sW(netW1 = W1[[1:Kmax]]) +
          def.sW(netW2 = W2[[1:Kmax]]) +
          def.sW(mean.netW2 = mean(W2[[1:Kmax]]), replaceNAw0 = TRUE) +
          def.sW(sum.netW3 = sum(W3[[1:Kmax]]), replaceNAw0 = TRUE)

#*****************************************************************************
# Define some summary measures for sA
#*****************************************************************************
```

```
def_sA <- def.sA(netA = A[[0:Kmax]]) +
          def.sA(sum.netAW2 = sum((1-A[[1:Kmax]])*W2[[1:Kmax]]), replaceNAw0 = TRUE)

#*****************************************************************************************
# Evaluate the summary measures applied to the  (O)bserved data (data.frame) and network
#*****************************************************************************************
# load observed data:
data(df_netKmax6)
# load the network ID matrix:
data(NetInd_mat_Kmax6)
res <- eval.summaries(sW = def_sW, sA = def_sA,  Kmax = 6, data = df_netKmax6,
  NETIDmat = NetInd_mat_Kmax6, verbose = TRUE)

#*****************************************************************************************
# Contents of the list returned by eval.summaries():
#*****************************************************************************************
names(res)
# matrix of sW summary measures:
head(res$sW.matrix)
# matrix of sA summary measures:
head(res$sA.matrix)
# matrix of network IDs:
head(res$NETIDmat)
# Observed data summary measures (sW,sA) and network
# stored as "DatNet.sWsA" R6 class object:
res$DatNet.ObsP0
class(res$DatNet.ObsP0)

#*****************************************************************************************
# Using DatNet.ObsP0 as input to tmlenet():
#*****************************************************************************************
options(tmlenet.verbose = FALSE) # set to TRUE to print status messages
res_K6 <- tmlenet(DatNet.ObsP0 = res$DatNet.ObsP0,
                  Qform = "Y ~ sum.netW3 + sum.netAW2",
                  hform.g0 = "netA ~ netW2 + sum.netW3 + nF",
                  hform.gstar = "netA ~ sum.netW3",
                  Anode = "A", Ynode = "Y", f_gstar1 = 0L)

res_K6$EY_gstar1$estimates
res_K6$EY_gstar1$vars
res_K6$EY_gstar1$CIs
```

`mcEvalPsi` *R6 class for Monte-Carlo evaluation of various substitution estimators for exposures generated under the user-specified stochastic intervention function.*

Description

This R6 class performs the Monte-Carlo evaluation of the target parameters using the data generated under the user-specified arbitrary intervention `gstar`. For a given dataset, take `E[Y|sA,sW] = m.Q.init` and calcualte estimate of `psi_n` under `g_star` using Monte-Carlo integration: (*) `W` can be iid or not (W's are never resampled). (*) Use `P_n(W) = 1` for the distribution of `W = (W_1,...,W_n)` and draw `n` new exposures `A=(A_1,...,A_n)` from the distribution of `g_star`. (*) Evaluate `n` summary measures `sA=(sA_1,...,sA_n)` using these `n` newly sampled exposures `A`. (*) Evaluate the subsititution estimators as an average of n predictions `E[Y=1|sA,sW]`. (*) Repeat `nrep` times until convergence of `psi_n`.

Usage

```
mcEvalPsi
```

Format

An `R6Class` generator object

Details

- `DatNet.ObsP0` - .
- `DatNet.gstar` - .
- `m.Q.init` - .
- `m.Q.starA` - .
- `m.Q.starB` - .
- `QY.init` - .
- `QY.starA` - .
- `QY.starB` - .
- `nOdata` - .
- `p` - .

Methods

`new(DatNet.ObsP0, DatNet.gstar, ...)` ...

`get.gcomp(m.Q.init)` ...

`get.tmleA(m.Q.starA, m.h.fit)` ...

`get.tmleB(m.Q.starB)` ...

`get.fiW()` ...

NetInd_mat_Kmax6 *An example of a network ID matrix*

Description

This matrix demonstrates an internal data structure of how `tmlenet` stores the input network. The network in this matrix is derived from the column `"Net_str"` of the dataset `df_netKmax6`. Both, this matrix and the vector of strings in column `"Net_str"`, represent the same network and both can be used for specifying the input network to `tmlenet` function. The network matrix may be specified by using the argument `NETIDmat` of the `tmlenet` function. Inputing the network via this type of a matrix may lead to significant reduction in total run time, since any network specified as a vector of strings, such as in column "Net_str", will be first converted to this type of matrix representation.

Usage

```
data(NetInd_mat_Kmax6)
```

Details

See below and Example 3 in `tmlenet` help file for examples constructing this matrix from the initial network input in column `"Net_str"` of `df_netKmax6`.

This matrix consists of `1000` rows and `6` columns. Each row `i` encodes a vector of network IDs of observation `i` in `df_netKmax6`, i.e., `NetInd_mat_Kmax6[,i]` contains a vector of observation row numbers in `df_netKmax6` that are presumed "connected to" (or "friends of") observation `i`. Each observation can have at most 6 friends and if an observation `i` has fewer than 6 friends the remainder row of `NetInd_mat_Kmax6[,i]` is filled with `NA`s.

Examples

```
data(df_netKmax6)
Net_str <- df_netKmax6[, "Net_str"]
IDs_str <- df_netKmax6[, "IDs"]
net_ind_obj <- simcausal::NetIndClass$new(nobs = nrow(df_netKmax6), Kmax = ncol(df_netKmax6))
net_ind_obj$makeNetInd.fromIDs(Net_str = Net_str, IDs_str = IDs_str, sep = ' ')
NetInd_mat <- net_ind_obj$NetInd
```

print_tmlenet_opts *Print Current Option Settings for* `tmlenet`

Description

Print Current Option Settings for `tmlenet`

Usage

```
print_tmlenet_opts()
```

Value

Invisibly returns a list of `tmlenet` options.

See Also

`tmlenet_options`

`RegressionClass` *R6 class that defines regression models evaluating P(sA|sW), for summary measures (sW,sA)*

Description

This R6 class defines fields and methods that controls all the parameters for non-parametric modeling and estimation of multivariate joint conditional probability model `P(sA|sW)` for summary measures `(sA,sW)`. Note that `sA` can be multivariate and any component of `sA[j]` can be either binary, categorical or continuous. The joint probability for `P(sA|sA) = P(sA[1],...,sA[k]|sA)` is first factorized as `P(sA[1]|sA) * P(sA[2]|sA, sA[1]) * ... * P(sA[k]|sA, sA[1],...,sA[k-1])`, where each of these conditional probability models is defined by a new instance of a `SummariesModel` class (and a corresponding instance of the `RegressionClass` class). If `sA[j]` is binary, the conditional probability `P(sA[j]|sW,sA[1],...,sA[j-1])` is evaluated via logistic regression model. When `sA[j]` is continuous (or categorical), its estimation will be controlled by a new instance of the `ContinSummaryModel` class (or the `CategorSummaryModel` class), as well as the accompanying new instance of the `RegressionClass` class. The range of continuous `sA[j]` will be fist partitioned into `K` bins and the corresponding `K` bin indicators (`B_1,...,B_K`), with `K` new instances of `SummariesModel` class, each instance defining a single logistic regression model for one binary bin indicator outcome `B_j` and predictors (`sW, sA[1],...,sA[k-1]`). Thus, the first instance of `RegressionClass` and `SummariesModel` classes will automatically spawn recursive calls to new instances of these classes until the entire tree of binary logistic regressions that defines the joint probability `P(sA|sW)` is build.

Usage

```
RegressionClass
```

Format

An `R6Class` generator object

Details

- `outvar.class` - Character vector indicating a class of each outcome var: `bin` / `cont` / `cat`.
- `outvar` - Character vector of regression outcome variable names.
- `predvars` - Either a pool of all character predictors (`sW`) or regression-specific predictor names.

- reg_hazard - Logical, if TRUE, the joint probability model P(outvar | predvars) is factorized as \prod_jP(outvar[j] | predvars) for each j outvar (for fitting hazard).
- `subset` - Subset expression (later evaluated to logical vector in the envir of the data).
- `ReplMisVal0` - Logical, if TRUE all gvars$misval among predicators are replaced with with gvars$misXreplace (0).
- `nbins` - Integer number of bins used for a continuous outvar, the intervals are defined inside `ContinSummaryModel$new()` and then saved in this field.
- `bin_nms` - Character vector of column names for bin indicators.
- `useglm` - Logical, if TRUE then fit the logistic regression model using `glm.fit`, if FALSE use `speedglm.wfit`..
- `parfit` - Logical, if TRUE then use parallel `foreach::foreach` loop to fit and predict binary logistic regressions (requires registering back-end cluster prior to calling the fit/predict functions)..
- `bin_bymass` - Logical, for continuous outvar, create bin cutoffs based on equal mass distribution.
- `bin_bydhist` - Logical, if TRUE, use dhist approach for bin definitions. See Denby and Mallows "Variations on the Histogram" (2009)) for more..
- `max_nperbin` - Integer, maximum number of observations allowed per one bin.
- `pool_cont` - Logical, pool binned continuous outvar observations across bins and only fit only regression model across all bins (adding bin_ID as an extra covaraite)..
- `outvars_to_pool` - Character vector of names of the binned continuous outvars, should match `bin_nms`.
- `intrvls.width` - Named numeric vector of bin-widths (`bw_j : j=1,...,M`) for each each bin in `self$intrvls`. When `sA` is not continuous, `intrvls.width` IS SET TO 1. When sA is continuous and this variable `intrvls.width` is not here, the intervals are determined inside `ContinSummaryModel$new()` and are assigned to this variable as a list, with `names(intrvls.width) <- reg$bin_nms` Can be queried by `BinOutModel$predictAeqa()` as: `intrvls.width[outvar]`.
- `intrvls` - Numeric vector of cutoffs defining the bins or a named list of numeric intervals for `length(self$outvar) > 1`.
- `cat.levels` - Numeric vector of all unique values in categorical outcome variable. Set by `CategorSummaryModel` constructor.

Methods

`new(outvar.class = gvars$sVartypes$bin, outvar, predvars, subset, intrvls, ReplMisVal0 = TRUE, useg`
: Uses the arguments to instantiate an object of R6 class and define the future regression model.

`ChangeManyToOneRegresssion(k_i, reg)` Take a clone of a parent `RegressionClass` (`reg`) for `length(self$outvar)` regressions and set self to a single univariate `k_i` regression for outcome `self$outvar[[k_i]]`.

`ChangeOneToManyRegresssions(regs_list)` Take the clone of a parent `RegressionClass` for univariate (continuous outvar) regression and set self to `length(regs_list)` bin indicator outcome regressions.

`resetS3class()` ...

Active Bindings

S3class ...

get.reg ...

SummariesModel *R6 class for fitting and predicting model P(sA|sW) under g.star or g.0*

Description

This R6 class Class for defining, fitting and predicting the probability model P(sA|sW) under g_star or under g_0 for summary measures (sW,sA). Defines and manages the factorization of the multivariate conditional probability model P(sA=sa|...) into univariate regression models sA[j] ~ sA[j-1] + ... + sA[1] + sW. The class self$new method automatically figures out the correct joint probability factorization into univariate conditional probabilities based on name ordering provided by (sA_nms, sW_nms). When the outcome variable sA[j] is binary, this class will automatically call a new instance of BinOutModel class. Provide self$fit() function argument data as a DatNet.sWsA class object. This data will be used for fitting the model P(sA|sW). Provide self$fit() function argument newdata (also as DatNet.sWsA class) for predictions of the type P(sA=1|sW=sw), where sw values are coming from newdata object. Finally, provide self$predictAeqa function newdata argument (also DatNet.sWsA class object) for getting the likelihood predictions P(sA=sa|sW=sw), where both, sa and sw values are coming from newdata object.

Usage

SummariesModel

Format

An R6Class generator object

Details

- n_regs - .
- parfit_allowed - .

Methods

new(reg, ...) ...

length ...

getPsAsW.models ...

getcumprodAeqa ...

copy.fit(SummariesModel) ...

fit(data) ...

predict(newdata) ...

predictAeqa(newdata, ...) ...

Active Bindings

`wipe.alldat` ...

`tmlenet` *Estimate Average Network Effects For Arbitrary (Stochastic) Interventions*

Description

Estimate the average network effect among dependent units with known network structure (in presence of interference and/or spillover) using **TMLE** (targeted maximum likelihood estimation), **IPTW** (Horvitz-Thompson or the inverse-probability-of-treatment) and **GCOMP** (parametric G-computation formula).

Usage

```
tmlenet(DatNet.ObsP0, data, Kmax, sW, sA, Anode, Ynode, f_gstar1,
  Qform = NULL, hform.g0 = NULL, hform.gstar = NULL, NETIDmat = NULL,
  IDnode = NULL, NETIDnode = NULL, verbose = getOption("tmlenet.verbose"),
  optPars = list(alpha = 0.05, lbound = 0.005, family = "binomial", n_MCsims =
  10, runTMLE = c("tmle.intercept", "tmle.covariate"), YnodeDET = NULL, f_gstar2
  = NULL, sep = " ", f_g0 = NULL, h_g0_SummariesModel = NULL,
  h_gstar_SummariesModel = NULL))
```

Arguments

`DatNet.ObsP0`	Instance of class `DatNet.sWsA` returned under the same name by the `eval.summaries` function. Stores the evaluated summary measures (`sW`,`sA`) for the observed data, along with the network information. When this argument is specified, the following arguments no longer need to be provided: `data`, `Kmax`, `sW`, `sA`, `IDnode`, `NETIDnode`, `optPars$sep`, `NETIDmat`.
`data`	Observed data, a `data.frame` with named columns, containing the baseline covariates, exposures (`Anode`), the outcomes (`Ynode`) and possibly the network column (`NETIDnode`), where network is specified by a vector of strings of friend IDs, each string using `optPars$sep` character to separate different friend IDs (default is `optPars$sep=' '`).
`Kmax`	Integer constant specifying the maximal number of friends for any observation in the input `data` data.frame.
`sW`	Summary measures constructed from baseline covariates alone. This must be an object of class `DefineSummariesClass` that is returned by calling the function `def.sW`.
`sA`	Summary measures constructed from exposures `Anode` and baseline covariates. This must be an object of class `DefineSummariesClass` that is returned by calling the function `def.sW`.
`Anode`	Exposure (treatment) variable name (column name in `data`); exposures can be either binary, categorical or continuous.

Ynode	Outcome variable name (column name in data), assumed normalized between 0 and 1. This can instead be specified on the left-side of the regression formula in argument Qform.
f_gstar1	Either a function or a vector of counterfactual exposures. If a function, must return a vector of counterfactual exposures evaluated based on the summary measures matrix (sW,sA) passed as a named argument "data", therefore, the function in f_gstar1 must have a named argument "data" in its signature. The interventions defined by f_gstar1 can be static, dynamic or stochastic. If f_gstar1 is specified as a vector, it must be of length nrow(data) or 1 (constant treatment assigned to all observations). See Details below and Examples in "EQUIVALENT WAYS OF SPECIFYING INTERVENTION f_gstar1" for demonstration.
Qform	Regression formula for outcome in Ynode, when omitted (Ynode=NULL), the outcome variable is regressed on all variables defined in sW and sA. See Details.
hform.g0	Regression formula for estimating the conditional density of P(sA | sW) under g0 (the observed exposure mechanism), when omitted, this regression is defined by sA~sW where sA are all summary measures defined by argument sA and sW are all baseline summary measures defined by argument sW.
hform.gstar	Regression formula for estimating the conditional density P(sA | sW) under interventions f_gstar1 or f_gstar2. When omitted, the same regression formula as in hform.g0 will be used for hform.gstar. See Details.
NETIDmat	Network specification via matrix of friend IDs (ncol=Kmax, nrow=nrow(data)), where each row i is a vector of i's friends IDs or i's friends row numbers in data if IDnode=NULL. See Details.
IDnode	Subject identifier variable in the input data, if not supplied the network string in NETIDnode is assumed to be indexing the row numbers in the input data
NETIDnode	Network specification by a column name in input data consisting of strings that identify the unit's friends by their IDs or their row numbers (two friends are separated by space, e.g., "1 2"; unit with no friends should have an empty "" string). See Details.
verbose	Set to TRUE to print messages on status and information to the console. Turn this on by default using options(tmlenet.verbose=TRUE).
optPars	A named list of additional optional parameters to be passed to tmlenet, such as alpha, lbound, family, n_MCsims, runTMLE, YnodeDET, f_gstar2, sep, f_g0, h_g0_SummariesModel and h_gstar_SummariesModel. See Details below for the description of each parameter.

Value

A named list with 3 items containing the estimation results for:

- EY_gstar1 - estimates of the mean counterfactual outcome under (stochastic) intervention function f_gstar1 ($E_{g_1^*}[Y]$).
- EY_gstar2 - estimates of the mean counterfactual outcome under (stochastic) intervention function f_gstar2 ($E_{g_2^*}[Y]$), NULL if f_gstar2 not specified.

- ATE - additive treatment effect ($E_{g_1^*}[Y]$ - $E_{g_2^*}[Y]$) under interventions f_gstar1 vs. in f_gstar2, NULL if f_gstar2 not specified.

Each list item above is itself a list containing the items:

- estimates - various estimates of the target parameter (network population counterfactual mean under (stochastic) intervention).
- vars - the asymptotic variance estimates, for **IPTW** and **TMLE**.
- CIs - CI estimates at alpha level, for **IPTW** and **TMLE**.
- other.vars - Placeholder for future versions.
- h_g0_SummariesModel - The model fits for P(sA|sW) under observed exposure mechanism g0. This is an object of SummariesModel **R6** class.
- h_gstar_SummariesModel - The model fits for P(sA|sW) under intervention f_gstar1 or f_gstar2. This is an object of SummariesModel **R6** class.

Currently implemented estimators are:

- tmle - Either weighted regression intercept-based TMLE (tmle.intercept - the default) with weights defined by the IPTW weights h_gstar/h_gN or covariate-based unweighted TMLE (tmle.covariate) that uses the IPTW weights as a covariate h_gstar/h_gN.
- h_iptw - Efficient IPTW based on weights h_gstar/h_gN.
- gcomp - Parametric G-computation formula substitution estimator.

Details

Note that in case when both arguments NETIDnode and NETIDmat are left unspecified the input data are assumed independent, i.e., no network dependency between the observations. All inference will be performed based on the i.i.d. efficient influence curve for the target parameter ($E_{g_1^*}[Y]$).

Also note that the ordering of the friends IDs in NETIDnode or NETIDmat is unimportant.

A special non-negative-integer-valued variable nF is automatically calculated each time tmlenet or eval.summaries functions are called. nF contains the total number of friends for each observation and it is always added as an additional column to the matrix of the baseline-covariates-based summary measures sWmat. The variable nF can be used in the same ways as any of the column names in the input data frame data. In particular, the name nF can be used inside the summary measure expressions (calls to functions def.sW and def.sA) and inside any of the regression formulas (Qform, hform.g0, hform.gstar).

The regression formalas in Qform, hform.g0 and hform.gstar can include any summary measures names defined in sW and sA, referenced by their individual variable names or by their aggregate summary measure names. For example, hform.g0 = "netA ~ netW" is equivalent to hform.g0 = "A + A_netF1 + A_netF2 ~ W + W_netF1 + W_netF2" for sW,sA summary measures defined by def.sW(netW=W[[0:2]]) and def.sA(netA=A[[0:2]]).

Additional parameters

Some of the parameters that control the estimation in tmlenet can be set by calling the function tmlenet_options.

Additional parameters can be also specified as a named list optPars argument of the tmlenet function. The items that can be specified in optPars are:

- alpha - alpha-level for CI calculation (0.05 for 95
- lbound - One value for symmetrical bounds on P(sW | sW).
- family - Family specification for regression models, defaults to binomial (CURRENTLY ONLY BINOMIAL FAMILY IS IMPLEMENTED).
- n_MCsims - Number of Monte-Carlo simulations performed, each of sample size nrow(data), for generating new exposures under f_gstar1 or f_gstar2 (if specified) or f_g0 (if specified). These newly generated exposures are utilized when fitting the conditional densities P(sA|sW) and when evaluating the substitution estimators **GCOMP** and **TMLE** under stochastic interventions f_gstar1 or f_gstar2.
- runTMLE - Choose which of the two TMLEs to run, "tmle.intercept" or "tmle.covariate". The default is "tmle.intercept".
- YnodeDET - Optional column name for indicators of deterministic values of outcome Ynode, coded as (TRUE/FALSE) or (1/0); observations with YnodeDET=TRUE/1 are assumed to have constant value for their Ynode.
- f_gstar2 - Either a function or a vector of counterfactual exposure assignments. Used for estimating contrasts (average treatment effect) for two interventions, if omitted, only the average counterfactual outcome under intervention f_gstar1 is estimated. The requirements for f_gstar2 are identical to those for f_gstar1.
- sep - A character separating friend indices for the same observation in NETIDnode.
- f_g0 - A function for generating true treatment mechanism under observed Anode, if known (for example in a randomized trial). This is used for estimating P(sA|sW) under g0 by sampling large vector of Anode (of length nrow(data)*n_MCsims) from f_g0 function;
- h_g0_SummariesModel - Previously fitted model for P(sA|sW) under observed exposure mechanism g0, returned by the previous runs of the tmlenet function. This has to be an object of SummariesModel **R6** class. When this argument is specified, all predictions P(sA=sa|sW=sw) under g0 will be based on the model fits provided by this argument.
- h_gstar_SummariesModel - Previously fitted model for P(sA|sW) under (stochastic) intervention specified by f_gstar1 or f_gstar2. Also an object of SummariesModel **R6** class. When this argument is specified, the predictions P(sA=sa|sW=sw) under f_gstar1 or f_gstar2 will be based on the model fits provided by this argument.

Specifying the counterfactual intervention function (f_gstar1 and optPars$f_gstar2)

The functions f_gstar1 and f_gstar2 can only depend on variables specified by the combined matrix of summary measures (sW,sA), which is passed using the argument data. The functions should return a vector of length nrow(data) of counterfactual treatments for observations in the input data.

Specifying the Network of Friends

The network of friends (connections) for observations in the input data can be specified in two alternative ways, using either NETIDnode or NETIDmat input arguments.

NETIDnode - The first (slower) method uses a vector of strings in data[, NETIDnode], where each string i must contain the space separated IDs or row numbers of all units in data thought to be connected to observation i (friends of unit i);

NETIDmat - An alternative (and faster) method is to pass a matrix with Kmax columns and nrow(data) rows, where each row NETIDmat[i,] is a vector of observation i's friends' IDs or i's friends' row numbers in data if IDnode=NULL. If observation i has fewer than Kmax friends, the remainder of NETIDmat[i,] must be filled with NAs. Note that the ordering of friend indices is irrelevant.

IPTW estimator

****

- As described in the following section, the first step is to construct an estimator $P_{g_N}(sA|sW)$ for the common (in i) conditional density $P_{g_0}(sA|sW)$ for common (in i) unit-level summary measures (sA,sW).
- The same fitting algorithm is applied to construct an estimator $P_{g^*_N}(sA^*|sW^*)$ of the common (in i) conditional density $P_{g^*}(sA^*|sW^*)$ for common (in i) unit-level summary measures (sA^*,sW^*) implied by the user-supplied stochastic intervention f_gstar1 or f_gstar2 and the observed distribution of W.
- These two density estimators form the basis of the IPTW estimator, which is evaluated at the observed N data points $O_i = (sW_i, sA_i, Y_i), i = 1, ..., N$ and is given by

$$\psi_n^{IPTW} = \sum_{i=1,...,N} Y_i \frac{P_{g^*_N}(sA^* = sA_i|sW = sW_i)}{P_{g_N}(sA = sA_i|sW = sW_i)}.$$

GCOMP estimator

****

TMLE estimator

****

Modeling P(sA|sW) for summary measures (sA,sW)

****

Non-parametric estimation of the common **unit-level** multivariate joint conditional probability model P_g0(sA|sW), for unit-level summary measures (sA,sW) generated from the observed exposures and baseline covariates $(A, W) = (A_i, W_i : i = 1, ..., N)$ (their joint density given by $g_0(A|W)Q(W)$), is performed by first constructing the dataset of N summary measures, $(sA_i, sW_i : i = 1, ..., N)$, and then fitting the usual i.i.d. MLE for the common density P_g0(sA|sW) based on the pooled N sample of these summary measures.

Note that sA can be multivariate and any of its components sA[j] can be either binary, categorical or continuous. The joint probability model for P(sA|sA) = P(sA[1],...,sA[k]|sA) can be factorized as P(sA[1]|sA) * P(sA[2]|sA, sA[1]) * ... * P(sA[k]|sA, sA[1],...,sA[k-1]), where each of these conditional probability models is fit separately, depending on the type of the outcome variable sA[j].

If sA[j] is binary, the conditional probability P(sA[j]|sW,sA[1],...,sA[j-1]) is evaluated via logistic regression model. When sA[j] is continuous (or categorical), its range will be fist partitioned into K bins and the corresponding K bin indicators (B_1,...,B_K), where each bin indicator B_j is then used as an outcome in a separate logistic regression model with predictors given by

sW, sA[1],...,sA[k-1]. Thus, the joint probability P(sA|sW) is defined by such a tree of binary logistic regressions.

For simplicity, we now suppose sA is continuous and univariate and we describe here an algorithm for fitting $P_{g_0}(sA|sW)$ (the algorithm for fitting $P_{g^*}(sA^*|sW^*)$ is equivalent, except that exposure A is replaced with exposure A^* generated under f_gstar1 or f_gstar2 and the predictors sW from the regression formula hform.g0 are replaced with predictors sW^* specified by the regression formula hform.gstar).

1. Generate a dataset of N observed continuous summary measures (sa_i:i=1,...,N) from observed ((a_i,w_i):i=1,...,N). Let sa\insa_i:i=1,...,M.
2. Divide the range of sA values into intervals S=(i_1,...,i_M,i_M+1) so that any observed data point sa_i belongs to one interval in S, namely, for each possible value sa of sA there is k\in1,...,M, such that, i_k < sa <= i_k+1. Let the mapping B(sa)\in1,...,M denote a unique interval in S for sa, such that, i_B(sa) < sa <= i_B(sa)+1. Let bw_B(sa):=i_B(sa)+1-i_B(sa) be the length of the interval (bandwidth) (i_B(sa),i_B(sa)+1). Also define the binary indicators b_1,...,b_M, where b_j:=I(B(sa)=j), for all j <= B(sa) and b_j:=NA for all j>B(sa). That is we set b_j to missing ones the indicator I(B(sa)=j) jumps from 0 to 1. Now let sA denote the random variable for the observed summary measure for one unit and denote by (B_1,...,B_M) the corresponding random indicators for sA defined as B_j := I(B(sA) = j) for all j <= B(sA) and B_j:=NA for all j>B(sA).
3. For each j=1,...,M, fit the logistic regression model for the conditional probability P(B_j = 1 | B_j-1=0, sW), i.e., at each j this is defined as the conditional probability of B_j jumping from 0 to 1 at bin j, given that B_j-1=0 and each of these logistic regression models is fit only among the observations that are still at risk of having B_j=1 with B_j-1=0.
4. Normalize the above conditional probability of B_j jumping from 0 to 1 by its corresponding interval length (bandwidth) bw_j to obtain the discrete conditional hazards h_j(sW):=P(B_j = 1 | (B_j-1=0, sW) / bw_j, for each j. For the summary measure sA, the above conditional hazard h_j(sW) is equal to P(sA \in (i_j,i_j+1) | sA>=i_j, sW), i.e., this is the probability that sA falls in the interval (i_j,i_j+1), conditional on sW and conditional on the fact that sA does not belong to any intervals before j.
5. Finally, for any given data-point (sa,sw), evaluate the discretized conditional density for P(sA=salsW=sw) by first evaluating the interval number k=B(sa)\in1,...,M for sa and then computing \prodj=1,...,k-11-h_j(sW))*h_k(sW) which is equivalent to the joint conditional probability that sa belongs to the interval (i_k,i_k+1) and does not belong to any of the intervals 1 to k-1, conditional on sW.

The evaluation above utilizes a discretization of the fact that any continuous density f of random variable X can be written as f_X(x)=S_X(x)*h_X(x), for a continuous density f of X where S_X(x):=P(X>x) is the survival function for X, h_X=P(X>x|X>=x) is the hazard function for X; as well as the fact that the discretized survival function S_X(x) can be written as a of the hazards for s<x: S_X(x)=\prods<xh_X(x).

Three methods for defining bin (interval) cuttoffs for a continuous one-dimenstional summary measure sA[j]

****

There are 3 alternative methods to defining the bin cutoffs S=(i_1,...,i_M,i_M+1) for a continuous summary measure sA. The choice of which method is used along with other discretization parameters (e.g., total number of bins) is controlled via the tmlenet_options() function. See ?tmlenet_options argument bin.method for additional details.

Approach 1 (equal.len): equal length, default.

****

The bins are defined by splitting the range of observed sA (sa_1,...,sa_n) into equal length intervals. This is the dafault discretization method, set by passing an argument bin.method="equal.len" to tmlenet_options function prior to calling tmlenet(). The intervals will be defined by splitting the range of (sa_1,...,sa_N) into nbins number of equal length intervals, where nbins is another argument of tmlenet_options() function. When nbins=NA (the default setting) the actual value of nbins is computed at run time by taking the integer value (floor) of n/maxNperBin, for n - the total observed sample size and maxNperBin=1000 - another argument of tmlenet_options() with the default value 1,000.

Approach 2 (equal.mass): data-adaptive equal mass intervals.

****

The intervals are defined by splitting the range of sA into non-equal length data-adaptive intervals that ensures that each interval contains around maxNperBin observations from (sa_j:j=1,...,N). This interval definition approach can be selected by passing an argument bin.method="equal.mass" to tmlenet_options() prior to calling tmlenet(). The method ensures that an approximately equal number of observations will belong to each interval, where that number of observations for each interval is controlled by setting maxNperBin. The default setting is maxNperBin=1000 observations per interval.

Approach 3 (dhist): combination of 1 & 2.

****

The data-adaptive approach dhist is a mix of Approaches 1 & 2. See Denby and Mallows "Variations on the Histogram" (2009)). This interval definition method is selected by passing an argument bin.method="dhist" to tmlenet_options() prior to calling tmlenet().

See Also

tmlenet-package for the general overview of the package, def.sW for defining the summary measures, eval.summaries for evaluation and validation of the summary measures, and df_netKmax2/df_netKmax6/NetInd_mat for examples of network datasets.

Examples

```
#**********************************************************************************
data(df_netKmax6) # Load observed data
data(NetInd_mat_Kmax6) # Load the network ID matrix
Kmax <- 6 # Max number of friends in the network
#**********************************************************************************

#**********************************************************************************
# EXAMPLES OF INTERVENTION FUNCTIONS:
#**********************************************************************************
# Returns a function that will sample A with probability x:=P(A=1))
```

```
make_f.gstar <- function(x, ...) {
  eval(x)
  f.A_x <- function(data, ...){
    rbinom(n = nrow(data), size = 1, prob = x[1])
  }
  return(f.A_x)
}
# Deterministic f_gstar setting every A=0:
f.A_0 <- make_f.gstar(x = 0)
# Deterministic f_gstar setting every A=1:
f.A_1 <- make_f.gstar(x = 1)
# Stochastically sets (100*x)% of population to A=1 with probability 0.2:
f.A_.2 <- make_f.gstar(x = 0.2)

#*********************************************************************************************
# DEFINING SUMMARY MEASURES:
#*********************************************************************************************
def_sW <- def.sW(netW2 = W2[[1:Kmax]]) +
          def.sW(sum.netW3 = sum(W3[[1:Kmax]]), replaceNAw0=TRUE)

def_sA <- def.sA(sum.netAW2 = sum((1-A[[1:Kmax]])*W2[[1:Kmax]]), replaceNAw0=TRUE) +
          def.sA(netA = A[[0:Kmax]])

#*********************************************************************************************
# EVALUATING SUMMARY MEASURES BASED ON INPUT DATA:
#*********************************************************************************************
# A helper function that can pre-evaluate the summary measures on (O)bserved data,
# given one of two ways of specifying the network:
res <- eval.summaries(sW = def_sW, sA = def_sA,  Kmax = 6, data = df_netKmax6,
                      NETIDmat = NetInd_mat_Kmax6, verbose = TRUE)

#*********************************************************************************************
# Specifying the clever covariate regressions hform.g0 and hform.gstar:
# Left side can consist of any summary names defined by def.sA (as linear terms)
# Right side can consist of any summary names defined by def.sW (as linear terms) & 'nF'
#*********************************************************************************************
hform.g01 <- "netA ~ netW2 + sum.netW3 + nF"
hform.gstar1 <- "netA ~ sum.netW3"

# alternatives:
hform.g02 <- "netA + sum.netAW2 ~ netW2 + sum.netW3 + nF"
hform.g03 <- "sum.netAW2 ~ netW2 + sum.netW3"

#*********************************************************************************************
# Specifying the outcome regression Qform:
# Left side is ignored (with a warning if not equal to Ynode)
# Right side can by any summary names defined by def.sW, def.sA & 'nF'
#*********************************************************************************************
Qform1 <- "Y ~ sum.netW3 + sum.netAW2"
Qform2 <- "Y ~ netA + netW + nF"
Qform3 <- "blah ~ netA + netW + nF"

#*********************************************************************************************
```

```
# Estimate mean population outcome under deterministic intervention A=0 with 6 friends:
# Estimation with regression formulas.
#******************************************************************************************
# Note that Ynode is optional when Qform is specified;
options(tmlenet.verbose = FALSE) # set to TRUE to print status messages
res_K6_1 <- tmlenet(data = df_netKmax6, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", f_gstar1 = 0L,
                    Qform = "Y ~ sum.netW3 + sum.netAW2",
                    hform.g0 = "netA ~ netW2 + sum.netW3 + nF",
                    hform.gstar = "netA ~ sum.netW3",
                    IDnode = "IDs", NETIDnode = "Net_str",
                    optPars = list(n_MCsims = 1))

res_K6_1$EY_gstar1$estimates
res_K6_1$EY_gstar1$vars
res_K6_1$EY_gstar1$CIs

#******************************************************************************************
# Run exactly the same estimators as above,
# but using the output "DatNet.ObsP0" of eval.summaries as input to tmlenet:
#******************************************************************************************
res_K6_1b <- tmlenet(DatNet.ObsP0 = res$DatNet.ObsP0,
                    Anode = "A", f_gstar1 = 0L,
                    Qform = "Y ~ sum.netW3 + sum.netAW2",
                    hform.g0 = "netA ~ netW2 + sum.netW3 + nF",
                    hform.gstar = "netA ~ sum.netW3",
                    optPars = list(n_MCsims = 1))

res_K6_1b$EY_gstar1$estimates
res_K6_1b$EY_gstar1$vars
res_K6_1b$EY_gstar1$CIs

#******************************************************************************************
# Same as above but for covariate-based TMLE.
#******************************************************************************************
res_K6_2 <- tmlenet(data = df_netKmax6, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", f_gstar1 = 0L,
                    Qform = "Y ~ sum.netW3 + sum.netAW2",
                    hform.g0 = "netA ~ netW2 + sum.netW3 + nF",
                    hform.gstar = "netA ~ sum.netW3",
                    IDnode = "IDs", NETIDnode = "Net_str",
                    optPars = list(runTMLE = "tmle.covariate", n_MCsims = 1))

res_K6_2$EY_gstar1$estimates
res_K6_2$EY_gstar1$vars
res_K6_2$EY_gstar1$CIs

#******************************************************************************************
# SPECIFYING THE NETWORK AS A MATRIX OF FRIEND ROW NUMBERS:
#******************************************************************************************
net_ind_obj <- simcausal::NetIndClass$new(nobs = nrow(df_netKmax6), Kmax = Kmax)
# generating the network matrix from input data:
NetInd_mat <- net_ind_obj$makeNetInd.fromIDs(Net_str = df_netKmax6[, "Net_str"],
```

```
                                                IDs_str = df_netKmax6[, "IDs"])$NetInd
nF <- net_ind_obj$nF # number of friends

data(NetInd_mat_Kmax6)
all.equal(NetInd_mat, NetInd_mat_Kmax6) # TRUE

print(head(NetInd_mat))
print(head(nF))
print(all.equal(df_netKmax6[,"nFriends"], nF))

res_K6_net1 <- tmlenet(data = df_netKmax6, Kmax = Kmax, sW = def_sW, sA = def_sA,
                       Anode = "A", f_gstar1 = f.A_0,
                       Qform = "Y ~ sum.netW3 + sum.netAW2",
                       hform.g0 = "netA ~ netW2 + sum.netW3 + nF",
                       hform.gstar = "netA ~ sum.netW3",
                       NETIDmat = NetInd_mat,
                       optPars = list(runTMLE = "tmle.intercept", n_MCsims = 1))

all.equal(res_K6_net1$EY_gstar1$estimates, res_K6_1$EY_gstar1$estimates)
all.equal(res_K6_net1$EY_gstar1$vars, res_K6_1$EY_gstar1$vars)
all.equal(res_K6_net1$EY_gstar1$CIs, res_K6_1$EY_gstar1$CIs)

#**********************************************************************************************
# EQUIVALENT WAYS OF SPECIFYING INTERVENTIONS f_gstar1/f_gstar2.
# LOWERING THE DIMENSIONALITY OF THE SUMMARY MEASURES.
#**********************************************************************************************
def_sW <- def.sW(sum.netW3 = sum(W3[[1:Kmax]]), replaceNAw0=TRUE)
def_sA <- def.sA(sum.netAW2 = sum((1-A[[1:Kmax]])*W2[[1:Kmax]]), replaceNAw0=TRUE)

# can define intervention by function f.A_0 that sets everyone's A to constant 0:
res_K6_1 <- tmlenet(data = df_netKmax6, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", Ynode = "Y", f_gstar1 = f.A_0,
                    NETIDmat = NetInd_mat)
res_K6_1$EY_gstar1$estimates

# equivalent way to define intervention f.A_0 is to just set f_gstar1 to 0:
res_K6_1 <- tmlenet(data = df_netKmax6, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", Ynode = "Y", f_gstar1 = 0L,
                    NETIDmat = NetInd_mat)
res_K6_1$EY_gstar1$estimates

# or set f_gstar1 to a vector of 0's of length nrow(data):
res_K6_1 <- tmlenet(data = df_netKmax6, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", Ynode = "Y", f_gstar1 = rep_len(0L, nrow(df_netKmax6)),
                    NETIDmat = NetInd_mat)
res_K6_1$EY_gstar1$estimates

#**********************************************************************************************
# EXAMPLE WITH SIMULATED DATA FOR AT MOST 2 FRIENDS AND 1 BASELINE COVARIATE W1
#**********************************************************************************************
data(df_netKmax2)
head(df_netKmax2)
Kmax <- 2
```

```
# Define the summary measures:
def_sW <- def.sW(W1[[0:Kmax]])
def_sA <- def.sA(A[[0:Kmax]])
# Define the network matrix:
net_ind_obj <- simcausal::NetIndClass$new(nobs = nrow(df_netKmax2), Kmax = Kmax)
NetInd_mat <- net_ind_obj$makeNetInd.fromIDs(Net_str = df_netKmax2[, "Net_str"],
                              IDs_str = df_netKmax2[, "IDs"])$NetInd

#*****************************************************************************************
# Mean population outcome under stochastic intervention P(A=1)=0.2
#*****************************************************************************************
# Evaluates the target parameter by sampling exposures A from f_gstar1;
# Setting optPars$n_MCsims=100 means that A will be sampled 1000 times (
# for a total sample size of nrow(data)*n_MCsims under f_gstar1)
options(tmlenet.verbose = TRUE)
res_K2_1 <- tmlenet(data = df_netKmax2, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", Ynode = "Y", f_gstar1 = f.A_.2,
                    NETIDmat = NetInd_mat, optPars = list(n_MCsims = 100))
res_K2_1$EY_gstar1$estimates
res_K2_1$EY_gstar1$vars
res_K2_1$EY_gstar1$CIs

#*****************************************************************************************
# Estimating the average treatment effect (ATE) for two static interventions:
# f_gstar1 sets everyone A=1 vs f_gstar2 sets everyone A=0;
#*****************************************************************************************
res_K2_2 <- tmlenet(data = df_netKmax2, Kmax = Kmax, sW = def_sW, sA = def_sA,
                    Anode = "A", Ynode = "Y",
                    f_gstar1 = 1,
                    NETIDmat = NetInd_mat,
                    optPars = list(f_gstar2 = 0, n_MCsims = 1))
names(res_K2_2)

# Estimates under f_gstar1:
res_K2_2$EY_gstar1$estimates
res_K2_2$EY_gstar1$vars
res_K2_2$EY_gstar1$CIs

# Estimates under f_gstar2:
res_K2_2$EY_gstar2$estimates
res_K2_2$EY_gstar2$vars
res_K2_2$EY_gstar2$CIs

# ATE estimates for f_gstar1-f_gstar2:
res_K2_2$ATE$estimates
res_K2_2$ATE$vars
res_K2_2$ATE$CIs
```

tmlenet_options *Setting Options for* tmlenet

Description

Additional options that control the estimation algorithm in tmlenet package

Usage

```
tmlenet_options(useglm = FALSE, parfit = FALSE,
  bin.method = c("equal.len", "equal.mass", "dhist"), nbins = NA,
  maxncats = 20, poolContinVar = FALSE, maxNperBin = 1000)
```

Arguments

useglm	Set to FALSE to estimate with speedglm.wfit and TRUE for glm.fit.
parfit	Default is FALSE. Set to TRUE to use foreach package and its functions foreach and dopar to perform parallel logistic regression fits and predictions for discretized continuous outcomes. This functionality requires registering a parallel backend prior to running tmlenet function, e.g., using doParallel R package and running registerDoParallel(cores = ncores) for integer ncores parallel jobs. For an example, see a test in "./tests/RUnit/RUnit_tests_04_netcont_sA_tests.R".
bin.method	The method for choosing bins when discretizing and fitting the conditional continuous summary exposure variable sA. The default method is "equal.len", which partitions the range of sA into equal length nbins intervals. Method "equal.mass" results in a data-adaptive selection of the bins based on equal mass (equal number of observations), i.e., each bin is defined so that it contains an approximately the same number of observations across all bins. The maximum number of observations in each bin is controlled by parameter maxNperBin. Method "dhist" uses a mix of the above two approaches, see Denby and Mallows "Variations on the Histogram" (2009) for more detail.
nbins	Set the default number of bins when discretizing a continous outcome variable under setting bin.method = "equal.len". If left as NA the total number of equal intervals (bins) is determined by the nearest integer of nobs/maxNperBin, where nobs is the total number of observations in the input data.
maxncats	Max number of unique categories a categorical variable sA[j] can have. If sA[j] has more it is automatically considered continuous.
poolContinVar	Set to TRUE for fitting a pooled regression which pools bin indicators across all bins. When fitting a model for binirized continuous outcome, set to TRUE for pooling bin indicators across several bins into one outcome regression?
maxNperBin	Max number of observations per 1 bin for a continuous outcome (applies directly when bin.method="equal.mass" and indirectly when bin.method="equal.len", but nbins = NA).

Value

Invisibly returns a list with old option settings.

See Also

print_tmlenet_opts

Index